MECANISMOS DE PROTEÇÃO
EM SISTEMAS DE COMPUTADOR

Leila Sodero Lima de Rezende

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO EM ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS (M.Sc.)

Aprovada por:

PIERRE JEAN LAVELLE
Presidente

Ten.Cel. ALMIR PAZ DE LIMA

SUELI MENDES DOS SANTOS

LUIZ ANTONIO CARNEIRO
DA CUNHA COUCEIRO

RIO DE JANEIRO, RJ - BRASIL
JULHO DE 1981

REZENDE, LEILA SODERO LIMA DE
Mecanismos de Proteção em Sistemas de Computador (Rio de Janeiro) 1981.
XIII, 261p. 29,7 cm (COPPE-UFRJ, M.Sc., Engenharia de Sistemas, 1981)

Tese - Universidade Federal do Rio de Janeiro. Faculdade de Engenharia

1. Mecanismos de proteção para a segurança de dados em sistemas de computador

I. COOPE/UFRJ II. Título (série).

DEDICATÓRIA

Dedicamos este trabalho a todas as pessoas que sentiram em algum instante de sua vida as conseqüências de uma falha de proteção no computador.

AGRADECIMENTOS

Agradecemos a todas as pessoas que com desprendimento nos transmitiram suas experiências legando à comunidade científica uma quantidade enciclopédica de artigos, sem os quais teria sido impossível a realização desse trabalho.

Também ao professor Pierre Jean Laveile por sua orientação objetiva e racional e ao professor Gerhard Schwarz por seu incentivo e desafio constantes.

RESUMO

Sistemas de computadores constituem alvo frequente de ataques para obtenção de privilégios a níveis pessoal, empresarial e nacional. São analisados os componentes de cada sistema de computador com o objetivo de se identificar suas vulnerabilidades e propor salvaguardas. Um conjunto amplo de mecanismos de proteção em projeto ou em uso no mundo é apresentado, analisado e criticado. Finalmente, é proposta uma classificação desses mecanismos segundo padrões de discrecionaridade.

ABSTRACT

Computer systems constitue a frequent target of attack to achieve privileges at personnel, corporate and national levels. Computer systems components are analysed with the objective of identifying their vulnerabilities and propose safeguards. A wide range of protection mechanisms in project or in use at world is presented, analysed and criticised. Finally, a classification of those mechanisms is proposed, according to discrecionarity patterns.

SIGLAS

ARPA	-	Advanced Research Projects Agency
ARPANET	-	ARPA Computer Network
CDC	-	Control Data Corporation
DBMS	-	Data Base Management System
FJCC	-	Fall Joint Computer Conference
ICS	-	Inventory Control System (IBM)
IMF	-	Instalation Management System (IBM)
IMS	-	Information Management System (IBM)
IPC	-	Interprocess Communication
JCL	-	Job Control Language (IBM)
LNS	-	Local Name Space (HYDRA)
MPC	-	Master Control Program (Burroughs)
NBS	-	National Bureau of Standards
NCC	-	National Computer Conference
NOS	-	Network Operating System
NRCC	-	National Research Concil of Canada
NSC	-	Network Security Center
NSF	-	National Science Foundation
OS/370	-	Operating System (IBM/370)
ROM	-	Read Only Memory
SDC	-	Systems Development Corporation
SJCC	-	Spring Joint Computer Conference
SRI	-	Stanford Research Institute
TCP	-	Transmission Control Protocol
TIP	-	Terminal Interface Processor
UCC	-	User Controled Cryptography
VM/370	-	Virtual Machine (IBM/370)

PALAVRAS EM INGLÊS

Pedimos licença aos apologistas da língua portuguesa para exprimir sem aspas, apóstrofe ou itálico termos da lingua inglesa que se tornaram corriqueiros no ambiente científico, para os quais a tradução ou é inexistente ou diminui poder para a palavra original. Outros termos em inglês, não tão técnicos nem tão específicos são utilizados entre apóstrofes. A seguir, uma lista das palavras em inglês que serão encontradas no texto:

back-up
batch
bit
buffer
'bypass'
byte

cache
call
capability
clip
cluster

dead-lock
default
dump

entry-point

facility
fault
front-door
front-end

'handshaking'
hardware
'house-keeping'

input
I/O
job

label
link
login
loop

'need-to-know'

off-line
offset
on-line
'one-way'
output
overflow
overhead

page-fault
password
'pipe'
pointer
'pool'
procedure
profile

scheduler
software
stack
staff
stream

tag
'ticket'
'turn-around'

working-set

ÍNDICE DE FIGURAS

Pág.

Fig 2.1 - Relação entre Segurança e Integridade da informação 7

Fig.3.1 - Matriz de acesso em um universo de 3 sujeitos e 9 objetos 18
Fig 3.2 - Estrutura interna de um mecanismo baseado em matriz de acesso 19
Fig 3.3 - Instruções que modificam a matriz de acesso 21
Fig 3.4 - Lista de capabilities de um job 23
Fig 3.5 - Capability etiquetada no sistema BCC-500 25
Fig 3.6 - Efeito da transmissão de capability na matriz de acesso 26
Fig 3.7 - Comparação entre mecanismos de capabilities e de listas de acesso 28
Fig 3.8 - Lista de chaves e fechaduras com atributos de acesso 31
Fig 3.9 - Mecanismo de chave-fechadura no CAL-TSS 32

Fig 4.1 - Níveis de classificação militares 37
Fig 4.2 - Conjuntos de categorias militares 38
Fig 4.3 - Elementos do modelo multinível 39
Fig 4.4 - Níveis de arquitetura do sistema KSOS 41
Fig 4.5 - Treliça de fluxo seguro de informações 43

Fig 5.1 - Mapeamento no mecanismo de memória virtual 51
Fig 5.2 - Endereçamento no sistema IBM S/38 58
Fig 5.3 - Diretórios para arquivos permanentes 61
Fig 5.4 - Um diretório paralelo ao organograma de uma empresa 62
Fig 5.5 - Comparação entre segmentação lógica e física de um arquivo 66

Fig 6.1 - Descritor de segmento 75
Fig 6.2 - Segmento de descritores 75
Fig 6.3 - Descritor de segmento contendo número de anel 79
Fig 6.4 - Capabilities de memória 82
Fig 6.5 - Capabilities de entrada 82
Fig 6.6 - Confinamento no I/O de uma procedure 91
Fig 6.7 - Operação na Data Mark Machine 91

Fig 7.1 - Domínios de execução em arquiteturas hierárquicas 97
Fig 7.2 - Inclusão do software do VM/370 na hierarquia de um sistema 98
Fig 7.3 - Hierarquia de processos e divisão da memória do RC 4000 99

Pág.

Fig 7.4 - Arquitetura do UCLA SECURE UNIX 111
Fig 7.5 - Arquitetura do KVM/370 117
Fig 7.6 - Estrutura do KSOS 118

Fig 8.1 - Testes de segurança para processamento de transação 122
Fig 8.2 - Esquema de validação de acesso discrecionário 125
Fig 8.3 - Grafo de autorização do sistema R 133
Fig 8.4 - Operação de recuperação no UCLA INGRES 134

Fig 9.1 - Vulnerabilidades de uma rede de computadores 142
Fig 9.2 - Algoritmo de criptografia do DES 150
Fig 9.3 - Transmissão de chaves entre dois nós de uma rede 152
Fig 9.4 - Visão do usuário da rede 153
Fig 9.5 - Fluxo de dados em canais criptografados processo-a-processo 156
Fig 9.6 - Configuração de uma rede com NSC 158
Fig 9.7 - Conexão entre KSOS e rede multinível 160

Fig A.1 - Arquitetura do PSOS 191

Fig B.1 - Extensões de objetos no HYDRA 197
Fig B.2 - Sistema distribuído MEDUSA 199
Fig B.3 - Organização de um CM 199
Fig B.4 - Terminal multinível do SIGMA 204
Fig B.5 - Configuração inicial do XNOS 206
Fig B.6 - Lista de acesso de um arquivo no XNOS 207

C O N T E Ú D O

Pag.

I. - INTRODUÇÃO 2

II. - FUNDAMENTOS

1. Conceitos Básicos 6
2. Mecanismos de Proteção e Sistemas Operacionais 9
3. Características de um Mecanismo de Proteção 12
4. Componentes de um Mecanismo de Proteção 15

III. - MECANISMOS DISCRECIONÁRIOS

1. Modelo da Matriz de Acesso 18
2. Capabilities 23
3. Listas de Acesso 28
4. Chaves-fechaduras 31

IV. - MECANISMOS NÃO-DISCRECIONÁRIOS

1. Modelo Multinível 35
2. Níveis de Segurança e Arquiteturas Hierárquicas 40
3. Controle de Fluxo 42
4. Momento de Ligação 46

V. - PROTEÇÃO DA INFORMAÇÃO ARMAZENADA

1. Memória Virtual 50
2. Arquiteturas Etiquetadas 53
3. Microprogramação 56
4. Diretórios 60
5. Acesso Lógico a Arquivos 64
6. Criptografia Aplicada a Arquivos 68

VI. - EXECUÇÃO PROTEGIDA DE PROCESSOS

1. Processos 72
2. Domínios e Segmentação 74
3. Anéis de Proteção 78
4. Domínios Definidos por Capabilities 81

Pag.

5. Comunicação e Cooperação 85
6. Confinamento 89

VII. - SISTEMAS NUCLEADOS

1. Arquiteturas Hierárquicas e Nucleadas 95
2. Núcleos de Segurança 101
3. Núcleos Discrecionários - Evolução Histórica 106
4. Núcleos Não-discrecionários - Evolução Histórica 113

VIII. - BANCOS DE DADOS

1. Controle de Acesso em Bancos de Dados 121
2. Modelos Discrecionários 124
3. Modelos Não-discrecionários 128
4. Mecanismos para BD Relacionais 131
5. Mecanismos para BD Estatísticos 135

IX. - SISTEMAS DISTRIBUÍDOS

1. Proteção em Redes 141
2. Autenticação 143
3. Criptografia 148
4. Sistemas Operacionais para Redes 153
5. Controle Centralizado 157
6. Controle Descentralizado 160

X. - RESUMO E CONCLUSÕES

1. Resumo 165
2. Conclusões 167

Pag.

APÊNDICE A - SISTEMAS DE HARDWARE CONCENTRADO

A.1 - ADEPT-50 170
A.2 - BCC-500 172
A.3 - MULTICS 175
A.4 - CAL-TSS 178
A.5 - UCLA UNIX 182
A.6 - KSOS 185
A.7 - S/38 188
A.8 - PSOS 191

APÊNDICE B - SISTEMAS DE HARDWARE DISTRIBUÍDO

B.1 - HYDRA 195
B.2 - MEDUSA 199
B.3 - SIGMA 203
B.4 - XNOS 206

BIBLIOGRAFIA 210

ÍNDICE DE PALAVRAS-CHAVE 223

I - INTRODUÇÃO

I - INTRODUÇÃO

Presentemente os danos realizados através de computador atingem mais de 200 milhões de dólares anuais somente nos Estados Unidos. Os conhecedores do assunto estimam que isso é apenas a ponta de um iceberg já que menos de 1% de todas as fraudes por computador são detetadas, e quando o são as companhias mostram-se embaraçadas em torná-las públicas.

Os problemas de segurança associados a um complexo sistema de computadores não dizem respeito apenas aos atos intencionalmente realizados em benefício próprio. Há casos em que, acidentalmente, toda uma comunidade pode ser posta em perigo. Por exemplo no ano de 1980 os computadores do Departamento de Defesa dos Estados Unidos emitiram vários alarmes falsos anunciando a captação de sinais provenientes de mísseis soviéticos. Por duas vezes a guerra esteve prestes a estourar.

Deliberados ou não, os atos que causam a exposição, modificação ou destruição de informação sensível devem ser prevenidos. As características especiais que compõem esses sistemas de computadores, como: grande quantidade de informação simultaneamente disponível a vários usuários, acesso de longas distâncias e retenção de resíduos mesmo depois de parcialmente utilizados, fazem da segurança uma meta difícil de ser atingida.

Os mecanismos de proteção empregados nos sistemas de computadores são às vezes complexos e caros. Entretanto, se o problema for equacionado na sua raíz, soluções simples e baratas podem ser encontradas. Sistemas inseguros não podem se tornar magicamente seguros pela adição de um ou mais mecanismos de proteção, como não se pode tornar saudável um organismo doente através de um simples remédio.

A expansão dos sistemas, a distribuição do processamento, a ligação de vários componentes em redes tornaram o problema da segurança mais sério. O tamanho ao qual os sistemas chegaram, impede muitas vêzes, que se façam testes completos e acompanhamento das suas tarefas. Técnicas de certificação e metodologias de especificação formal ajudam os projetistas a testar cada item relacionado à proteção do sistema.

Muito tem sido pesquisado na área de segurança de computador. Desde a confiabilidade de cada microelemento do hardware até às características psicológicas do profissional de processamento de dados o caminho é extenso e emocionante. De ambos os prismas: dos que se protegem e dos que querem atacar. Cada detalhe é importante, uma pequena fresta pode ser fatal. Mesmo que não haja danificação na informação, mesmo que não haja escoamento de segredos, a perda de um simples arquivo pode significar a destruição de meses de trabalho realizado por toda uma equipe.

Hoje em dia é difícil encontrar uma geração de computadores que não tenha dispositivos de segurança, providos por uma tecnologia em contínuo aperfeiçoamento. Todos já se conscientizaram de que o computador é uma ferramenta e uma arma, tanto a nível indi-

dividual como a empresarial e internacional. Invenções de ficção científica como espiões eletrônicos se tornaram realidade em tempo inferior ao que levaram para ser imaginadas. Instrumentos tão potentes não podem cair em mãos de pessoas inescrupulosas ou "ingênuas".

A pesquisa para esta Tese se situou na análise dos mecanismos de proteção implementados em sistemas desenvolvidos ao longo desses quinze últimos anos. Esses mecanismos, criados para equipamentos, sistemas e ambientes diferentes foram comparados com o objetivo de encontrar os elementos essenciais de cada aproximação. Características comuns foram filtradas e reunidas a fim de se estabelecer um conjunto básico correspondente aos componentes principais de um mecanismo de proteção. Além de procurar os elementos estruturais, analisamos as modulações criadas em torno dêles e sua validade.

Esta Tese foi escrita para o estudante ou profissional de processamento de dados que, tendo uma visão sistêmica geral não tem necessariamente conhecimento detalhado de cada assunto tratado. Assim, ao analisar mecanismos de proteção, por exemplo, de um banco de dados, partimos da definição de banco de dados, características e modelos, para em seguida atingir o assunto que nos diz respeito. Essa preocupação foi uma constante durante todo o nosso trabalho, e embora deixe margem para críticas pelos especialistas, optamos por um conteúdo menos elitista e mais didático. De certa forma, essa decisão foi praticamente uma necessidade pois dada à amplitude dos assuntos enfocados seria irreal supor conhecimento prévio de cada um deles.

A Tese está dividida em oito capítulos, dez se considerarmos a Introdução e a Conclusão. O capítulo II contém os fundamentos para os seguintes. São apresentadas definições de termos básicos em segurança, características principais e componentes de um mecanismo de proteção. O capítulo III introduz os mecanismos de proteção discrecionários apresentando uma razoável base teórica e alguns exemplos. O capítulo IV repete o mesmo para o caso dos mecanismos não-discrecionários. No capítulo V são analisados vários mecanismos de proteção da informação armazenada, estaticamente falando. No capítulo VI essa mesma informação é considerada dinâmicamente, durante a execução dos processos do sistema. O capítulo VII enfoca uma nova tendência na construção de mecanismos de proteção, a de agrupá-los numa parte privilegiada do sistema operacional, o núcleo, onde desfrutam de todas as vantagens que o poder lhes confere e de onde controlam o funcionamento de todo o sistema. No capítulo VIII são apresentados mecanismos de proteção específicos para Banco de Dados e no capítulo IX os específicos para Sistemas Distribuídos.

Os Apêndices, como na maioria dos casos, são parte integrante desse trabalho e na versão inicial estavam incluídos no corpo da Tese. Entretanto, com o objetivo de separar MECANISMOS de SISTEMAS achamos por bem criar os Apêndices, contendo a descrição de alguns sistemas estudados. Na realidade estes constituem o subconjunto mais detalhadamente analisado do conjunto total, que inclui em torno de 40 sistemas, como se verá pelas referências.

Esta Tese, sendo tanto uma análise como uma síntese, é antes de tudo, a exemplo do assunto estudado, uma proposta preventiva de uma mentalidade de segurança. Está na hora de conscientizarmos a nossa sociedade, em especial os que trabalham direta ou indiretamente em computadores, da necessidade de se utilizar mecanismos de proteção. Este é, sem dúvida o nosso objetivo primordial.

II - F U N D A M E N T O S

II.1 CONCEITOS BÁSICOS

No contexto de proteção da informação em computadores existem alguns conceitos que devem ser especificados antes de se proceder às referências.

1. ACESSO - A habilidade de fazer uso de uma informação armazenada em um sistema de computador. Tipos de acesso são: ler, escrever, copiar, alterar, executar, etc.

2. AUTORIZAÇÃO - A atitude de conceder o ACESSO à informação através de um conjunto de regras previamente estabelecidas. Exemplo: autorização para ler.

3. POLÍTICA DE SEGURANÇA - Um conjunto finito de regras que delimitam o ACESSO à informação durante a execução de uma computação. Políticas de segurança são implementadas através de mecanismos de proteção. Exemplo: privacidade é uma política de segurança que regula a disseminação de informação relativa a uma pessoa ou a um grupo de pessoas.

4. POLÍTICA DE INTEGRIDADE - Um conjunto de regras que especificam a ALTERAÇÃO da informação durante a execução de uma computação. Como no caso da política de segurança, existem mecanismos de proteção que suportam a política de integridade e estes podem ser um subconjunto daqueles (já que ALTERAÇÃO é um tipo de ACESSO).

5. SEGURANÇA X INTEGRIDADE - Uma violação da segurança pode implicar ou não em uma violação na integridade, embora a possibilidade de uma modificação correta da informação após uma violação na segurança seja remota. O diagrama da Fig. 2.1 explica porque.

6. MECANISMO DE PROTEÇÃO - O conjunto de facilidades de hardware e software colocado à disposição dos programadores como base para a implementação de políticas.

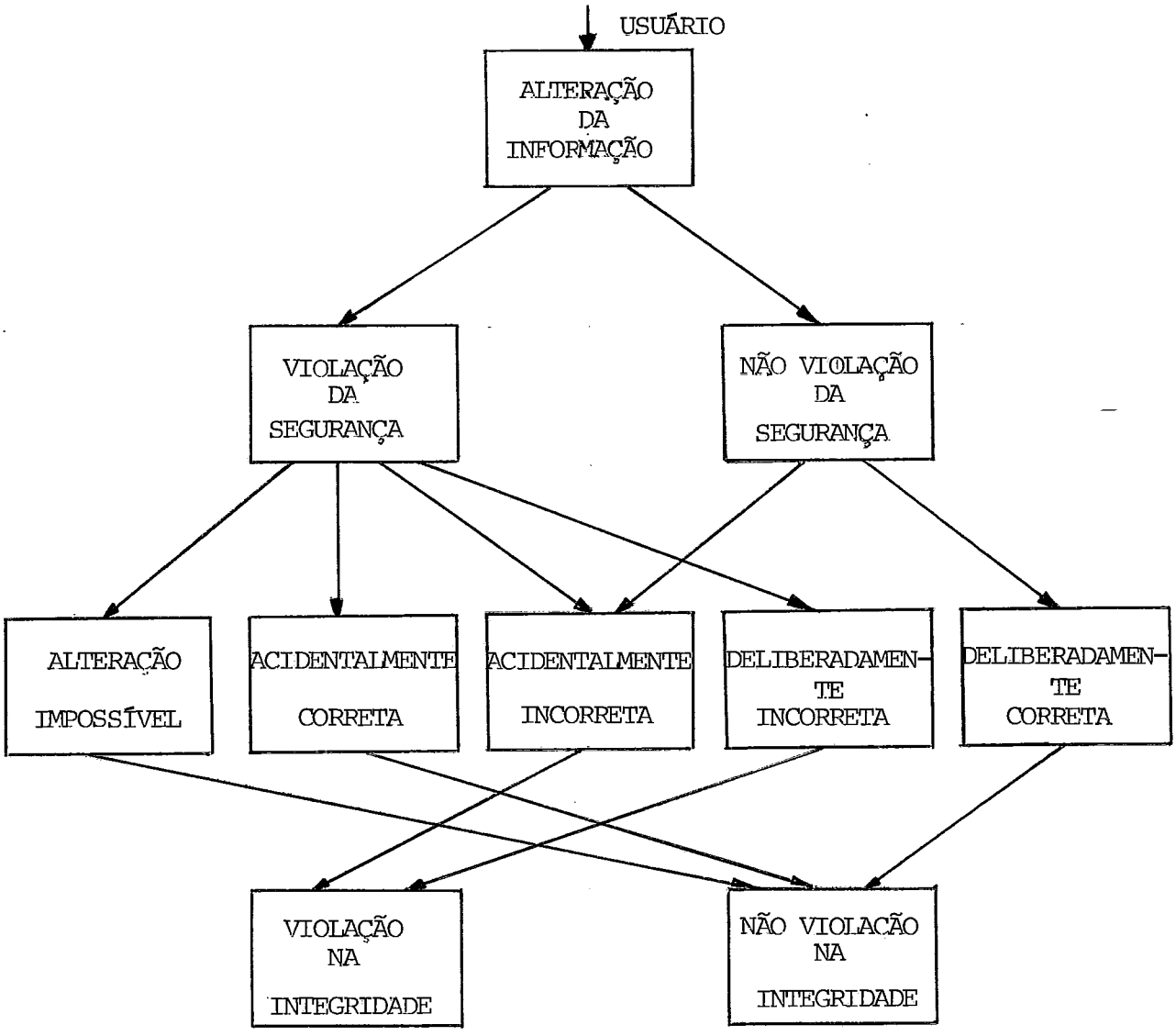


Fig.2.1 Relação entre Segurança e Integridade da Informação(Sha 77)

7. DISCRECIONARIDADE - É a propriedade de um mecanismo segundo a qual o ACESSO aos recursos do sistema é especificado pelos próprios usuários individualmente ou por grupos de usuários, de modo que satisfaçam suas necessidades particulares de utilização do sistema.

8. NÃO-DISCRECIONARIDADE - É a propriedade de um mecanismo segundo a qual todos os usuários e recursos do sistema são compartimentalizados em níveis pré-fixados e o ACESSO só é concedido em acordância com a hierarquia estabelecida.

9. CONFIABILIDADE - É a propriedade de um mecanismo que desempenha comprovadamente as funções que lhe são atribuídas.

10. RECUPERABILIDADE - É a propriedade de um mecanismo que após qualquer erro, acidente, desastre ou queda do sistema pode ser restaurado sem causar perda de serviço ou de segurança.

11. AUDITABILIDADE - É a propriedade que envolve a monitoração contínua da segurança e confiabilidade de um mecanismo incluindo a deteção de comportamento anômalo ou potencialmente perigoso.

II.2 MECANISMOS DE PROTEÇÃO E SISTEMAS OPERACIONAIS

Peter Denning (Den 71) definiu um sistema de computação em termos das várias funções de controle e supervisão que provê para os processos criados pelos seus usuários, onde o termo PROCESSO denota um programa em execução. O conjunto de funções realizadas pelo sistema pode ser especificamente dividido em 7 grupos:

1. Criação e remoção de processos.

2. Controle do progresso dos processos, isto é, garantindo que cada processo evolua a uma taxa positiva e não bloqueie o progresso dos outros.

3. Alocação de recursos de hardware para os processos, exemplo: dispositivos de I/O.

4. Alocação de recursos de software, exemplos: arquivos, editores, compiladores, assemblers, bibliotecas de rotinas e sistemas de programação.

5. Conexão entre processos através dos meios de comunicação inter processuais.

6. Tratamento das condições excepcionais que surgem durante a execução dos processos, exemplos: erros aritméticos ou de máquina, interrupções, instruções ilegais ou privilegiadas.

7. Proteção e controle de acesso à informação.

O software que realiza ou auxilia o hardware a realizar essas funções é chamado Sistema Operacional.

Enquanto que os seis primeiros grupos de funções estão intrinsecamente ligados à própria existência do sistema, o sétimo grupo apresenta uma característica de opcionalidade que pode ser comprovada pela existência de um grande número de sistemas operacionais, aceitos e utilizados em larga escala, que não possuem as funções do sétimo grupo.

O conceito de proteção em sistemas de computação começou a aparecer contemporaneamente aos sistemas de multiprogramação. Anteriormente, os recursos do computador como CPU, memória principal, sistemas de arquivos e periféricos eram utilizados simultaneamente por um só usuário, que blocava um certo tempo durante o qual a máquina lhe era dedicada. Assim, programas, arquivos, e todo o material sensível poderiam ser protegidos pelos métodos tradicionais, uma vez que não necessàriamente permaneciam armazenados no próprio sistema de computação.

Com o advento dos sistemas de multiprogramação, a possibilidade de se compartilhar recursos da máquina paralelamente entre vários usuários, trouxe a necessidade de se protegê-los de uma maneira apropriada. Inicialmente a proteção era incluída na tarefa dos

schedulers ou gerentes de recursos que, ao alocarem cada recurso, testavam a "competência" do processo solicitante. Não se poderia dizer que isso fosse realmente um mecanismo de proteção, melhor seria considerá-lo apenas um 'housekeeping'.

Dos primeiros gerentes de recursos até hoje muito tem sido pesquisado na área de sistemas operacionais seguros, a maioria das pesquisas financiadas pelo Departamento de Defesa dos Estados Unidos como se pode ver pelas referências. O sistema operacional é o responsável pelo acesso aos recursos da máquina. É através das suas decisões e ordens que todo o complexo de equipamentos desempenha cada função específica. Assim, uma vez obtido o controle do sistema operacional, pode-se conseguir qualquer informação, bastando que ela exista em algum lugar do sistema.

Teoricamente existem 2 alternativas para se atingir maior segurança num sistema operacional, uma basicamente corretiva e uma basicamente preventiva (Neu 78). A primeira envolve o acesso à estrutura dos sistemas existentes para descobrir falhas de segurança e a tentativa de cobrir as trajetórias que levam a essas falhas. A segunda envolve o projeto de novos sistemas que são intrinsecamente seguros e cuja segurança pode ser comprovada de alguma maneira convincente.

Com respeito à aproximação CORRETIVA várias técnicas tem sido utilizadas para descobrir erros que reduzem a segurança. Estas incluem codificação de pesquisa para certos modelos de instruções dos programas correspondentes a chaves de prováveis falhas e também as tentativas tradicionais de penetração operacional. Entretanto, várias razões levam a crer que esta aproximação não é a melhor:

- não é necessário procurar muito para descobrir falhas nos sistemas existentes.
- trajetórias que tentam remover essas falhas, são elas mesmas frequentemente falhas.
- tentativas de caracterizar as falhas não detetadas são no máximo apenas especulativas.

Assim, em geral, adaptações de segurança possuem efetividade limitada e inevitavelmente deixam muitas incógnitas.

A longo prazo o uso da aproximação PREVENTIVA é significantemente mais produtivo. Esta aproximação significa que o sistema foi projetado com o objetivo fundamental de prover segurança. Deve incluir identificação explícita dos propósitos e especificação formal do projeto. Deve usar uma linguagem de programação moderna e deve empregar provas formais das propriedades mais significantes para o comportamento do sistema, exemplo: segurança. Pesquisas recentes mostram que é possível assegurar a segurança de um projeto especificado precisamente antes da sua implementação e depois assegurar a segurança da implementação.

Existe porém um problema de custo na aproximação preventiva. Milhares de instalações de computadores em todo o mundo estariam

dispostas a pagar pela segurança das suas aplicações mas não a modificá-las para compatibilizá-las com novos sistemas. Os custos de modificação, recompilação, redocumentação, treinamento, etc, relativos a mudança de software ou de hardware são elevados demais na maioria das situações. Para evitá-los existe uma 3a. alternativa que é uma combinação das aproximações corretiva e preventiva. O sistema existente é reprojetado de tal forma que suas partes relevantes para a segurança são isoladas em uma pequena e claramente definida coleção de programas chamada núcleo. Entretanto, a efetividade dessa combinação pode também ser reduzida pelas restrições impostas ao projeto como por exemplo a necessidade de manter compatibilidade com a interface original insegura. Pode também confundir POLÍTICA com MECANISMOS, tornando o sistema específico para um número restrito de políticas. Mas de qualquer forma, em muitos casos esta alternativa já foi escolhida e satisfaz às necessidades específicas a que se propõe.

Concluindo, podemos observar a intensa modificação pela qual passaram os mecanismos de proteção num sistema operacional: de "função opcional" para NÚCLEO. Agora já se sabe que um mecanismo de proteção é tanto mais seguro quanto mais fundo estiver no sistema do computador. Muitos são implementados em firmware e muitos também no próprio hardware. Sistemas de computadores não podem mais se dar ao luxo de possuir Security Facilities opcionais, como no passado, pois a segurança só pode existir se estiver na sua base.

Nos capítulos seguintes examinaremos as 2 últimas alternativas e os artifícios que foram utilizados para resolver suas vicissitudes.

II.3 CARACTERÍSTICAS DE UM MECANISMO DE PROTEÇÃO

Num sistema de computador um mecanismo de proteção deve ser capaz de controlar internamente a interação entre os vários usuários do sistema, garantindo separação quando requerida, permitindo cooperação irrestrita quando desejada e proporcionando tantos degraus de controle intermediário quantos forem necessários.

Os múltiplos usuários de um computador possuem diferentes objetivos e são responsáveis por diferentes níveis de autoridade. Um grupo tão diverso só poderá usar o mesmo sistema se este for geral o suficiente para satisfazer as necessidades específicas de cada um, proporcionando-lhes a capacidade de proteger seus próprios bens e de liberar o acesso, de maneira controlada, a outros usuários.

O projeto dos mecanismos de proteção é a sua fase mais importante. Nela são definidas as técnicas que levarão à implementação das políticas de segurança. Nela são especificadas formalmente as estruturas que serão mais tarde implementadas de acordo com o que foi projetado.

Independentes da política, do mecanismo e da técnica empregados, existem algumas características que devem orientar o projeto de todo mecanismo de proteção. Estas foram surgindo a partir de erros cometidos e compõem agora o conjunto das características necessárias para tornar a construção do sistema mais rápida e para tornar o mecanismo mais seguro e compatível. São as seguintes:

1. FUNCIONALIDADE - O mecanismo deve ser capaz de atender às necessidades do conjunto de usuários de uma maneira correta e natural.

2. SIMPLICIDADE - Mecanismos simples podem ser exaustivamente testados e todas as trajetórias de dados podem ser investigadas.

3. PROJETO NÃO SECRETO - Os mecanismos de proteção não podem depender da ignorância dos seus usuários, que na realidade são pessoas de altíssimo nível intelectual. O projeto deve ser divulgado, debatido em palestras, para apresentar todos os problemas antes da implementação. A proteção real deve depender dos parâmetros, estes sim, secretos, passíveis de serem alterados periodicamente. Esta separação entre mecanismos e parâmetros, embora primitiva, enviou um número surpreendente de projetos de volta à prancheta de desenho.

4. EFICIÊNCIA DE DESEMPENHO - Embora os mecanismos de proteção ocasionem implicitamente um aumento no overhead geral do sistema, este deve ser o mínimo possível. Especialmente em sistemas que são projetados para substituir outros, comparações são inevitáveis.

5. ECONOMIA - O custo da implementação de um tipo qualquer de restrição de acesso não deve ser fator importante na determinação da política de segurança, portanto, os mecanismos de proteção devem ser capazes de realizar economicamente qualquer uma delas.

6. MÁXIMO APROVEITAMENTO DO HARDWARE - Um software "feito sob medida" para um hardware específico pode não ser comercializável independentemente, mas as vantagens em simplicidade de algoritmos, economia de linhas de programação e rapidez na execução compensam plenamente.

7. FALTA DE ACESSO COMO DEFAULT - Se os mecanismos são baseados em permissão ao invés de em exclusão, os próprios usuários comunicarão erros eventuais que lhes impeçam de acessar recursos aos quais tem direito.

8. COMPLETARIDADE - Cada tentativa de acesso deve ser validada, o mecanismo não pode "memorizar" resultados de testes anteriores e conceder acesso baseado nessas informações, pois isso cria um sério embaraço nas situações de revogação de direitos.

9. MÍNIMO DE PRIVILÉGIOS - Derivado do 'need to know' militar, esse princípio postula que cada usuário do sistema deve utilizar somente o mínimo necessário de privilégios para realizar seu trabalho. Assim, o número de interações potenciais entre programas privilegiados é menor, o sistema fica menos exposto e a auditoria é mais fácil.

10. COMPARTILHAMENTO MÍNIMO - Cada objeto compartilhado é uma trajetória em potencial entre usuários, podendo ser utilizada para escoamento ilícito de informação. Além disso, funções que servem a vários usuários tem maior dificuldade em manter a satisfação individual e podem levar à construção de portas falsas driblando o mecanismo de proteção.

11. NATURALIDADE NA INTERFACE HUMANA - O uso rotineiro e automático dos mecanismos de proteção é uma característica essencial, caso contrário canais "simplificadores" serão automaticamente construídos burlando os critérios pré-estabelecidos.

12. DESCENTRALIZAÇÃO DAS ESPECIFICAÇÕES DE PROTEÇÃO - Por dois motivos: para evitar "gargalos" nas decisões administrativas que se tomadas centralmente poderiam gerar 'bypass' nas situações de urgência e para não aumentar a vulnerabilidade do sistema, que é tanto maior quanto mais concentrada for.

13. FLEXIBILIDADE - Deixar margem para inclusões, adaptações, implementações de novas políticas, criação de ambiente de proteção privativo do usuário e de subsistemas protegidos que possibilitem a implementação de qualquer esquema de controle de acesso.

14. CERTIFICAÇÃO - Provar que o projeto do sistema atende às especificações formais e que a implementação atende ao projeto é o mesmo que provar que a implementação do sistema atende à especificação formal. Isto é chamado certificação e já existem alguns sistemas cujos mecanismos de proteção foram totalmente certificados.

Para finalizar, algumas considerações sobre a confiabilidade do hardware. O hardware é o prérequisito para a construção de qualquer sistema, e em se tratando de um sistema seguro, é indispensável que tenha o mínimo de erros. A diferença fundamental entre a confiabilidade do hardware e do software é que um componente de hardware nunca pode ser completamente livre de erros. Ambos são suscetíveis a erros algorítmicos, mas o hardware é também suscetível a erros probabilísticos. Estes representam as falhas imprevisíveis dos elementos e a proteção contra elas é geralmente obtida através da redundância (estática e dinâmica), deteção e localização de erros, tolerância de faltas e reconfiguração/recuperação dinâmica. O sistema PRIME, desenvolvido em Berkeley por Fabry é um excelente exemplo de um hardware construído para manter disponibilidade, privacidade e custo efetivos através da redundância (Fab 73).

II.4 COMPONENTES DE UM MECANISMO DE PROTEÇÃO

Tradicionalmente um sistema formal é dividido em três partes:

- uma sintaxe para descrever elementos e propriedades de elementos.
- uma semântica descrevendo os exemplos concretos dos elementos e dando um significado às sentenças da sintaxe.
- uma teoria de provas para raciocínios sobre os elementos e suas propriedades.

Uma lógica de proteção é um sistema formal cujos elementos são os componentes do mecanismo de proteção. Nesta seção serão descritos alguns componentes de um sistema de proteção, os mais comuns. Os exemplos concretos serão vistos nos outros capítulos e também nos apêndices. Os raciocínios estão também dispersos nos outros capítulos, mas não nos arriscamos ainda a elaborar uma teoria de provas. Os primeiros passos nesse sentido ainda estão sendo dados e achamos melhor esperar por eles.

Os componentes mais comuns de um mecanismo de proteção são:

1. OBJETOS - São elementos do sistema operacional que podem ser acessados e cujo acesso deve ser controlado pelo mecanismo de proteção. Cada objeto deve estar univocamente associado a uma identificação que lhe é atribuída no momento da sua criação. Exemplos de objeto são: programas e arquivos de usuário, páginas de memória principal, dispositivos de I/O, etc.

2. EXTENSÕES DE OBJETOS - Existem dois tipos de objetos: os primitivos, produzidos diretamente pelo sistema e as extensões, que são objetos criados pelos usuários. Extensões de objetos são por exemplo sistemas de arquivos especiais, subsistemas protegidos, etc. Além de tornar o sistema mais flexível as extensões transferem ao usuário parte da responsabilidade na proteção dos seus próprios objetos.

3. SUJEITOS - São os elementos ativos do sistema, ou seja: os que podem acessar objetos, como por exemplo os programas, terminais, etc. Sujeitos também podem ser acessados, portanto são um subconjunto dos objetos e devem possuir identificação única.

4. PROCESSOS - Dos componentes de um mecanismo de proteção, o processo é o mais importante, por ser um elemento ativo, portanto um sujeito, e por ser na maioria das vezes o responsável por todas as funções realizadas no sistema. Exemplos de processos são: programas de usuário, gerentes de memória e de I/O, etc.

5. REGRAS DE ACESSO - É um conjunto de regras explicitamente especificadas que definem o tipo de acesso autorizado entre o conjunto de sujeitos e o conjunto de objetos. Devem ser simples, completas e flexíveis para gerarem um ambiente de proteção compatível com as aspirações do grupo de usuários. Exemplos de regras são: autorização para ler, para executar, etc.

6. NÍVEL DE SEGURANÇA - É a combinação de uma categoria de classificação ordenada hierarquicamente e de um conjunto de compartimentos. Sujeitos e objetos podem ser distribuídos em vários níveis de segurança, que caracterizam os privilégios de acesso de cada grupo. Exemplos: SECRETO, NÃO DISSEMINÁVEL A ESTRANGEIROS, etc.

7. DOMÍNIO DE EXECUÇÃO - É o conjunto de elementos que um processo em execução pode acessar com seus respectivos direitos de acesso. Um sujeito pode ser visto como um par (processo, domínio) em que o processo é um programa em execução e o domínio é o ambiente de proteção em que o processo atua.

8. SUBSISTEMA PROTEGIDO - É uma coleção de objetos de procedures e dados que é encapsulado em um domínio de si mesmo de maneira que a estrutura interna dos seus objetos de dados só é acessível através dos seus objetos de procedure, que só podem ser chamados em pontos de entrada protegidos.

9. NÚCLEO - A reunião de todos os mecanismos de proteção numa parte protegida do sistema. É um artifício utilizado para simplificar a depuração e a certificação de sistemas seguros.

10. CAPABILITY - Uma referência protegida a um objeto do sistema. Contém normalmente o endereço e os direitos de acesso ao objeto.

III - MECANISMOS DISCRECIONÁRIOS

III.1 MODELO DA MATRIZ DE ACESSO

Um dos modelos mais influentes nos mecanismos de proteção é o da matriz de acesso, desenvolvido por Lampson e complementado por Graham e Denning (GrD 72). A base desse modelo é uma matriz cujos elementos contêm as regras de acesso de SUJEITOS a OBJETOS. A escolha dos sujeitos, objetos e regras de acesso é feita à discreção do projetista, de maneira a preencher as necessidades específicas de cada sistema; devido a isso, é um modêlo discrecionário e pode ser utilizado na construção de mecanismos de proteção discrecionários.

Numa matriz de acesso o conjunto de sujeitos é representado pelas linhas da matriz s_1, s_2... e o conjunto de objetos pelas suas colunas x_1, x_2.... Cada entrada da matriz A (s,x) contém cadeias, chamadas ATRIBUTOS DE ACESSO que especificam os direitos ou privilégios de acesso de cada elemento. Como foi definido no capítulo anterior, sujeitos são também objetos, portanto devem constar das colunas da matriz (fig 3.1).

	s_1	s_2	s_3	f_1	f_2	p_1	p_2	t_1	t_2
s_1	propried. controle	* prop.		* ler	propried. ler	acordar	acordar	ler escrever	
s_2			controle	* escrever	executar				ler
s_3					escrever	parar		escrever	

* Indica autorização de "cópia" do atributo

Fig. 3.1 - Matriz de acesso em um universo de 3 sujeitos e 9 objetos (Den 71).

Associado a cada tipo de objeto, existe um monitor, através do qual o acesso a este tipo de objeto é validado. Exemplos de monitores são o SISTEMA DE ARQUIVOS para os arquivos, o HARDWARE para a execução de instruções e endereçamento de memória e o SISTEMA DE PROTEÇÃO para os sujeitos. Assim, quando um sujeito s_i solicita um tipo de acesso α ao objeto x_j o monitor checa a matriz para verificar se $\exists \alpha$ em A (s_i, x_j). Basicamente o conjunto de etapas percorridas entre a solicitação e a obtenção do acesso é o seguinte:

1. O sujeito s solicita acesso ao objeto x da maneira α .

2. O sistema de proteção ativa o monitor de x e lhe fornece o triplet(s, α , x).

3. O monitor de x vai à matriz de proteção e verifica se existe α em A (s, x). Se positivo o acesso é concedido, se negativo é sinalizada uma violação de proteção.

Pode-se observar que os atributos de acesso são examinados pelo monitor do objeto a cada tentativa de acesso, o que é uma característica importante num mecanismo de proteção. A organização deste mecanismo pode ser representada pelo diagrama da fig. 3.2, onde os testes de acesso são invisíveis para os sujeitos, mas estão presentes em cada solicitação.

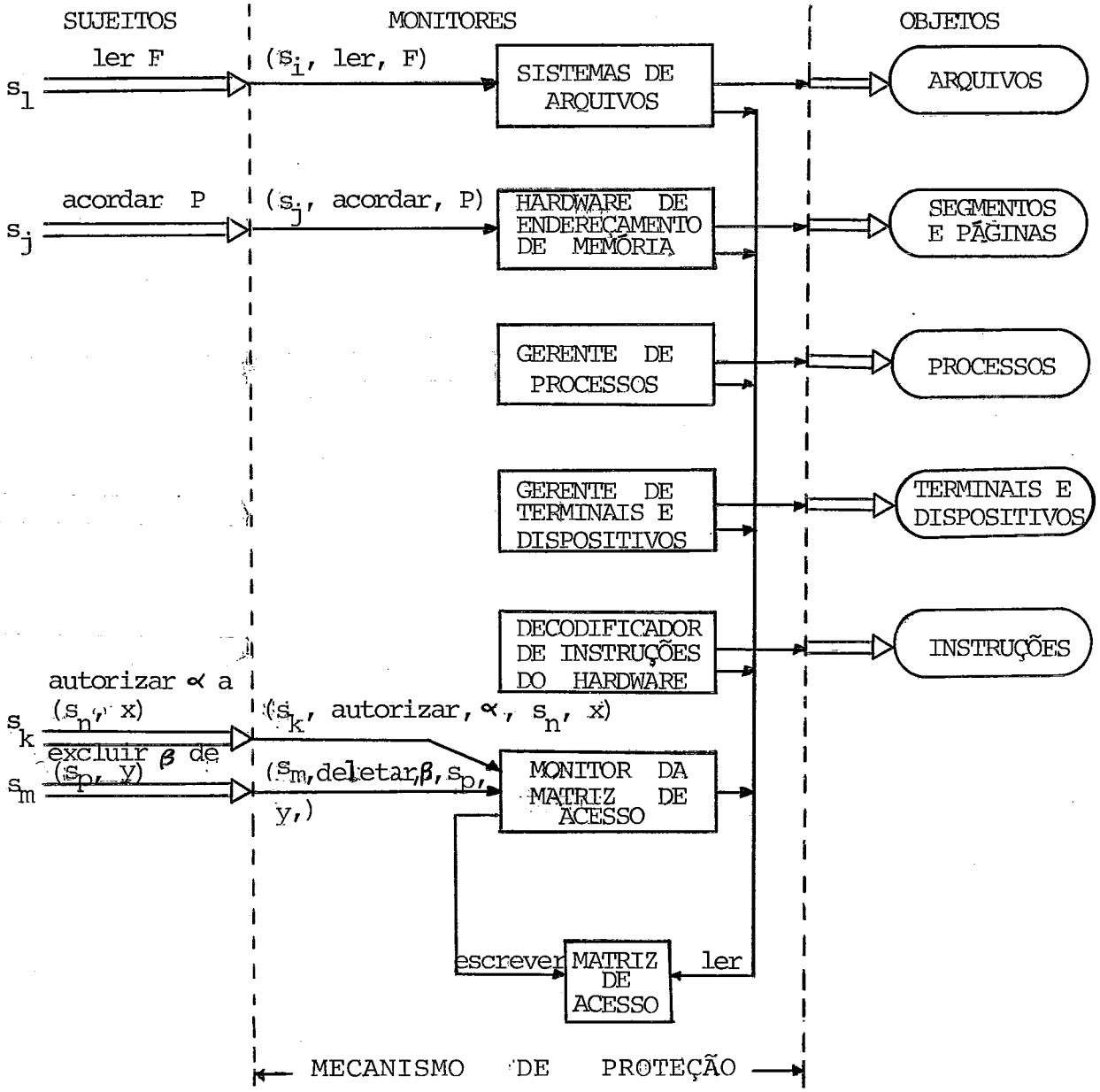


Fig. 3.2 Estrutura interna de um mecanismo baseado em matriz de acesso (GrD 72).

É conveniente interpretar toda a informação especificando os tipos de acesso de sujeitos a objetos como constituindo um ESTADO de proteção do sistema. Uma vez que a matriz de acesso é dinâmica, cada alteração na matriz significa uma mudança no estado de proteção. É condição fundamental que qualquer mudança sempre leve a outro estado protegido, portanto, alterações na matriz devem ser cuidadosamente controladas. Para isto, existe o monitor da própria matriz e da sua exatidão depende todo o sistema de proteção. Na organização da fig. 3.2, por exemplo, a matriz pode ser lida a partir de qualquer monitor de objetos, mas modificada apenas pelo seu próprio monitor.

As regras que gerenciam alterações na matriz de acesso são implementadas através de um conjunto de operações permitidas.

Por exemplo:

Transferir $\propto$ a A(s, x) - autoriza o sujeito a transferir os direitos de acesso que ele possui para outro sujeito.

Incluir $\propto$ em A(s, o) - autoriza um sujeito a incluir direitos de acesso em outro sujeito.

Muitas outras operações são facultadas a sujeitos na alteração da matriz de acesso, como inclusão e deleção de sujeitos e objetos, etc.

A idéia lançada por Lampson inclui o uso de direitos denominados PROPRIEDADE, CONTROLE e CÓPIA como regras para criar, deletar e transferir atributos de acesso em uma matriz. O direito de propriedade pode ser exercido para qualquer objeto (inclusive sujeitos) e o direito de controle apenas para sujeitos. Desses direitos decorrem as seguintes autorizações:

1. C R I A Ç Ã O - Um sujeito s pode adicionar qualquer direito de acesso a A (s', x) para qualquer s' se s tem direito de propriedade sobre o objeto x . Por exemplo, na fig. 3.1 s_1 pode adicionar "controle" em A (s_2, s_2) , ou "ler" em A (s_3, f_2) porque s_1 tem PROPRIEDADE sobre s_2 e f_2.

2. D E L E Ç Ã O - Um sujeito s pode deletar qualquer direito de acesso de A (s', x) para qualquer x se s tem direito de controle sobre o sujeito s'. Por exemplo, s_2 pode deletar "escrever" de A (s_3 , f_2) e "parar" de A (s_3 , p_1) porque s_2 tem CONTROLE sobre s_3.

3. TRANSFERÊNCIA - Se * $\propto$ aparece em A (s, x), s pode colocar * $\propto$ ou $\propto$ em A (s', x) para qualquer s'. Por exemplo, s_1 pode colocar "ler" em A (s_2, f_1) ou "*propriedade" em

A (s_3, s_2) porque s_1 tem * (CÓPIA) em A (s_1, f_1) e A (s_1, s_2).

(OBS: a cópia é feita na mesma coluna).

O asterisco de cópia é necessário para prevenir qualquer sujeito não privilegiado de passar adiante atributos de acesso que lhe foram concedidos. Na figura 3.3 é encontrado um resumo das instruções protegidas que modificam a matriz de acesso.

Comando (por s_0)	Autorização(em $A(s_0,x)$)	Operação resultante
Transferir $\{\alpha, \alpha^*\}$ a $A(s,x)$	α^*	armazenar $\{\alpha, \alpha^*\}$ em $A(s,x)$
Autorizar $\{\alpha, \alpha^*\}$ a $A(s,x)$	proprietário	armazenar $\{\alpha, \alpha^*\}$ em $A(s,x)$
Deletar α de $A(s,x)$	controle ou proprietário	deletar α de $A(s,x)$
Ler $A(s,x)$	controle ou proprietário	copiar $A(s,x)$
Criar objeto x	nenhuma	adicionar coluna x em A armazenar "proprietário" em $A(s_0,x)$
Destruir objeto x	proprietário	deletar coluna x de A
Criar sujeito s	nenhuma	adicionar linha e col. s em A armazenar "controle" em $A(s_0,s)$
Destruir sujeito s	proprietário	deletar linha e col. s de A

Fig. 3.3 - Instruções que modificam a matriz de acesso (GrD 72)

Graham defendeu a tese de que não é necessário haver distinção entre os direitos de propriedade e controle. Além disso, a noção de transferência pode ser extendida para incluir um modo TRANSFERÊNCIA PURA, no qual o direito de acesso transferido desaparece do sujeito que transfere. Por exemplo, se o símbolo "." indica transferência pura, o sujeito s pode colocar $.\alpha$ ou α em $A(s',x)$ sempre que $.\alpha$ aparecer em $A(s,x)$, mas nesse caso, $.\alpha$ é deletado de $A(s,x)$. Pode-se desejar limitar o número de proprietários de um objeto em exatamente 1. Assumindo que cada objeto tem inicial

mente um proprietário esta condição será perpetuada permitindo apenas ". propriedade" ou "propriedade", mas não "*propriedade".

É importante notar que um modelo serve para ajudar a compreensão do funcionamento lógico de um sistema e não implica em nenhuma implementação particular. Assim, regras de acesso não precisam ser armazenadas na forma de uma matriz. De fato, esta seria uma solução muito ineficiente, porque em geral, a matriz de proteção é esparsa e qualquer sujeito tem acesso apenas a um pequeno subconjunto do conjunto de objetos.

Existem no mínimo 3 implementações práticas do modelo da matriz de acesso. A primeira usa a idéia de armazenar a matriz A por linhas, isto é, com cada sujeito s associado a uma lista de pares $(x, A(s, x))$, onde cada par é chamado CAPABILITY e a lista é chamada C-lista (lista de capabilities). A segunda, usa a idéia de armazenar a matriz A por colunas, isto é, com cada objeto x associado a uma lista de pares $(s, A(s, x))$, chamada LISTA DE ACESSO. A terceira aproximação, CHAVE-FECHADURA, é uma combinação das duas primeiras, pois cada sujeito tem uma C-lista e cada objeto uma lista de acesso. A C-lista é composta por entradas (x,K) onde K é uma chave e a lista de acesso é composta por entradas (L, α) onde L é uma fechadura e α um atributo de acesso. O acesso é concedido quando a chave K "abrir" a fechadura L. Essas três implementações são estudadas em detalhe nas próximas seções.

Finalmente, utilizando o modêlo da matriz de proteção, Harrison, Ruzzo e Ullman (HRU 76) desenvolveram um modelo formal de sistemas de proteção e demonstraram que em casos restritos pode ser demonstrado se um sistema é seguro, mas isto se torna impossível de uma maneira generalizada. Surpreendentemente e sob hipóteses bastante fracas, encontraram que não pode ser decidido se um sistema geral possui situações seguras. Isto acontece porque os mecanismos de proteção são empregados à critério dos usuários e liberam para estes os direitos sobre seus objetos. Na realidade, se formos considerar êsse ponto de vista, nenhum sistema é seguro no sentido de guardar hermeticamente objetos, mas apenas possibilita aos usuários que tenham seus objetos "sob controle".

Particularmente interpretamos a tese acima como uma consequência normal de qualquer esquema com tendências liberalizantes: a responsabilidade de proteção, que era potencialmente do sistema, fica descentralizada sobre seus usuários. É essa a filosofia básica de qualquer mecanismo de proteção discrecionário.

III.2 - CAPABILITIES

Provavelmente a palavra mais usada nos meios ligados à segurança de computadores na última década tenha sido CAPABILITY. Tantos foram os sistemas, as teorias, as teses de mestrado e doutorado, que a idéia de capability é aplicada em pelo menos metade dos mecanismos de proteção de hardware e de software.

Basicamente, capability é um ticket que referencia um objeto e contém um conjunto de direitos de acesso. A posse de uma capability é considerada como uma evidência de que o sujeito pode acessar o objeto das formas e apenas dessas formas descritas pela capability.

A idéia de capability foi lançada em 1966 por Dennis e Van Horn (DeV 66) ao descrever mecanismos de proteção para sistemas de multiprogramação. Para controlar o acesso aos objetos do sistema, particularmente aos segmentos de memória, os autores sugeriram uma lista de capabilities, em que cada capability era constituída por 4 partes:

- um índice numérico - através do qual a capability era chamada pelos processos.
- um indicador de propriedade - O para o objeto possuído, N para não possuído.
- um pointer contendo o endereço do objeto.
- vários indicadores de acesso - X para executar, R para ler e W para escrever.

O acesso a cada objeto protegido, por exemplo, um segmento de memória, era obtido pelo fornecimento do índice numérico da capability correspondente ao segmento. Cada job era constituído por um conjunto de processos possuindo a mesma C-lista (fig. 3.4), e 2 processos com independentes C-listas nunca poderiam fazer parte do mesmo job, mesmo se elas fossem idênticas.

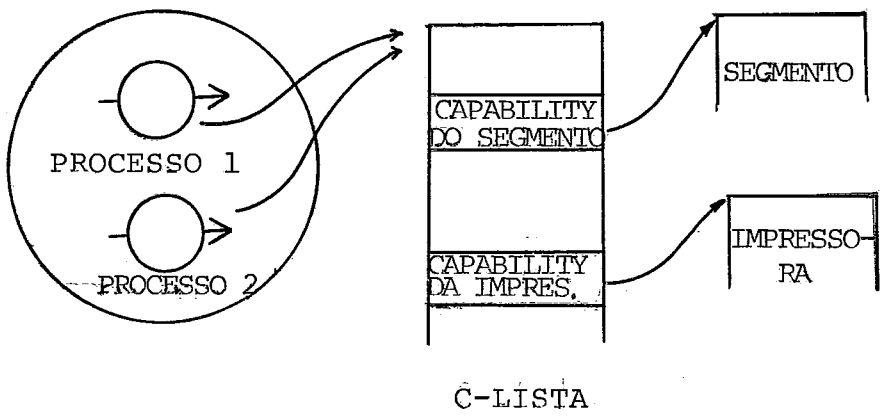


Fig. 3.4 - Lista de capabilities de um job

Uma vez que os dispositivos periféricos também eram recursos físicos do sistema, as funções de I/O também deveriam ser controladas por capabilities da C-lista do job. Durante a execução de um job, capabilities poderiam ser adicionadas ou deletadas da C-lista através de um mecanismo de esferas de proteção em que o processo executante "preparava" a esfera do processo seguinte, transferindo-lhe algumas de suas capabilities com autoridade semelhante ou restringida.

Finalmente, cada usuário ou grupo de usuários do sistema chamado PRINCIPAL podia ter um grupo de "objetos retidos", independentes da execução de jobs, os quais poderiam ser segmentos de memória, funções de I/O, entradas ou diretórios organizados em árvores, onde cada entrada continha o nome do componente e uma capability para ele.

Posteriormente muitas modificações foram feitas nos conceitos iniciais de capabilities (Lin 76). Embora diferentes sistemas as usem de formas bastante diversas, capabilities em geral possuem 3 propriedades:

- Dado que existem capabilities para um objeto do sistema, êsse é o único caminho possível para se chegar a ele.
- Uma parte da capability determina os modos de acesso para o objeto.
- A criação e modificação das capabilities estão subordinadas a uma parte do sistema de baixo nível e acesso restrito.

Para impedir a criação de capabilities por usuários não autorizados que através delas ganhariam acesso a objetos, existem 2 possibilidades: partição reservada e etiqueta.

O método da PARTIÇÃO RESERVADA consiste em manter na memória algumas localizações especiais como segmentos e registros apenas para capabilities, cujo acesso é vedado aos usuários. A solução por partição é usada na Chicago Magic Number Machine e no Plessey System 250. Para cada segmento é definido no instante da sua criação se o mesmo deve conter capabilities ou dados. Além disso, existe um grupo de registros de processador para dados e um para capabilities. O conjunto de instruções satisfazem regras como as seguintes: dados só podem ser copiados em segmentos de dados, capabilities só podem ser copiadas em segmentos de capabilities, etc.

O método da ETIQUETA consiste em incluir em cada palavra da memória um bit extra ou um conjunto de bits, inacessíveis aos usuários, que identificam se a palavra é uma capability, caso em que apenas o sistema de proteção tem permissão para modificá-la. Usam etiquetas o Burroughs 6700, o Rice, a Basic Machine, entre outros. O S/38 da IBM utiliza etiquetas para autenticar pointers. Nesse sistema, as capabilities são pointers, munidos de direitos de acesso, que podem ser transcritos e armazenados indefinidamente para uso posterior (Ber80). Também o BCC-500 utiliza etiquetas para validar capabilities, cuja estrutura é a da fig. 3.5.

ETIQUETA	TIPO	VALOR	ATRIBUTOS DE ACESSO

Fig. 3.5 - Capability etiquetada no sistema BCC-500 (Lam 69)

A etiqueta é legível apenas pelo supervisor. O tipo contém o tipo do objeto, por exemplo, arquivo. O valor contém o endereço, por exemplo o índice no disco. Os atributos de acesso são os modos permitidos de acesso ao objeto (Lam69).

As duas alternativas, partição e etiqueta são equivalentes, no sentido de que uma estrutura representada por uma delas pode ser transladada para equivalente na outra. Qual das duas é a melhor? Fabry (Fab 74) apresenta as vantagens e desvantagens da partição reservada. É mais simples para suportar diferentes formatos de capabilities e dados, o que é importante pois as capabilities são longas: em geral 64 bits. Além disso permitem fácil alocação, recuperação e modificação pelo sistema de proteção, pois estão todas em locais conhecidos, eliminando os mecanismos de memória necessários para trabalhar com etiquetas. Entretanto, como a maioria dos objetos necessita simultaneamente capability e dados, dois segmentos precisam ser manuseados em vez de apenas um, como no caso das etiquetas.

A coexistência de capabilities e dados no mesmo objeto também pode ser obtida sem a existência de arquiteura etiquetada como propõe Anita Jones (Jon 80) pela introdução de uma CERCA de separação entre dados e capabilities. Esta cerca é usada para testes de verificação de limite, os quais garantem que as capabilities são manipuladas apenas por operações apropriadas do sistema.

O índice numérico criado por Dennis e Van Horn para identificar cada capability numa C-lista, atualmente se chama identificador e não necessita ser numérico, podendo ser implementado de duas maneiras:

- o identificador é um pointer para o objeto, contendo seu endereço e limite, ou alternativamente um pointer para uma tabela de indireção ou paginação.

- o identificador é um código único em todo o sistema e está permanentemente associado ao objeto, sendo chamado identificador único (Lin 76).

Na primeira maneira, existe o inconveniente de se precisar atualizar periodicamente as capabilities sempre que o objeto mudar de endereço (ou as tabelas de indireção). A segunda maneira tem sido mais utilizada, embora tenha o inconveniente de ser 'one-way' cada identificador tem que ser único para toda a vida do sistema e seu tamanho varia em torno de 50 bits.

Frequentemente um sujeito tem permissão para transmitir uma cópia ou outorgar uma capability a outro sujeito. Na matriz de proteção isto significa a inclusão das regras de acesso da capability na interseção da linha do novo sujeito com a coluna do objeto, como na fig 3.6.

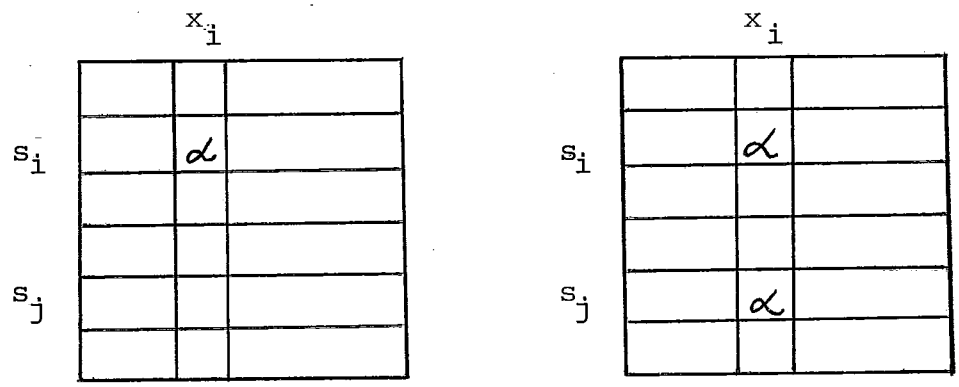


Fig. 3.6 - Efeito da transmissão de capability na matriz de acesso (Pop 74)

No sistema HYDRA, por exemplo, as capabilities podem ser passadas de um usuário para outro, inclusive podendo ser retidas por usuários entre sessões de terminais. Objetos não possuem "proprietários". Todos os portadores de capabilities para um objeto dividem o controle entre si na proporção dos seus direitos. É claro, se um usuário retiver para si todos os direitos em relação a um objeto, se torna proprietário na prática. Apenas o núcleo do sistema pode manipular capabilities, assim, é impossível falsificar uma capability ou obter acesso a um objeto sem possuí-la. Além disso, ao contrário da maioria dos sistemas baseados em capabilities, no HYDRA, não apenas os sujeitos, mas também os objetos podem conter C-listas, por exemplo, um diretório pode conter capabilities para objetos do tipo arquivo e semáforo (CoJ 75, Jon 80).

Um ponto importante envolvendo transmissão de capabilities diz respeito à REVOGAÇÃO de direitos. Em algumas situações é necessário revogar os direitos de acesso que tenham sido previamente concedidos. Se as capabilities que representam esses direitos tiverem sido copiadas muitas vezes, medidas especiais precisam ser tomadas. A mais simples de todas é fazer uma cópia do objeto e deletar a versão antiga, mas isso pode destruir os direitos de acesso de outros sujeitos autorizados e causar a existência de lixo na memória representado pelas capabilities de "objetos-fantasmas". No sistema CAL-TSS as capabilities não apontam para objetos diretamente, mas para uma entrada de uma tabela global, a qual aponta para o objeto. A revogação é realizada simplesmente pela deleção da entrada na tabela global. Torna-se necessário obter uma nova capability para o objeto e distribuí-la pelos sujeitos que permanecem autorizados. As entradas na tabela podem ser reusadas sem risco, porque tanto a capability como a entrada na tabela contém o mesmo nome interno do objeto, que é verificado. Mas este sistema causa o aparecimento de diferentes cópias de uma capability apontando para a mesma entrada (LaS 76).

Revogação seletiva desativando apenas as capabilites desejadas pode ser feita construindo-se um grande número de pointers retroativos, que acompanham a trajetória das capabilities entre os processos. Isto foi feito na 15a. revisão do Multics para refletir as mudanças de uma entrada no diretório diretamente nos descritores de registros, mas é uma alternativa cara.

A revogação seletiva pode ser implementada de uma forma mais econômica, pela criação de capabilities revogáveis que apontam para um objeto indiretamente através da capability principal do objeto. A capability revogável pode ser desativada sem alterar a principal.

Outra solução, contida na tese de Anita Jones é a deleção periódica de todas as capabilities dos processos forçando a sua reaquisição. É controvertida no aspecto de que o período de atividade de cada processo é muito pequeno e a revogação só pode ocorrer quando o processo é suspenso ou encerrado. Por exemplo, não seria possível ao proprietário de um arquivo revogar o acesso de um processo que já o tenha aberto e o esteja utilizando (Lin 76).

Finalmente, as capabilities podem ser colocadas em "caixas fechadas" e passar como num correio entre processos que tem autorização para transmití-las mas não para exercê-las até um processo final que as recebe e utiliza. Capabilities especialmente aplicadas à criação de domínios protegidos para processos são vistas na seção VI.4.

III. 3 - LISTAS DE ACESSO

O armazenamento da matriz de acesso por colunas dá origem às LISTAS DE ACESSO, às vezes chamadas listas de controle de acesso. A cada objeto é associada uma lista contendo nome de todos os sujeitos que podem acessá-lo com os respectivos modos de acesso. Listas de acesso não necessitam conter endereços e pointers como no caso das capabilities, embora isso possa acontecer. A fig.3.7 mostra a diferença entre as duas formas de armazenamento.

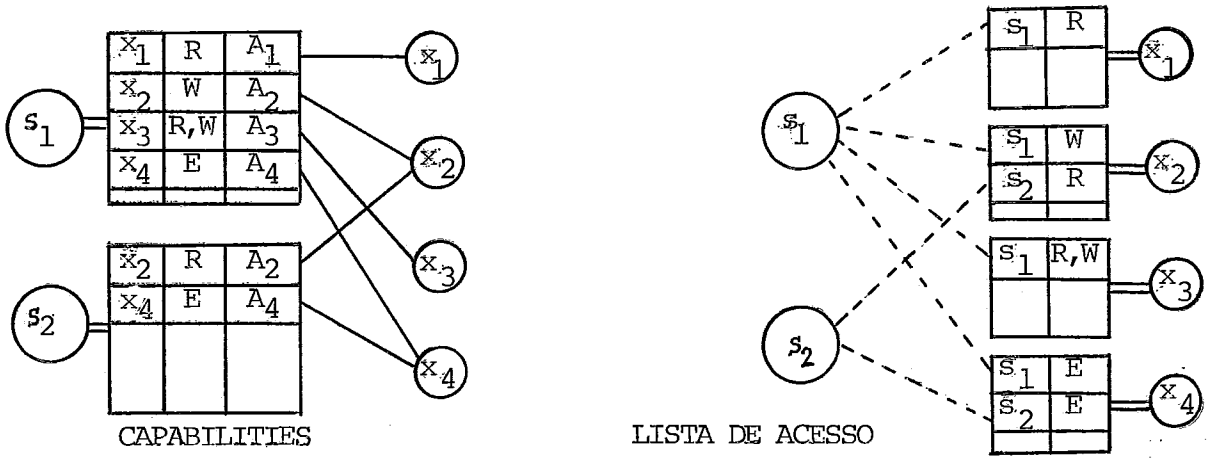


Fig. 3.7 - Comparação dos mecanismos de capabilities e de listas de acesso.

A grande vantagem do mecanismo de listas de acesso é a reversibilidade dos ligamentos, tão difícil de ser executada no mecanismo de capabilities. Para se modificar ou excluir o acesso de um sujeito, basta alterar ou eliminar esse sujeito da lista de acesso do objeto.

O exemplo mais simples de uma lista de acesso é encontrado KSOS (CaD 79). Embora este sistema tenha sido projetado especificamente para implementar o mecanismo de multiníveis não-discricionários, razões de flexibilidade ocasionaram a criação de uma pequena lista de acesso para cada objeto. Esta lista é composta por apenas 9 bits que especificam os tipos de acesso: ler, escrever e executar/pesquisar para o proprietário, outros usuários do mesmo nível, todos os outros usuários. Além de ser simples, é eficiente e barata, pois a maior parte do overhead é transferida para o controle dos níveis.

Listas de acesso trazem à tona um problema muito característico: o controle de quem pode modificar a informação de controle de acesso. No mecanismo de capabilities, esta consideração está difusa, pois um sujeito que possui uma capability pode ter a autoridade de propagá-la. Na lista de acesso, o controle está mais preciso e centralizado, o que é vantagem, mas também exige uma forma eficiente de ser exercido. O objetivo principal deste mecanismo é gerar dentro do computador uma estrutura de autorização que reflita ou modele a estrutura da organização que utiliza o computador. Basicamente, 2 políticas podem ser modeladas: autocontrole e controle hierárquico (SaS 75).

AUTO-CONTROLE é o esquema mais simples. Neste, o conceito de bits de permissão pode ser extendido para incluir não apenas autorização para ler e escrever, mas também autorização para modificar a própria lista de acesso em que estão contidos. Suponhamos por exemplo que a criação de um novo objeto esteja condicionada à criação de uma lista de acesso cuja entrada inicial contém todas as permissões para o sujeito que a criou. Assim, o criador recebe um pointer para esta lista e pode construí-la contendo quaisquer sujeitos com quaisquer direitos de acesso ao objeto recém-criado. Provavelmente a maior objeção ao auto-controle seja o seu "absolutismo": não há previsão para mudanças de autoridade que não tenham sido antecipadas pelo seu criador. Por exemplo, num sistema de time-sharing comercial, se o chefe de um departamento qualquer estiver doente, não há meio de um funcionário de sua confiança adquirir acesso temporário a um arquivo recém-criado. Pior que isso, o auto-controle pode criar objetos completamente inacessíveis, gerando lixo na memória.

Para eliminar esses problemas tem sido usado o esquema de CONTROLE HIERÁRQUICO. Neste, sempre que um novo objeto é criado, o criador tem que especificar algum controlador de acesso previamente existente para regular as futuras mudanças de acesso. Cada controlador pode ser em si mesmo uma lista de acesso de outro objeto, o que origina uma árvore de listas de acesso. A interpretação de "bit de modificação da lista" é modificada para um pointer que vem da lista imediatamente superior à lista corrente. A aproximação hierárquica modela o controle de acesso, pois um usuário com autoridade para modificar uma lista de acesso tem também a autoridade para modificar todas as listas de acesso que se encontram abaixo desta.

O controle hierárquico é usado em sistemas de tempo compartilhado como se segue. Ao primeiro controlador de acesso é associada uma lista de acessos no nome do administrador do sistema. Este cria vários controles de acesso, por exemplo, um para cada departamento na sua companhia e confere permissão de modificar acesso ao administrador de cada departamento. Cada administrador pode, adicionalmente, criar controladores de acesso para os subdepartamentos e assim por diante. Qualquer forma de compartilhamento pode ser modelada pela simples criação de listas de acesso.

Em uma emergência, os administradores (ou seus back-ups) podem intervir e modificar qualquer lista de acesso das suas subárvores.

A objeção a este esquema é que os administradores ficam muito poderosos. Pode-se argumentar que o sistema está simplesmente espelhando a realidade de uma empresa em que a responsabilidade e o poder estão associados a uma hierarquia. Entretanto, no computador, existe uma diferença: o abuso do poder quase sempre passa desapercebido. Para solucionar esse problema foi sugerido que se adicionasse ao controlador de acesso um campo chamado PRESCRIÇÃO. Sempre que uma tentativa de modificar uma lista de acessos é feita, a permissão de modificação só é concedida após a verificação do campo de prescrição, que contém diretrizes, as quais podem ser:

. nenhuma ação.

. gravar no histórico do sistema a modificação.

. adiar a alteração por um dia (período de "descongelamento").

. aguardar que a alteração seja confirmada por outra pessoa.

Nos casos mais sérios, a prescrição pode significar que a alteração só pode ser realizada se for confirmada por um comitê de representantes de vários departamentos ou grupos de usuários. Todas estas políticas de segurança podem ser implementadas utilizando o mecanismo das listas de acesso com controle hierárquico e não apenas uma hierarquia pura e simples como se imagina a priori. A noção de "prescrição" na realidade, muda as estruturas empresariais típicas permitindo que o controle seja realizado em todos os níveis. Embora aparentemente essencial a um mecanismo de proteção, não se conhece um sistema real que a tenha implementado.

Nos sistemas de arquivos os diretórios funcionam analogamente às listas de acesso. Associado a cada arquivo há um número de bits especificando quem pode acessá-lo e de que maneira. Como a grande maioria dos diretórios são organizados em árvores, pode-se concluir que empregam o esquema do controle hierárquico. Na seção V.4 serão revistas as listas de acesso usadas em diretórios.

A política de isolamento completo pode ser implementada através do mecanismo de listas de acesso: basta restringir cada lista a apenas uma entrada, que representa apenas um usuário. Este é um caso extremo e também não foi implementado na prática.

Um dos sistemas pioneiros no uso de listas de acesso para proteger objetos é o MULTICS. Neste sistema, sempre que um objeto é criado ganha como default uma lista semelhante à do diretório em que o objeto é adicionado. Se o proprietário desejar pode modificá-la a seguir, e a lista resultante ficará "ligada" ao objeto, sendo consultada a cada tentativa de acesso. Todos os diretórios fazem parte de uma estrutura onde sujeitos autorizados a modificar elementos dos níveis superiores tem permissão para modificar os que lhes são subordinados, inclusive a lista de acesso inicial (Sal 74a).

Listas de acesso correspondem mais aos objetivos dos usuários, pois é mais fácil controlar o acesso a um objeto ou a um grupo de objetos, do que, seletivamente controlar os sujeitos que podem acessá-lo. Entretanto possuem diversas desvantagens. A informação de um processo em execução não está localizada e conversões entre nomes globais e locais devem ser constantemente realizadas antes dos testes de proteção. Isto significa que mesmo nas situações em que o acesso é recusado, o sistema já sofreu o impacto de todas as conversões e operações de endereçamento. Portanto, as capabilities são consideradas melhores para representações inversas e processamento. Um sistema pode lucrativamente utilizar ambas se tiver um bom mapeamento e a atualização das listas de acesso for convenientemente refletida na atualização das capabilities. Tanto o CAL-TSS como o MULTICS são sistemas que usam ambos os mecanismos (Pop 74).

III.4 - CHAVES - FECHADURAS

O mecanismo de proteção conhecido por CHAVE-FECHADURA ('key - lock') é uma combinação dos mecanismos de capabilities e lista de acesso (GrD 72). Cada sujeito do sistema tem associada uma "C-lista" de entradas (x, K), onde x é o nome de um objeto do sistema e K é uma chave. Paralelamente, cada objeto tem associada uma "lista de acesso" de entradas (L, α), onde L é uma fechadura e α uma cadeia de atributos de acesso (fig. 3.8).

Quando um sujeito s quer acessar um objeto x da maneira β, seleciona na sua lista de chaves uma entrada (x, K) tal que exista na lista de fechaduras do objeto uma entrada (L, α) para a qual K = L e $\alpha = \beta$. Assim, a chave "abre" a fechadura que então libera os atributos de acesso α. Dessa forma podem existir várias maneiras de acessar o mesmo objeto, dependendo da cadeia α que é liberada quando a fechadura é aberta. O proprietário de um objeto x pode conceder "α" acesso a x "simplesmente incluindo (K, α) na lista de fechaduras e distribuindo (x, K) aos vários sujeitos. Pode também revogar esta concessão simplesmente deletando (K, α) da lista de fechaduras. É possível para um proprietário criar várias chaves e fechaduras correspondentes à mesma cadeia α, isto é, distribuindo (x, K_1), (x, K_2), ... e colocando (K_1, α), (K_2, α),... na lista de fechaduras. Isto se assemelha do sistema de chaves de memória usado no IBM S/360, com uma diferença importante: não há o conceito de C-lista e é possível para alguns sistemas falsificar chaves da memória.

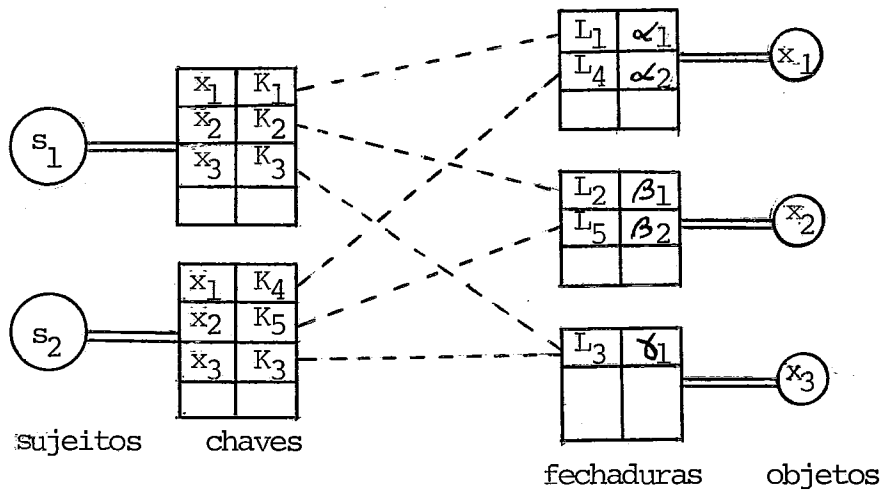


Fig. 3.8 - Lista de chaves e fechaduras com atributos de acesso

O mecanismo de chave-fechadura pode ser combinado com o de capabilities como no CAL-TSS (LaS 76). Neste sistema, sujeitos são representados por (processo, domínio) em que o processo é a entidade ativa e domínio o seu campo de ação. Cada domínio é constituído por uma C-lista e o processo só pode acessar elementos que pertençam a ela. Cada elemento desta C-lista é uma capability composta por 4 partes:

. um nome único para todo o sistema.

. um tipo t que identifica o tipo do objeto.

. um conjunto de direitos ou atributos de acesso que é um sub conjunto do conjunto R de todos os direitos possíveis.

. um valor v que é simplesmente um inteiro, cuja interpretação depende do tipo t do objeto.

Entre os vários tipos de objeto (8 do núcleo, 5 do usuário e quantas extensões forem incluídas) existem os "diretórios". Um diretório é uma lista de entradas que contém 3 partes:

. um nome simbólico.

. uma especificação do objeto, como um pointer para o disco, um pointer para outro diretório, etc.

. Uma lista de fechaduras composta por um valor v e uma lista de direitos de acesso.

Para obter acesso a uma dada entrada em um diretório, um sujeito deve apresentar uma chave de acesso que está armazenada na parte de valor de uma das suas capabilities e que seja igual à uma das fechaduras da lista do objeto. Ao abrí-la, o sujeito (processo, domínio) adquire os direitos de acesso correspondentes à fechadura (fig. 3.9).

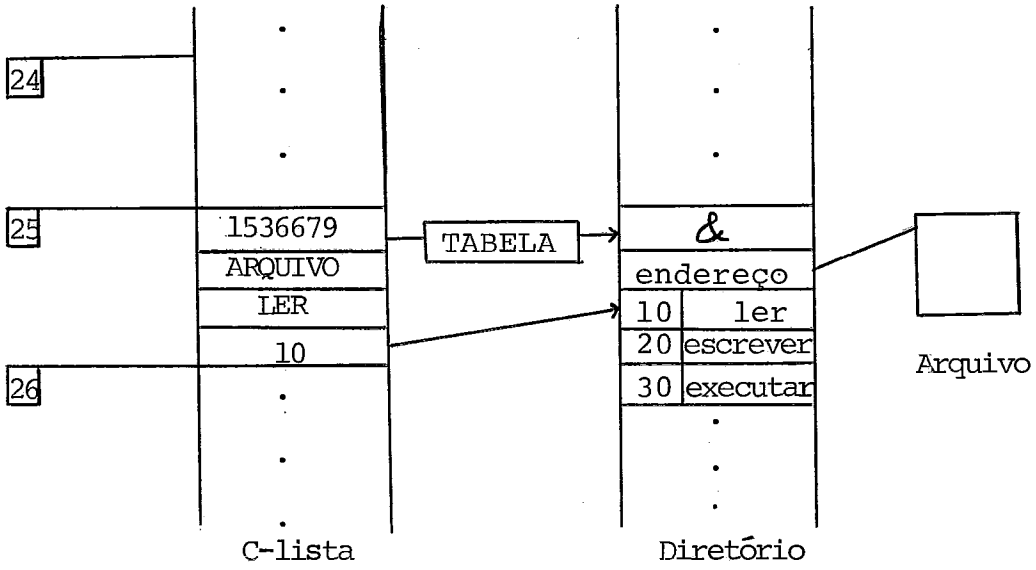


Fig. 3.9 - Mecanismo de chave-fechadura no CAL-TSS (LaS 76).

Sistemas hierárquicos podem ser interpretados como mecanismos simplificados de chave-fechadura em que as chaves e fechaduras são pequenos inteiros e o acesso é permitido se a chave for menor ou igual à fechadura.

Uma família de mecanismos de proteção pode ser criada a partir dos mecanismos de chave e fechadura (Nee 72). Considere-se uma situação em que qualquer regime de proteção pode potencialmente ocorrer em um determinado instante. Um sistema de chave-fechadura pode ser imaginado formado por uma chave contendo simplesmente o nome do regime de proteção corrente e por uma fechadura que abre um segmento de memória onde todas as regras de acesso correspondentes àquele regime estão armazenadas. Não há restrições quanto ao número de modelos que podem ser implementados, mas o tamanho das chaves e fechaduras deve ser fixado em um número de bits conveniente.

Neste mecanismo pode-se listar todas as chaves que abrem uma determinada fechadura, todas as fechaduras que são abertas por uma chave, todos os caminhos que podem ser seguidos por aberturas consecutivas. Pode-se expressar a variedade total de regimes de proteção e fazer uma atribuição definitiva de chaves e fechaduras de forma que acesso inválido nunca seja permitido. Em casos não triviais isto se torna um problema de análise combinatorial complicado e uma tarefa difícil encontrar uma solução que satisfaça todas as restrições. Mas uma vez que se limite o espaço e se defina regimes de proteção específicos, se torna exequível e interessante.

A característica principal dos mecanismos de chave e fechadura é que possuem vantagens dos mecanismos de capabilities e dos de listas de acesso. Tanto a atribuição como a revogação dos direitos são facilmente executadas. A desvantagem é a duplicação da informação, que gasta um dos bens mais preciosos do computador: memória. Entretanto, se bem aproveitada, essa duplicação é mais um dispositivo de proteção servindo para cheques duplicados de autorização.

IV - MECANISMOS NÃO-DISCRECIONÁRIOS

III - MECANISMOS DISCRECIONÁRIOS

IV.1 - MODELO MULTINÍVEL

Num ambiente de política não-discrecionária, cada objeto do sistema está associado a um qualificador com características e propriedades previamente definidas. As decisões de acesso são tomadas levando-se em conta as características do qualificador e não de cada sujeito/objeto em particular, como no caso discrecionário. Mecanismos de proteção que implementam política não-discrecionária são, é claro, pouco flexíveis, mas são mais simples para implementar, utilizar e controlar.

O modelo não-discrecionário mais conhecido é o dos níveis de segurança militares, também chamado MODELO MULTINÍVEL. Neste, cada objeto está associado a um nível de segurança, que é o seu qualificador.

A idéia de usar os níveis de segurança militares como qualificadores de objetos do computador tornou-se pela primeira vez uma realidade no sistema ADEPT-50 da Systems Development Corporation (Wei69). Para explicar as regras de controle de acesso no ADEPT-50, Weissman criou um modelo matemático que foi o primeiro modelo multinível descrito formalmente.

O modelo multinível de Weissman supunha a existência de 4 tipos de objetos de segurança no sistema: usuário, terminal, job e arquivo, e atribuía a cada objeto um triplet de qualificadores: autoridade, categoria e franquia. O qualificador AUTORIDADE era extraído de um conjunto hierarquicamente ordenado

$$A = \{ a_0 < a_1 < \ldots < a_v \}$$

que se referia aos níveis de autoridade militares:

$$A = \{ \text{NÃO-CLASSIFICADO} < \text{CONFIDENCIAL} < \text{SECRETO} < \text{ULTRA-SECRETO} \}$$

O qualificador CATEGORIA era um conjunto discreto de compartimentos específicos mutuamente exclusivos

$$C = \{ c_1, c_2, \ldots, c_w \}$$

cujo objetivo era restringir o acesso por projeto e área:

$$C = \{ \text{OLHOS APENAS, CRIPTOGRAFADO, RESTRITO, SENSITIVO, } \ldots \}$$

Finalmente, o qualificador FRANQUIA era a implementação do princípio militar 'NEED-TO-KNOW'. Cada terminal (t), job (j) e arquivo (a) possuía uma lista correspondente aos usuários com "franquia" para utilizá-lo:

$$F_u = \{ u \}$$

$$F_t = \{ u_{t0}, u_{t1}, \ldots u_{tx} \}$$

$$F_j = \{ u_{j0}, u_{j1}, \ldots u_{jy} \}$$

$$F_a = \{ u_{a0}, u_{a1}, \ldots u_{az} \}$$

As regras de acesso desse modelo eram baseadas na comparação relativa entre os qualificadores do objeto e os do sujeito que estava solicitando o acesso. Um sujeito (α) e um objeto (β) eram "comparáveis" da seguinte maneira:

$$A_\alpha \geq A_\beta \longleftrightarrow a_\alpha \geq a_\beta$$
$$C_\alpha \geq C_\beta \longleftrightarrow C_\alpha \supseteq C_\beta$$
$$F_\alpha \geq F_\beta \longleftrightarrow F_\alpha \supseteq F_\beta$$

O acesso só era autorizado se as 3 comparações acima obtivessem resultado positivo.

A especificação de cada qualificador variava com o tipo do objeto. Para usuários e terminais, autoridade, categoria e franquia eram CONSTANTES. Para jobs, a autoridade era a MENOR das duas anteriores (usuário e terminal), a categoria era a INTERSEÇÃO das duas anteriores (usuário e terminal) e a franquia era CONSTANTE. E para arquivos, se arquivo novo, uma equação era aplicada para determinar os seus qualificadores, a partir daí ficava CONSTANTE (arquivo antigo).

Embora aparentemente fosse um modelo geral permitindo um número qualquer de níveis de segurança, compartimentos, usuários e terminais, era restrito a apenas 4 tipos de objetos de segurança. Portanto tinha um espectro de ação reduzido a sistemas com essa característica.

A generalização do modelo multinível foi feita, com algumas alterações, por Bell e La Padula, no "Secure Computer Systems - Mathematical Foundation and Model" lançado em 1974 pela MITRE Corporation. Além de ter sido generalizado, o modelo foi simplificado, com a eliminação, por exemplo, do qualificador FRANQUIA.

O modelo multinível de Bell e La Padula considera o ESTADO de um sistema seguro, que é descrito por:

. acesso corrente, dado por um triplet: sujeito (s), objeto (x), tipo de acesso (t).

. nível de segurança de cada objeto.

. nível máximo e corrente de segurança do sujeito.

. um conjunto de regras de acesso baseadas em níveis.

Qualquer mudança de estado no sistema é causada por um pedido. Pedidos podem ser feitos para acessar objetos, alterar níveis de segurança, acessar o conjunto de regras, criar ou destruir objetos. A resposta do sistema a um pedido é chamada decisão. Dado o pedido e o estado corrente, a decisão e o estado novo são determinados por uma regra que prescreve como cada pedido será tratado.

Um sistema é seguro se cada regra de acesso preserva a sua segurança, ou seja:

"Se um sistema está em estado seguro, qualquer mudança de esta

do só pode levar a outro estado seguro".

Existem 4 tipos de acesso t nos modelos multiníveis:

. Nem observar nem alterar.
. Observar apenas (LER).
. Alterar apenas (ESCREVER).
. Observar e alterar (LER/ESCREVER).

Mudanças seguras entre estados são conduzidas pela observação de duas propriedades dos modelos não-discrecionários: simples e estrela.

PROPRIEDADE SIMPLES DE SEGURANÇA - "Para cada acesso corrente (s, x, t) com tipo de acesso LER, o nível do sujeito deve dominar o nível do objeto". Esta propriedade pode ser entendida como: um sujeito só pode LER um objeto se o seu nível de segurança fôr superior ou igual ao do objeto.

PROPRIEDADE ESTRELA - "Para cada acesso corrente (s, x, t) com tipo de acesso ESCREVER, o nível do objeto deve dominar o nível do sujeito". Esta propriedade significa que nenhum sujeito pode "degradar" a informação escrevendo-a em objeto de nível mais baixo que o seu próprio.

Assim, considerando os 3 tipos efetivos de acesso dos modelos multiníveis, a decisão em relação ao triplet (s, x, t) fica assim determinada:

. Se t = LER, o nível de s deve dominar o nível de x.
. Se t = ESCREVER, o nível de s deve ser dominado pelo nível de x.
. Se t = LER/ESCREVER, os níveis de s e x devem ser iguais.

As propriedades acima são implementadas através do seguinte mecanismo. Inicialmente é atribuído a cada SUJEITO e OBJETO do sistema um nível de segurança, também chamado classe de segurança, composto por 2 partes: um nível de classificação e uma categoria.

NÍVEL DE CLASSIFICAÇÃO é uma réplica dos níveis de segurança militares, cada objeto é classificado de acordo com a sensitividade da informação em ULTRA-SECRETO, SECRETO, CONFIDENCIAL ou NÃO-CLASSIFICADO. A esses níveis também é dado o nome de NÍVEL DE AUTORIDADE correspondendo à autoridade hierárquica que permite a sujeitos de um nível mais alto acessar objetos do seu nível e dos inferiores (fig. 4.1).

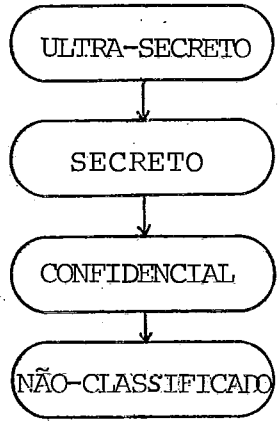
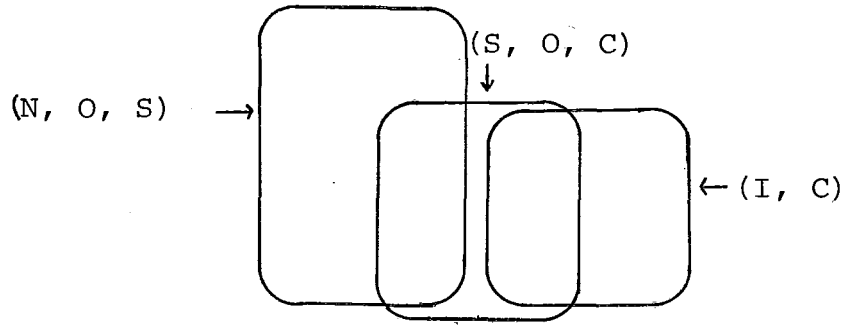


Fig. 4.1 - Níveis de classificação militares

CATEGORIA é uma combinação de compartimentos de controle usados para restringir o acesso por projeto e por área, como as do FBI e Energia Atômica. Exemplos de compartimentos são: SENSÍVEL, IRRESTRITO, NUCLEAR, OLHOS APENAS, CRIPTOGRAFADO, etc. O número de categorias é igual a 2^m compartimentos onde quase todas as combinações são possíveis (fig. 4.2).



(N, O, S) = NUCLEAR, OLHOS APENAS e SENSÍVEL.

(I, C) = IRRESTRITO e CRIPTOGRAFADO.

(S, O, C) = SENSÍVEL, OLHOS APENAS e CRIPTOGRAFADO.

Fig. 4.2 Conjuntos de Categorias Militares

Um nível de segurança ou uma classe de segurança é então especificado por:

NÍVEL DE SEGURANÇA = (NÍVEL DE CLASSIFICAÇÃO, CATEGORIA).

Diz-se que um nível de segurança domina outro se e somente se seu nível de classificação é maior ou igual ao do outro e seu conjunto de categorias contém o do outro. Apesar de todos os níveis de classificação serem ordenados hierarquicamente, o mesmo não se pode dizer dos níveis de segurança, pois nem todos os conjuntos estão contidos em outros (WFS 80).

Devido a esta situação real a implementação das propriedades simples e estrela é feita como se segue:

Simples - Um sujeito com classe de segurança $\underline{a} = (i,x)$ só pode LER um objeto com classe de segurança $\underline{b} = (j,y)$ se:

$$i \geq j \quad \text{e} \quad x \supset y$$

ESTRELA - Um sujeito com classe de segurança $\underline{a} = (i,x)$ só pode ESCREVER em um objeto com classe $\underline{b} = (j,y)$ se:

$$i \leq j \quad \text{e} \quad x \subset y$$

A mais importante vantagem dos modelos não-discrecionários é que especificações formais podem ser feitas. Além disso, o controle é realizado não apenas sobre o ACESSO à informação, mas também sobre o seu FLUXO dentro do sistema, o que é praticamente impossível nos mecanismos discrecionários (Den 76a).

O modelo multinível tem originado a construção de uma quantidade de sistemas operacionais dos quais analisaremos o ADEPT-50, o núcleo da MITRE, o KSOS e de um grande número de aplicações voltadas para comunicação como o SIGMA e o XNOS. Pesquisas estão sendo feitas para se construir redes de computadores que permitam a transmissão e o processamento de múltiplos níveis.

A tabela da fig. 4.3 sintetiza os elementos do modelo de níveis de segurança apresentado.

ELEMENTO	INTERPRETAÇÃO
Sujeito, s	Processos, etc.
Objeto, x	Dados, arquivos, etc.
Classificação	Nível de autorização do sujeito. Nível de classificação do objeto.
Categoria	Privilégios de acesso.
Nível de segurança	(classificação, categoria).
Tipo de acesso	Não observar e não alterar, observar, alterar, observar e alterar.
Pedido	Mudança no acesso corrente ou outros aspectos do estado do sistema.
(s, x, t)	Acesso corrente.
Decisão	Sim, não, erro?
Regras	Determinam decisão e próximo estado.

Fig. 4.3 - Elementos do modelo multinível (WFS 80)

IV.2 - NÍVEIS DE SEGURANÇA E ARQUITETURAS HIERÁRQUICAS

Arquiteturas hierárquicas são formadas por vários níveis ou camadas de software, mas não existe correspondência alguma entre esses níveis e os níveis de segurança de um modelo multinível. É o que vamos explicar.

Um sistema de ARQUITETURA HIERÁRQUICA é aquele em que, à maneira de Dijkstra (Dij68) as funções de alocação de recursos, ativação de processos, controle e supervisão estão distribuídos em várias camadas e o controle exercido sobre cada uma depende de sua posição na hierarquia.

Assim, a hierarquia de um sistema operacional é heterogênea, os componentes são agrupados pelas suas FUNÇÕES, como numa grande empresa: funcionários dos níveis mais altos exercem funções de controle e "não sabem" executar os serviços realizados pelos funcionários dos níveis mais baixos.

De constituição diferente é a hierarquia de NÍVEIS DE SEGURANÇA. Os componentes não são agrupados pelas suas funções, mas sim pelo seu CONTEÚDO. A hierarquia é homogênea, pois em todos os níveis as funções são as mesmas: ler, escrever, alterar informações. Os componentes dos níveis superiores "sabem" acessar os dos níveis inferiores e vice-versa, só que os últimos não estão autorizados a fazê-lo.

Poderíamos dizer que enquanto a hierarquia da arquitetura é interna ao sistema, a hierarquia dos níveis de segurança é externa ao mesmo, depende do organograma da instalação, um agente completamente externo ao universo do computador. Em consequência, a hierarquia da arquitetura é estática e a dos níveis de segurança é dinâmica e sofre os mesmos efeitos das mudanças ambientais sofridas pelos usuários do sistema.

Sistemas de arquitetura hierárquica podem implementar mecanismos discrecionários e não-discrecionários, o mesmo acontecendo para sistemas de arquitetura plana. O MULTICS, por exemplo é um sistema de arquitetura hierárquica. Pelos anéis de proteção da memória, numerados de 0 a 7, todas as funções são estratificadas, de modo que as funções dos anéis superiores podem ativar processos e controlar as dos anéis mais baixos. À primeira vista poderia parecer que o sistema implementa um mecanismo não-discrecionário de 7 níveis de segurança. Entretanto, o MULTICS é tipicamente um sistema discrecionário que implementou mecanismos de capabilities e de listas de acesso (Sal 74a).

Há outros sistemas, como o KSOS, de arquitetura hierárquica, que realmente suportam uma política não-discrecionária. Porém existem 20 níveis na arquitetura do KSOS e apenas 4 níveis de segurança sem nenhuma correlação de uns com outros como se vê na tabela da fig. 4.4.

NÍVEL	NOME	ABSTRAÇÃO
19	KER	interface de chamada do núcleo
18	SPF	funções especiais
17	PRO	operadores de processo
16	IPC	comunicação interprocessual
15	FCA	capabilities de arquivos
14	SUB	subtipos de arquivos, extensões
13	MFS	sistemas de arquivos montáveis
12	PST	estado do processo
11	PVM	memória virtual do processo
10	SEG	segmentos
9	FIL	resumos de arquivos
8	FST	estados de arquivos
7	SMX	modelo de segurança
6	PRV	controle privilegiado
5	DIF	funções independentes de dispositivo
4	TII	informação independente de tipo
3	SYL	nível do sistema
2	SEN	nomes de entidades de segurança
1	DIF	funções dependentes do dispositivo
0	MAC	máquina

Fig. 4.4 - Níveis de arquitetura do sistema KSOS (BeB 79)

Como "nível do sistema" é a informação fornecida no nível 3 da hierarquia do sistema operacional, deduz-se que a partir daí, todas as funções são executadas em um dos níveis de segurança: ultra-secreto, secreto, confidencial e não classificado (BeB 79).

Desde Dijkstra praticamente todos os sistemas operacionais foram construídos com arquiteturas hierárquicas (máquinas virtuais, núcleos, anéis são formas de hierarquia). Há poucos exemplos de sistemas planos (o HYDRA é um deles). Por isso, há que se tomar muito cuidado com a palávra NÍVEL: será empregada nos dois sentidos, os quais não devem, sob hipótese alguma ser confundidos.

IV.3 - CONTROLE DE FLUXO

Um FLUXO EXPLÍCITO de informação ocorre de um objeto x a um objeto y quando uma sequência de instruções lê dados de x e escreve em y. Um FLUXO IMPLÍCITO de informação ocorre quando uma sequência qualquer de instruções permite a um objeto y "deduzir" dados de um objeto x sem haver explicitamente transferência de informação.

O controle de fluxo é uma forma de evitar que haja fluxo ilícito entre objetos, seja explícito ou implícito, seja por canais normais ou cobertos. Uma política de fluxo especifica os canais nos quais a informação está autorizada a escoar.

Mecanismos completos de controle de fluxo devem monitorar detalhadamente o fluxo de dados entre variáveis de cada programa mas são extremamente complexos e dispendiosos. Um subconjunto destes se refere aos mecanismos para controlar o fluxo entre classes de segurança, e são portanto, mecanismos não-discrecionários de controle de fluxo. Embora sejam mais rígidos, são usualmente eficientes e desempenham seu papel com alta margem de segurança.

A política de fluxo mais simples especifica apenas 2 classes de informação: confidencial (c) e não confidencial (N) e só permite fluxo de dados da classe N para a classe C. Por exemplo, se um software de serviço manuseia dados de usuário, parte dos quais é confidencial, o programa pode reter apenas a parte que não é confidencial e deve ser impedido de reter a outra.

Sistemas de computadores governamentais e militares baseados nas classes de segurança possuem política de fluxo um pouco mais complicada, decorrente das duas propriedades: simples e estrela. Como cada classe de segurança a é um par (i,x) correspondente ao nível de autorização i e à categoria x, a informação só pode escoar para outra classe b = (j,y) se $i \leq j$ e $x \subseteq y$. Suponhamos por exemplo que existam 3 níveis de classificação: 1 (confidencial), 2 (secreto) e 3 (ultra-secreto) e 4 compartimentos: I (irrestrito), R (restrito), S (sensitivo) e C (crypto). Então a informação de (2, RS) pode escoar para (3, RS) e para (2, RSC), mas não para (1, RS) nem (3,R).

Dorothy Denning (Den 76a) desenvolveu um modelo de fluxo de informações baseado em objetos, classes de segurança e processos. Basicamente, o modelo define um fluxo FM representado por:

$$FM = \langle N, P, SC, \oplus, \rightarrow \rangle$$

N é o conjunto de objetos ou receptáculos de informação como arquivos, segmentos, variáveis de programa e também usuários.

$$N = \{ a, b, \ldots \}$$

SC é o conjunto de classes de segurança, que podem ser as militares ou outras quaisquer.

$$SC = \{ \underline{a}, \underline{b}, \ldots \}$$

Cada objeto "a" está ligado a uma classe de segurança $\underline{a}$ que especifica o nível de segurança associado à informação armazenada em "a". Processos podem também estar ligados a classes de segurança. Ligações podem ser estáticas (constantes) ou dinâmicas (variáveis).

O operador de combinação de classes, " $\oplus$ " é associativo e comutativo e especifica para qualquer par de classes de operandos a classe que resultará de qualquer função binária nos operandos (AND, NOT, OR,...). Assim, a classe resultante de qualquer função binária entre os objetos "a" e "b" é "$\underline{a} \oplus \underline{b}$". Por extensão, a classe resultante de qualquer função n-ária $f(a_1,...,an)$ é "$\underline{a_1} \oplus \ ... \ \oplus \underline{a_n}$".

Uma relação de fluxo " $\longrightarrow$ " é definida sôbre pares de classes de segurança. Para as classes $\underline{a}$ e $\underline{b}$, escrevemos $\underline{a} \longrightarrow \underline{b}$ se e somente se a informação da classe $\underline{a}$ está autorizada a escoar para a classe $\underline{b}$. Ou seja, o operador $\longrightarrow$ é uma regra de PERMISSÃO DE FLUXO a qual deve ser atendida por qualquer sequência de operações que causam transferência de informação de um objeto a outro.

A regra de segurança do modelo é simplesmente a seguinte: um modelo de fluxo FM é seguro se e só se a execução de uma sequência de operações não provoca um fluxo que viola a relação $\longrightarrow$. Esta regra pode ser formalizada:

"Se um valor $f(a_1, ..., an)$ escoa para um objeto b, que está estaticamente ligado a uma classe de segurança $\underline{b}$, então a relação $\underline{a1} \oplus ... \oplus \underline{an} \longrightarrow \underline{b}$ deve ser verdadeira".

Exemplo: seja um sistema que contenha informações pessoais de três qualidades: médicas, financeiras e criminais. Então, os fluxos de informação só serão permitidos como na fig. 4.5. Não estão autorizados fluxos de objetos pertencentes à classe de segurança MED para CRIM ou FIN. Esta estrutura é chamada TRELIÇA de subconjuntos de sistema e o modelo de D. Denning é chamado MODELO DE TRELIÇA para um fluxo seguro de informações.

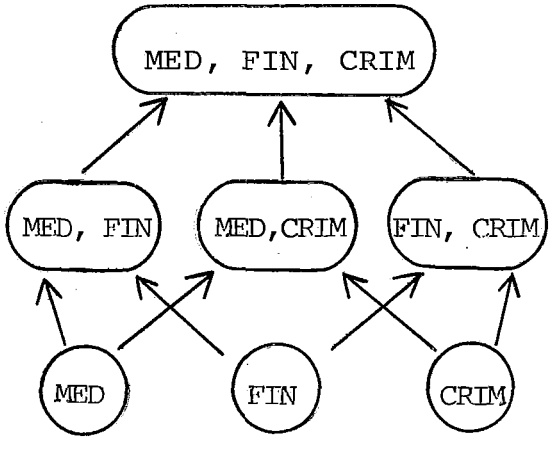


Fig. 4.5 - Treliça de fluxo seguro de informações (WFS80)

Fluxos explícitos de informação são mais fáceis de serem controlados que os fluxos implícitos. Existem operações que causam

fluxo de informação mesmo sem serem executadas. Consideremos por exemplo a instrução:

IF X = FALSE THEN Y: = FALSE ELSE Y: = TRUE

Esta instrução contém um fluxo implícito de X para Y. Após a execução da mesma, qualquer que seja o resultado do IF, o valor de X fica conhecido. Testar Y é o mesmo que testar X.

O modelo de treliça também prevê o caso dos fluxos implícitos. Seja:

$S = c: S_1, \ldots, S_n$ uma estrutura condicional

$B = \{ b_1, \ldots, b_n \}$ o conjunto de objetos que recebem fluxo implícito de S

Então todos os fluxos implícitos são seguros se a relação

$$\underline{c} \longrightarrow \underline{b_i} \quad (1 \leq i \leq n)$$ for verdadeira.

Ou equivalentemente, se $\underline{c} \longrightarrow \underline{b_1} \oplus \ldots \oplus \underline{b_n}$ for verdadeira após a execução de S.

O modelo que acabamos de analisar se refere aos fluxos explícitos e implícitos decorrentes de operações realizadas por canais LEGÍTIMOS quais sejam as instruções de cópia, I/O, passagem de parâmetros e mensagens. Entretanto não se aplica aos fluxos de correntes de canais COBERTOS como: efeito de um processo na carga do sistema, mudança na taxa de transmissão de I/O, etc. Esses casos são complexos e caros e dificilmente podem ser prevenidos. Na maioria das vezes sua ocorrência não é eliminada totalmente mas são utilizados dispositivos para deteção de fluxos em canais cobertos especificados. Peter e Dorothy Denning afirmaram (DeD79) que não existem modelos de custo efetivo para fechar todos os canais cobertos de fluxo de informação.

O controle de fluxo tem sido pesquisado principalmente nos mecanismos não-discrecionários e particularmente nos militares multiníveis. Controlar o fluxo de informações através de mecanismos discrecionários é praticamente impossível, pois se chega a um sistema n-dimensional de proporções impalpáveis. Se forem permitidas algumas restrições, ou seja: um meio-termo entre a discrecionaridade e a não-discrecionaridade pode-se obter algumas alternativas. Uma delas é o mecanismo de controle de fluxo por vetores (SaG 78) onde a cada processo são concedidos vetores de capabilities cujos elementos só podem ser usados dentro do mesmo vetor. Por exemplo, se um processo tiver 2 vetores de capabilities:

<(A, READ) ; (B, WRITE)>

<(C, READ) ; (D, WRITE)>

este processo só pode escrever o conteúdo de A em B e o conteúdo de C em D, prevenindo todos os outros fluxos de informação cruzados.

Este esquema é restritivo, por conter elementos do princípio do mínimo de privilégios, mas pode ser utilizado para controlar o fluxo em um mecanismo de proteção baseado em capabilities, e este é o seu maior mérito. Todos os outros esquemas conhecidos de controle de fluxo trabalham apenas com mecanismos não-discrecionários.

IV.4 - MOMENTO DE LIGAÇÃO

Momento de ligação ('binding-time') é o momento na vida do sistema em que um mecanismo de proteção multinível realiza a ligação entre o objeto e o nível de segurança.

Existem 2 possibilidades: ESTÁTICA, quando a classe do objeto é constante e DINÂMICA quando varia com seu conteúdo. Usuários são ligados estaticamente aos níveis de autorização. Cada programa de usuário pode herdar uma ligação estática do seu proprietário, ou ser ligado dinâmicamente em decorrência de suas atividades.

LIGAÇÃO ESTÁTICA

Ligações estáticas são feitas em objetos de maneira previamente definida por seus proprietários. Pode ser feita em dois momentos: compilação ou execução (Den 76a).

Nos sistemas CASE e MITRE cada processo p está associado a um nível de segurança $\underline{p}$, que especifica a classe mais alta que o processo pode ler e a classe mais baixa em que pode escrever. Um mecanismo de proteção ativado durante a EXECUÇÃO do processo permite que p adquira acesso de leitura a um objeto "a" apenas se a relação $\underline{a} \longrightarrow \underline{p}$ é válida e acesso de escritura a um objeto "b" apenas se a relação $\underline{p} \longrightarrow \underline{b}$ é válida. Assim, p pode ler de $a_1 \ldots, a_m$ e pode escrever em $b_1, \ldots, b_n$ apenas se

$$\underline{a_1} \oplus \cdots \oplus \underline{a_m} \rightarrow p \rightarrow \underline{b_1} \oplus \cdots \oplus \underline{b_n}$$

Este mecanismo garante a segurança de todos os fluxos, explícitos e implícitos, pois nenhum fluxo pode ocorrer a menos que $\underline{a} \rightarrow \underline{p} \longrightarrow \underline{b}$ o que implica que $\underline{a} \longrightarrow \underline{b}$.

Outro mecanismo de ligação estática feita durante a EXECUÇÃO é o da Data Mark Machine de Fenton (Fen74). Nesta máquina são usadas etiquetas para ligar objetos à classes de segurança. Cada processo p deve estar associado a uma classe de segurança $\underline{p}$. Esta classe é determinada como se segue: sempre que uma estrutura condicional c: $S_1, \ldots S_n$ é encontrada, a classe corrente $\underline{p}$ é empilhada numa stack e é substituída por uma classe $\underline{p} \oplus \underline{c}$. Na saída da estrutura, $\underline{p}$ é recuperada da stack. Desta forma a classe $\underline{p}$ representa o limite mínimo superior das classes $\underline{c1} \ldots \underline{ck}$ imediatamente antes da execução do conjunto S, o qual está condicionado aos valores de k variáveis de condição: $c_1, \ldots, c_k$. Se S especifica um fluxo explícito dos objetos $a_1, \ldots, a_n$ ao objeto b, o mecanismo de execução de instrução verifica se $\underline{a_1} \oplus \ldots \oplus \underline{a_n} \oplus \underline{p} \longrightarrow \underline{b}$ é válida e inibe a execução de S em caso contrário. Este mecanismo está também automaticamente checando todos os fluxos implícitos para b quando qualquer fluxo explícito ocorre pois Fenton provou que em ligações estáticas não é necessário verificar os fluxos implícitos durante a execução que ocorrem na ausência de fluxos explítitos.

No sistema proposto por D. Denning (Den 76b) a ligação é feita estaticamente durante a COMPILAÇÃO. A cada instrução, o próprio compilador verifica se o valor $\underline{f}(a_1,\ldots,a_n) = \underline{a_1} \oplus \ldots \oplus \underline{a_n} \rightarrow \underline{b}$ é verdadeira. Caso positivo, a instrução é válida, caso negativo é emitida mensagem de erro. Isto é uma forma de CERTIFICAÇÃO. Mecanismos que produzem certificação possuem 3 vantagens. Primeira, a execução de um programa é garantida antes que ele inicie, portanto o programa não pode escoar informação causando um número exato de violações de segurança. Segunda, aumenta a velocidade de execução pois todos os checks são feitos antes. Terceira, o processo de certificação pode ser especificado em linguagem de alto nível. Por outro lado, existem 2 limitações: primeira , os fluxos não especificados diretamente por defeito da linguagem não podem ser verificados e segunda, programas certificados podem tornar-se inseguros por erros de hardware. Estas limitações podem ser removidas utilizando-se uma linguagem adequada e um mecanismo que checa todos os fluxos novamente durante a execução. Técnicas acuradas de certificação que incluem assertivas formais das dependências de I/O permitidas entre módulos de programas, e verificação de fluxos de dados entre variáveis de programas, foram desenvolvidas na Mitre Corporation (Mil 76), UCLA (Pop 78) e SRI International (Neu 78).

LIGAÇÃO DINÂMICA

Um sistema puramente baseado em ligações dinâmicas é inviável. Poderia gerar situações absurdas como um usuário de nível de autoridade NÃO-CLASSIFICADO alterar sua classificação para ULTRA-SECRETO.

Um dos mecanismos de ligação dinâmica conhecidos existe em uma versão modificada da Data Mark Machine de Fenton. Atualizando a classe de um objeto ao receber informação superior à classe antiga, o mecanismo força a validação da relação de fluxo. Assim, o que nos casos estáticos é um teste ($\underline{a_1} \oplus \ldots \oplus \underline{a_n} \oplus \underline{p} \longrightarrow \underline{b}$?) no caso dinâmico é uma atribuição:

$$\underline{b} := \underline{a_1} \oplus \ldots \oplus \underline{a_n} \oplus \underline{p}$$

Entretanto, fluxos implícitos não são resolvidos, como se pode verificar pela consideração de 2 processos p e q:

```
(p)  c: = false
     if    a then c: = true

(q)  b: = false
     if    c then b: = true
```

Se p e q são executados sequencialmente e c é uma variável global a ambos, o processo q termina com b = a mesmo que b pertença a uma classe inferior à de a.

Duas alternativas foram propostas para solucionar êsse problema. D.Denning propos que cada objeto b, que possa receber um fluxo implícito de um processo p tenha sua classe atualizada para $\underline{b} \oplus \underline{p}$ quando o processo termina. Anita Jones propos que o conteúdo de b seja apagado quando p terminar sempre que a relação $\underline{p} \longrightarrow \underline{b}$ não for satisfeita (JoL 75).

O mecanismo de proteção do ADEPT-50 (Wei 69) também é baseado no princípio de nunca decrescer a classe de um objeto, atualizando-a quando necessário para corresponder à informação contida. Sempre que um programa p escreve uma informação num objeto b, a classe do objeto é atualizada para $\underline{b} = \underline{b} \oplus \underline{f}$, onde $\underline{f}$ é determinada pelo $\oplus$ de todas as classes dos arquivos abertos pelo programa.

O grande problema dos mecanismos de ligação dinâmica é uma tendência a superclassificar informação, já que apenas alterações "para cima" são realizadas. Este problema pode ser resolvido por um processo de degradação manual da informação, realizada por uma pessoa autorizada. No sistema SIGMA (AmO 78) este procedimento é permitido. É citado como exemplo o caso de um usuário que solicita ao sistema uma lista de todas as suas mensagens contendo uma palavra qualquer. Para realizar esta operação, o usuário deve estar num nível de segurança maior ou igual ao de cada mensagem do arquivo. Suponhamos agora que o usuário desejasse fazer uma modificação numa das suas mensagens que tem nível de segurança abaixo do corrente. Neste caso, a propriedade estrela impediria a modificação ou obrigaria a mensagem resultante a ter seu nível atualizado, superclassificando-a. Portanto, ao usuário deve ser dado o direito de "escrever abaixo". PESSOAS são interpretadas de forma diferente de PROGRAMAS, e agem como verdadeiros filtros seletivos na classificação das informações.

Finalizando, combinações de mecanismos de ligação estática e dinâmica podem ser eficientes. Por exemplo: compiladores que após identificar todas as possibilidades de fluxos implícitos entre as variáveis de um programa, inserem no texto compilado instruções de atualização de níveis que serão preenchidas durante a execução. Compiladores de linguagens podem também prover mecanismos de proteção de memória e de tipo, gerar selos de autenticação e uma série de outros medidas de segurança estáticas e dinâmicas combinadas, recriando no escopo de um programa o que o sistema operacional realiza a nível do sistema (Mor 73).

V - PROTEÇÃO DA INFORMAÇÃO ARMAZENADA

V.1 - MEMÓRIA VIRTUAL

De todos os recursos do sistema operacional, a memória é provavelmente o mais raro, por isto técnicas e políticas de alocação e proteção de memória tem sido pesquisadas com interesse.

Memórias tem sido organizadas em no mínimo 2 níveis: memória principal (diretamente endereçável) e memória auxiliar. A necessidade de grandes quantidades de memória tem forçado um compromisso entre a memória principal, rápida porém cara, e a auxiliar, lenta porém barata.

Dentre as técnicas utilizadas na gerência da memória principal, a memória virtual tem sido das mais frequentes. Também chamada "memória de 1 nível", foi introduzida pelo computador ATLAS em 1962 (Den 71). Esta máquina foi a primeira a fazer a distinção entre "endereço" e "localização" de uma palavra de informação interpretando endereço como nome da palavra e localização como sua residência física. Assim, o conjunto de todos os endereços que um programa pudesse referenciar (endereços de programa), era o seu ESPAÇO DE NOMES OU ENDEREÇAMENTO e o conjunto de todas as localizações físicas que pudesse acessar (endereços de hardware) era o seu ESPAÇO DE MEMÓRIA.

Com isso, a gerência da memória tornou-se independente da programação. O espaço de nomes ficou permanentemente associado ao programa e o espaço da memória ficou sendo problema do sistema, que transladava endereços de programa em endereços de hardware durante a execução. Vários sistemas de alocação de memória e mecanismos de endereçamento de hardware foram definidos e implementados baseados nessa técnica de "memória de 1 nível". Esta denominação se refere ao fato de que para o programador a memória se assemelha apenas a uma enorme loja simulada (virtual) de espaços de memória.

Um mecanismo de memória virtual é nada mais que uma função de mapeamento f do ESPAÇO DE NOMES ao ESPAÇO DE MEMÓRIA (fig. 5.1). A cada endereço de programa x, a função associa univocamente um endereço de hardware y = f (x). Se y está na memória auxiliar, f(x) é indefinida, o mecanismo sinaliza uma page-fault que causa a interrupção na execução do programa, a página que contém y é trazida para a memória principal e a f é atualizada. Esta translação entre memórias é necessária porque existem mais páginas na memória virtual do que blocos na memória real, consequência da diferença de custo entre ambas. Entretanto é assumido que existe sempre um bloco disponível na memória principal para receber uma página solicitada por uma interrupção de programa. Esta disponibilidade é gerenciada por um mecanismo de remoção de páginas, que é invocado sempre que o número de blocos disponíveis cai abaixo de um certo valor pré-fixado. É utilizado um algoritmo na seleção das páginas que serão removidas, os algoritmos mais comuns são FIFO (primeira a entrar primeira a sair), LRU (menos recentemente utilizada) entre outros.

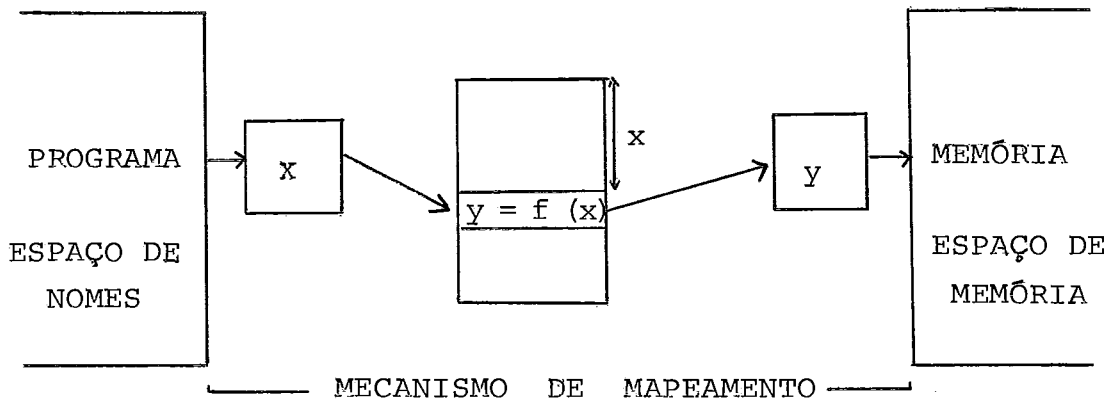


Fig 5.1 - Mapeamento no mecanismo de memória virtual (Den 71)

Muitas estratégias tem sido utilizadas na implementação de memórias virtuais. Um tipo comum, NÚCLEO-TAMBOR, usa memória principal de núcleo magnético e memória auxiliar de tambor, a translação de endereços sendo implementada por uma combinação de hardware e software. Outro tipo usa memória principal de registro e memória auxiliar de núcleo magnético, a translação de endereços sendo implementada inteiramente em hardware. Este é chamado MEMÓRIA DE CACHE ou "memória de escravo" e é usado no IBM 360/85, séries IBM 370 e CDC 7600 (Den 71).

Não tendo sido criado especificamente com propósitos de proteção, o mecanismo de memória virtual proporciona um estágio de segurança (BCD72) devido a:

1. INTEGRIDADE - Desviando a responsabilidade de alocação da memória das mãos do programador para as do sistema, o mecanismo se torna mais íntegro, as decisões são uniformes para todos os casos e as possibilidades de erros são mais cuidadosamente examinadas.

2. ISOLAMENTO - Cada programa só pode referenciar a informação contida no seu ESPAÇO DE ENDEREÇAMENTO, qualquer outra informação lhe lhe é inacessível, uma vez que é impossível obter dados pela leitura direta de endereços da memória.

3. COMPARTILHAMENTO SEGURO - O isolamento definido acima permite que sistemas de multiprogramação e time-sharing sejam possíveis, e que os recursos do sistema, basicamente a memória sejam compartilháveis sem depender estritamente da boa fé dos seus usuários.

4. PROTEÇÃO - É possível embutir chaves de proteção em cada entrada da função de mapeamento, contendo restrições, modos de acesso (ex : ler, escrever, executar), tipos de programa ou qualquer outra informação de proteção.

Com efeito, essa quarta característica tem sido usada como base para a maior parte dos mecanismos de proteção de memória desenvolvidos até o momento. O próprio conceito de CAPABILITIES surgiu das tabelas de endereçamento da memória virtual cujas entradas foram desmembradas em quantidades independentes (Fab 74).

O mecanismo de memória virtual não é nem pretende ser um mecanismo de proteção eficiente. Provê algumas funções de proteção e realiza algumas tarefas que em outros casos são desempenhadas por mecanismos de proteção. É, em si mesmo, apenas um mecanismo de endereçamento. Mas é a base, indispensável e fundamental para a implementação dos outros, e nos sistemas comuns, que não foram projetados com propósitos de segurança, é o único mecanismo de que dispõe a memória para se proteger.

V.2 ARQUITETURAS ETIQUETADAS

Arquiteturas etiquetadas foram inventadas para uma variedade de aplicações além de proteção. O Burroughs 5700 e seus ancestrais (Org 73) e o computador da universidade Rice (Feu 72) são exemplos de arquiteturas que utilizam etiquetas (tags) multibits para identificar separadamente instruções, descritores e os vários tipos de dados. Êsses bits são interpretados pelo hardware como informações sobre os dados existentes na localização, ou como seu status com respeito ao programa.

O conceito básico dos TAG BITS não é novo. Quase todos os computadores empregam um bit de paridade utilizado pelo hardware para detetar falhas de memória. Além disso, muitos computadores utilizam um byte tipo "fechadura", que limita o acesso de uma área da memória somente ao sistema operacional ou a outro do sistema que possua a "chave" para abrí-la.

Máquinas mais antigas como o Burroughs 5500 empregam um bit de flag para informar ao hardware que aquela palavra contém um valor não numérico que deve ser interpretado. O computador Rice R-1 contruído em 1959 empregava 2 bits de etiqueta para cada palavra, que eram usados na depuração e monitoração do sistema (Feu 72).

Sistemas mais recentes como o EAI 8400 e o Telefunken TR4 empregam também etiquetas de 2 bits. O Burroughs 6700 e o B7700 empregam etiquetas de 3 bits para identificar tipos de operandos numéricos e informações especiais pelo sistema operacional (Org 73).

Em 1968 Iliffe (referência 1 de Feu 72) publicou seus conceitos sobre a realização de uma máquina etiquetada que era na realidade uma rejeição da máquina clássica de Von Neumann. Na máquina de Von Neumann, programas e dados eram equivalentes, no sentido de que os dados de um programa poderiam ser o próprio programa. Enquanto essa prática era admissível em minicomputadores com um único usuário, tornava-se uma ameaça à proteção nos grandes computadores, onde o compartilhamento de dados e programas deveria ser encorajado. O que Iliffe propos e tornou realidade no computador Basic Language Machine (BLM) foi uma arquitetura em que toda a informação relacionada a um elemento está contida numa etiqueta na própria localização do elemento. Por exemplo, um algoritmo para realizar a soma de um vetor, em vez de usar um índice e um loop, usa a informação contida no início do vetor. O algoritmo é o mesmo quer o vetor tenha 10 quer 100 elementos. Da mesma forma, em vez de haver vários algoritmos diferentes como somar inteiro, somar ponto flutuante, somar complexo, etc, há apenas um algoritmo, pois a etiqueta do dado é quem vai definir o seu conteúdo.

A utilização de etiquetas deve seguir algumas regras básicas conforme definido por Iliffe e formalizado por Fabry (Fab 74).

1. A etiqueta deve ser testada a cada acesso.

2. Quando uma palavra é copiada, à cópia deve ser fornecida a mesma etiqueta da original.

3. O resultado de operações em palavras de mesma etiqueta deve ser igual aos operados.

4. O resultado de operações em palavras de etiquetas diferentes deve ser decidido por um conjunto de regras pré-determinadas.

5. Operações que alteram o valor das etiquetas devem ser controladas cuidadosamente.

Por essas regras pode-se observar que arquiteturas etiquetadas produzem um excelente ambiente para a implementação de mecanismos de proteção. Com efeito, isso tem sido feito com frequência, principalmente no caso do mecanismo de CAPABILITIES.

A grande vantagem de se utilizar etiquetas para autenticar capabilities é que elas não podem ser falsificadas. Sendo uma tarefa específica do sistema, a atribuição de etiquetas a capabilities não pode ser realizada por programas de usuário. Duas possibilidades existem para a modificação das capabilities autenticadas por etiquetas: por software e por hardware.

No sistema BCC-500 as capabilities (fig. 3.5) são modificadas pelo sistema operacional. É possível armazenar e carregar palavra completa, incluindo a etiqueta, dando uma certa liberdade aos programas de usuários para manipular os nomes dos objetos nos quais estão trabalhando. Êstes últimos, entretanto, não possuem a propriedade de alterar os bits de etiqueta, portanto, não podem "construir" capabilities para si próprios (Lam 69).

No sistema IBM S/38 as capabilities só podem ser modificadas pelo sistema operacional. Qualquer tentativa de alterar o conteúdo de uma delas causa a remoção da sua etiqueta, o que lhe tira a validade (Ber 80).

Além de simplificar a construção do mecanismo de capabilities, outras vantagens são inerentes às arquiteturas etiquetadas (Feu72):

1. O HARDWARE pode realizar testes de consistência durante a execução de cada operação, por exemplo, verificação de tipo e limite.

2. O projeto do SISTEMA OPERACIONAL é simplificado, pois diferenças como label e endereço podem ser determinadas na hora da execução. A construção de entry-points protegidos é uma consequência imediata dessa simplificação pois pode ser impedido o acesso a subrotinas via endereço, sendo permitido apenas via palavra de controle.

3. COMPILADORES podem ser construídos mais simples e mais eficientes liberando-os da tarefa de testar a semântica das operações que passam para o nível do hardware. Podem também introduzir cheques nos programas compilados para serem preenchidos durante a execução através de palavras cuja etiqueta contem bits de controle para o hardware.

4. O projeto e o uso de SISTEMAS DE DEPURAÇÃO também pode ser aperfeiçoado. Dumps de memória podem ser emitidos usando o tipo de cada dado: complexos, inteiros, etc. Localizações simbólicas podem ser utilizadas facilitando a depuração dinâmica. Interrupções relativas a ações de controle podem ser geradas e resolvidas através das etiquetas. Finalmente, como as etiquetas podem ser colocadas pelo compilador ou pelo usuário (através do sistema operacional) a codificação correta não necessita ser recompilada, mas apenas as instruções erradas, simplificando o processo da depuração.

5. A MEMÓRIA do computador pode ser seccionada em várias partes lógicas, e o ACESSO a cada uma ser controlado pelos bits da etiqueta. Alguns sistemas operacionais utilizam uma memória de 2 níveis, um do supervisor e outro dos usuários. Podem ser criadas tantas partes quanto são os usuários em multiprogramação, gerando isolamento controlado pelo próprio hardware.

6. PALAVRAS que contem endereços podem ser manipuladas apenas por um conjunto restrito de instruções. As instruções mais poderosas podem ser negadas ao usuário, etc.

Arquiteturas podem ser etiquetadas de um grande número de maneiras. As etiquetas podem existir em cada palavra da memória, em cada página, ou em cada segmento. Segmentos são conjuntos de páginas, e possuem uma etiqueta chamada descritor. Segmentos e descritores serão estudados com mais detalhe no capítulo VI.

Finalmente, só encontramos uma restrição para o uso de etiquetas de memória: gastam espaço. Mas levando em consideração a simplificação considerável realizada no sistema operacional, nos compiladores, nos programas de aplicação, etc., acreditamos que o saldo é positivo.

V.3 MICROPROGRAMAÇÃO

O termo microprogramação foi inicialmente enunciado pelo Professor Wilkes do Laboratório Matemático da Universidade de Cambridge em 1951 (Ros 69). Sua tese era a seguinte: é possível interpretar a parte de controle de um computador como efetuando um número de transferências de informação de registro a registro, algumas em sequência, algumas em paralelo, para obter a execução de uma simples instrução de máquina. Cada um desses passos pode ser interpretado como a execução de uma instrução para alguma máquina desconhecida pelo programador. O conjunto desses passos, que representa a execução de uma simples instrução na máquina do usuário é chamado microprograma. Microprogramas podem também ser usados para outras operações necessárias que são invisíveis ao programador, por exemplo: obter a próxima instrução e calcular o endereço efetivo.

Em meados dos anos 60 já era possível e prático construir computadores em que o controle era explicitamente realizado por microprogramas, residentes em uma memória separada da memória do usuário. Essa memória, em geral só de leitura (ROM) era preparada pelo fabricante do computador.

Por outro lado, os sistemas operacionais dos anos 60, com raras excessões, não estavam preocupados com a segurança e proteção. Qualquer usuário que tivesse conhecimento da linguagem Assembler era potencialmente capaz de devassá-los, e o que é pior, de devastá-los. Dezenas de caminhos curiosos foram encontrados para a sua penetração, às vezes, por mera vaidade pessoal.

No início dos anos 70 essas deficiências foram estudadas cuidadosamente e mecanismos de proteção de hardware e software foram pesquisados. O advento da microprogramação de baixo custo tornou possível seu uso em várias aplicações de segurança. A partir daí a microprogramação tem sido empregada para implementar formas simples de proteção de memória e estados de execução múltipla em máquinas que, de outra maneira, não poderiam possuir essas características. O próprio núcleo do sistema operacional tem sido implementado em microprogramção, como o RTOS (BES 79), sem falar dos microcomputadores programados para agir como dispositivos de criptografia (capítulo IX).

Mecanismos de proteção implementados em microprogramação são muito menos vulneráveis a mudanças não autorizadas do que em programação convencional, pois é impossível alterar a microcodificação por um programa ordinário. Além disso, existe microcodificação que é fisicamente impossível de ser alterada (ROM).

Microinstruções são instruções de baixíssimo nível que compõem um microprograma. Como introduzido por Wilkes, especificam transferências de dados simples entre a memória principal e registradores de alta velocidade, entre os próprios registradores e entre registradores e processadores, tais como somadores, multiplicadores, etc. Existem microinstruções condicionais, lógicas e funcionais, como deslocamenteo dos elementos de um registrador. No sistema VENUS, por exemplo, também as operações de sincronização de processos e I/O são realizadas por microprogramas.

O VENUS (Lis 72) foi um sistema projetado para testar o efeito da arquitetura do hardware na complexidade do software. Queriam demonstrar que quanto mais sofisticado o hardware, mais simples o software. Uma excelente oportunidade para a microprogramação. Foi escolhida uma arquitetura hierárquica, à maneira de Dijkstra (Dij68) cujos níveis inferiores foram microprogramados no computador Interdata 3. O resultado foi um conjunto de 2.000 instruções armazenadas em uma ROM, que proveem suporte para as seguintes características:

. MEMÓRIA VIRTUAL - uma tabela central única é usada pelo micro programa para mapear endereços virtuais em reais. Existem segmentos de 64 Kbytes de dados e páginas de 256 bytes.

. MULTIPROGRAMAÇÃO - até 16 processos concorrentes podem ser executados, consistindo de um espaço de endereçamento (segmentos) e uma área de trabalho (informações relacionadas ao processo). O compartilhamento de recursos e a sincronização entre processos são controladas por microprogramas, que realizam, por exemplo, alocação da CPU.

. CANAIS DE I/O - totalmente microprogramados, liberam o software de todas as restrições de tempo real associadas a dispositivos. Utilizam também semáforos para sincronização.

. PROCEDURES - cada procedure é armazenada em um único segmento, segmentos compartilhados são usados como passagem de parâmetros e procedures reentrantes podem ser compartilhadas.

A partir desse "hardware" microprogramado, a construção do software se tornou simples e econômica. Foram implementados dicionários (traduzem nomes externos e internos de segmentos), filas de mensagens (para comunicação interprocessual) e outras construções de alto nível.

Em sistemas mixtos, ou sejam, que utilizam tanto a micro como a macro programação, a memória dos microprogramas sempre fica isolada da memória do sistema. No Burroughs B 1700 (Wil 72), por exemplo, existem 1 ou 2 memórias de 25.665.536 bits (S-memory) para o sistema e de 1 a 4 módulos de memória microprogramável (M-memory). Escrever microprogramas para o B 1700 é muitas vezes mais simples do que escrever subrotinas Fortran pelas seguintes razões:

1. Microprogramas consistem de sentenças pequenas e imperativas, em inglês e comentários descritivos. Exemplo:

```
READ 8 BITS TO t COUNTING fa UP AND f1 DOWN
```

2. Não é necessário saber o formato das microinstruções, apenas as que se referem à S-memory que o programa está interpretando.

3. A multiprogramação dos microprogramas é função do MCP (sistema operacional do Burroughs), sendo transparente ao programador. Não há diferença alguma decorrente do fato de haver um ou mais interpretadores rodando simultaneamente.

4. O uso da M-memory é função apenas do MCP, usuários não podem mover informações de ou para lá. Agindo como intermediário entre o usuário e a memória de microprogramação o sistema operacional diminui a probabilidade de acessos não autorizados.

5. Uma vez que todas as referências são codificadas simbolicamente, a proteção é facilmente assegurada. Microprogramas podem referenciar apenas objetos do seu ESPAÇO DE NOMES o qual inclui apenas objetos que pertencem aos microprogramas e às suas máquinas virtuais. Mesmo os nomes gerados artificialmente (como subscritos negativos para vetores de FORTRAN) tem sua validade checada pelo hardware.

No IBM S/38 (Ber 80) a microprogramação também é utilizada como proteção da memória. Objetos, que são construções de alto nível tais como programas, filas e índices são armazenados em um ou mais segmentos da memória. Esses segmentos possuem um cabeçalho de informação que é um tipo de etiqueta, legível apenas pela microprogramação, que identifica o tipo de objeto existente no segmento e os seus atributos. A etiqueta enumera também os segmentos adicionais que podem ser requeridos para um dado tipo de objeto, os quais são encadeados por rotinas microprogramadas que testam os direitos de acesso do sujeito solicitante.

A mais importante utilidade da microprogramação no S/38 porém não é esta. A memória total do S/38, somadas principal e secundária em disco possui uma configuração de 2.6 bilhões de bytes, organizados em segmentos de 16 megabytes e páginas de 512 bytes. O endereço virtual de 64 bits é composto por um identificador de segmentos de 40 bits e um offset de 24 bits (fig. 5.2). Tudo isso sairia extremamente caro, em tempo e espaço, se fôsse inteiramente suportado pelo hardware. No entanto, o hardware suporta apenas segmentos de 64 Kbytes e endereços de 48 bits. Um simples componente da microprogramação estende o tamanho do endereço virtual aos 64 bits com a restrição de que não existam ao mesmo tempo mais do que 16 milhões de segmentos (situação impossível).

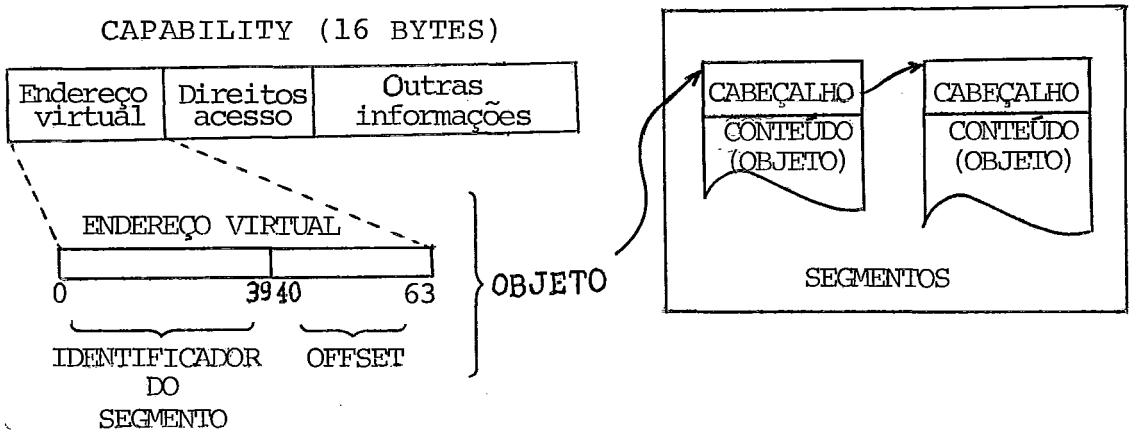


Fig. 5.2 - Endereçamento no Sistema IBM S/38

Não apenas segmentos são protegidos por rotinas de microprogramação no sistema S/38, mas todos os objetos do sistema. O acesso a cada objeto só é autorizado após cheques realizados por duas rotinas. A primeira verifica o tipo e a validade da capability e a existência do objeto endereçado. A segunda rotina verifica se o processo tem autoridade para usar o objeto, sincroniza acessos múltiplos e realiza outros cheques.

Instruções de máquina nunca são executadas diretamente, mas antes transformadas em microinstruções, as quais, ou realizam a operação 'in-line' ou invocam outra subrotina da microprogramação. Uma vez que um programa está transladado, torna-se um objeto protegido do sistema e não pode mais ser modificado. As variáveis estáticas são alocadas em SPACES (objetos de memória protegidos) na primeira vez em que o programa é executado e as variáveis automáticas são alocadas e inicializadas em SPACES a cada chamada do programa em um processo. Constantes são armazenadas com o programa. Essas variáveis e constantes são diretamente endereçáveis pelo microprograma mas sempre que uma instrução referencia um objeto em outra região da memória, os microprogramas realizam testes para verificar os direitos de acesso.

Como foi visto, a microprogramação pode ser utilizada praticamente para construir qualquer mecanismo de proteção que é normalmente construído em macroprogramação. Núcleos de proteção microprogramados, mecanismos discrecionários e não-discrecionários, capabilities, controle de fluxo, criptografia, etc., todos já possuem alguma experiência microprogramada. Especialmente no que se refere à proteção dos próprios mecanismos, que, microprogramados, tornam-se sem dúvida mais seguros.

V.4 DIRETÓRIOS

Num sistema de computação, mesmo que nenhum job esteja sendo executado, cada usuário tem um conjunto de objetos "próprios". Estes podem ser criados, modificados e deletados à sua discreção. Além disso, a cada usuário deve ser dada a capacidade de utilizar objetos do sistema e de oustros usuários, que serão compartilhados por vários jobs. Deve haver garantias de que os objetos "próprios" não são acessados por outros usuários a não ser que ele autorize. Para controlar dessa forma o uso e o compartilhamento de objetos, o sistema precisa organizá-los de uma forma eficiente.

Três esquemas podem ser utilizados com êsse objetivo: sistemas de arquivos, espaço de nomes segmentado e bancos de dados (Den 71). Bancos de dados serão estudados detalhadamente no capítulo VIII, por isso só trataremos aqui dos dois primeiros.

Um SISTEMA DE ARQUIVOS é um mecanismo que gerencia a criação e a utilização de objetos de dados pertencentes aos usuários identificando-os por grupamentos denominados arquivos. Um arquivo é uma coleção de informações em geral homogêneas e em geral protegidas pelas mesmas restrições de acesso. A tarefa do sistema de arquivos é manter tabelas com informações sobre cada arquivo tais como: nome do proprietário, departamento, data de criação, data de expiração, meio de armazenamento (disco, fita, etc), sistema a que pertence, e outras. Um sistema de arquivos não contém os arquivos, mas apenas referências a eles e frequentemente está repleto de entradas fantasmas decorrentes de deleções não comunicadas pelos usuários. O sistema de arquivos do OS/370 é um exemplo, em que existem duas instruções diferentes: uma para deletar o arquivo e outra para deletar a entrada do arquivo.

O ESPAÇO DE NOMES SEGMENTADO é também um mecanismo para gerenciar a criação e utilização de objetos de dados de usuários, porém o faz de forma diferente. Toda a informação armazenada pelo sistema está on-line e é potencialmente endereçável por qualquer processo ativo e por qualquer job. Isto significa que não há necessidade de haver cópia de instruções ou itens de dados através de dispositivos periféricos. Programas e dados não necessitam ter imagens na memória central de núcleo, porque toda a grande memória segmentada é diretamente endereçável. Isso evita duplicações que aumentam as necessidades de espaço e podem introduzir erros. Pedidos de I/O, buffers e toda a parafernália relativa à transferência de informação entre meios "permanentes" (disco, fita, etc) e "voláteis" (memória central) são desnecessários. O próprio termo "arquivo" perde o seu significado, dando lugar ao termo "segmento de dados". Êste é o mecanismo existente no MULTICS (BCD 72). A informação relacionada à proteção como modos de acesso, etc., de cada segmento, está contida numa parte chamada descritor do segmento que é manuseada pelo sistema de memória. Assim, um só mecanismo, o de endereçamento, desempenha duas tarefas que em outros sistemas são realizadas por dois mecanismos: o de endereçamento e o de arquivos (DeV 66).

Em qualquer dessas duas aproximações, tanto na de sistemas de arquivos como na de espaço de nomes segmentado encontramos uma

construção de alto nível comum: os diretórios, utilizados tanto para administração como para proteção de objetos do usuário.

Um DIRETÓRIO é um conjunto de itens, cada item sendo uma associação entre o nome de um objeto, a sua localização e as suas restrições de acesso. Os itens de um diretório podem teoricamente representar qualquer objeto do sistema: segmentos, funções de I/O, arquivos de usuários e até outros diretórios. Isto leva à formação de hierarquias, comumente chamadas árvores e ao diretório mais baixo na hierarquia é dado o nome de raiz.

Muitas variações podem ser feitas em torno do conceito de diretórios (DeD79). Normalmente cada usuário é proprietário de um diretório, onde pode criar e deletar arbitrariamente seus objetos e deixá-los ficar por um longo período de tempo. Todos os diretórios podem fazer parte de uma raiz, que é o diretório geral do sistema. As entradas em cada diretório de usuário podem conter, por exemplo: o nome do objeto, seu comprimento, endereço, uma lista de usuários autorizados e um identificador único do objeto. Cada entrada na lista de autorização pode especificar o nome e o tipo de acesso permitido a cada indivíduo designado pelo proprietário do objeto. Por exemplo, na fig. 5.3, Jones concedeu acesso tipo "ler" seu arquivo Y a Smith e a si própria e acesso tipo "escrever" somente a si própria, não tendo concedido nenhum tipo de acesso a Cox.

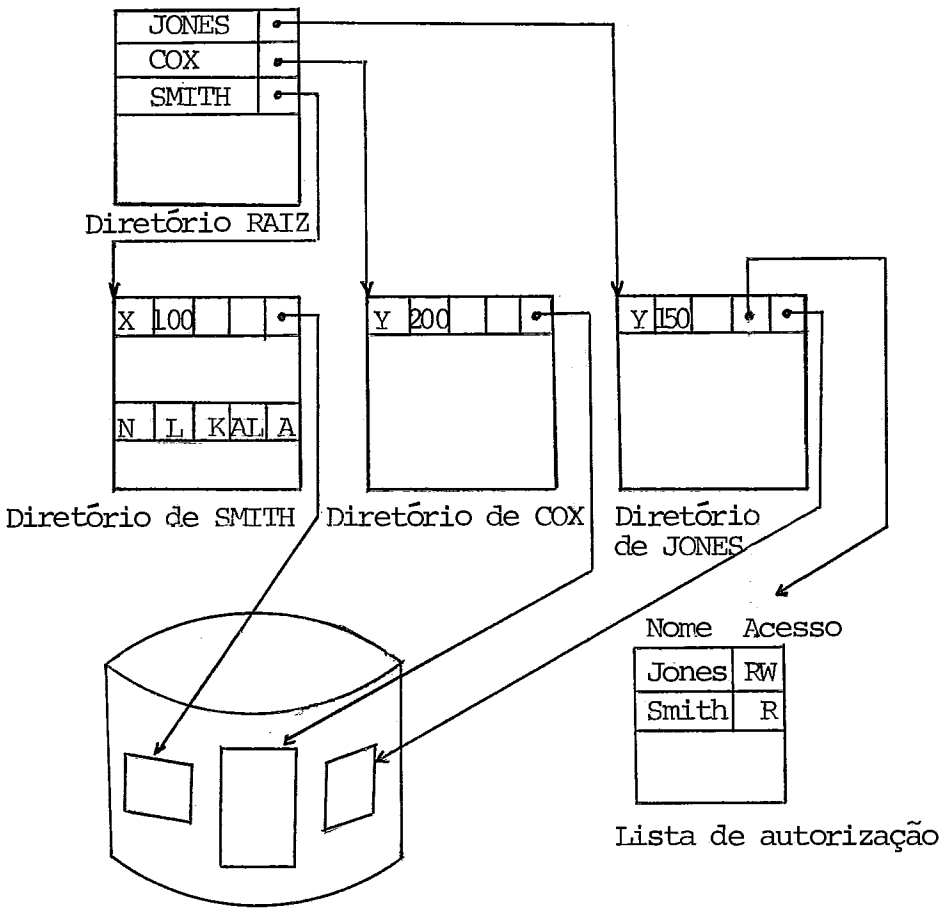


Fig. 5.3 - Diretórios para arquivos permanentes (DeD79)

Arranjar os diretórios em árvores traz 3 vantagens: primeira, cada diretório pode ser determinado univocamente por sua trajetória na árvore, segunda, o mesmo nome pode ser reutilizado em vários diretórios e terceira, a que nos interessa mais diretamente , permite que o controle de acesso seja feito por um mecanismo de proteção organizado "sobre" a estrutura da árvore.

Um mecanismo de proteção de arquivos que utiliza essa estrutura é a Instalation Management Facility (IMF) da IBM (GWM 75). Desenvolvida para gerenciar a utilização e o armazenamento de objetos do usuário, a IMF foi implementada no S/360-91, no OS/360-MVT, OS/VS e similares. O objetivo da facility é encorajar os usuários a estabelecer linhas claras de autorização e responsabilidade sobre os próprios objetos, realizando as funções de controle que na maioria dos casos são desempenhadas pelo sistema operacional. Aliás, a IBM tem sido coerente em não procurar salvaguardas e mecanismos de proteção a nível de sistema, mas em transferir a responsabilidade de uma maneira descentralizada para os proprietários de objetos.

A IMF também foi chamada Sistema de Controle de Inventário - (ICS), porque à maneira dos seus homônimos detém o controle de todos os "bens" relativos ao uso do computador, provendo meios para sua segurança e integridade. A idéia básica da IMF é: objetos são os ARQUIVOS a serem protegidos e sujeitos podem ser de 3 tipos: AUTOR (quem cria um objeto), PROPRIETÁRIO (quem define os privilégios de acesso de outros usuários àquele objeto) e USUÁRIO (quem acessa o objeto de acordo com os privilégios possuídos). Além disso, usuários podem ser reunidos em GRUPOS ou ser conectados por ligações adicionais a outros conjuntos de arquivos gerando vários diretórios interconectados. Na fig. 5.4 encontramos o exemplo de uma empresa com 3 divisões A, B, C, onde é construído um diretório que espelha a estrutura de seu organograma.

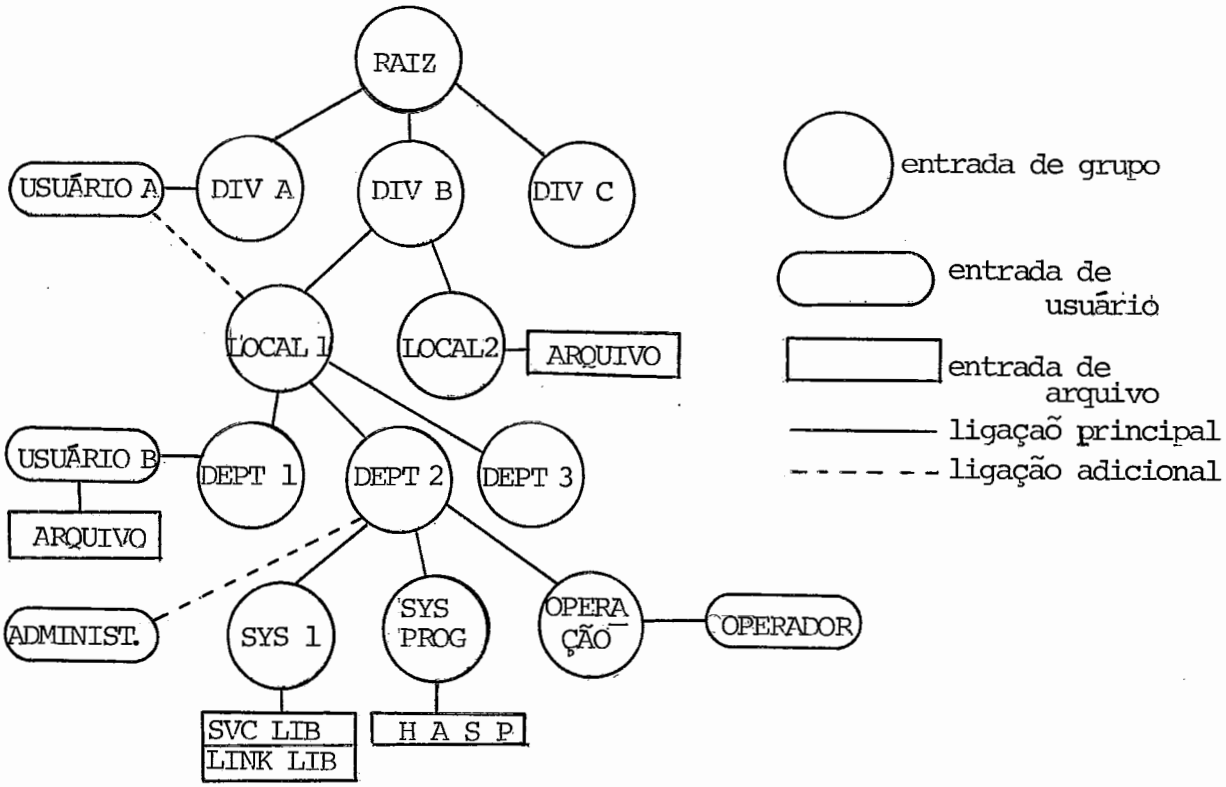


Fig. 5.4 - Um diretório paralelo ao organograma de uma empresa

Além de controlar o acesso, a IMF realiza funções de auditoria, suporte para 4 níveis de memória (on-line, migrada, arquivada e back-up), estatísticas de utilização, migração (mover arquivo de volume on-line para volume off-line) e suporte para TSO.

A IMF é apenas uma parte de um mecanismo completo de proteção, como foi reconhecido pelos próprios autores (GWM 75), os quais deveriam englobar todo o sistema operacional. O grau de proteção produzido por ela é limitado e se restringe a um único tipo de objeto do sistema: arquivos. Qualquer usuário mais determinado pode encontrar formas de driblar a facility. A justificativa dada para isso é que instalações comerciais não necessitam um nível muito elevado de proteção, da qual discordamos. Entretanto os conceitos de grupos, árvores e conexões são amplamente válidos e podem ser expandidos para outros objetos do sistema.

No MULTICS (Sal 74a) cada objeto, inclusive os diretórios, devem estar catalogados em algum diretório, originando uma árvore única de diretórios. Sendo o MULTICS um sistema baseado no mecanismo discrecionário de listas de acesso, essa estrutura de diretórios é também a residência de todas as listas de acesso, correspondentes a cada objeto do sistema. Cada entrada num diretório consiste do nome do objeto e sua lista de acesso. Uma autorização para modificar alguma entrada do diretório é interpretada também como autorização para modificar a lista de acesso da entrada. Não existe outro controle da lista de acesso. Por exemplo: deletar o nome de um usuário da lista de acesso de um diretório, não significa que será deletado também dos diretórios que estão abaixo deste na hierarquia. No MULTICS sempre que uma entrada é adicionada num diretório ganha como default uma lista de acessos inicial, que é cópia da lista de acessos corrente daquele diretório. Se o usuário desejar, pode alterá-la depois. Todos os objetos do sistema: processos, segmentos, filas de mensagens, descritores removíveis e os próprios diretórios são protegidos através dessa estrutura hierárquica de diretórios. O MULTICS implementa o espaço de nomes segmentado .

Outros sistemas utilizam diretórios, tanto para funções administrativas como de segurança. No CAL-TSS (LaS 76) entradas em diretórios são associadas a cada objeto de usuário que ocupa espaço em disco. No KSOS (CaD 79) o objeto denominado SUBTIPO DE ARQUIVO é um diretório que permite a construção de extensões de objetos.

Particularmente no caso de proteção da informação armazenada os diretórios desempenham um papel importante e indispensável, pois simultaneamente organizam e provem estruturas de controle e supervisão.

V.5 ACESSO LÓGICO A ARQUIVOS

Paradoxalmente a maioria dos sistemas de memória virtual, que é uma camuflagem da memória secundária, também provem sistemas de arquivos, que é a aceitação desta memória secundária. Isto porque a "grande loja de espaços de memória" não é suficientemente grande para armazenar toda a informação dos usuários e nem suficientemente flexível para lhes garantir controle e supervisão a qualquer nível desejado (Den 71).

As propriedades físicas das memórias auxiliares (discos, tambores, fitas) tem forçado a organização de arquivos em registros. Dependendo do meio utilizado, cada arquivo pode ser estruturado em uma de diversas formas: direto, sequencial, sequencial indexado , particionado, etc. Entretanto, da mesma forma que ninguém se preocupa com os blocos de memória no mapeamento feito pela memória virtual, os registros de uma fita e as trilhas de um disco não necessitam ser visíveis ao programador. Existem sistemas que confundem "propriedades lógicas dos arquivos" com "implementação de mapeamento" e apresentam uma estrutura ultra-complicada ao programador. Nesta seção nos preocuparemos em analisar os tipos de acesso lógico a arquivos, independentemente da sua organização ou do meio físico em que está armazenado.

A forma mais intuitiva de acesso lógico aos dados de um arquivo especifica 2 tipos de acesso: independente de conteúdo ou dependente de conteúdo (BaH 80). No primeiro, a decisão de conceder o acesso é tomada antes da leitura dos dados e antes mesmo da abertura do arquivo. No segundo, a decisão de acesso só pode ser tomada após a leitura dos dados, que são "parâmetros" do mecanismo de proteção.

A forma mais sofisticada extende essas duas categorias para 4 e as decisões são baseadas em testes realizados nos seguintes itens:

1. ARQUIVO - Esta categoria inclui as decisões de acesso e uso que dependem apenas do sujeito e do objeto para serem resolvidos. Exemplos: empregados externos ao departamento de pessoal não podem acessar arquivos de dados pessoais.

2. AMBIENTE - Consiste das decisões de acesso relacionadas ao ambiente, ao momento, à topologia corrente, etc. Exemplo: empregados do departamento de pessoal só podem acessar arquivos pessoais durante o expediente e em terminais especificados.

3. ELEMENTO - São as decisões de acesso que envolvem elementos particulares do objeto, como campos especiais de um arquivo. Nesse caso, subconjuntos de campos do arquivo são formados para atender à especificação de acesso. Exemplo: o campo de salário só é acessível aos funcionários que trabalham na seção de pagamento.

4. DADO - Contém todas as decisões de acesso que são resolvidas após a inspecção do dado. Há que ser feita uma leitura anterior de cada dado antes de ser concedido o acesso. Exemplo: salários acima de um teto só podem ser acessados pelo chefe do departamento de pessoal.

As categorias 1 e 2 se aplicam a qualquer objeto do sistema e são assunto de estudo em todo o escopo desta tese. Examinaremos aqui apenas as categorias 3 e 4 que são específicas para arquivos de dados, ou seja: para objetos cujo conteúdo influi na decisão de acesso. Dois mecanismos serão descritos: segmentação lógica e segmentação física (BaH 80).

SEGMENTAÇÃO LÓGICA

O mecanismo de segmentação lógica usa o ELEMENTO para a reestruturação lógica do arquivo em um conjunto de vários subarquivos, cada qual com seus direitos de acesso específicos. Quando um usuário solicita o arquivo, seu pedido é interpretado e transladado em um pedido para o subarquivo lógico específico, assim, só é necessário realizar as 3 decisões restantes (arquivo, ambiente e dado). Na realidade essa reestruturação é feita apenas nos caminhos de acesso ao arquivo e não no conteúdo dele mesmo. Não são criadas cópias múltiplas, mas o arquivo fica apenas logicamente dividido em subarquivos de elementos "semelhantes" (essa "semelhança" de elementos é definida pelo mecanismo de proteção).

Esta aproximação é parecida à do CODASYL (GKS79) o qual mantendo o arquivo intacto apresenta ao usuário diferentes arquivos lógicos ou "vistas" através do uso de fechaduras da passagem de chaves ao DBMS (vistas serão analisadas detalhadamente na seção VIII.4). A diferença entre o CODASYL e a segmentação lógica acima é que nesta última são mantidos subarquivos lógicos para cada usuário, que obtém a informação sempre do mesmo local e não de "vistas" diferentes como no primeiro.

Deve ser enfatizado que a rotina de reestruturação não precisa ser processada a cada vez que o usuário abre o arquivo. Ao contrário, as declarações são geradas quando o arquivo é criado, neste momento são construídos os links apropriados para os subarquivos e no cabeçalho de cada subarquivo é armazenada a informação de proteção. As únicas procedures usadas durante a execução do programa são as de acesso e processamento, que colocam o usuário imediatamente em contato com o seu subarquivo.

A possibilidade de pré-processar tanto quanto possível as decisões de segurança é a maior vantagem da segmentação lógica, pois minimiza o tempo de execução. Entretanto, requer alguma memória para os cabeçalhos de subarquivos, tempo do pré-processamento e tempo de processamento adicional para atualização das estruturas durante mudanças de política de segurança.

SEGMENTAÇÃO FÍSICA

O objetivo geral da segmentação física de um arquivo ('file segmentation' ou 'file clustering') é agrupar os valores de elementos que são mais frequentemente requeridos juntos em subarquivos físicos independentemente acessíveis. Como na segmentação lógica o ELEMENTO é utilizado na reestruturação do arquivo, porém no caso da segmentação física existe uma separação física entre os subarquivos (fig 5.5).

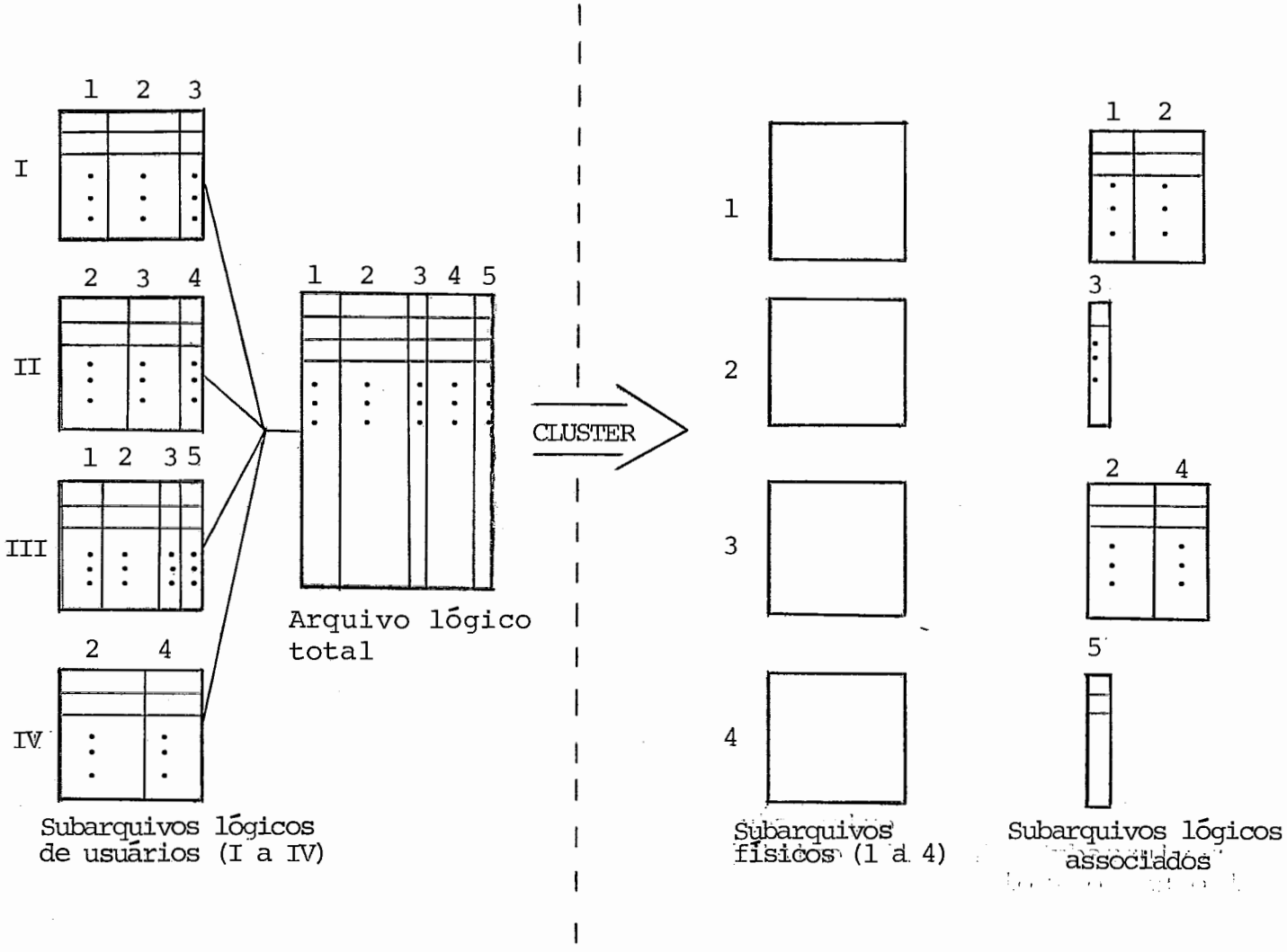


Fig. 5.5 - Comparação entre segmentação lógica e física de arquivos

É assumindo que todos os pedidos são conhecidos, feitos por programas "enlatados", que são usados por vários usuários, os quais possuem diferentes necessidades lógicas ou "vistas" para a mesma aplicação. Assim, há um mecanismo de segurança seletiva embutido na própria estrutura do arquivo. É demonstrado que o custo adicional de memória para armazenar as definições de subarquivos é irrelevante em comparação à memória total necessária e os custos da adição ou modificação de elementos variam com a segmentação escolhida. Por exemplo, um subarquivo que tenha direitos idênticos em

todos os seus elementos para um dado usuário pode ser mais eficientemente processado que um subarquivo em que os direitos estão misturados.

Comparativamente que alternativa é melhor em cada caso? Valem duas observações:

- Em ARQUIVOS ESTÁTICOS, onde o número de consultas é muito maior que o de atualizações, a segmentação física pode suportar as eventuais mudanças de estratégia de segurança seletiva.

- Em ARQUIVOS VOLÁTEIS, onde o número de atualizações é comparável ao de consultas, a melhor alternativa é a segmentação lógica, embora a física possa ser realizada a um preço elevado, exceto se realizada por um bom algoritmo.

Finalmente, esses dois mecanismos podem ser utilizados para decisões dependentes de DADO da mesma forma que o foram para as decisões dependentes de ELEMENTO. Agrupando-se os DADOS de iguais restrições em um cluster apropriado de registros como foi feito nos seus campos, pode-se tornar a decisão 4 a mais barata de todas (BaH 80).

V.6 CRIPTOGRAFIA APLICADA A ARQUIVOS

Uma descrição mais detalhada de mecanismos de criptografia encontra-se no capítulo IX-Sistemas Distribuídos, onde estes mecanismos encontram 80% de sua aplicabilidade.

Os outros 20% da utilidade da criptografia dizem respeito à proteção de arquivos de dados, ou seja, da informação armazenada. Sua maior virtude é liberar o usuário da necessidade de confiar nos administradores de sistemas. Nenhum mecanismo de proteção criptográfico pode ser enganado por autoridades do sistema, já que dependem de uma chave cujo conhecimento é restrito ao proprietário do arquivo. Para acessar um arquivo cifrado os programadores de sistema teriam que espioná-lo durante o seu processamento, o que é mais difícil de ser consequido e portanto de ser bem sucedido.

Olhando pelo outro lado da moeda, a criptografia também libera os programadores de sistemas dos dilemas associados à entrega de arquivos de um usuário a outro em vista de solicitação escrita. Entregando o arquivo criptografado, qualquer problema de exposição de informação corre por conta do proprietário.

Para que arquivos criptografados sejam processados sem grande aumento do overhead, o mecanismo de cifragem pode estar integrado diretamente no mecanismo de acesso do arquivo. O ideal seria se o próprio hardware se encarregasse dessas tarefas, por exemplo, através de unidades periféricas providas de etapas de cifragem e decifragem associadas às de escritura e leitura, respectivamente (DiH79). Cada dispositivo poderia possuir a sua própria chave, única, que não seria transmitida nem compartilhada com outros dispositivos. A criptografia seria completamente transparente a todos os dados armazenados em qualquer meio. Dessa forma as unidades portáteis do sistema poderiam ser lidas só no próprio sistema em que foram escritas. Não conhecemos até o momento nenhum dispositivo assim.

Essa criptografia de baixo nível entretanto não é suficiente para evitar exposições de informações causadas por funcionários da instalação. Vários algoritmos podem ser oferecidos, em hardware ou em software para produzir uma cifragem de dados cuja chave é conhecida apenas pelo proprietário. Os próprios programas de aplicação podem possuir no seu início e no seu final algumas linhas de codificação que decriptografem os dados antes de utilizá-los e os criptografem depois.

Pode-se observar dessa análise que existem vários níveis possíveis para se criptografar arquivos de dados. O mesmo dado pode ser criptografado várias vêzes, por vários processos diferentes , desde que depois se conheça o caminho inverso para a sua recuperação. Em cada nível um tipo de perigo está sendo evitado. Por exemplo, a cifragem produzida pelas unidades periféricas é uma salvaguarda contra roubos de arquivos, além de ser uma excelente proteção para cópias back-ups que são passíveis de serem guardados em locais remotos, visto que só podem ser lidas em terminais específicos.

O nível mais elevado de cifragem de dados já realizado é a construção de arquivos com cada campo criptografado por uma chave diferente. Por este método, a implementação de uma regra de acesso dependente de elemento é simplesmente a administração da chave correspondente a cada elemento e a posse física de um arquivo ao qual se tem apenas direito parcial de leitura não implica na sua exposição total. Gudes (Gud 80) propõe algumas alternativas para o projeto de sistemas de arquivo com essas características. Várias políticas de segurança podem ser implementadas: compartimentalizadas, hierárquicas e dependentes de dados. As especificações de proteção são basicamente realizadas através da cifragem dos dados e da distribuição seletiva das chaves de criptografia entre os usuários. Devido a isso, esses mecanismos são chamados de Criptografia Controlada pelo Usuário (UCC).

Um sistema baseado em UCC considera 3 partes principais: arquivos, mecanismos de proteção ou "sistema" e usuários. Quando um usuário solicita acesso a um arquivo, o sistema pede a chave, quer o acesso seja de simples leitura, quer de processamento dos dados do arquivo. Algumas cifras "processáveis" podem ser construídas, o que elimina a necessidade de serem decifradas antes do uso. A proteção das chaves durante a transferência usuário-sistema é fundamental e deve ser realizada pelos métodos de proteção convencionais, entretanto algumas regras podem ser estabelecidas, como: nenhuma chave pode residir no sistema, devem ser realizados testes de identificação e autenticação de usuários e todos os algoritmos de criptografia devem ser de conhecimento público. O mecanismo proposto por Gudes é baseado na existência virtual de uma matriz de acesso, cujas linhas são os usuários, cujas colunas são os arquivos e cujos elementos são as chaves de acesso. Daí valem as duas aproximações dos mecanismos discrecionários: C-lista e lista de acessos.

A C-lista é armazenada pelo sistema sob uma forma criptográfica, cuja chave é conhecida apenas pelo usuário. Assim, em vez de memorizar n chaves de n arquivos, o usuário memoriza apenas 1, que abre uma C-lista de n capabilities. A lista de acessos é armazenada no início de cada arquivo juntamente com um registro de validação, que é uma chave Kj cifrada, utilizada para confirmar a chave do usuário no momento do seu pedido de acesso.

Também DIRETÓRIOS podem ser implementados neste modelo. Especificações hierárquicas são construídas sobre árvores de diretórios, tais que a cada diretório corresponde uma chave de criptografia e para acessar um diretório abaixo da árvore deve-se antes acessar o seu supervisor. Complementarmente, as especificações dependentes de DADOS introduzem a idéia de átomos de informação, em que um átomo é um conjunto de registros no arquivo que está cifrado sob a mesma chave e, portanto, a menor parte de um arquivo que pode ser separadamente protegida.

Finalmente, técnicas de criptografia podem ser combinadas com outros mecanismos de proteção causando ou não redundância. Needham (Nee 79) nos apresenta um sistema de arquivos convencional que é acessado remotamente numa rede via capability criptografada. O monitor do arquivo prepara um registro contendo as informações de proteção, criptografa-o e fornece a chave ao usuário. Este pode exercê-la ou distribuí-la entre alguns companheiros selecionados.

Cada arquivo pode possuir várias chaves (correspondentes a várias capabilities diferentes) e ser acessado de qualquer parte da rede. Este mecanismo é uma forma modificada do mecanismo de chave-fechadura. A maior vantagem é que não há necessidade de distribuir as chaves de criptografia entre vários sistemas pois a cifragem e decifragem sempre ocorrem no mesmo local.

A criptografia não pode substituir os mecanismos de proteção convencionais dos arquivos, mas pode aumentar consideravelmente a segurança total do sistema. Um arquivo é mais vulnerável durante os longos períodos em que reside ocioso em disco ou fita. Se estiver criptografado nesses momentos, um impostor só pode comprometê-lo durante os curtos períodos em que está sendo processado, tarefa incomparavelmente mais difícil.

VI - EXECUÇÃO PROTEGIDA DE PROCESSOS

VI.1 PROCESSOS

A importância do estudo dos processos nesta análise de mecanismos de proteção decorre do fato de serem eles, na maior parte das vezes, as únicas entidades ativas (sujeitos) do sistema. São os responsáveis pela movimentação da informação, pela sua criação, modificação e destruição. De nada adiantaria um sistema que protegesse a informação em "cofres de aço" se o ato de armazená-la e recuperá-la desses cofres não fosse revestido de vários dispositivos de segurança. Situação similar existe no interior do sistema operacional onde os processos são os responsáveis pela manipulação dos objetos protegidos.

Na seção II.2 o sistema operacional foi definido em termos das funções de controle e supervisão que provê para os processos criados pelos usuários onde "processo" denota um programa em execução (Den71). A interpretação de processo pode ser bem mais ampla, incluindo também os criados pelo próprio sistema para realizar funções não acessíveis aos usuários. Dijkstra (Dij68) por exemplo definiu um sistema como uma sociedade de processos sequenciais cooperando harmoniosamente onde cada processo sequencial pode ser um programa de usuário, um periférico de input, um periférico de output, um controlador de segmento ou um interpretador de mensagem.

A idéia de processo originou-se de projetos no início dos anos 60, por exemplo, no MULTICS e OS/360 (onde se chamava task). Era entendida como a abstração da atividade de uma unidade de processamento. Outras interpretações foram sugeridas:

"local de controle de uma sequência de instruções" Dennis

"funcionário realizando os passos de um algoritmo" Van Horn

"programa em execução em um pseudo-processador" Saltzer.

"sequencia de estados de um programa" Horning.

O objetivo dessas definições é basicamente o mesmo: distinguir o programa estático do processo dinâmico, sugerindo que programa é uma sequência de "instruções" e processo é uma sequência de "ações" realizadas pelo programa.

Existe uma ambiguidade na associação entre os termos processo e programa. Há sistemas nos quais cada programa de usuário dá origem a apenas 1 processo e mesmo chamando várias procedures, o processo permanece o mesmo. Este é o caso mais comum. Outros sistemas, entretanto, associam o processo à função realizada por um processador virtual e assim um programa de usuário determina a criação de vários processos. Esta ambiguidade existe, mas não é difícil, nas primeiras referências à palavra processo se identificar o quão amplo (ou restrito) é o seu significado em cada caso.

Processos são chamados sequenciais pelo fato de possuirem um conjunto de ESTADOS. Cada estado é atingido discretamente pela execução de uma ação numa sequência, partindo do estado inicial inativo e retornando ciclicamente ao estado final inativo após percorrer os outros estados.

Um processo é dito "rodando" se o seu estado indica que existe algum processador no sistema executando instruções desse processo. Um processo é chamado "pronto" se pode entrar em execução em algum processador. Um processo que não está rodando nem pronto é chamado "suspenso" e está esperando por um evento externo, por exemplo, término de uma operação de I/O (Lis 72).

Em um sistema físico de computador um processo é representado pela informação que pode ser carregada no processador para permitir a execução do conjunto de instruções. Esta informação está associada ao estado do processo e deve incluir palavras do acumulador, palavras de índice, próxima instrução a ser executada e indicadores dos objetos que o processo está autorizado a acessar durante a sua execução (DeV 66). Estes indicadores constituem o "domínio de execução" do processo e é a parte mais relevante para o estudo dos mecanismos de proteção. Formas de implementar domínios de execução são por exemplo, descritores de segmentos, listas de capabilities e anéis de proteção. A transferência de controle entre domínios de execução deve ser manuseada com grande cuidado, já que geralmente implica na aquisição de novos direitos de acesso. A ocorrência de eventos excepcionais como overflow, violação de memória, fim de arquivo, geralmente causa uma mudança no domínio de execução que deve ser prevista e tratada com antecedência. Muitos atacantes forçam a ocorrência dessas situações imprevisíveis para ganhar acesso não autorizado a objetos protegidos (Lam 69).

Como existem vários processadores num sistema, podem existir vários processos ativos simultâneamente, o que significa que vários processos irão compartilhar os recursos do sistema. Para que isto seja feito de uma forma controlada, é preciso que hajam mecanismos de comunicação e sincronização entre os processos que permitam uma distribuição proporcional às necessidades de cada um. Se cada processo tiver meios de executar num ambiente protegido, comunicar-se de maneira segura com seus companheiros e realizar suas funções na ordem de precedência e de paralelismo especificada, o sistema contará com um grupo eficiente de "funcionários" trabalhando e cooperando entre si para o seu bom desempenho. Mecanismos de proteção propostos com esses objetivos serão examinados nas seções seguintes.

VI.2 DOMÍNIOS E SEGMENTAÇÃO

Um mecanismo de proteção preenche duas funções básicas: isolamento e acesso controlado. Isolamento refere-se à restrição de cada sujeito (no caso o processo) só ter permissão de usar objetos cujo acesso lhe tenha sido autorizado. Controle de acesso refere-se aos vários tipos de permissão que os sujeitos possuem para acessar o mesmo objeto de diferentes maneiras.

A implementação de um ambiente de isolamento ou domínio de execução é feita pela escolha de uma regra de controle de acesso. O tipo mais primitivo de hardware que implementa domínios de execução é a CHAVE DE MODO para a execução de instruções. Dois modos são especificados: mestre e escravo. No modo mestre qualquer instrução pode ser executada, incluindo um subconjunto denominado "instruções privilegiadas". No modo escravo qualquer tentativa de executar essas instruções privilegiadas causa uma fault. O grupo das instruções privilegiadas inclui instruções de I/O, instruções de mudança de modo e mudança do registro de limite de memória. Este registro bloqueia qualquer acesso de usuário, no modo escravo, à informação escrita abaixo de um certo nível da memória (usualmente onde reside o supervisor). Exemplos de sistemas baseados em mestre/escravo são: OS/360-MVT, CDC 6600 e CTSS (Gra 68).

Proteção baseada nesse tipo de hardware é uma solução tipo tudo ou nada. Se um programa tem qualquer privilégio, tem todos. Se tem qualquer acesso, tem acesso completamente irrestrito. Como mecanismo de proteção não é satisfatório. Em um sistema onde usuários precisam compartilhar dados na memória de trabalho são necessários mais controles, como a habilidade de variar os direitos de acesso para cada bloco lógico de informação.

Máquinas posteriores tiveram seu hardware modificado para permitir a divisão da memória em um grande número de partes ou SEGMENTOS cada qual com chaves de controle de acesso do tipo: mestre/escravo, ler/escrever, executar/não-executar. Entretanto ainda aí qualquer usuário que tivesse acesso em geral tinha o mesmo acesso que os outros usuários àquele segmento físico da memória. Nas máquinas subsequentes o próprio hardware possuia características em que o acesso variava de acordo com o segmento lógico, permitindo que diferentes usuários tivessem diferentes tipos de acesso ao mesmo segmento físico. Foi assim que apareceram os mecanismos de proteção baseados em segmentação.

Enquanto a página é uma unidade física de memória invisível ao usuário, o segmento é uma unidade lógica de memória cuja informação é controlada e protegida segundo regras determinadas por este usuário. A maioria dos computadores que usam segmentação possuem também hardware de paginação, mas este detalhe é irrelevante para esta análise.

Um segmento é um bloco de palavras cujo comprimento pode variar durante a execução de um processo. É a menor quantidade de informação que pode ser protegida separadamente no mecanismo de memória segmentada. Os endereços absolutos do hardware não são utilizados, mas cada palavra é endereçada pelo par de inteiros (S, W) onde S é o número do segmento e W o número da palavra no segmento.

Associado a cada segmento da memória existe um DESCRITOR DE SEGMENTO (Fig. 6.1) contendo a sua localização absoluta, tamanho corrente e indicadores de controle de acesso. Os indicadores de controle de acesso podem por exemplo ser os seguintes:

X	executável como procedure incluindo referências internas a constantes
R	legível como dados mas não executável
XR	executável como procedure e legível como dado
RW	legível e redigível
XRW	executável, legível e redigível

Cada processo do sistema possui um segmento especial chamado SEGMENTO DE DESCRITORES que contém os descritores de todos os segmentos que o processo pode acessar (Fig.6.2). Este segmento de descritores representa exatamente o domínio de execução do processo, portanto, no sistema, o número de segmentos de descritores é igual ao número de processos ativos. Sempre que um processo está executando um registro do hardware chamado registro de base de descritor contém a localização absoluta do seu segmento de descritores. O número do segmento S, usado no endereçamento, é de fato o índice no segmento de descritores do descritor desse segmento. Pode alternativamente ser usada uma chave K como na fig. 6.2 e nesse caso cada palavra no segmento é referenciada pelo par (K, W). Cada descritor é armazenado com um bit de presença que diz se o segmento está na memória principal. Como as chaves não se modificam, programas não são afetados pela relocação de segmentos.

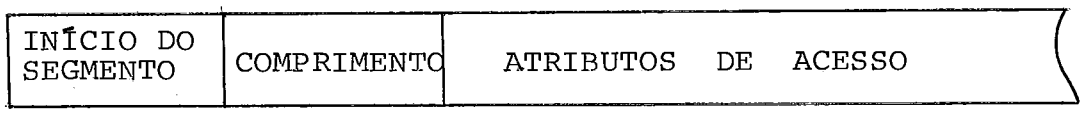


Fig. 6.1 - Descritor de segmento

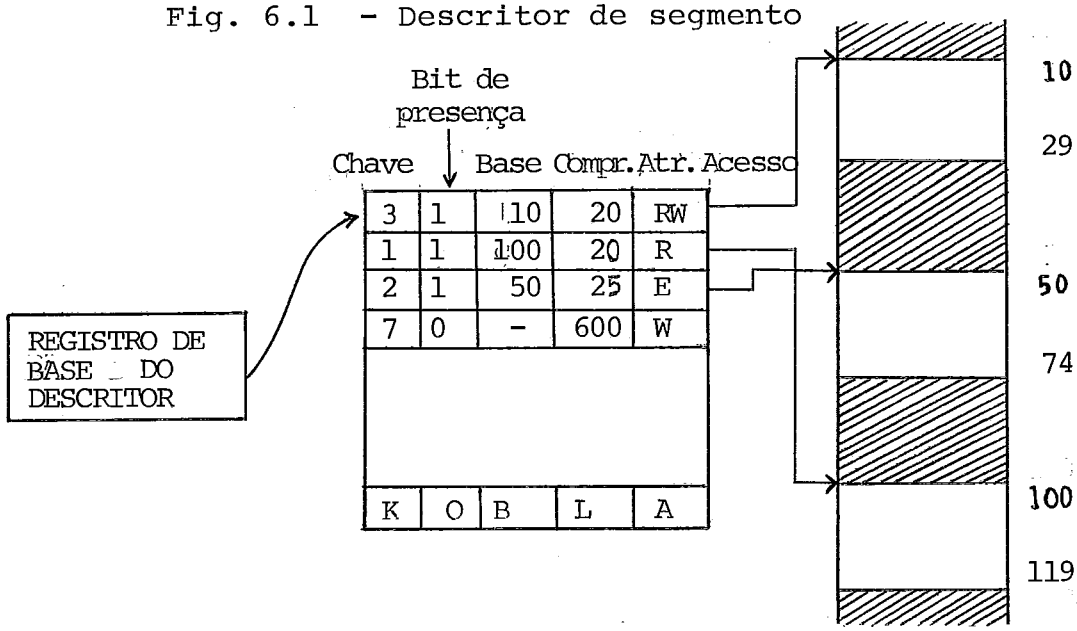


Fig. 6.2 - Segmento de descritores (DeD 79)

Quando segmentos são compartilhados entre vários processos, o problema se torna mais complicado. Existem 4 soluções possíveis (Fab 74):

ENDEREÇAMENTO DIRETO - Os descritores dos vários segmentos compartilhados devem ser definidos centralmente, uma tabela para cada programa. Assim, é impossível que um processo execute vários subprogramas constituídos independentemente. Todo o programa, inclusive todas as subrotinas, devem ser compilados de uma só vez. É o caso do Burroughs B 5700.

ENDEREÇAMENTO INDIRETO - Cada processo tem uma tabela de segmentos e tantos segmentos de ligação quantas forem as subrotinas. Esses segmentos de ligação fazem a correspondência entre os índices da tabela e os dos segmentos compartilhados. Gasta muita memória para alocar todos os segmentos de ligação e sobrecarrega o sistema para manusear a informação de indireção. É usado no Multics.

MÚLTIPLAS TABELAS DE SEGMENTOS - Existem tantas tabelas de segmentos quantas forem as subrotinas mais uma para o programa principal. Cada tabela de segmentos agora é privativa de um programa. A desvantagem principal é que desaparece a tabela de segmentos única do processo, que poderia ser compartilhada por vários programas do mesmo processo e utilizada para passagem de parâmetros, etc. Esta solução é encontrada no Burroughs B 6700.

CAPABILITIES - Todas as tabelas de segmentos são substituídas por capabilities, não são necessárias informações de indireção e uma única tabela é suficiente para todo o processo. Essa solução será analisada detalhadamente na seção VI.4.

Domínios de proteção não dizem respeito apenas às referências à memória, mas a qualquer outro objeto do sistema. Entretanto, podem ser sintetizados pelos objetos de memória (páginas e segmentos) se considerarmos que outros objetos terão que ser na maioria dos casos armazenados naqueles. Daí decorre que alguns sistemas utilizam a segmentação como um mecanismo básico de proteção, sofisticando-o de acordo com as especificações de cada caso. Frequentemente a decisão de projeto mais importante se refere ao tamanho de cada segmento. Este deve ser tal que permita ao usuário flexibilidade na proteção dos seus vários tipos de objetos e simultâneamente não sobrecarregue demais o sistema (Fab 74). Experiências com máquinas Burroughs indicam que quando segmentos são alocados em termos de unidades "naturais" para o problema que está sendo solucionado, o tamanho dos segmentos é em média menor que as páginas tipicamente usadas hoje em dia. Usuários tendem a juntar vários

objetos no mesmo segmento para fazer seus programas rodarem mais eficientemente. Entretanto, esta prática deve ser desencorajada, pois não há mecanismos de proteção que permitam a esses objetos serem tratados individualmente. M. O'Halloran, um dos projetistas do Plessey System 250 sugeriu o contrário, isto é: muitos segmentos por página, em vez de muitas páginas por segmento, o que permitiria um tratamento adequado para tantos segmentos pequenos. T. Linden (Lin 76) sugeriu a criação de pequenos domínios de proteção, que seriam ambientes de execução de módulos de programas, onde cada módulo era independentemente protegido dos seus similares e cada programa executaria em vários pequenos domínios. A criação de pequenos domínios é uma das formas mais eficientes de simplificar o desenvolvimento e a manutenção de software: é mais difícil escrever 1 programa de 50.000 linhas do que 1.000 programas de 50 linhas de programação.

VI.3 ANÉIS DE PROTEÇÃO

Processos ao executarem, necessitam vários domínios de execução. ANÉIS DE PROTEÇÃO são uma forma de implementar domínios de execução, pois consistem de uma coleção encapsulada de objetos uniformemente protegidos. Um processo em um anel só pode acessar aqueles objetos que pertencem ao seu anel. Uma pequena diferença existe entre ANÉIS e DOMÍNIOS propriamente ditos: os anéis contém realmente os objetos, enquanto os domínios contém apenas nomes ou referências aos objetos. Devido a isso, os anéis podem se sobrepor, pois há objetos que pertencem a vários anéis. (Pop 74).

O mecanismo dos anéis de proteção foi idealizado e implementado no MIT para permitir aos usuários a criação de subsistemas isolados. Mais tarde verificou-se que era também um excelente mecanismo de proteção para defender o supervisor dos acessos não autorizados de usuários (BCD 72). Toda esta seção se refere ao MULTICS, único sistema conhecido que implementa anéis de proteção, embora a filosofia seja aplicável a qualquer outro sistema (Sal 74a).

Como foi dito, num sistema de memória segmentada, cada segmento tem um descritor contendo as suas regras de acesso e para cada processo ativo existe um segmento especial chamado segmento de descritores contendo os descritores de todos os segmentos que aquele processo pode potencialmente acessar.

Entretanto, ao contrário da maioria dos sistemas operacionais, no MULTICS o supervisor não reside em espaço protegido da memória nem opera em processo dedicado. Em vez disso, os segmentos reais que contêm a codificação do supervisor são potencialmente acessíveis por todos os usuários, o que é o mesmo que dizer que no MULTICS o supervisor executa debaixo de um processo do usuário.

A situação se desenrola da seguinte maneira: sempre que um job entra no sistema, um novo processo é criado e o seu segmento de descritor é inicializado com descritores para todos os segmentos do supervisor, permitindo que o processo realize todas as funções básicas do supervisor por si próprio. A execução do supervisor no espaço de endereçamento de cada processo facilita a comunicação entre as procedures do usuário e do supervisor. Por exemplo, o usuário pode chamar uma procedure do supervisor como se estivesse chamando uma própria. Além disso, o compartilhamento do supervisor facilita a execução simultânea, por vários processos, das funções do supervisor, exatamente como o compartilhamento das procedures de usuários facilita a execução simultânea de funções escritas por usuários. Mais ainda, a organização do supervisor em segmentos de procedures e dados permite que ele desfrute das mesmas facilities dos segmentos de usuários, quais sejam: paginação, convenções de chamada-retorno, etc.

Tudo bem até aí. Mas o fato de tornar o supervisor tão acessível, é claro, tornou-o também mais vulnerável. Uma vez que os segmentos do supervisor fazem parte do domínio de execução de cada processo, é necessário que ele seja protegido adicionalmente, contra referências não autorizadas. Este papel é desempenhado pelos anéis de proteção (Gra 68).

O mecanismo dos anéis de proteção foi feito para criar um ambiente de isolamento na execução de cada processo, de maneira que o acesso a cada segmento, em qualquer momento seja controlado por diferentes tipos de privilégios. A realização prática do princípio 'need to know' (característica 9 da seção II.3) gerou a idéia de várias camadas de proteção em um mesmo processo, suficientes para proteger ambos, os processos do usuário e do supervisor.

Portanto, os anéis de proteção podem ser vistos com "múltiplos domínios de execução de um processo" (Sch 75). A cada segmento é atribuído um número de anel especificado no seu descritor (fig 6.3). Quanto mais baixo o número do anel em que uma procedure se encontra, maiores os seus privilégios de acesso. Uma procedure executando no anel j não tem acesso a nenhum segmento do anel i se $i < j$. Em compensação tem acesso a todos os segmentos do anel k se $k > j$.

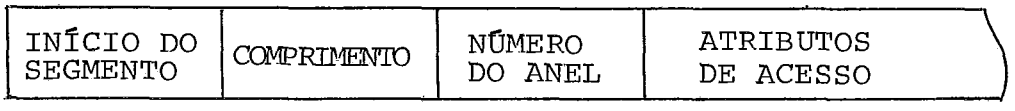


Fig. 6.3 Descritor de segmento contendo número do anel

Para realizar esta restrição o sistema deve estar consciente de todas as trocas de controle entre anéis. Quando uma procedure executando no anel j tenta transferir o controle para um procedure em qualquer outro anel, ocorre uma fault que é dirigida para o supervisor (processos do supervisor) e manuseada por ele. Todos os argumentos passados devem ser verificados, inclusive endereços, cuja validação é feita para evitar que procedures de anéis inferiores causem dano às dos anéis superiores e vice-versa. Após a chamada ter sido validada o ponto de retorno e seu correspondente número de anel são registrados numa stack, de maneira que na volta, as funções inversas são realizadas e a procedure anterior recebe novamente o controle.

Na primeira versão do MULTICS os anéis de proteção eram realizados por software, através das chamadas ao supervisor mencionadas acima. Chamadas que atravessavam anéis eram caras por isso e muitos processos foram desnecessariamente colocados no supervisor, tornando-o excessivamente complexo. Mais tarde foi desenvolvido um mecanismo de hardware, capaz de controlar essas transferências, embutido nas próprias sequências de código objeto que realiza todas as chamadas e retornos (ScS 72). Esse mecanismo é tal que quando uma procedure, por exemplo do anel j chama outra por exemplo do anel i, onde $i < j$, a validação dos argumentos em i é feita levando em consideração apenas os privilégios de j, exatamente como seria se a procedure estivesse executanto em j. Após completada a validação é que a procedure adquire todos os privilégios correspondentes ao anel i. Portanto, a procedure se protege a si mesma, através do hardware, de executar por ordem da outra, operações que a outra não estava autorizada a realizar. Chamadas que atravessavam anéis passaram a custar o mesmo que as internas ao anel, permitindo a remoção de muitos módulos do supervisor e a utilização ampla da filosofia de anéis.

Finalmente vamos examinar como foi feita a distribuição de funções supervisor-usuário no MULTICS:

. ANEL 0 - procedures de mais baixo nível do SUPERVISOR: operações primitivas de controle de acesso, I/O, multiplexação de memória e de processador, etc.

. ANEL 1 - segunda camada do SUPERVISOR: contabilidade, gerenciamento do stream de I/O, etc.

. ANÉIS 2 e 3 - disponíveis para a construção de subsistemas protegidos de USUÁRIOS, exemplos: um compilador ou um subsistema para historiar acessos a um banco de dados, etc.

. ANEL 4 - procedures normais de USUÁRIOS.

. ANÉIS 5, 6 e 7 - disponíveis para a autoproteção de USUÁRIOS exemplos: depuração de programas, utilização de programas emprestados, etc.

A invocação de algumas procedures do anel 0 é possível apenas de forma implícita e em situações especiais. Outras procedures do anel 0 e 1 podem ser explicitamente invocadas por procedures dos anéis 2 a 5 através de "portas" padronizadas, mas nunca por procedures dos anéis 6 e 7. Algumas portas do anel 0 são acessíveis a todos os processos de usuários, mas apenas se chamadas por procedures executando no anel 1. Essas portas são a interface interna entre as 2 camadas do supervisor. Usuários selecionados executando em 2-5 tem acesso a portas especiais de 1, por exemplo, a porta para registrar novos usuários.

Os subsistemas executando em 2 e 3 podem ser protegidos das procedures de 4 a 7. Diferentes subsistemas protegidos podem operar simultâneamente nos anéis 2 e 3 de vários processos e vários processos podem compartilhar simultâneamente o mesmo subsistema protegido. A construção desses subsistemas abaixo do supervisor elimina a necessidade de inclusão dos mesmos no supervisor.

As vantagens da estrutura de anéis para o sistema são inúmeras: limitação da propagação de erros, maior simplicidade na modificação, eliminação no supervisor de muitas funções que agora são executadas por processos dos usuários, proteção pelo esquema de portas de chamada. Além disso, através dos anéis 2 e 3 os usuários podem construir seus próprios subsistemas protegidos que suportam a sua política de segurança específica.

VI.4 DOMÍNIOS DEFINIDOS POR CAPABILITIES

Um domínio de execução pode ser definido discrecionàriamente por CAPABILITIES ou LISTAS DE ACESSO. No primeiro caso o domínio corresponde à C-lista do processo, no segundo caso é o conjunto de todos os objetos que incluem o processo na sua lista de acesso.

A maior atração do método de capabilities é a sua eficiência de execução. Suponha-se que cada domínio tem acesso a uma média de "m" objetos e cada objeto seja acessível a "n" domínios. A memória esperada para armazenar as capabilities de um processo ativo no método das capabilities é do tamanho da C-lista, ou m localizações de memória. Já no método da lista de acessos, para garantir a mesma velocidade, seria necessário usar uma média de m listas de acesso, o que corresponderia a nm localizações de memória rápida. Isto porque o working-set das capabilities está armazenado em um só local e o das listas de acesso está disperso.

Outra distinção entre os dois métodos diz respeito à verificação da autorização. O método da lista de acessos necessita checar a autorização a cada acesso, o método das capabilities associa a permissão com o domínio e checa apenas uma vez.

A desvantagem das capabilities é, como se sabe, a revogação de permissões, que nunca é feita com facilidade. Pode-se dizer que o método das capabilities permite um eficiente USO dos direitos de acesso e o método das listas de acesso permite uma eficiente ADMINISTRAÇÃO dos mesmos (Den 76b).

O mecanismo de CAPABILITIES é o meio conhecido mais eficiente para a construção de domínios de proteção: permite estruturas de "pequenos domínios" e processos de múltiplos domínios, além de prover um sistema único e simples de proteger tanto os objetos do sistema como os dos usuários. Enquanto outros sistemas possuem hardware de endereçamento para controlar o acesso aos segmentos, mecanismo de execução de instruções para controlar o acesso a procedures e sistemas de arquivos para controlar o acesso a arquivos, o mecanismo de capabilities pode sozinho controlar todos esses objetos.

Os dois tipos mais usados de capabilites são o de memória e o de entrada. Uma CAPABILITY DE MEMÓRIA autoriza o acesso a um segmento na memória virtual de um processo. Pode ter código de acesso como ler, escrever ou executar, dependendo se a referência à memória foi feita durante a porção "execute" ou "fetch" de uma instrução. Uma CAPABILITY DE ENTRADA autoriza a invocação de uma procedure em um domínio diferente do domínio da procedure que chamou.

A fig. 6.4 ilustra o mecanismo de acesso para capabilities de memória. Cada domínio d tem associada sua própria lista de capabilities CL (d) cujas entradas (j: c, x) especificam que o nome local (j) do objeto pode ser transladado no nome real (x) com seu código de acesso (c). Uma tabela central (CMT) contém os endereços dos objetos que podem estar na forma base-limite (b, l) se o objeto estiver presente na memória principal ou na forma endereço (a) se o objeto estiver na memória secundária. Memórias associativas podem armazenar diretamente a trajetória (j, c, b, l) omitindo a tabela principal e caindo no caso do "descritor de segmento". Este meca-

nismo, é claro aumenta a velocidade da translação e torna a lista de capabilities permanente visto não ser necessário alterações decorrentes da paginação.

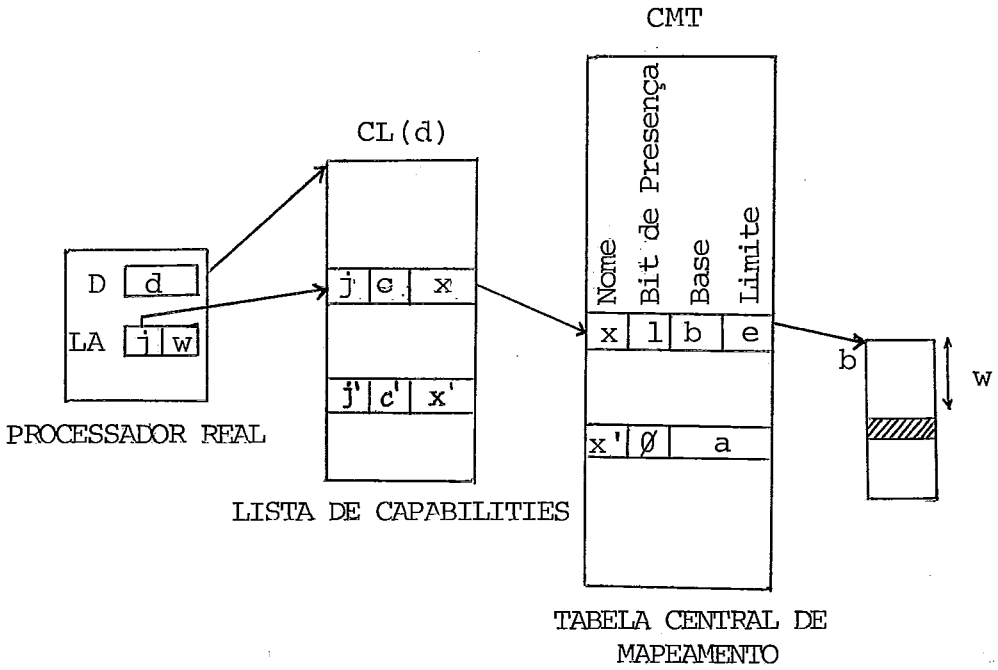


Fig. 6.4 - Capabilities de memória (Den 76b)

Capabilities de entrada são mais complicadas, pois devem passar parâmetros, mudar o registro correspondente ao domínio de execução vigente e forçar o pointer de instrução para o início da nova procedure. A fig. 6.5 ilustra como a procedure p_1 de domínio d_1 chama a procedure p_2 de domínio d_2. A capability de j_1, "execute", autoriza a execução da procedure p_1, que ao encontrar a instrução "enter p_2" vai à lista verificar se existe uma capability autorizando a execução de p_2. Caso positivo, as variáveis de p_1 são armazenadas na stack e a execução de p_2 é iniciada pelo exercício da primeira capability do domínio novo d_2. No retorno, as variáveis são recuperadas da stack.

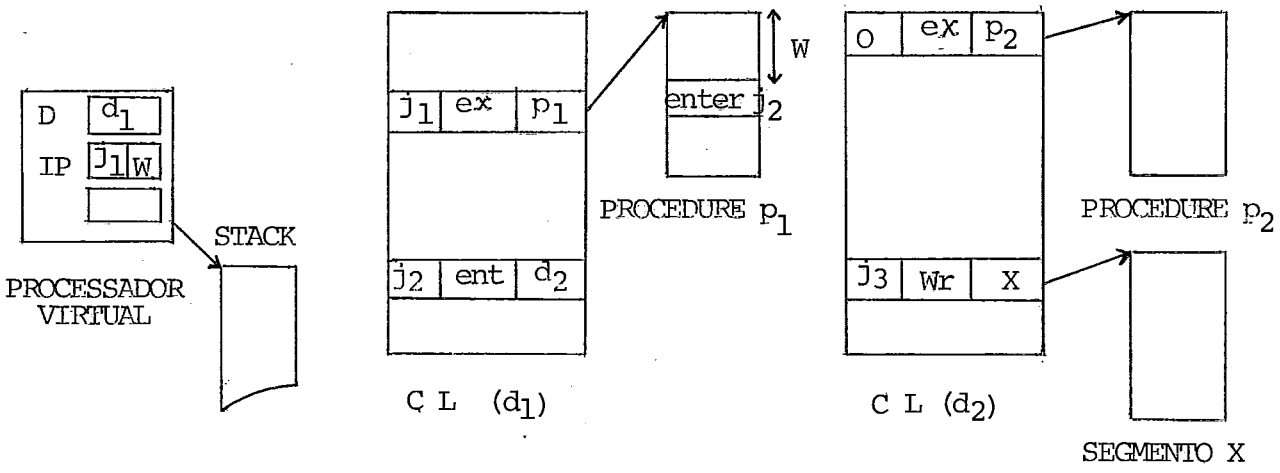


Fig. 6.5 - Capabilities de entrada (Den 76b)

No sistema HYDRA (CoJ75) por exemplo, os domínios de execução são definidos por capabilities. Cada processador, dos 16 existentes, executa em um domínio protegido, que corresponde à lista de todos os objetos do sistema (com suas capabilities) que o processo pode potencialmente acessar. Essa lista é também um objeto protegido do sistema, chamado LNS (espaço de nomes locais) e contém 3 tipos diferentes de capabilities: as independentes de chamada, as dependentes de chamada e as templates. Capabilities independentes de chamada são inerentes ao processo, estão sempre presentes em qualquer execução do mesmo. As dependentes de chamada agem como parâmetros e só são conhecidas pelo processo durante a execução, pois dependem do processo ou do usuário que chamou e são perdidas ao término da execução. As templates são capabilities conhecidas pelos dois processos, pelo que envia e pelo que recebe e exercem a função de validação da chamada. O domínio de execução do processo é portanto a soma de todas essas capabilities e das outras capabilities que podem existir eventualmente na parte de representação das primeiras, gerando uma árvore de objetos acessíveis, em cuja raiz está o LNS.

No sistema BCC 500 (Lam 69) cada domínio é também um objeto protegido e tem as suas próprias capabilities. A qualquer instante um processo pode estar executando em um domínio utilizando as capabilities desse domínio. Uma delas pode referenciar outro domínio e o processo, ao utilizá-la, mudará seu domínio de execução. Assim, um processo rodando em um domínio pode executar, expandir os cálculos, acessar arquivos e outros objetos sem se preocupar por quem ou quando foram criados. Do ponto de vista de controle, existe uma verdadeira hierarquia de domínios em que um domínio pode controlar outro já que as capabilities do controlado são um subconjunto das capabilities do que controla. Esta estrutura de processos foi idealizada para tornar o sistema mais eficiente, visto que processos são entidades caras, precisam manter dados em memória privativa, enquanto que domínios são baratos, basta um conjunto de bits para representá-los.

O mecanismo de proteção proposto por Needham no CAP System (Nee 72) usa 4 tabelas de capabilities simultaneamente: uma para as capabilities globais ao processo, uma para as capabilities associadas ao processo corrente e compartilhadas por todos os processos que usam esse programa, uma para as capabilites associadas ao programa corrente e privativas de 1 processo e uma para os argumentos da ativação corrente do processo. O domínio de proteção corrente de 1 processo é definido por essas 4 listas. Ao chamar uma procedure protegida (instrução de máquina ENTER) um processo troca o seu domínio corrente para o domínio da procedure protegida. Cook (Coo78) realizou experiências comparando o custo da troca de domínios através de chamadas a procedures protegidas no CAP e o mesmo custo num sistema operacional tipo mestre-escravo. Suas conclusões foram de que o uso das capabilities, naturalmente, aumenta o tempo de execução mas o mecanismo é muito mais poderoso.

Listas de capabilities podem ser criadas estaticamente durante a compilação em ambientes onde os módulos de programas correspondem a gerentes de extensões de objetos e as capabilities apontam para objetos específicos. Esta situação existe no CAP, no HYDRA e no Plessey 250.

Finalizando, vamos resumir as propriedades mais importantes do uso das capabilities na formação de domínios de execução protegidos.

- Capabilities são permanentes, podem ser armazenadas em programas ou na memória auxiliar por um período indefinido.

- Capabilities podem ser transmitidas eficientemente como endereços a procedures em vários domínios e como componentes de mensagens interprocessuais.

- Outros objetos do sistema, ex: fitas de mensagens ou do usuário, arquivos, podem utilizar-se do mesmo mecanismo.

- Pequenos domínios podem ser criados para permitir que subunidades ou módulos de programas sejam executados em ambientes restritos.

- O mecanismo de capabilities é uma base natural para a implementação de ambientes fechados.

VI.5 COMUNICAÇÃO E COOPERAÇÃO

Processos que são logicamente independentes não cooperam entre si e o controle na concorrência aos recursos é normalmente exercido pelo sistema. Entretanto, processos que são logicamente dependentes interagem diretamente e controlam explicitamente sua concorrência aos recursos. Além disso, qualquer processo realizando uma tarefa pode gerar um número finito de tarefas para outros processos. Em arquiteturas hierárquicas, como no THE (Dij 68) processos só podem gerar tarefas para processos em níveis mais altos da hierarquia. Em arquiteturas planas como no HYDRA (WCC 74) processos podem gerar tarefas para quaisquer outros processos. Três condições são necessárias para haver cooperação entre processos:

- Qualquer processo só pode gerar um número finito de tarefas ou chamadas a outros processos. Essa condição é necessária para que o sistema não entre em loop.

- Enquanto houver alguma tarefa pendente no sistema tem que haver algum processo ativo. Ou seja: é impossível que todos os processos estejam "prontos" se ainda há serviço a executar.

- Em algum momento do sistema cada processo deve retornar ao seu estado inativo. Ou seja: nenhum processo pode ficar permanentemente ativo (como nos casos de deadlock).

Assim, quando um sistema é baseado na cooperação entre processos os mecanismos de comunicação devem ser eficientes, fáceis de entender, fáceis de usar e seguros contra uso não autorizado e deadlock. Existem duas formas básicas de se estabelecer a comunicação entre processos: por compartilhamento de dados e por mensagens (SAG 72).

COMPARTILHAMENTO DE DADOS

Um caso de cooperação realizada através do compartilhamento de dados é o conhecido problema do "produtor/consumidor". Dois processos cíclicos, o produtor e o consumidor compartilham um buffer de n células de memória, onde o produtor coloca ítens para serem usados mais tarde pelo consumidor. O produtor pode ser, por exemplo, um processo que gera uma linha de saída por instante e o consumidor um processo que opera a linha da impressora. A sincronização é imprescindível nesses casos. O produtor deve ser impedido de colocar um ítem num buffer cheio e o consumidor deve ser impedido de remover um ítem de um buffer vazio.

No sistema THE (Dij 68) a sincronização é realizada através de dados compartilhados chamados semáforos. Semáforo é uma variável inteira que tem associada uma lista de espera. Se a variável inteira é negativa, seu valor absoluto é igual ao número de processos da lista. Duas operações podem ser realizadas nos semáforos: P e V. P é realizada quando um processo deseja esperar pela ocorrência de um evento ou pela disponibilidade de um recurso. Ao realizar a operação P o processo incrementa a lista de espera em 1 unidade. V é realizada quando um processo libera um recurso. Ao realizar V o processo decrementa a lista de espera de 1 unidade.

O sistema VENUS (Lis 72) utiliza também semáforos para a sinalização entre processos. Como em outros sistemas, no VENUS todos os processos estão em um de 3 estados possíveis: prontos, rodando ou bloqueados. Uma lista especial J contém os processos prontos para rodar. Processos bloqueados estão nas listas de espera dos semáforos. Se um processo rodando solicita um recurso que não está disponível ou espera pela ocorrência de algum evento fica bloqueado e incrementa a lista do semáforo (P). Neste ponto, é escolhido na lista J o processo pronto, de mais alta prioridade, para executar. Se uma V é realizada, algum processo bloqueado se torna pronto e é adicionado à lista J, exceto se tiver maior prioridade que o processo em execução no momento, caso em que este vai para a lista J e o novo processo pronto inicia imediatamente a execução. Assim, o processo rodando tem sempre prioridade no mínimo igual aos processos prontos.

A comunicação por dados compartilhados geralmente é feita usando um segmento compartilhado especificamente definido para êsse fim. No momento da criação o segmento já é tornado compartilhável e seu acesso só pode ser feito através de um protocolo combinado entre todos os processos que o usarão. Esta forma de comunicação, entretanto, só é interessante quando a taxa de dados é baixa. Isto porque é uma forma preemptiva de comunicação: há necessidade de suspender a execução de um processo enquanto espera pela resposta do outro (CaD 79). Por exemplo, dispositivos periféricos de baixa taxa de dados comunicam-se por compartilhamento de buffer, implementando um tipo de cooperação produtor/consumidor. O processo solicita a transmissão de uma unidade de informação (palavra, bloco, etc) e espera em estado suspenso até que a informação seja transmitida.

MENSAGENS

Uma mensagem é uma cadeia de caracteres onde está prefixado o número de identificação do emissor (SAG 72). Todas as mensagens direcionadas a um determinado objeto são colocadas num buffer associado a esse objeto de onde serão retiradas mais tarde pelo destinatário. É uma forma não-preemptiva de comunicação, pois nem o emissor nem o destinatário necessitam suspender sua execução como consequência do envio ou recebimento da mensagem. Estabelecer o canal de comunicação pode ser uma operação cara, mas uma vez que a ligação esteja feita, um mínimo de verificação é necessária na transmissão de cada mensagem.

O princípio mais importante que rege a comunicação por mensagem é que o número de identificação do emissor não pode ser esquecido. Este é um dispositivo de segurança que impede por exemplo um processo de enviar uma mensagem maliciosa a outro processo. Chamadas protegidas a procedures podem ser feitas através de mensagens. Uma mensagem contendo a lista de parâmetros é enviada à procedure e o emitente entra em estado de espera apenas se precisar do resultado para continuar a sua execução. Quando o destinatário encontra a mensagem no seu buffer realiza a função solicitada e envia outra mensagem com os resultados. O sistema RC 4000 (Han 70) opera desta maneira.

Quando se escolhe o método da transmissão de mensagens, a comunicação interprocessual tem 2 formas: PORTAS ou FACILITIES.

1. PORTAS OU CAIXAS DE CORREIO - Neste esquema cada processo possui um número de PORTAS que são a interface de comunicação com outro processo. Um mecanismo de comunicação através de portas interessante é o do UCLA SECURE UNIX (PoF 78). Baseado em capabilities, o mecanismo é basicamente composto por duas instruções executadas pelo núcleo do sistema operacional:

   . NOTIFY (capability-2, processo-2) - O processo 1 envia ao processo 2 uma notificação de que existe para ele uma mensagem na sua porta.

   .RECEIVE (capability-1, processo-1) - O processo 2, que recebeu o NOTIFY, busca na sua porta a mensagem que lhe foi destinada e notifica esta ação ao processo 1.

Como ambas as instruções são interpretadas e executadas pelo núcleo, que verifica se as capabilities estão corretas, o mecanismo protege contra tentativas maléficas de um processo influir incorretamente na execução de outro. Uma CAIXA DE CORREIO pode ser inserida na linha de comunicação com vários pontos de mensagens, proporcionando uma buferização automática de mensagens.

No Cm* (JCD 77) as caixas de correio são objetos protegidos do sistema. Cada caixa é capaz de conter um número fixo de mensagens mantidas em ordem FIFO. Cada processo só pode depositar uma mensagem numa caixa se houver espaço.

Uma arquitetura modificada do CAP System foi proposta por Herbert (Her 78) introduzindo caixas de correio na comunicação interprocessual. No CAP comum a comunicação interprocessual é implementada inteiramente em software, sendo lenta e cara. Na arquitetura proposta um sistema de mensagens microprogramado permite que domínios compartilhem a mesma caixa de correio, transmitindo entre sí vários blocos de argumentos.

2. FACILITIES DO SISTEMA - São processos que provem serviços do sistema como alocação de dispositivos ou gerenciamento de arquivos e necessitam características especiais de comunicação. Isto porque o número de processos ativos que querem utilizar esses serviços simultâneamente é muito grande e não deve, teoricamente, ser limitado (na realidade em muitos sistemas há um limite máximo do número de processos por serviço). Prover cada processo desse tipo com um número de portas suficientemente grande implica na utilização de uma quantidade excessiva de memória. Assim, os processos responsáveis pela execução dos serviços do sistema são transformados em "facilities" e usados através de monitores contactados por "chamadas". A chamada não é bufferizada e o processo solicitante não pode prosseguir enquanto esta não for atendida pela facility. Usando facilities, um mecanismo de proteção deve ser empregado para determinar que facility pode ser chamada por cada processo. Dois mecanismos podem ser empregados: matriz de acesso e árvore de criação.

Na MATRIZ DE ACESSO, cada elemento (i, j) indica se o processo i pode chamar a facility j. Esta matriz se torna de difícil atualização pois os processos são criados e destruídos dinamicamente. A ÁRVORE DE CRIAÇÃO é a alternativa em que as chamadas das facilities obedecem à estrutura hierárquica de criação dos processos.

O sistema SUE (Lis 72) é um exemplo de comunicação por árvore de criação. Neste, uma facility pode ser chamada apenas pelos seus descendentes. A conversação entre 2 quaisquer processos pode ser realizada por chamadas da facility de cada um dos seus ancestrais comuns. As solicitações de serviços podem ser buferizadas simplesmente pela criação de um descendente para buferizar o pedido. Como todos os calls de facility são dirigidos em direção à raiz da árvore de criação, o deadlock pode ser evitado impondo a cada facility um limite de tempo para terminar cada pedido de usuário. Uma vez que o SUE é baseado em capabilities, cada processo possui uma lista das capabilities correspondentes às facilities que pode acessar.

Sistemas baseados em máquinas virtuais não permitem que duas máquinas virtuais diferentes compartilhem as mesmas páginas de memória virtual ou áreas comuns de memória virtual de disco. Portanto, a comunicação é feita também por mensagens. É o caso do sistema PRIME (Fab 73), em que cada máquina virtual pode enviar mensagens às outras máquinas. Cada vez que uma mensagem é enviada a um subsistema virtual, segue com ela uma identificação inconfundível do subsistema que a enviou. Para que o subsistema possa tomar decisões sobre o envio das suas informações privativas a outros subsistemas, é mantida em cada máquina virtual uma tabela de todas as outras máquinas que estão autorizadas a receber suas informações. Este esquema pode, entretanto, ser alterado para incluir um protocolo de 'handshaking' ou password em vez de uma lista pré-determinada e a identificação dinâmica será usada como um teste a curto prazo de que a informação está sendo passada ao mesmo subsistema que iniciou o protocolo de 'handshaking'.

Finalizando, em nenhuma arquitetura é tão necessário o uso de mensagens como nas arquiteturas distribuídas. Exemplos destas são o HYDRA, o STAROS, o MEDUSA e todas as redes de computadores. Nestes casos, a comunicação por mensagens atinge seu ponto de maior amplitude e mérito.

VI.6 CONFINAMENTO

O princípio fundamental da construção de um sistema operacional é o do FECHAMENTO, isto é: nenhuma ação é permitida a menos que tenha sido explicitamente autorizada (Den 76b).

Um princípio decorrente deste é o do Isolamento de processos, isto é: cada processo não deve ter capacidade de agir exceto no espaço estritamente necessário para realizar sua tarefa. Isto significa que todas as trajetórias de acesso devem ser previamente especificadas, não devem haver caminhos camuflados e o processo não deve reter privilégios concedidos estritamente para executar suas funções.

Na maioria dos sistemas operacionais, sempre que um usuário chama um programa pertencente a outro usuário ou um utilitário pertencente ao sistema operacional está correndo um risco. Ele não tem formas de estar seguro de que o programa que chamou não vai, maliciosamente ou inadvertidamente fazer alguma coisa errada com seus arquivos. A maioria dos usuários simplesmente aceita esse risco confiando nos sistemas de back-up para ajudar na recuperação em caso de desastre. Porém, em sistemas em que a segurança é importante, isso não é suficiente. O usuário precisa ter alguma forma ou caminho de evitar os danos que uma procedure chamada pode causar.

Um problema similar acontece com o autor de um programa ou utilitário que será colocado à disposição de muitos usuários diferentes. O utilitário presumivelmente deverá ter que manipular muitos arquivos privativos e estruturas de dados que não são nem podem ser conhecidos por quem os está utilizando. O autor do programa precisa ter algumas garantias de que, exceto durante a execução do programa, os usuários não terão acesso a essas estruturas de dados e não poderão danificá-las nem ao utilitário.

Esses dois problemas juntos são conhecidos como SUSPEIÇÃO MÚTUA (Lam 73). Nos sistemas projetados com preocupação de segurança significa que:

1. O dono de uma procedure não pode ter acesso aos objetos privativos do sujeito que a chama.

2. O sujeito que chama uma procedure não pode ter acesso aos objetos privativos da procedure.

A primeira parte do problema de Suspeição Mútua é a mais difícil de ser resolvida, e é conhecida como: Problema do Cavalo de Troia, pois envolve um "presente" de um estranho introduzido entre as "paredes" de um domínio de proteção.

O Problema do Cavalo de Troia tem merecido muito estudo e é bastante complicado. Programas que possuem rotinas subversivas são usados constantemente, podem existir em programas emprestados de outros usuários, em procedures do sistema como editores, compiladores, rotinas de biblioteca, etc.

Uma lista de possíveis escoamentos originados por serviços tipo Cavalo de Troia foi enunciada por Lampson (Lam 73).

1. Se o serviço tem memória, pode armazenar dados, esperar que seu proprietário o chame e retornar-lhe os dados.

2. O serviço pode escrever em um arquivo permanente no diretório de seu proprietário.

3. O serviço pode criar um arquivo temporário, e autorizar acesso ao seu proprietário, o qual testa o sistema periodicamente para verificar se o arquivo existe e copiá-lo antes que seja destruído.

4. O serviço pode enviar uma mensagem a um processo controlado por seu proprietário.

5. A informação pode ser sutilmente codificada na conta que é enviada ao proprietário.

6. O serviço pode enviar bits de informação a um programa rodando em paralelo simplesmente acendendo e apagando semáforos de procedures do sistema.

7. O serviço pode variar sua taxa de I/O, de paginação e introduzir variações na performance do sistema.

Uma solução razoável para o Problema do Cavalo de Troia é a criação de programas confinados. CONFINAMENTO é a qualidade da procedure que ao ser chamada não causa escoamento dos dados privativos de quem chama. Existem 3 condições básicas para que uma procedure seja confinada:

. FALTA DE MEMÓRIA - Um programa confinado não deve possuir memória. Um subsistema sem memória é um programa ou uma procedure que não guarda informação quando termina sua tarefa. Se o serviço tem memória, pode causar os escoamentos tipos 1 e 2 acima.

. ISOLAMENTO TOTAL - Um programa confinado não deve se comunicar com outro programa nem autorizar acesso a seus arquivos temporários. A falta de isolamento é a causa dos escoamentos tipo 3 e 4 acima.

. TRANSITIVIDADE - Se for absolutamente necessário a chamada ou a comunicação com outro programa, este deve também ser confinado. No caso do supervisor, por exemplo, a comunicação sempre é necessária, por isso, quando um programa emitir uma chamada ao supervisor, todos os canais devem ser testados. Pode-se distinguir 3 tipos de canais do supervisor: canais de memória (segmentos, etc.), canais legítimos (contas, etc.) e canais cobertos. Estes últimos representam as formas de escoar informação encobertas como os tipos 6 e 7, são difíceis de ser evitados mas são também complicados para serem realizados.

Lipner (Lip 75) sugere uma série de medidas para fechar os canais identificados por Lampson baseados na propriedade estrela dos modelos não-discrecionários (seção IV.1). Qualquer que seja o serviço produzido, um programa não pode escrever em um objeto de sensitividade inferior à do objeto que gerou a informação. Portanto, se o autor do programa não tem autoridade para acessar um objeto protegido, também não terá para acessar os objetos que o programa produz na sua saída. Todos os canais de memória e legítimos são protegidos por essa regra. Restam ainda os canais cobertos. Lipner observou que todos os canais cobertos estão associados a um único recurso do sistema: TEMPO, o qual pode ser observado e utilizado como veículo de informação. Baseado nessa descoberta, propôs que os canais cobertos sejam protegidos restringindo o usuário a ver apenas "tempo virtual", dependendo apenas da sua própria atividade. Um relógio virtual poderia ser associado a cada processo, que não teria mais formas de realizar operações ilícitas baseadas no tempo. Essa solução é cara, como todas as que se referem a canais cobertos e não se conhece nenhum sistema que a tenha implementado.

Uma solução plausível, também de hardware, é a de Fenton (Fen 74). Inicialmente, é proposto um modelo de serviço qualquer por exemplo uma procedure de cálculo de imposto. Para o usuário o serviço é como uma caixa preta. Requer informações confidenciais e não-confidenciais como entrada e emite dados confidenciais e não confidenciais na saída (fig. 6.6).

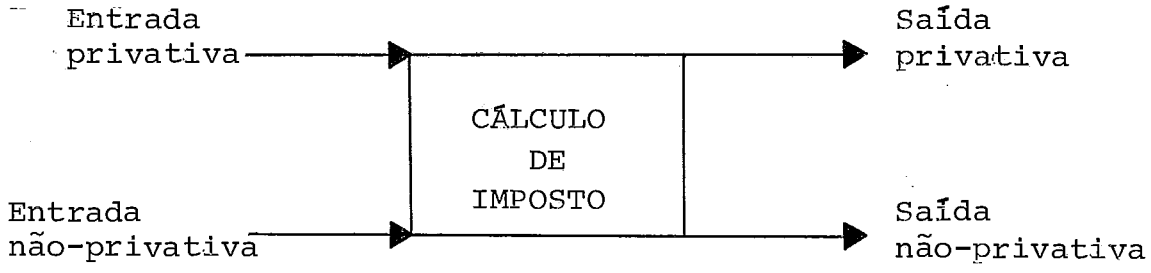


Fig. 6.6 Confinamento no I/O de uma procedure

A Data Mark Machine de Fenton possui um mecanismo de hardware associando a cada localização de memória uma marca fixa, que no caso mais simples significa "privativa" ou "nula". Informações privativas só podem ser armazenadas em localizações privativas, o mesmo acontecendo com as nulas. Uma tabela de decisões define o resultado de uma operação entre 2 dados (fig 6.7).

Dado 1 / Dado 2	nulo	privativo
nulo	nulo	privativo
privativo	privativo	privativo

Fig. 6.7 - Operações na Data Mark Machine

É claro que a condição necessária e suficiente para que um programa seja confinado é que nenhum bit de informação marcado privativo seja copiado ou alterado para nulo. Isso deve incluir também os fluxos implícitos. Fenton definiu e provou 5 teoremas que são a base de proteção do seu sistema:

1. Um sistema é seguro se e só se a trajetória nula do programa não precisar depender de nenhuma informação privativa. Assim, qualquer fluxo implícito de informação fica proibido.

2. Enquanto houver informação privativa sendo processada nenhum dado marcado nulo pode ser modificado.

3. Se a informação sendo processada for nula, não pode haver mudança de controle (ex: JUMP) que dependa de informação privativa. Se isto for necessário, a marca da informação deve ser alterada para privativa.

4. Uma vez atingido o estágio de privativo, a única forma de voltar para nulo é através de uma instrução de RETURN.

5. Uma procedure pode iniciar com informações nulas, executar inteiramente e parar sem nunca ser necessário acessar informações privativas.

Dessa forma, o próprio hardware da máquina se encarrega de propiciar as condições de confinamento.

Outro exemplo é o mecanismo de chamada de procedure do HYDRA (CoJ 75). O ambiente de execução de uma procedure não é determinado apenas pelo sujeito que chama, mas existem também informações "próprias" da procedure. O problema da Suspeição Mútua, completo, está presente e é solucionado. Como foi visto na seção VI.4 cada procedure do HYDRA contém 3 conjuntos de capabilities: as "próprias" que nunca são transmitidas a quem chama, os "parâmetros de execução", que nunca são levadas de volta com a procedure e as "templates" que podem ser conhecidas por ambas. Essas capabilities são todas alinhadas no LNS (espaço de nomes locais) de execução da procedure e cada processo é formado por uma pilha de LNS. O único LNS disponível a cada procedure é o do topo da pilha, ou seja: o seu próprio. Portanto, nem a mais maliciosa procedure pode acessar ou danificar outros objetos através de seus parâmetros. Todos os arquivos, diretórios e outros objetos que não são passados explicitamente estão absolutamente seguros. Ao retornar o controle, o LNS é destruído, portanto, a procedure que chamou não tem acesso às capabilities privativas da chamada, resolvendo a outra parte do problema da Suspeição Mútua.

Quando a comunicação entre processos é feita através da passagem de mensagens o confinamento se torna mais exequível pois toda a transmissão de dados é feita de forma explícita. Comunicação por mensagens não tem efeitos colaterais, ou seja: uma mensagem só é recebida quando é solicitada e os únicos efeitos que pode causar no receptor são os provocados pelo processo que envia. No caso de memória compartilhada, isso não é verdade, visto que qualquer "sócio" pode arbitrariamente modificar a memória sem conhecimento dos outros.

Sistemas operacionais de hardware distribuído como o MEDUSA

(OSS80) tem maiores chances de realizar confinamento entre seus processos, visto que há isolamento físico entre os vários processadores. A memória local de cada um só pode ser acessada pelo próprio processador e uma vez definido o formato dos pipes de mensagens, a organização interna e a localização das procedures não importa. O uso dos pipes de mensagens tem dupla vantagem: limita os danos e permite que a reconfiguração do sistema seja feita de forma confinada.

A conclusão final desta seção é que só pode haver confinamento em sistemas cujo hardware está preparado para suportá-lo. Sistemas operacionais sem adequada assistência do hardware tendem a ser ineficientes e similarmente, mecanismos de proteção não integrados ao hardware são amorfos. Isto significa que o projeto do sistema deve ser iniciado pela máquina abstrata e prosseguido pelo hardware para suportá-la. Dessa forma, a implementação de domínios protegidos que levarão à execução de processos confinados se tornará possível e lógica.

VII - SISTEMAS NUCLEADOS

VII.1 - ARQUITETURAS HIERÁRQUICAS E NUCLEADAS

Mesmo o sistema operacional mais plano que existe, ao ser utilizado por um programador, está criando uma hierarquia. Na base está o hardware do computador. No segundo nível (ou terceiro, se existe microprogramação no segundo) estão as rotinas do sistema operacional, que proveem a simulação de um conjunto de primitivas e estruturas de dados não suportadas diretamente pelo hardware. Em seguida vem o compilador, alterando por sua vez as estruturas dos sistemas operacionais. Uma nova camada de software é produzida pelo programa do usuário, que roda sobre o compilador e simula a operação de um computador especializado para aquela aplicação. E finalmente vem os dados do programa, a última camada . Por mais simples que seja o sistema, exceto em casos de computadores dedicados a aplicações, essas 5 camadas estão sempre presentes.

Baseados nessas assertivas, a idéia intuitiva de que há certos objetos do sistema que são "programas" e outros que são "dados" perde seu significado. É uma distinção apenas aparente, porque na realidade, o que é "programa" em um contexto, pode ser "dado" em outro. Por exemplo: uma codificação de ALGOL é "programa" para os dados, mas é "dado" para o compilador. O sistema operacional é "programa" para o compilador, mas é "dado" para o hardware (Pra 75).

Vamos agora nos abstrair da hierarquia completa de um sistema de computador e observar apenas o nível correspondente ao sistema operacional. Independentemente da forma estrutural, um sistema operacional é uma sociedade de processos cooperantes. Muitos sistemas possuem um grande número de processos e um pequeno número de processadores reais. O gerente de processos utiliza-se de algoritmos de schedulling para alocar processadores a processos. O termo "processador virtual" foi inventado para sugerir a simulação inerente a esse procedimento. Da mesma forma, uma grande "memória virtual" é simulada pelo gerente de memória em uma pequena memória real. Assim, um "processo virtual" pode ser visto como um processo executado em um "processador virtual" e tendo acesso a uma "memória virtual" (Den 76b).

Como processos são ativados por outros processos, um sistema operacional também pode, por sua vez, ser subdividido em vários níveis cada qual definido e representado pelo conjunto de processos virtuais de que é composto, e gerando os processadores virtuais que serão utilizados pelo nível seguinte.

Os conceitos fundamentais da construção de sistemas operacionais em hierarquias de processos foram lançados por Dijkstra (Dij 68). Na época em que Dijkstra projetou o THE para a máquina holandesa ELXS, palavras como segurança e mecanismos de proteção eram ainda irrelevantes para o contexto de um sistema operacional. O sistema, de multiprogramação, tinha como objetivos reduzir o tempo 'turn-around' para jobs de pequena duração e fazer uma distribuição econômica e racional dos recursos da máquina entre usuários, tornando-se viável para sistemas de propósitos gerais.

Entretanto, para atender a esses objetivos, o sistema precisa ter um mínimo de mecanismos de isolamento, que propiciem a cada usuário da multiprogramação um ambiente privativo de trabalho, como se

tivesse um computador dedicado à sua disposição. No THE cada programa de usuário é abstraído como um processo do sistema e periféricos também são gerenciados por processos independentes. O fato de ter sido o primeiro sistema a definir uma hierarquia clara entre seus processos, merece que analisemos detalhadamente as implicações dessa hierarquia do ponto de vista de segurança. Foi a seguinte a distribuição adotada por Dijkstra:

. NÍVEL 0 - responsável pela alocação dos processadores segundo uma regra de prioridade. Neste nível um relógio de tempo real impede que um processo monopolize a CPU. Acima deste nível o processador perde a sua identidade permanecendo visível apenas a atividade dos processos.

. NÍVEL 1 - onde se localiza o "controlador de segmento" um processo responsável pelo gerenciamento da memória, realizando paginação e despaginação para os processos dos níveis superiores. Acima deste nível as páginas perdem a identidade, e a identificação da informação é realizada através de segmentos.

. NÍVEL 2 - neste nível o "interpretador de mensagens" cuida da alocação do teclado para conversações entre o operador e os processos dos níveis superiores. Acima deste nível é como se cada processo "possuisse" um teclado privativo, embora na realidade compartilhe o mesmo dispositivo físico, o qual perde sua identidade.

. NÍVEL 3 - onde os processos responsáveis pela alocação dos dispositivos periféricos se localizam, realizando bufferização da entrada e desbuferização da saída. Acima deste nível os periféricos reais perdem a identidade e se tornam transparentes aos usuários.

. NÍVEL 4 - composto pelos programas independentes de usuários, a cada programa correspondendo um processo sequencial.

. NÍVEL 5 - o operador (não implementado).

Embora não houvesse preocupação específica com segurança, a hierarquia criada por Dijkstra possui implicitamente as seguintes características encontradas em vários mecanismos de proteção:

1. O usuário não tem acesso a posições específicas da memória central. Nem mesmo sabe em que parte seu programa está sendo armazenado, visto que o processo responsável pela alocação da memória está visível apenas no nível 1.

2. O interpretador de mensagens não utiliza nenhum endereço fixo na memória, assim as conversações entre operador e processos dos níveis superiores sofrem paginação da mesma maneira que os programas de usuários.

3. A cada nível um recurso ou um conjunto de recursos do hardware perde a sua identidade, tornando-se inacessível diretamente ao programador. Além disso, não há montador para programas

de usuários escritos em liguagem de máquina, apenas um compilador para linguagem de alto nível.

4. O isolamento entre usuários é o maior possível, o único objeto diretamente compartilhado é uma biblioteca de procedures onde se localiza o compilador Algol 60.

5. Os recursos são gerenciados descentralizadamente, de maneira que o esforço para se obter controle total do sistema é consideravelmente superior ao necessário no caso em que um mesmo supervisor detém centralmente a responsabilidade de alocar cada recurso.

Visto isso, do ponto de vista de segurança está reconhecida a superioridade dos sistemas organizados hierarquicamente sobre os sistemas em que todos os recursos estão agrupados num mesmo 'pool'. Além disso, verifica-se que construir o supervisor em vários níveis limita a propagação de erros e torna fácil a modificação em qualquer nível sem alterar os vizinhos, aumentando a confiança na integridade de cada função do sistema.

O conceito de arquiteturas hierárquicas foi generalizado por Peter Denning (Den 76b). Para este, a idéia básica é construir o sistema operacional em termos de uma hierarquia de máquinas abstratas (ou virtuais) aninhadas, cada qual gerenciando privativamente um conjunto específico de recursos não manuseado por nenhuma outra máquina. Se denotamos as máquinas abstratas por M_0, M_1,... M_k, então as operações definidas em cada M_i são implementadas como procedures que operam nas estruturas locais de dados de M_i e podem empregar operações definidas em M_{i-1}.

A máquina abstrata M_0 foi interpretada como o hardware básico e a M_k como o ambiente em que os usuários rodam seus programas, embora existam arquiteturas que permitam aos usuários rodar em qualquer dos níveis, inclusive construir outros, criando seu próprio ambiente de execução, exemplos: MULTICS (Sal 74a), PSOS (FeN 79), etc.

Denning examinou o que acontece em cada nível i da hierarquia. Os processos P_{ij} em M_i implementam novas operações do sistema, que não estão disponíveis em M_{i-1}, manipulam recursos e dados representados por uma estrutura D_i e invocam operações definidas no nível inferior M_{i-1}. Cada operação em M_i é indivisível e os dados em D_i não podem ser acessados fora do nível i (fig. 7.1).

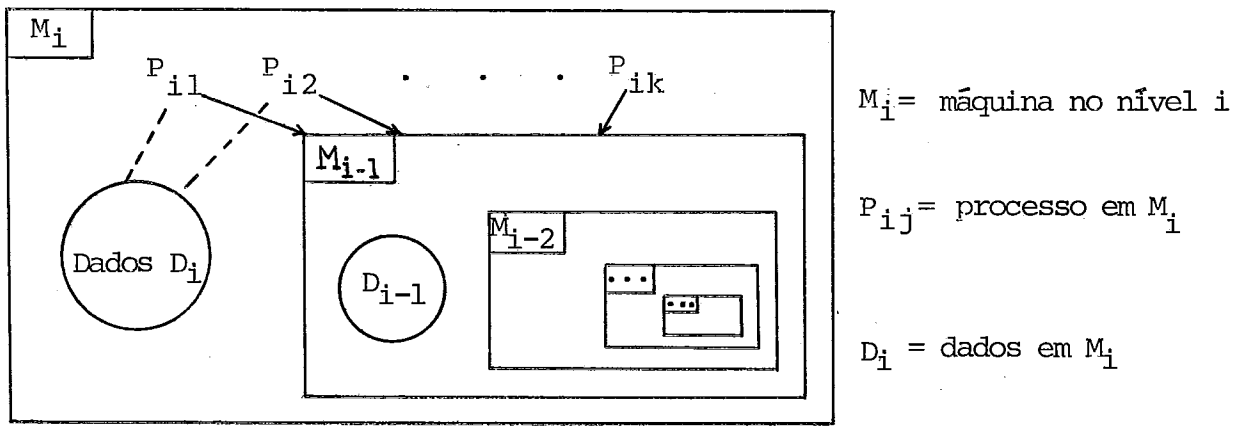


Fig. 7.1 Domínio de execução em arquiteturas hierárquicas (Den 76b)

O número de níveis, k, depende do sistema e dos seus objetivos. Dijkstra utilizou 5 níveis no THE, Liskov utilizou 6 no VENUS (Lis 72), Neuman utilizou 13 no PSOS (FeN 79). O sistema VENUS nos chama a atenção por ter sido implementado através de micro-programas. Cada dispositivo de I/O foi microprogramado, aparecendo para o sistema como um dispositivo virtual. O VENUS suporta a execução concorrente de 16 máquinas virtuais microprogramadas, ou seja: 16 processos rodando em processadores virtuais e contendo espaço de endereçamento e área de trabalho em memória virtual.

A partir daí torna-se fácil entender as técnicas que levaram à construção dos sistemas baseados em máquinas virtuais e dos sistemas nucleados tão em moda ultimamente.

A título de ilustração, vamos examinar o VM/370 da IBM (Att79). Este é um conjunto de procedures que criam múltiplas cópias do hardware provendo processadores virtuais para as várias máquinas virtuais que rodam sobre ele (ou sob ele, dependendo do ângulo). Através deste software, várias máquinas virtuais (OS/VS1, OS/VS2, DOS, etc) podem rodar simultaneamente utilizando o mesmo hardware real. Na realidade, o software do VM significa apenas a introdução de mais um nível na hierarquia, colocado exatamente entre o hardware e o sistema operacional antigo (fig. 7.2).

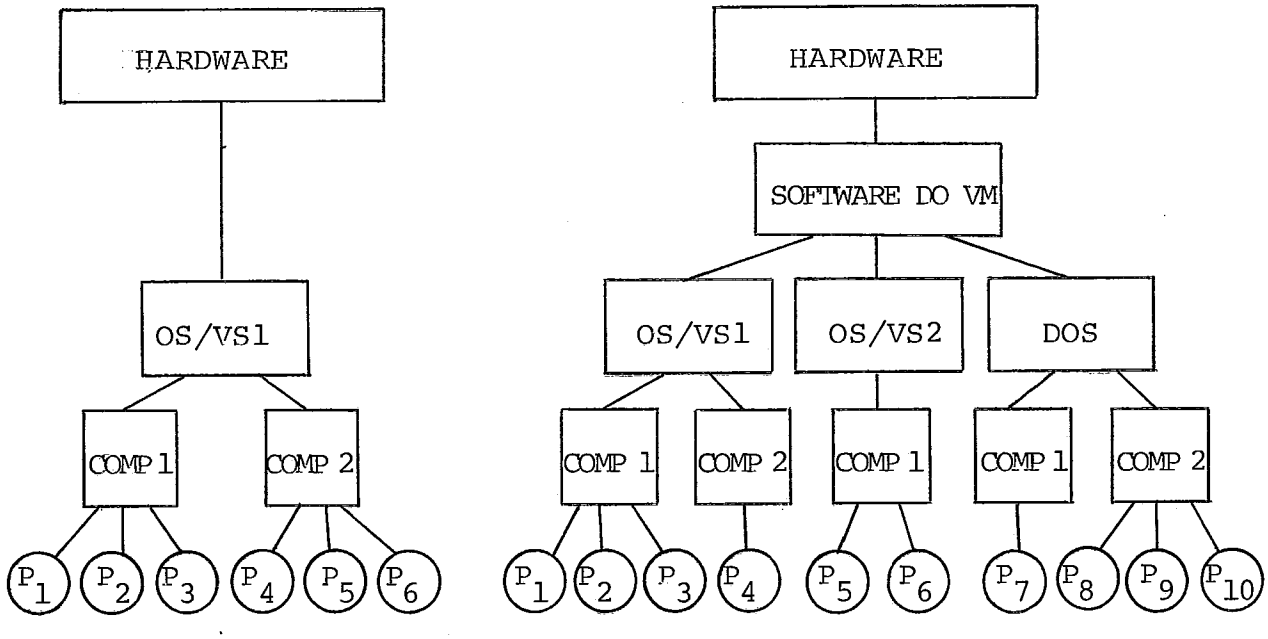


Fig. 7.2 - Inclusão do software do VM/370 na hierarquia de um sistema (Att 79)

O objetivo da IBM ao introduzir o VM foi compatibilizar as várias versões de sistemas operacionais. O pretenso isolamento entre essas máquinas não acontece na realidade porque o software do VM não é confinado (AMP 76, ChS 76).

Sucede porém, que, quando bem utilizadas, arquiteturas hierárquicas podem gerar excelentes mecanismos de proteção, pelas características mencionadas para o caso do THE e que podem ser generalizadas para qualquer outro sistema hierárquico. Ambientes de execu

ção protegidos podem ser gerados para cada usuário do sistema e a possibilidade do desenvolvimento modular proporciona acurados métodos para certificação e teste, que são a base de qualquer sistema seguro.

Baseados nessa evidência foram construídos muitos sistemas operacionais hierárquicos e se chegou ao refinamento dessa categoria de sistemas: os NÚCLEOS.

O termo núcleo foi utilizado pela primeira vez por Brinch Hansen na construção de um sistema de multiprogramação para o computador dinamarquês RC 4000 (Han 70). Preocupado com as incompatibilidades existentes entre um sistema operacional e as necessidades reais de cada programador, Hansen decidiu criar um sistema de multiprogramação que em vez de possuir funções para satisfazer necessidades específicas, possuísse um "núcleo" que pudesse ser estendido à critério do programador.

O núcleo do RC 4000 é um conjunto de processos paralelos cooperantes. Cada programa de usuário assume a forma inteira de um processo. A sincronização entre os processos é feita por semáforos binários e a comunicação entre eles é feita por um mecanismo de mensagens em que o núcleo administra um 'pool' comum de buffers de mensagens e uma fila de mensagens para cada processo.

Entretanto, a característica mais interessante do núcleo RC4000 é a hierarquia dinâmica de processos. Enquanto a hierarquia de processos de Dijkstra e de todos os sistemas hierárquicos até o momento era estática, Hansen realizou um sistema em que os processos são criados e controlados dinamicamente, de acordo com as necessidades de cada job. A situação se desenrola da seguinte maneira: após o carregamento inicial do sistema, a memória contém o "núcleo" e um sistema operacional básico, S, que pode criar processos paralelos A, B, C. Estes processos, por sua vez podem criar outros D, E e F, que podem criar outros, limitados apenas pelo número total de processos. Assim é constituída uma família de processos (avô, pai, filho, etc) que concorrem igualmente aos recursos do sistema e podem se comunicar com quaisquer outros processos, independentemente da sua posição na hierarquia (fig. 7.3).

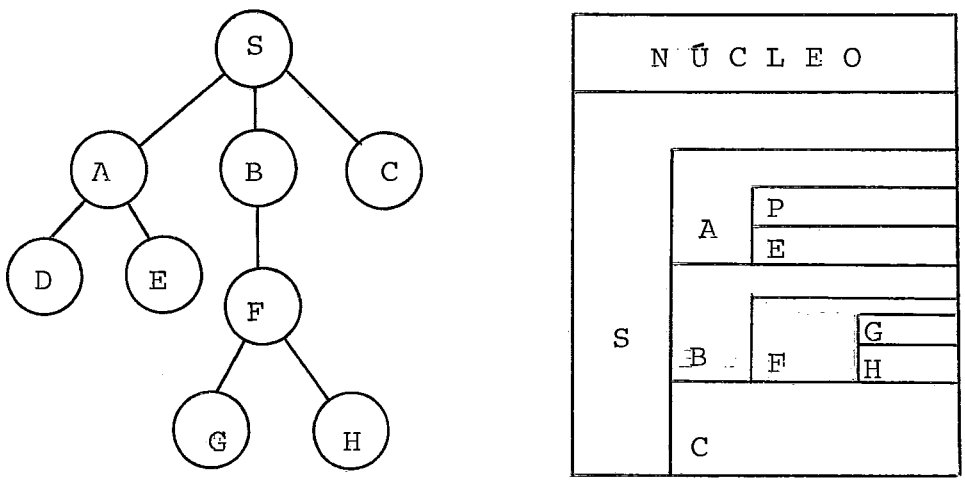


Fig. 7.3 - Hierarquia de processos e divisão da memória no RC4000

Com uma estrutura tão simples o RC4000 conseguiu atingir seus objetivos: novos sistemas operacionais podem ser implementados como programas sem nenhuma modificação no núcleo do sistema e vários sistemas operacionais podem rodar simultaneamente sem obrigar a instalação a fazer "mudança de modo".

A idéia de núcleo e árvores de criação lançadas por Hansen tem sido base para a implementação de inumeráveis sistemas nucleados, dos quais um subconjunto nos interessa especialmente: NÚCLEOS DE SEGURANÇA.

VII.2 NÚCLEOS DE SEGURANÇA

A maior dificuldade encontrada no projeto de um sistema hierárquico é a determinação do que cada camada deve conter. Determinar, entre todas as funções de um sistema quais são dependentes das outras, e quais devem ser utilizadas na construção das máquinas virtuais que serão primitivas para as outras, é uma tarefa que necessita um profundo conhecimento dos propósitos do sistema que se vai construir.

É de consenso geral que se um sistema operacional tem "segurança" entre os seus propósitos, as primitivas que contém codificação relativa à proteção devem estar no seu nível mais baixo: imediatamente junto ao hardware ou no próprio hardware. Isolando a codificação de proteção no coração do sistema, o seu funcionamento correto não depende do comportamento dos outros módulos. A esta camada, de baixíssimo nível que contém as primitivas dos mecanismos de proteção do sistema é dado o nome de NÚCLEO DE SEGURANÇA.

Um núcleo de segurança é a MENOR quantidade de mecanismos comuns necessários para implementar os modelos de compartilhamento da informação, comunicação interprocessual e multiplexação dos recursos físicos que são desejados num sistema seguro (Sch 75). De acordo com essa caracterização de núcleo de segurança, este não é a mesma coisa que um supervisor. A razão usual que inclui um mecanismo no supervisor é protegê-lo dos danos acidentais ou maliciosos causados por programas de usuários. Esta razão não é suficiente para incluir um mecanismo em um núcleo de proteção, pois podem ser criados alguns domínios protegidos fora do núcleo aos quais programas de usuário não tenham acesso. Isto foi com efeito realizado em alguns sistemas, exemplos: KSOS (CaD 79) e KVM (Sch 75).

A dúvida permanece: que funções devem ser colocadas no núcleo de segurança? Em outras palavras; que funções são primitivas para os mecanismos de proteção?

Dezenas de possibilidades existem e as várias aproximações nos forneceram um espectro bastante amplo de implementações práticas de núcleos de segurança. Tantas que nos permitem afirmar que apesar de as arquiteturas nucleadas exibirem muitas características semelhantes e diferirem substancialmente das arquiteturas convencionais, também diferem significantemente entre si.

A característica mais influente na decisão do que deve constituir um núcleo de segurança é a POLÍTICA DE SEGURANÇA que será suportada por este. Por exemplo, um núcleo que suporta uma política de isolamento é consideravelmente diferente de um núcleo que suporta uma política de compartilhamento com chaveamento de domínios entre processos individuais. Quanto mais sofisticada a política a ser implementada, tanto mais mecanismos precisam ser colocados no núcleo. Por outro lado, um núcleo é por definição "a menor quantidade de mecanismos comuns necessários", por isso, a escolha da política deve levar em conta essa restrição de volume. Não se trata apenas de escolher uma política discrecionária ou não-discrecionária, mas em decidir até que ponto a característica de discrecionari

dade será implementada por mecanismos internos ao núcleo. Incluídas aí regras que governam a alteração dos dados de política de segurança que podem ser gerenciadas por mecanismos internos ou externos ao núcleo (PoK 74).

Uma vez definidos os mecanismos que implementarão a política de segurança, devem ser escolhidos os OBJETOS protegidos do sistema. Estes podem representar desde componentes do hardware (segmentos, dispositivos de I/O, processadores, etc.) até construções de usuários de alto nível (diretórios, árvores, registros, etc.). A granularidade que envolve essa escolha é imensa. Deve ser decidido se todos os objetos serão primitivos ou se haverá também flexibilidade para a implementação de extensões, caso em que devem ser criados mecanismos para protegê-los como aos primeiros.

Outro aspecto do sistema que invariavelmente afeta a construção de um núcleo de segurança se refere ao suporte dos DISPOSITIVOS DE I/O. Algumas arquiteturas necessitam software de núcleo adicional para suportar cada dispositivo. É o caso dos núcleos construídos para hardware de PDP (PoK 78) que não possui canais de I/O, portanto toda a codificação de canal deve ser escrita pelo sistema operacional. Outros sistemas, embora possuam canais devem criar proteção adicional para estes. Geralmente, quanto mais simples for o hardware, mais complexo deve ser o núcleo.

Também o DESEMPENHO geral do sistema deve ser levado em conta, principalmente em núcleos que são construídos para substituir sistemas antigos e que fatalmente sofrerão comparações do tipo "qual é mais eficiente?". Em certos casos algumas funções não relacionadas à segurança, se colocadas fora do núcleo causam diminuição na eficiência, devido às chamadas frequentes. Mesmo quando não há restrições de desempenho nas especificações do núcleo, sua complexidade pode ser aumentada pela necessidade de codificar certas funções de uma maneira eficiente. Por exemplo, uma simples pesquisa sequencial pode ser otimizada por um algoritmo complexo que usa encadeamento em área de overflow.

Outra característica diz respeito à FUNCIONALIDADE do núcleo. Poderá ser diretamente utilizado pelos usuários ou dependerá de outro sistema como interface? Já houve casos em que a dificuldade apresentada na utilização direta do núcleo foi elemento decisivo para a sua descontinuação (LaS 76). Aprendendo com esse erro, a maioria dos núcleos projetados ultimamente apresentam uma interface simples ou um software intermediário, já pronto e utilizado pelos usuários, que torna quase transparente a existência do núcleo de segurança.

Segundo Popek e Kline (PoK 78) um dos caminhos mais eficientes de se reduzir o tamanho e a complexidade de um núcleo é remover tanto quanto possível as funções de GERÊNCIA DE OBJETOS para tipos particulares de objetos. O grau em que isso pode ser feito depende, é claro, do sistema e do tipo do objeto. Incluída na gerência de recursos existem 3 aspectos significantes que podem ser removidos do núcleo e suportados por software externo: integridade de

tipo, controle de acesso e nomeação. Os únicos objetos para os quais a integridade deve ser checada pelo próprio núcleo são aqueles objetos reconhecidamente utilizados como base para outros: segmentos, páginas, blocos de disco e similares. Todos os outros objetos podem ser protegidos por codificação privilegiada de fora do núcleo. Quando decididamente não é possível excluir o suporte de tipo do núcleo, é no mínimo possível excluir o controle de acesso, deixando apenas a parte que realmente opera no objeto. Em outras palavras, a POLÍTICA que decide as operações está na codificação privilegiada de fora do núcleo, mas as operações são realizadas por MECANISMOS do núcleo. Essa separação entre política de segurança e mecanismos de proteção é uma característica indispensável no projeto de qualquer sistema operacional, particularmente nos nucleados (ver definições seção II.1). E finalmente, o terceiro aspecto que pode ser removido do núcleo diz respeito aos nomes de objetos. Já se observou que nos sistemas operacionais comuns um grande número de funções são utilizadas para gerenciar "nomes" de recursos, em vez do próprio recurso. Todo esse software de mapeamento pode fazer parte dos domínios de usuários e pode ser particionado em vários espaços de nomes, cada qual gerenciado por um processo diferente.

As etapas de projeto e construção de um núcleo de segurança devem ser rigidamente acompanhadas e certificadas. A certificação de um núcleo é composta basicamente por 2 partes: a primeira provando que as especificações correspondem aos objetivos e a segunda provando que a codificação corresponde às especificações.

Quatro estágios podem ser identificados na construção de um núcleo de segurança (Mil 76):

1. MODELO ABSTRATO - em que são definidos os objetivos e os propósitos do sistema, as políticas que devem ser suportadas e as características gerais dos elementos do sistema. Palavras chaves nesse estágio são: estados de proteção, sujeitos, objetos, direitos de acesso. Para modelos discrecionários podem ser adicionadas também capabilities, listas de acesso, chave-fechadura, propagação de direitos, revogação de acesso, etc. Para modelos não-discrecionários podem ser incluídas: níveis de segurança, categorias, propriedades simples e estrela, controle de fluxo, etc.

2. ESPECIFICAÇÃO FORMAL - em que é adotada uma técnica de especificação formal para descrever os elementos do modelo abstrato. Pode ser escolhida uma técnica de especificação modular como a de Parnas (Par 72), uma linguagem de especificação formal como a SPECIAL da SRI International (Neu 78) ou a PASCAL (PoF 78)

ou outra qualquer. Palavras chaves nesse estágio são: variáveis de estado, funções executáveis, segmentos, tabelas de mapeamento, processos, comunicação interprocessual, compartilhamento, isolamento, arquivos, diretórios, etc. Nesse estágio é feita a primeira parte da certificação: a prova de que as especificações correspondem aos objetivos.

3. ESPECIFICAÇÃO ALGORITMICA- em que são codificados os programas em linguagem de máquina, em microprogramação ou em linguagem de alto nível, se a semântica for bem compreendida. Cada módulo do estágio anterior pode ser programado e testado independentemente, baseado apenas na especificação formal. Os testes são conduzidos no sentido de provar: a especificação algorítmica (codificação) corresponde à especificação formal, que é a segunda e última parte da certificação do núcleo.

4. IMPLEMENTAÇÃO - em que são carregados os módulos conjuntamente na máquina e iniciada a operação regular do núcleo. Os testes não se referem mais à certificação, pois nesse estágio ela já é assumida como feita, e sim ao funcionamento do sistema como um todo.

A maior vantagem das arquiteturas nucleadas é a necessidade da especificação clara dos propósitos do sistema. Outros sistemas, que não foram obrigados a isto, tornaram-se inconsistentes ou prolixos. Além disso, a identificação das funções relevantes e a sua separação de todas as outras funções do sistema operacional permite que a certificação seja realizada a um custo efetivo. É muito mais fácil e econômico dedicar um esforço maciço à certificação de um pequeno conjunto de funções, do que realizar essa tarefa em todo o sistema. Assim, em sistemas nucleados, apenas o núcleo é certificado, o resto do sistema passa pelos testes convencionais dos sistemas operacionais.

Existem, entretanto, algumas contra - indicações ao uso de núcleos. Núcleos não podem ser modificados casualmente, já que por definição toda a sua codificação é essencial para satisfazer os propósitos do sistema e qualquer modificação causaria um desvio nesses propósitos. Deve-se tomar cuidados extremos para se ter certeza de que uma mudança no núcleo não comprometerá sua operação correta. Além disso, para se construir um núcleo pequeno, os objetivos devem ser razoavelmente estreitos e altamente específicos. Esses objetivos limitam de certa forma as aplicações para as quais o núcleo é util.

Outro problema é a extrema dificuldade na determinação das funções que são essenciais e das que não são. Muitas vêzes ocorre uma superestimativa e o núcleo se torna maior do que deveria realmente ser. Núcleos grandes não merecem tanta confiança já que a certificação é mais difícil de ser realizada. Outras vezes o tamanho do núcleo aumenta porque não há suporte de hardware suficiente para prover comunicação interprocessual econômica e processos que são notadamente irrelevantes acabam aninhados junto aos relevantes, para não causar aumento de overhead.

Com tudo isso, a tecnologia de núcleos, especialmente núcleos de segurança está se propagando vertiginosamente. Talvez devido à tendência especializadora da ciência humana, que aperfeiçoa continuamente instrumentos de aplicação cada vez mais restrita. É um problema de julgamento decidir se as vantagens da alternativa núcleo ultrapassam as desvantagens. Para situações em que as necessidades estão claramente definidas e um núcleo pequeno possa ser construído, a alternativa núcleo é a ideal.

Nas 2 seções seguintes será apresentada uma evolução do projeto e construção de núcleos de segurança. Acreditamos que o estudo dos sistemas nucleados desta forma histórica é fundamental para a compreensão das soluções mais avançadas.

VII.3 - NÚCLEOS DISCRECIONÁRIOS - EVOLUÇÃO HISTÓRICA

Núcleos discrecionários foram desenvolvidos através da combinação da idéia de núcleo de um sistema operacional (Han 70) com a de mecanismos de proteção discrecionários (capítulo III). O objetivo central de um núcleo discrecionário é que no coração do sistema seja construído um conjunto de facilities de aplicabilidade universal e confiabilidade absoluta através das quais possam ser implementados quaisquer propósitos ou políticas que se queira.

Dentre os mecanismos de proteção discrecionários, o mais empregado tem sido o de capabilities por suas inúmeras vantagens (seção III.2, seção VI.4). Esta tendência foi refletida na construção de núcleos discrecionários, que uniram as vantagens de uma arquitetura nucleada às do uso de capabilities. O nosso maior interesse nesta seção é observar como os núcleos discrecionários evoluiram, do simples ao sofisticado, aprendendo com os erros e acertos de cada antecessor. Vamos examinar 4 núcleos discrecionários obedecendo à ordem cronológica: SUE, CAL-TSS, HYDRA e UCLA SECURE UNIX.

SUE

O projeto SUE da Universidade de Toronto visava o desenvolvimento de um sistema operacional seguro para a família de computadores IBM/360 (SAG 72). Na época em que foi criado o SUE, o conceito de núcleo de um sistema operacional estava começando a aparecer e não se cogitava em idéias de certificação. O objetivo era reunir os conceitos dos sistemas THE (Dij 68), BCC 500 (Lam 69), RC 4000 (Han 70) e MULTICS (Sal 74a) construindo um sistema operacional nucleado cujos objetos fossem protegidos por capabilities.

A estrutura do sistema resultou numa série de camadas hierárquicas tipo cascas de cebola, onde a camada mais interna foi chamada Kernel. O Kernel foi definido como "a camada de software que usa o hardware para implementar processos (criação, destruição e comunicação), proteção, gerência simples da memória e dos canais e facilities de controle de tempo". As camadas seguintes são formadas por processos relacionados entre si por uma árvore de criação: um processo é filho do processo que o criou e pai de cada processo que criar. Dessa forma, várias máquinas virtuais podem ser definidas em quaisquer níveis da hierarquia e ordenadas parcialmente com respeito à sua sofisticação. A máquina virtual mais sofisticada, já na periferia do sistema, foi chamada Nucleus e produz arquivos em discos, facilities de I/O para os periféricos, mecanismos de mensuração e contabilidade.

Vemos que no SUE os termos Kernel e Nucleus possuem significados diferentes [1]. As funções que pertencem ao Nucleus do SUE são irrelevantes para a segurança e podem pertencer a qualquer parte do sistema operacional. As funções relacionadas à segurança, ou seja, que são desempenhadas por mecanismos de proteção, pertencem ao Kernel.

1 'Kernel' pode ser traduzido por semente, parte central, âmago. 'Nucleus' pode ser traduzido por núcleo, miolo, ponto central. A diferença é mais dialética do que semântica e nesta Tese, ambas são traduzidas igualmente por Núcleo (exceto no caso do sistema SUE).

No SUE foram adotados os conceitos de objetos protegidos, domínios e capabilities. Cada "capability" é um bloco de controle associado a um processo que indica se o processo está autorizado a usar um recurso particular. O conjunto de capabilities de um processo é o seu domínio de proteção. Existem 3 tipos de capabilities: QUALITATIVA, que autoriza o uso de um recurso, QUANTITATIVA, que controla o número de vezes que um recurso é utilizado e ESPECÍFICA de processo. As duas primeiras só podem ser criadas e modificadas pela rotina de proteção do Kernel, a terceira é criada e modificada por processos do sistema. Mecanismos de mensuração e contabilidade são implementados usando capabilities: cada processo é contabilizado pelo uso de um recurso ao exercer a capability para o recurso.

Dentre as funções do Kernel, criação e destruição de processos representam transmissão de capabilities. Capabilities quantitativas são registradas pelos pais antes de serem transmitidas aos filhos, portanto, no retorno, cada pai possui meios de se inteirar do número de vezes que o recurso foi utilizado pelo processo-filho. Comunicação interprocessual é realizada através de 2 formas: caixas de correio e facilities, onde cada facility pode ser chamada por seus descendentes ou pelos seus irmãos mais novos e respectivos descendentes (árvore de criação - Seção VI.5).

O SUE foi projetado para ser um sistema extensível, no sentido de que vários subsistemas protegidos podem ser adicionados, cada qual servindo a uma comunidade de usuários. Para isso, seu Kernel provê as funções básicas e seu Núcleo proporciona uma hierarquia parcialmente ordenada de máquinas virtuais. Um subsistema protegido pode ser criado pelo usuário em qualquer nível da hierarquia, pode usar ou não capabilities e pode escolher entre caixas de correio e facilities para comunicação interprocessual. Uma linguagem de implementação foi especialmente idealizada para o projeto SUE, para permitir que o sistema seja facilmente compreensível, modificável, extensível e verificável. Foi um dos primeiros sistemas a se preocupar com a demonstração das propriedades de cada função e sua certificação, o que lamentavelmente não foi concluído.

Embora o projeto e a implementação dessa linguagem de implementação tenha consumido mais recursos do que o previsto, os objetivos gerais foram atendidos. Como reportado pela equipe (SAG 72), estes não eram apenas construir um sistema operacional seguro, mas aprender como construí-lo utilizando um conjunto de técnicas: árvores de processos, hierarquias de máquinas virtuais, capabilities, mecanismos de comunicação interprocessual. Uma metodologia de desenvolvimento resultou da geração de extensiva documentação proveniente de relatórios quadrimestrais. Projeto e programação estruturados foram utilizados. E finalmente, alguns módulos do Nucleus foram efetivamente certificados. O sistema SUE foi implementado na Universidade de Toronto, onde os estudantes foram encorajados a criar diversos subsistemas protegidos.

CAL-TSS

Construir um sistema operacional para o CDC 6400, baseado em capabilities, cuja arquitetura contivesse várias camadas independentes e permitisse aos usuários a adição de novas camadas e a criação de novos objetos protegidos eram os objetivos fundamentais do projeto CAL-TSS em Berkeley (LaS 76).

O sistema incluía no mínimo 4 camadas, cada qual definindo um universo virtual protegido dos outros. A primeira representava o Sistema do Núcleo as três últimas o Sistema do Usuário.

O Sistema do Núcleo era formado pelo conjunto mínimo de facilities necessário para a construção das outras camadas. O Núcleo suportava 8 tipos de objetos protegidos: arquivos do núcleo, canais de eventos, blocos de alocação, C-listas, labels, operações, processos e tipos. Uma característica de hardware do CDC-6400, a ECS ("Extended Core Store") com taxa de transmissão rapidíssima para a CM ("Central Memory") permitia isolar todos os objetos do núcleo, tornando-os inacessíveis aos usuários.

O Sistema do Usuário era formado por todas as facilities necessárias para executar programas de usuários e suportava 5 tipos de objetos protegidos: arquivos em disco, chaves de acesso, diretórios, etiquetas de nomes e descritores de domínios. Objetos do usuário eram armazenados em disco, daí serem também chamados "objetos de disco".

No CAL-TSS toda a codificação que não fazia parte do Núcleo devia executar em algum domínio protegido. Como o domínio não era um objeto do núcleo e sim do usuário, em geral domínios eram entidades estáticas, em que as operações de criação e destruição eram muito menos frequentes que as de chamada. Domínios, no CAL-TSS eram constituídos por 3 partes: alguns registradores, uma porção de CM e uma C-lista. A C-lista continha as capabilities para os objetos que podiam ser acessados naquele domínio. Como esses objetos podiam ser outras C-listas, ficava implícita uma estrutura de árvore.

Processos eram objetos do núcleo, só podiam ser iniciados por facilities do núcleo. Cada processo incluia uma árvore de domínios uma stack de chamadas, alguns registros de máquina, alguns canais de eventos, um bloco de alocação e alguns timers. Canais de eventos eram objetos protegidos utilizados na comunicação interprocessual. Blocos de alocação eram objetos que alocavam espaço na ECS ou tempo de CPU. O CAL-TSS tinha o audacioso objetivo de contabilizar cada recurso da máquina utilizado por um usuário, por isso, processos eram tão complicados.

Arquivos de núcleo e arquivos de disco possuiam a mesma estrutura. Por opção do usuário através da operação ATTACH, uma parte dos seus arquivos podia residir na ECS e uma parte no disco, acelerando as execuções. Diretórios eram listas dos objetos armazenados no disco. Cada entrada no diretório era adicionada no momento da criação do objeto e também a árvore de diretórios era utilizada para contabilização. Um esquema de chave-fechadura era empregado para proteger os objetos de um diretório: cada emtrada possuia uma lista de fechaduras contendo diferentes privilégios de acesso e um domínio só podia "abrir" a fechadura para a qual tivesso a chave de acesso (objeto protegido). O fato de as capabilities do núcleo não

poderem aparecer em diretórios, construções de usuários, obrigou a criação de outro tipo de objeto protegido: as etiquetas de nomes. Cada etiqueta de nome era um objeto contendo um inteiro inalterável que a associava univocamente a uma capability; em vez de colocar capabilities nos diretórios, eram colocadas etiquetas de nomes. Finalmente, label era o nome global para domínios equivalentes em diferentes processos e operações eram meios de trocar o domínio corrente por outro.

Com uma variedade tão grande de tipos de objetos protegidos, o CAL-TSS ainda possuia mecanismos para a criação de extensões de objetos. Para isso, objetos tipo Tipo autorizavam a associação de uma capability a um objeto de qualquer tipo criado pelo usuário. Uma vez associada, esta capability nunca poderia ser modificada, agindo como um verdadeiro lacre do núcleo para aquele objeto do usuário.

No projeto do CAL-TSS foi utilizada a linguagem de especificação PASCAL, através da qual foram definidos os componentes de cada tipo de objeto e as operações permitidas por tipo. A estrutura de camadas levou à contrução de um núcleo altamente confiável que durante vários meses de operação originou apenas 4 falhas no sistema. O mecanismo de capabilities também foi utilizado com segurança permitindo uma maneira consistente e uniforme de referenciar e proteger os objetos do sistema. O I/O externo também foi bem manuseado pelo núcleo, através dos canais de eventos que realizavam a comunicação entre o sistema do núcleo e os dispositivos periféricos. Foi construído um simulador do SCOPE, sistema operacional do fabricante, que foi colocado inicialmente no topo do núcleo e posteriormente no topo do sistema do usuário. A extensibilidade do núcleo, testada durante a construção do sistema do usuário foi bem sucedida.

Entretanto, o sistema CAL-TSS, quase pronto, foi descontinuado por falta de recursos. Isso aconteceu principalmente porque após os 3 anos de desenvolvimento, o sistema não era nem eficiente, nem usável para ser colocado a serviço de um centro de processamento. Três razões contribuiram para isso. Primeira, havia um grande número de idéias novas e não testadas no sistema. Segunda, o grupo gerente do projeto era inexperiente e muitas vezes foi dedicado um esforço considerável a alguma coisa que parecia interessante e não era essencial. Terceira, a equipe falhou em compreender que "núcleo" não é a mesma coisa que um sistema operacional e subestimaram o trabalho necessário para torná-lo usável por um programador comum.

Essas e outras falhas foram reportadas em vários artigos pela equipe do projeto, nos quais foram incluídos um conjunto de idéias e sugestões relevantes para a construção de um núcleo de segurança, experiência lamentavelmente adquirida no insucesso, mas que representou importante contribuição para a comunidade científica. A experiência de Lampson e sua equipe no CAL-TSS é considerada em praticamente todos os núcleos de segurança que o sucederam.

HYDRA

A noção intuitiva de que o núcleo de um sistema operacional é um conjunto de linhas de codificação severamente protegido numa área da memória não corresponde à realidade. Uma vez que existam mecanismos de proteção intimamente embutidos na estrutura do sistema, estes não precisam estar fisicamente agrupados. Um exemplo típico de

distribuição do núcleo é o HYDRA, da Carnegie - Mellon University, um sistema seguro, de propósitos gerais, e suficientemente amplo para implementar quaisquer políticas que se queira (WCC74, COJ75).

O núcleo de segurança do HYDRA é um conjunto de facilities que permitem a aplicação de operações (procedures) aos recursos (objetos). Uma variedade de sistemas operacionais podem ser criados a partir dessas primitivas, entretanto, os conceitos de árvore de criação e "paternidade" não são aplicados. O HYDRA rejeita frontalmente a idéia de hierarquias, argumentando que diminuem a flexibilidade, e estrangulam a experimentação. Não há, com efeito, necessidade delas, pois seus 16 processadores executam realmente em paralelo, já que consistem de um PDP-11 com memória privativa, memória secundária e dispositivos de I/O. Não existe controle central desses "pedaços" autônomos de hardware, mas é fundamental que haja um bom mecanismo de comunicação entre eles.

Cada processador executa em um ambiente protegido definido por uma lista de capabilities chamada LNS (espaço de nomes locais). Apenas os objetos que contêm capabilities no LNS podem ser referenciados pela procedure em execução. Um mesmo processo (programa de usuário) pode estar sendo executado simultaneamente em mais de um processador, cada qual em um domínio diferente. O núcleo provê as primitivas CALL e RETURN que permitem a invocação de procedures e a criação de outro LNS, mudando o domínio de execução. Um LNS é formado por 3 tipos de capabilities: as dependentes da chamada (parâmetros de chamada) as independentes da chamada (inerentes à procedure) e as templates (conhecidas por quem ativa e quem é ativada). As templates são uma verificação dinâmica dos direitos possuídos por uma procedure ao ativar outra.

Em cada processador existe uma réplica das funções mínimas de núcleo, ou seja, as que suportam as noções de recursos (objetos) e procedures (operações). Outras funções de núcleo estão distribuídas e duplicadas, mas não aparecem necessariamente em cada um. Um mecanismo de comunicação interprocessual eficiente permite transmissão de instruções e dados entre os processadores, 4 níveis de interrupção, um relógio central e hardware de relocação em cada bus de processador, para permitir mapeamento de endereços virtuais no endereço físico da memória compartilhada. Mensagens são enviadas através de "portas" que se comunicam por meio de canais.

A separação entre mecanismos e política é bem determinada no HYDRA. Mecanismos são todas as funções que proveem primitivas para o controle de acesso aos recursos. Políticas são objetos protegidos que agem como caixas de correio entre o núcleo e os mecanismos.

Além de permitir ao usuário a criação de seu próprio sistema operacional ou subsistema protegido, existe a flexibilidade para a implementação de extensões de objetos. Nesse caso apenas, fica implícita uma estrutura hierárquica através do objeto TIPO, que permite a criação de qualquer outro tipo de objeto não provido pelas primitivas do sistema. São providos também mecanismos que definem operações nos novos tipos de objetos e mecanismos para prover proteção e controle de acesso a eles.

Um usuário do HYDRA não tem ao seu dispor um sistema operacional no sentido de uma entidade monolítica que oferece várias facili

ties ao usuário. Ao contrário, o ambiente do usuário é composto por um conjunto de recursos e um conjunto de operações, a partir dos quais quaisquer regras podem ser construídas, o que é intrinsecamente a filosofia de um núcleo discrecionário. Essa característica gerou críticas como o fato do HYDRA não ser um sistema utilizável por programadores ordinários, que antes precisavam "construir" o sistema operacional para depois usá-lo. Versões mais recentes já incluem um conjunto de subsistemas específicos que podem ser facilmente utilizados.

UCLA SECURE UNIX

O núcleo do UCLA SECURE UNIX, ou simplesmente UCLA UNIX é um dos projetos mais avançados na tecnologia de núcleos de segurança e utiliza as mais modernas técnicas como: arquitetura hierarquica, capabilities, implementação em firmware, certificação (PKK 79).

O núcleo básico consiste de aproximadamente 760 linhas da PASCAL, excluído o suporte I/O, que para o PDP-11, deve ser escrito como função da CPU. Numa hierarquia de 3 níveis o núcleo corresponde ao 1º nível e é o único módulo do sistema que está autorizado a executar as instruções privilegiadas do hardware. Implementa uma dúzia de operações primitivas que podem ser chamadas a partir de processos do usuário.

Aprendendo com os erros do CAL-TSS e do HYDRA, o UCLA UNIX não é nem sofisticado nem difícil de ser utilizado. A falta de sofisticação corre por conta do número de objetos implementado, apenas 4: processos, páginas, dispositivos e capabilities, que nunca são criados nem destruídos mas são reaproveitados. A simplicidade de utilização decorre da criação de uma interface tão perto quanto possível do Unix.

Sendo o núcleo tão reduzido e objetivo, é necessário que seja complementado por processos auxiliares e estes correspondem ao 2º nível da hierarquia (fig. 7.4). O 3º nível corresponde aos processos de usuários que rodam nas máquinas virtuais criadas pelo 2º nível.

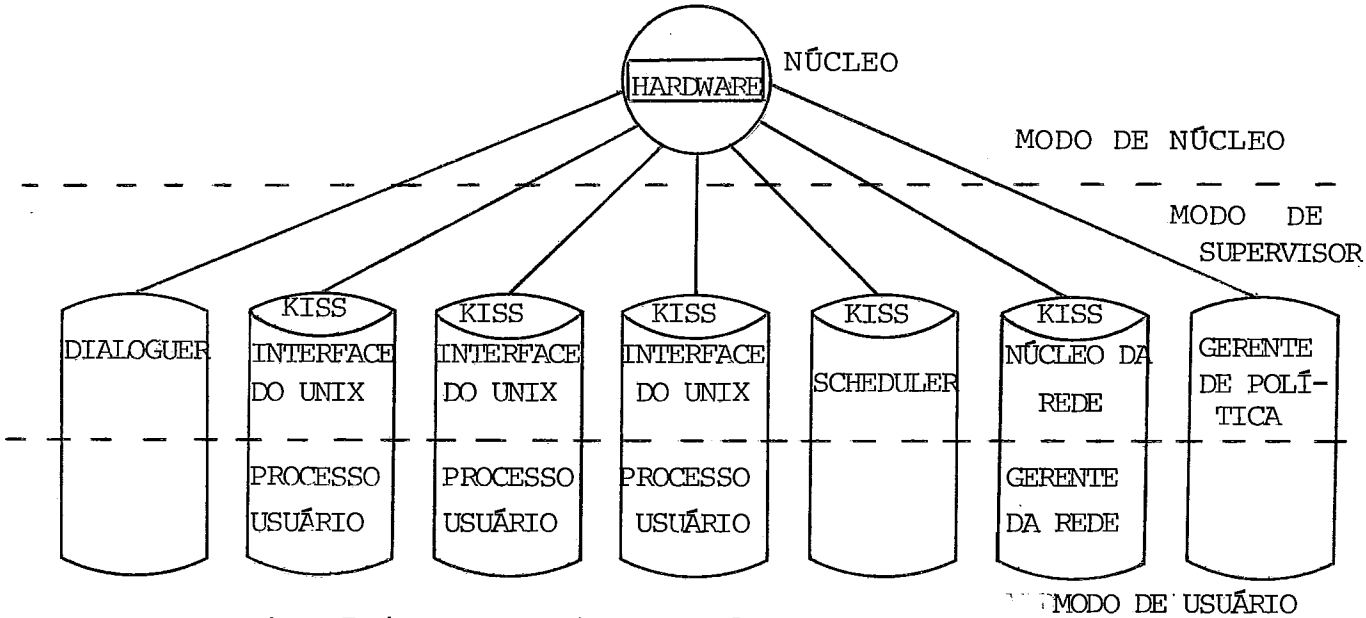


Fig. 7.4 - Arquitetura do UCLA SECURE UNIX

A segurança do sistema depende da operação correta de apenas 2 processos do 2º nível: o Dialoguer e o Gerente de Política. O Dialoguer é o processo responsável pela comunicação com o usuário, e inicialmente "possui" todos os terminais ou seja, as capabilities para eles, realizando as funções de autenticação. O Gerente de Política é responsável pela implementação de várias opções de política de segurança, como requer um núcleo discrecionário. Mais uma vez vemos a separação entre mecanismos de proteção (internos ao núcleo) e políticas de segurança (externas ao núcleo).

No Unix Standard, diretórios são organizados em árvores. Entretanto, a estrutura de diretórios e suporte de arquivos não precisa ser implementada por um processo protegido. A maioria da codificação que roda no domínio do usuário não precisa ser certificada como o Núcleo e o Gerente de Política. Por isso foram criadas as Interfaces com o UNIX, processos não-privilegiados que transformam chamadas do UNIX em chamadas do núcleo.

Além das Interfaces com o Unix, existem 2 processos que fazem parte do KISS (Subsistema de Interface com o Núcleo): o Scheduler e o Núcleo da Rede. O Scheduler contém políticas de gerência de recursos a curto prazo para CPU e memória: ativação de processos, paginação, etc. Finalmente o núcleo da Rede foi criado para assumir a segurança da transmissão de dados, não confiando no software inseguro da rede.

Simples, pequeno, microprogramado, totalmente certificado, eficiente e objetivo, o núcleo do UCLA SECURE UNIX representa o aperfeiçoamento de dezenas de técnicas e um trabalho de vários anos que gerou muitas Teses de Mestrado e Doutorado, basta ver as referências (PKK 79).

VII.4 NÚCLEOS NÃO-DISCRECIONÁRIOS - EVOLUÇÃO HISTÓRICA

A adoção da política não-discrecionária pela maioria dos organismos militares nos USA levou o Departamento de Defesa a patrocinar projetos de núcleos com o propósito específico de implementar mecanismos não-discrecionários. Considerando que muito dinheiro já havia sido investido nos sistemas em uso corrente, os núcleos projetados deveriam, além de implementar a política não-discrecionária, permitir ao usuário comum continuar usando o sistema anterior sem alterações ou recompilações.

O usuário comum estava habituado há anos a utilizar mecanismos discrecionários de controle de acesso. Era impraticável tentar eliminar totalmente o controle subjetivo que cada um deles possuia sobre seus próprios objetos. Por outro lado, expurgar mecanismos de acesso discrecionário para fora do núcleo causaria falhas irreparáveis na proteção do sistema. Devido a isso, a maioria dos núcleos não-discrecionários contém também mecanismos discrecionários de controle de acesso.

Pelo menos 5 núcleos conhecidos foram projetados especificamente para conter mecanismos de proteção de níveis de segurança: o núcleo do MULTICS, o núcleo da MITRE, o KVM/370, o KSOS e o SIGMA. Vamos dar uma olhada rápida nos 4 primeiros, o SIGMA pode ser visto no Apêndice B.

NÚCLEO DO MULTICS

O MIT em Cambridge, Massachusetts tem agido como um laboratório de importantes descobertas na área de mecanismos de proteção (em outras áreas também). Dele provieram as idéias de capabilities (DeV 66), anéis de proteção (Gra 68), listas de acesso (Sal 74a). Também a idéia dos núcleos de segurança foi incubada no MIT, por uma razão muito simples: o sistema operacional MULTICS é constituído por um "supervisor em camadas" e atingiu um tamanho tão grande que se tornou impossível de ser totalmente certificado. A partir daí foi fácil chegar à conclusão de que apenas as funções relevantes para a proteção deveriam ser certificadas e convencionou-se chamar a êsse conjunto de funções núcleo de segurança.

Construir um núcleo de segurança para um sistema existente pode ser feito de duas formas diferentes: aproveitando parte da codificação existente no sistema antigo e incluindo-se no núcleo ou projetando um núcleo totalmente novo para rodar sob o sistema operacional antigo. Ambas as alternativas são viáveis, dependendo apenas dos critérios e das características do sistema. No caso do MULTICS, foi escolhida a la. alternativa, como nos reporta Schroeder, arquiteto do sistema desde as suas primeiras versões (Sch 75).

Esta escolha foi baseada em vários argumentos. O MULTICS foi um sistema projetado e desenvolvido com propósitos de segurança, portanto já incluia vários mecanismos de proteção. Possuia um conjunto completo de capacidades funcionais como compartilhamento direto da informação entre usuários, suporte de hardware para informação de controle, modularidade, codificação PL/I, facilidades para modificação, inclusão e realização de testes.

Desta forma, o plano foi dividido em 3 estágios:

1. Remover do supervisor os mecanismos irrelevantes para segurança.

2. Reestruturar o núcleo remanescente.

3. Particioná-lo em vários ambientes de proteção incluindo os mecanismos de controle não-discrecionários de níveis.

O primeiro estágio significou a remoção de várias atividades do núcleo, principalmente das relacionadas ao sistema de arquivos. Também a gerência da associação entre nomes de referências e segmentos do espaço de endereçamento de um processo foi totalmente removido do núcleo. Outras atividades relativas à inicialização do sistema, criação de novos processos, etc., foram investigadas e removidas. O tamanho do núcleo foi bastante reduzido.

No segundo estágio, a reestruturação também contribuiu para a redução. Em alguns casos, um bloco completo de codificação pode ser eliminado e sua função assumida por outro, como a substituição de mecanismos de I/O externo pelo conjunto da ARPANET. Funções internas de memória virtual, back-up e carga do sistema foram também reestruturadas. Processos foram reimplementados usando um mecanismo de 2 camadas que simplificou a interação com o mecanismo de memória virtual e a comunicação interprocessual. Também a parte do sistema de arquivos que permaneceu no núcleo foi totalmente reestruturada. E outras mais.

Finalmente, o núcleo deveria ser particionado em vários ambientes de proteção para permitir várias especificações diferentes de segurança. A idéia mais cotada era particionar o núcleo que roda sob cada processo de usuário (como no MULTICS standard) em 2 camadas: na camada inferior ficavam os mecanismos de controle não-discrecionário e na outra mecanismos de controle discrecionário.

O projeto do núcleo do MULTICS, denominado Projeto Guardian foi um esforço conjunto do MIT, Honeywell e da Força Aérea Americana. Infelizmente os recursos deste projeto expiraram antes do seu término e o núcleo não foi implementado (Neu 78).

NÚCLEO DA MITRE

Paralelamente ao trabalho realizado pela equipe do MIT em Cambridge, uma equipe da Mitre Corporation em Bedford, também em Massachusetts se empenhava na tarefa de construir um núcleo de segurança para implementar a política não-discrecionária de níveis. Entretanto, partindo do ponto zero da construção de um sistema operacional, pode realizar os 4 estágios descritos anteriormente para a construção de um núcleo (Mil 76).

No estágio de modelo abstrato foram definidos uma matriz de acesso discrecionário, uma hierarquia de sujeitos e objetos e um conjunto de estados de proteção do sistema. Transições entre estados só eram autorizados se as propriedades simples e estrela fossem atendidas. Seis tipos de transições entre estados foram definidas:

1. Criar (ou ativar) um objeto.

2. Deletar (ou desativar) um objeto.

3. Obter acesso a um objeto para um sujeito.

4. Liberar um objeto de um acesso por um sujeito.

5. Colocar um sujeito na lista de controle de um objeto.

6. Remover um sujeito da lista de controle de um objeto.

Nota-se que o mecanismo discrecionário utilizado é o de lista de acesso. Cada objeto tem uma lista de acesso, na qual sujeitos só são incluídos em acordância com as propriedades não-discricionárias. O modelo multinível de classes hierarquicamente ordenadas e categorias não-hierárquicas, gerando níveis de segurança parcialmente ordenáveis é aplicado integralmente.

No estágio de especificação formal foi usada a técnica de desenvolvimento modular de Parnas (Par 72). Variáveis de estado foram chamadas funções-V (porque contém valores) e as funções executáveis foram chamadas funções-O (porque realizam operações). Ambas possuem parâmetros. Todas as referências de funções-V são objetos. Todos os acessos são devidos a execuções de funções-O. Cada tipo de transição entre estados (criar, deletar, etc.) é feito por funções-O especificadas no núcleo. Fora do domínio do núcleo as funções-O executam instruções comuns do sistema operacional.

Entre os objetos protegidos, os segmentos desempenham papel especial pelo fluxo implícito de informações que contém. Por exemplo: se 2 usuários compartilham a máquina e só existe espaço para criar mais um segmento, a criação ou não criação deste segmento por um deles envia um bit de informação implícita ao outro, sobre a sua decisão. Para evitar isso, o NÚCLEO DA MITRE atribui a cada usuário uma quota limite de memória, que, quando exaurida, não informa nada além de simples término de espaço. Além disso, cada nível de segurança só pode rodar em áreas específicas da memória para impedir deduções feitas a partir de números de índices. Se estas deduções forem feitas entre usuários do mesmo nível, não haverá escoamento ilícito de informação.

Todas as variáveis internas do núcleo são por sua vez associadas a um nível de segurança e o núcleo é obrigado a obedecer internamente às propriedades não-discricionárias. Isso para evitar que variáveis do núcleo ao trabalharem com vários níveis de segurança escoem inadvertidamente informações de um para outro.

A certificação do núcleo e a determinação de canais de fluxo inplícito foram os pontos altos do NÚCLEO DA MITRE. Os 2 últimos estágios: especificação algorítmica e implementação não trouxeram novidades. Mas os dois primeiros forneceram subsídios para um grande número de pesquisas posteriores na área de núcleos de proteção. Entretando, há críticos que consideram o NÚCLEO DA MITRE complicado para utilização por um programador comum. Acham que deveria ter sido feita uma interface mais simples para o programador ou uma adaptação a um software já divulgado. Essa segunda solução foi adotada nos dois núcleos que veremos agora.

KVM

O KVM/370 foi uma adaptação feita ao VM/370 da IBM por um grupo de pesquisa da System Development Corporation em Santa Monica , California. Além dos USA forneceu fundos para o projeto o Departamento de Defesa do Canadá. Os objetivos eram prover o VM dos mecanismos de controle de acesso não-discrecionários e eliminar as conhecidas falhas de segurança do sistema (GLP 79).

Como no NÚCLEO DO MULTICS a primeira decisão importante foi dividir o programa de controle do VM em duas partes: uma relevante e outra não relevante para a segurança. A segunda decisão importante foi atribuir a cada máquina virtual um nível de segurança. Desta forma, as máquinas virtuais podem rodar em ambientes protegidos e utilizar funções provenientes de duas fontes: do núcleo certificado (KERNEL) e do resto do programa de controle (NKCP). O núcleo roda em estado de supervisor e o NKCP ('Non Kernel Control Program') em estado de programa-problema. Numerosas cópias do NKCP são produzidas, cada uma rodando como uma máquina virtual sob o núcleo. Por sua vez, cada NKCP pode suportar diversas máquinas virtuais de usuário.

Basicamente, sujeitos no KVM/370 são interpretados como os NKCP's individuais. O núcleo provê isolamente completo entre os NKCP's, mas as máquinas virtuais sob o mesmo NKCP estão isoladas da mesma maneira que no VM standard. Devido a isso todas as máquinas virtuais sob o mesmo NKCP devem estar no mesmo nível de segurança tornando sem importância quaisquer fluxos de informação entre elas. Consequentemente devem existir tantos NKCP's quantos são os níveis de segurança.

Os objetos no KVM/370 são conjuntos de áreas de dados em DASD ou volumes inteiros de DASD, volumes de fitas, dispositivos de registros unitários, páginas de memória real, processos e todo o ambiente de trabalho de uma máquina virtual (blocos de controle, registros de memória de trabalho, etc).

As condições de segurança da política não-discrecionária são implementadas pelo núcleo: sujeitos só podem acessar objetos se as propriedades simples e estrela forem satisfeitas. Uma vez que o núcleo é o árbitro final e todos os acessos a dispositivos reais devem passar por ele, nenhuma ação dos NKCP pode comprometer a segurança.

Além dos processos incluídos no núcleo e no NKCP existem certos processos, não relacionados diretamente com a segurança mas que devem manusear dados de vários níveis de segurança. Esses são módulos que controlam o compartilhamento de recursos do sistema real entre as várias máquinas a níveis de segurança diferentes. Para manter a segurança, esses processos são submetidos à mesma verificação formal do núcleo, gerando os "processos privilegiados" que podem executar em estado real de supervisor. Quando a certificação não é possível e permanece a possibilidade de fluxo ilícito de informação os processos são chamados "semi-privilegiados", são submetidos a auditoria permanente e só podem executar em estado real de problema (fig. 7.5).

Das 30 falhas de segurança identificadas no VM/370 a maioria envolve funções de I/O. A complexidade da linguagem de canal frequentemente introduz erros na translação de comandos que possuem várias interpretações. Sob o KVM/370 as palavras de comando de canal não possuem vários significados e apenas o núcleo está autorizado a transladá-los e modificá-los no seu espaço de dados. Canais de memória e de tempo também são controlados pelo KVM/370 através da auditoria aos processos semi-privilegiados. Finalmente ao KVM/370 foram incorporados mecanismos de controle discrecionários para atender às modificações da legislação americana com respeito a segurança nacional.

Dessa forma, foram atingidos objetivos de prover o VM de mecanismos para implementação de uma política militar e eliminar as conhecidas falhas. A interface com o programador permaneceu a mesma, só que agora, tentativas ilegais que antes eram aceitas normalmente, são detetadas pelo mecanismo de proteção.

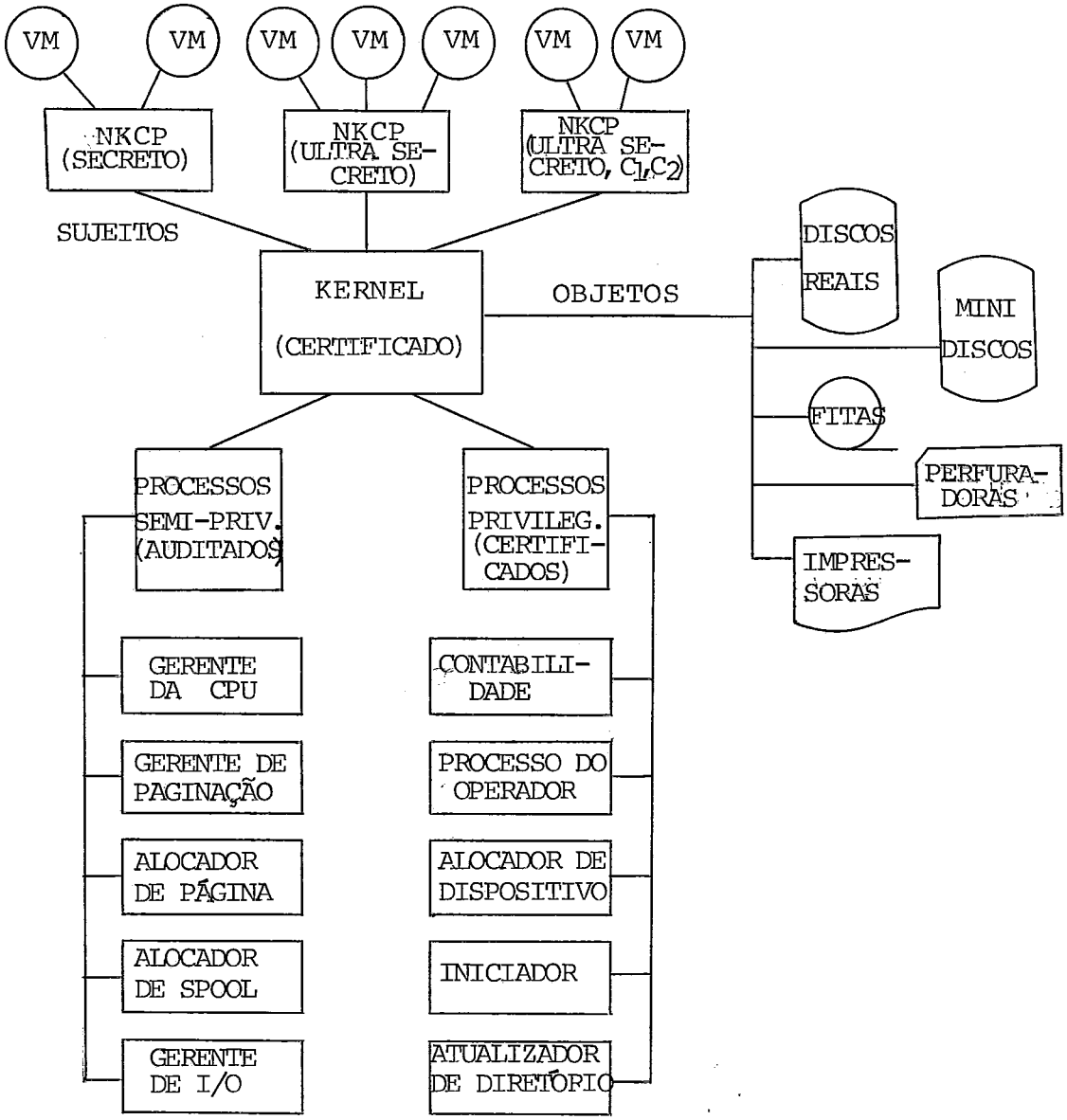


Fig. 7.5 - Arquitetura do KVM/370

KSOS

O núcleo KSOS - Kernelized Secure Operating System foi desenvolvido na Ford Aerospace & Communications Corporation, Palo Alto também Califórnia. Tem algumas semelhanças com o núcleo do VM desenvolvido em Santa Monica e aproveita alguns conceitos do núcleo da MITRE. É um dos núcleos mais bem construídos e sofisticados pois teve outros como base e como fundamentos (CaD 79).

O KSOS foi desenvolvido em uma instalação que já utilizava o sistema operacional Unix, num PDP 11/70. Aproveitando a experiência de outros sistemas nucleolizados, o KSOS se propunha a implementar a política não-discrecionária de níveis e ao mesmo tempo prover uma interface simples para o programador, via UNIX, ou diretamente no núcleo. O núcleo deveria ser ele próprio um sistema operacional provendo todas as funções inerentes e deveria possuir um bom rendimento, para evitar comparações desagradáveis.

Ambientes de máquinas virtuais que incluem controle direto dos dispositivos de I/O de usuários são incompatíveis com o PDP-11/70 , devido à granularidade do mapeamento de endereços virtuais a reais. Portanto, todos os dispositivos deveriam ser manuseados pelo próprio núcleo. Não obstante, o núcleo deveria ser tão pequeno quanto possível e totalmente certificado, para isso seria escolhida uma metotologia de projeto e uma linguagem de especificação.

A metodologia de desenvolvimento hierárquico (HDM) da SRI International (BeB 79) foi selecionada e como tal influencia fortemente a decomposição hierárquica do projeto. O KSOS possui 3 componentes estruturais: núcleo (KERNEL), NKSR e Emulador do Unix (fig. 7.6).

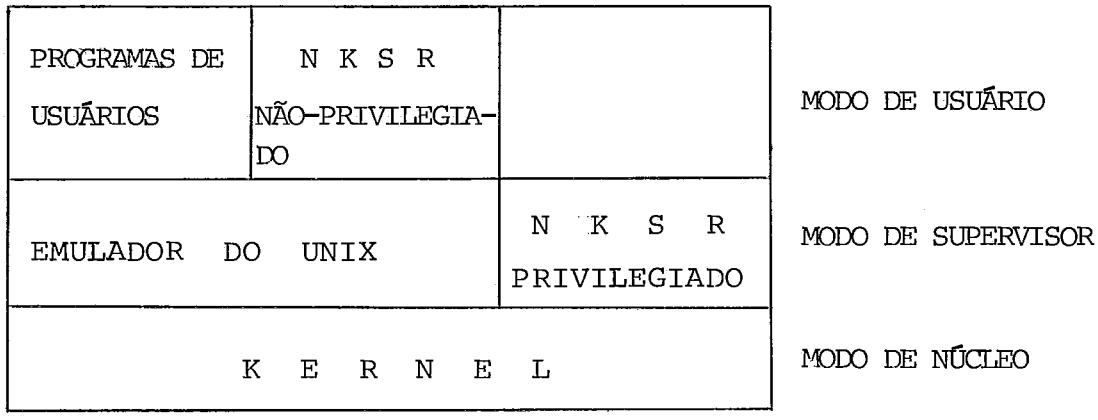


Fig. 7.6 - Estrutura do KSOS

O núcleo é o coração do sistema. Mediador de cada tentativa de acesso realizada por cada processo de usuário, centraliza o controle de todos os recursos do sistema. Reside no espaço de endereçamento mais privilegiado da máquina, de onde tem acesso a todas as facilities de gerência do hardware puro e da memória. Visto como uma máquina abstrata, a função do núcleo é criar, a partir do hardware, objetos para as suas interfaces (processos, segmentos, etc.). O núcleo realiza 3 tipos de cheques à solicitação de acesso: no primeiro certifica se as 2 propriedades não-discrecionárias básicas (simples e estrela) estão sendo atendidas, no segundo testa

se a integridade dos objetos não está sendo corrompida, no terceiro suporta um mecanismo de proteção discrecionário. Neste mecanismo o acesso está completamente sob controle do usuário. Para cada objeto existem 9 bits que especificam 3 tipos de acesso: ler, escrever e executar / pesquisar para o dono, outros usuários do mesmo grupo e todos os outros usuários.

O NKSR ('Non-Kernel Security Related Software'), é uma coleção de processos autônomos que desempenham serviços de suporte para o sistema. É composto pelas funções que realizam as operações do dia a-dia e de manutenção do sistema, descarregamento seguro da impressora, cheques de consistência nos arquivos, suporte de geração do sistema, interface com redes. Através do NKSR o usuário pode utilizar o núcleo independente do UNIX como se o núcleo fosse, ele próprio, um sistema operacional seguro. As funções mais sensitivas como o login, adição e deleção de usuários, etc., estão no NKSR privilegiado e interagem diretamente com o núcleo, as outras estão no NKSR não-privilegiado e interagem com o Emulador do UNIX.

O Emulador do Unix transforma chamadas de interface do Unix em sequências de chamadas ao núcleo, permitindo que sistemas em operação no Unix Standard rodem também no KSOS. Uma vez que o Emulador é não-privilegiado não precisa ser certificado e qualquer sistema operacional pode ter um emulador rodando seguramente no KSOS. Atualmente existe apenas o Emulador do Unix, mas podem haver 2, 3, etc "máquinas virtuais" em coexistência pacífica sob o núcleo.

Foi incluída uma interface para redes multiníveis composta por um Network Daemon privilegiado responsável pela separação da cadeia de dados em vários níveis e gerência de protocolo com a rede.

Finalmente, o núcleo do KSOS foi o primeiro a empregar industrialmente um conjunto de técnicas que haviam sido usadas apenas em ambientes acadêmicos de pesquisa e a demonstrar sua efetividade na construção de mecanismos de proteção para sistemas operacionais.

VIII - B A N C O S D E D A D O S

VIII.1 CONTROLE DE ACESSO EM BANCOS DE DADOS

A proteção de qualquer objeto de um sistema de computador deveria ser da responsabilidade do sistema operacional, entretanto, as soluções que os mecanismos de proteção normalmente sugerem, não são suficientes para resolver o problema de segurança do banco de dados. A maioria dos mecanismos de proteção dos sistemas operacionais são mecanismos de controle de acesso que suportam regras sobre quem pode realizar que operações em quais objetos. Nos sistemas de arquivos tradicionais, cada usuário é responsabilizado pela proteção dos seus objetos e utilizam esses mecanismos para implementar a sua política de segurança. Vários tipos de acesso são possíveis a um objeto: ler, escrever, ordenar, deletar, adicionar, etc., a maioria dos quais se referindo à ALTERAÇÃO. Além do fato de cada aplicação possuir os seus próprios arquivos, a quantidade de dados que é compartilhada é em geral pequena e os arquivos de comunicação são feitos sob medida, transmitindo apenas os dados estritamente necessários.

Num ambiente de banco de dados o problema se torna mais difícil de ser equacionado. Muitos usuários, com restrições de segurança diferentes tem acesso ao mesmo 'pool' de dados, com graus de sensibilidade variáveis. A maioria dos acessos são referentes à LEITURA e mecanismos de proteção que distinguem entre ler e alterar apenas não são suficientes (DJL 79). Além disso, uma diferença básica entre os mecanismos de proteção dos sistemas operacionais e os de bancos de dados é que nos primeiros a decisão de acesso pode ser subdividida em decisões individuais para cada objeto solicitado. E nos segundos é importante que todo o conjunto de respostas relativas às mesmas perguntas seja analisado globalmente, avaliando não apenas as relações entre os elementos mas também o histórico da informação que já foi divulgada.

Outras diferenças incluem o número de objetos do banco que é muito maior, a granularidade da informação, que é muito mais detalhada, a definição dos elementos e as trajetórias de acesso entre dados, que também devem ser protegidas (GKS 79). Resumindo, a segurança do banco de dados está relacionada à semântica da informação, não apenas às suas características físicas.

Pelo exposto, podemos concluir que devem existir procedimentos e mecanismos que satisfaçam essa necessidade adicional de segurança. Com PROCEDIMENTOS queremos nos referir às normas administrativas, que são necessárias para a segurança de sistemas comuns e que devem ser ampliadas para bancos de dados. E com MECANISMOS estamos nos referindo às funções internas desempenhadas pelo Gerente do Banco de Dados (DBMS) que complementam as funções de proteção realizadas pelo sistema operacional.

Vamos nesta seção examinar como são feitos os acessos a um banco de dados, para podermos entender os mecanismos propostos nas seções seguintes.

Cada acesso ao banco de dados é realizado através de um conjunto de passos, onde salvaguardas de proteção vão testando a sua viabilidade. Uma transação efetuada com sucesso significa que todos os

testes foram convenientemente atendidos. Por outro lado, uma solicitação recusada significa que pelo menos um dos testes não teve suas restrições cumpridas e bloqueou a passagem da informação. Pode-se numerar 7 passos (fig. 8.1) a serem percorridos no processamento de uma transação (WFS 80):

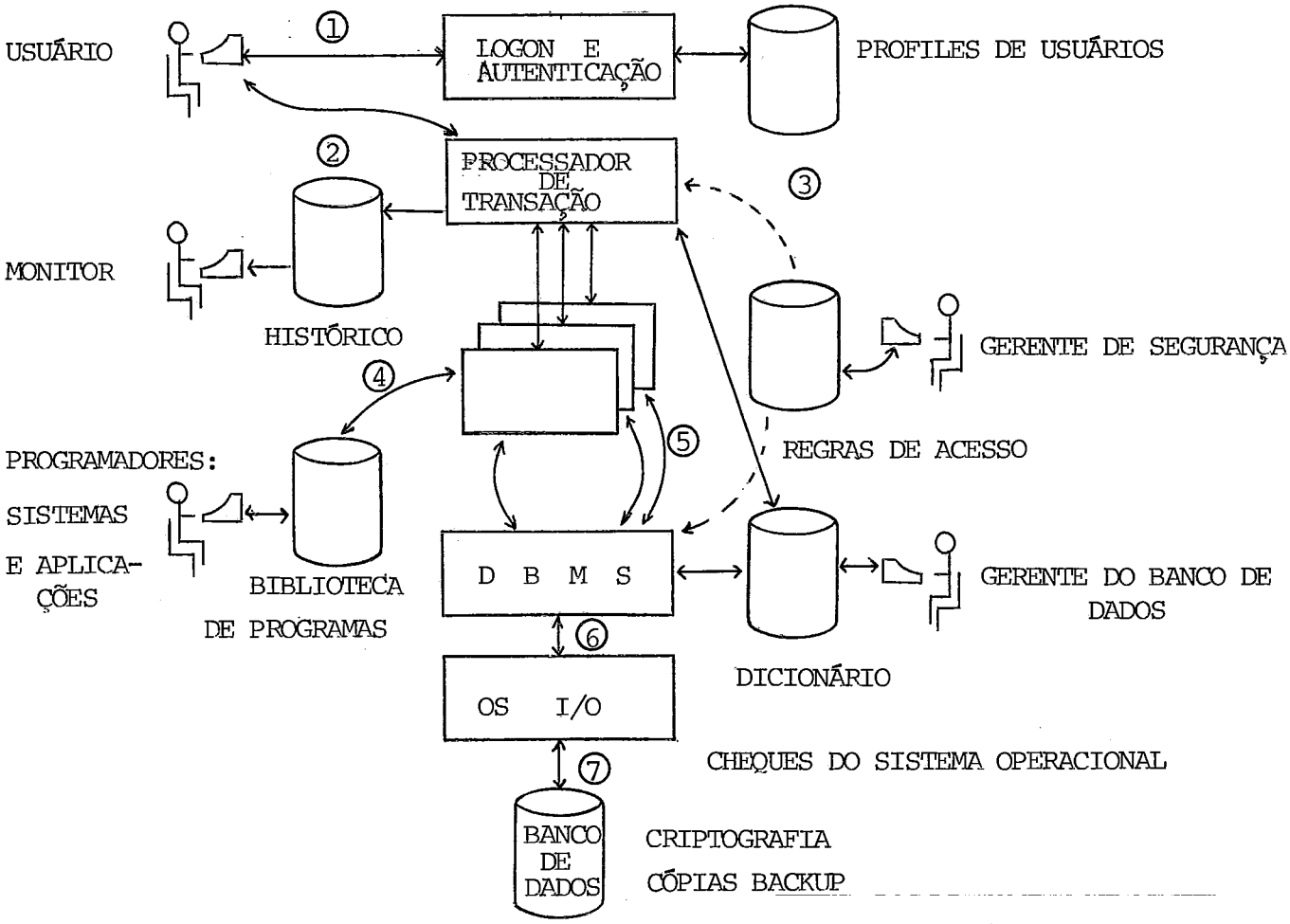


Fig. 8.1 - Testes de segurança para processamento de transação (WFS 80)

1. AUTENTICAÇÃO - o sistema verifica a identidade do usuário utilizando profiles dos usuários e dos terminais.

2. FORMULAÇÃO DA TRANSAÇÃO - o usuário especifica os objetos que deseja acessar e modos de acesso.

3. AUTORIZAÇÃO - o sistema verifica a autoridade do usuário para aquele tipo de transação. Regras de autorização são escritas e fiscalizadas pelo gerente de segurança, através do histórico de cada transação.

4. TRADUÇÃO DA TRANSAÇÃO - os programas de aplicação traduzem a transação que estava em linguagem do usuário para uma linguagem conhecida internamente no banco de dados. O atendimento a uma simples

transação pode envolver a execução de vários programas de aplicação armazenados na biblioteca de programas. Programadores de sistemas controlam e mantêm a biblioteca enquanto programadores de aplicação constroem os programas e os atualizam.

5. SUBMISSÃO DA TRANSAÇÃO AO DBMS - a transação traduzida é submetida ao DBMS que acessa o arquivo de controle à procura de regras de semântica e autorização para validar o acesso.

6. ACESSO AO BANCO DE DADOS - a transação traduzida e validada é transformada em chamada de I/O que é passada para o sistema operacional.

7. CHEQUES DO SISTEMA OPERACIONAL - mecanismos de proteção normais do sistema operacional como criptografia, uso de cópias back-up, verificação do uso de arquivos especiais, proteção implementada em hardware, etc. podem ser utilizados neste passo.

Podemos observar, portanto, que os mecanismos de proteção específicos para bancos de dados não excluem os mecanismos comuns do sistema operacional, mas os complementam. Várias políticas de segurança diferentes podem ser implementadas, dependendo do tipo do banco e das necessidades da instalação. Por exemplo, a política de segurança pode ser centralizada, como acima, em que um único gerente de segurança controla todos os aspectos de segurança do banco de dados, ou descentralizada, em que o controle é distribuído, muito útil para bancos de dados geograficamente dispersos.

Bancos de dados podem possuir mecanismos para implementar qualquer política que se queira: mínimo de privilégios, compartilhamento maximizado, multinível, etc., em granularidade maior do que os mecanismos comuns. Deve haver controle de acesso dependente de elemento e de conteúdo. Felizmente, a maioria dos modelos de proteção para os sistemas operacionais são também aplicáveis aos bancos de dados. Também para estes, a discrecionaridade (não-discrecionaridade) é a característica fundamental.

VIII.2 - MODELOS DISCRECIONÁRIOS

A segurança de um banco de dados pode ser considerada um subconjunto da segurança total do sistema, e nesse caso, o modelo discrecionário da matriz de acesso é aplicável (WFS 80). Entretanto, incluir na matriz geral do sistema os elementos do banco, tornaria o mecanismo de proteção excessivamente complexo, já que os objetos do banco possuem diferentes níveis arquitetônicos (interno, conceitual e externo) e requerem proteção a uma granularidade muito maior (arquivo, registro, campo dentro do registro).

Assim, é interessante que o DBMS seja responsável pela implementação do modelo de segurança, adaptando os mecanismos às suas necessidades.

No modelo da matriz de acesso de Lampson, Graham e Denning (seção III.1) objetos são as entidades do sistema cujo acesso deve ser protegido (páginas, dispositivos, etc). Para o DBMS (WFS 80) OBJETOS são conjuntos de ocorrências de ítens de dados. Os nomes desses conjuntos de dados são representados pela variável O, portanto, para cada banco de dados existe um conjunto finito de valores, conhecidos e acessíveis independentemente:

$$O_1, O_2, \ldots, O_n$$

SUJEITOS são sempre os usuários e seus programas, isto é, as entidades que solicitam o acesso ao banco de dados. Cada banco possui um conjunto de usuários potenciais, definidos pela variável s:

$$s_1, s_2, \ldots, s_m$$

REGRAS DE ACESSO, que constituem os elementos da matriz de acesso são as operações familiares de ler, escrever, atualizar, deletar e mais todas as outras definidas especialmente para bancos de dados. Para um dado DBMS, um conjunto de TIPOS de acesso é definido pela variável t:

$$t_1, t_2, \ldots, t_k$$

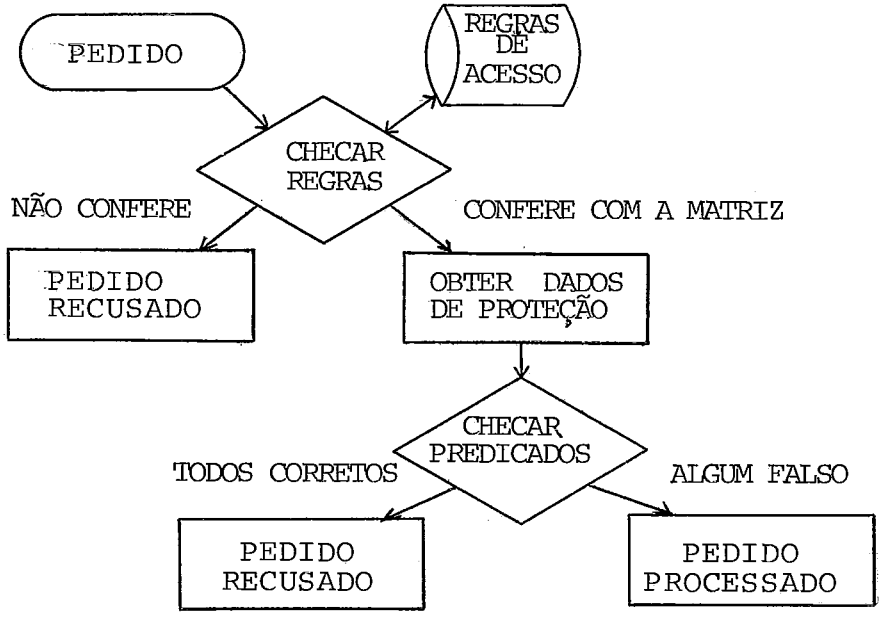
Observamos que cada elemento do conjunto de objetos pode representar um conjunto de dados, por isso foi usada letra maiúscula na notação. É importante notar que a matriz de acesso do banco de dados é mais estável que a do sistema operacional, onde sujeitos e objetos são criados e destruídos pelo sistema a cada instante. Aqui, ao contrário, modificações são feitas apenas para especificar uma nova regra de acesso ou revogar outra.

Para que o modelo seja geral o suficiente que permita representar qualquer controle de acesso dependente de tipo e conteúdo, é necessário que a regra de acesso contenha um PREDICADO p. Este predicado permite que um conjunto arbitrário O' de ocorrências de itens de dados seja definido como o objeto efetivo para a regra de acesso. O predicado pode também colocar restrições adicionais na regra de acesso, como permitir o acesso apenas em certas horas do dia.

Uma regra de acesso será então representada por (s, O, t, p), especificando que um sujeito s tem acesso t às ocorrências de O para as quais o predicado é p. Os dados que devem ser obtidos para avaliar o predicado são conhecidos como "dados de proteção". Por exemplo, p pode ser uma restrição de salário máximo permitido para leitura por um empregado do Pessoal. Nesse exemplo, o pedido de acesso seria composto assim:

s - empregado do Pessoal
O - relação de salários
t - ler
p - salário < 20.000

Um processo de validação deve existir para cada acesso, o qual é governado pelas regras de acesso descritas. No exemplo acima, um esquema possível de processo de validação é dado pela fig. 8.2.



Fig, 8.2 - Esquema de validação de acesso discrecionário (WFS 80)

Uma política de segurança que pode ser implementada através desses mecanismos é a da MODIFICAÇÃO DE PEDIDO. É muito utilizada para os casos em que um registro é solicitado e alguns dos seus campos não estão autorizados para aquele usuário (há restrições de predicado). O próprio sistema modifica o pedido fornecendo apenas os campos permitidos. Da mesma forma, se um conjunto de registros é solicitado e apenas alguns satisfazem o predicado, estes são fornecidos.

O momento certo para realizar o processo de validação depende da implementação. É mais seguro realizar todos os testes quando o acesso é solicitado, isto é, na hora da execução. Entretanto, para melhorar a eficiência, alguns sistemas os realizam mais cedo, armazenando seus resultados junto aos dados de proteção. Vamos descrever rapidamente algumas variações do modelo discrecionário aplicado a banco de dados:

IMS da IBM

Uma implementação parcial do modelo da matriz de acesso foi feita no IMS da IBM, que define o pedido de acesso da seguinte maneira:

s é o USERID
O é o código da transação
p é o predicado que especifica apenas uma password

O código de transação invoca um programa de aplicação para o qual os tipos de acesso a certos segmentos já foram especificados. Existe ainda uma regra adicional (s, O, t) em que:

s é o código da transação
O é o tipo do segmento
t é o tipo de acesso

Outras características de segurança do IMS permitem a restrição de certos códigos de transação e comandos do sistema a um terminal específico (terminais são interpretados como sujeitos e códigos de transação comandados são interpretados como objetos).

MODELO DA UCLA

Um mecanismo de chave-fechadura (seção III.4) ligeiramente modificado é utilizado num modelo de protocolo para coordenação de recursos em bancos de dados distribuídos na UCLA (MPM 80). Um protocolo de fechamento coordena o acesso aos bancos distribuídos e mantém a consistência do sistema em condições normais e anormais. Quedas parciais da rede e falhas de comunicação não afetam a integridade do mecanismo, que é capaz de recuperar a partir de um número qualquer de falhas provendo máximo aproveitamento do que já foi feito. Em cada nó existe réplica do controlador central de fechaduras (LC) o qual mantém uma tabela de todas as fechaduras ativas. Cada entrada na tabela é um triplet(H, T, P), onde:

H é a identificação do hospedeiro
T é um identificador de transação único em cada nó
P é uma descrição da porção lógica do banco de dados que vai ser fechada, associada ao modo de fechamento (ex: ler, escrever). Em um banco de dados relacional, a especificação de fechamento pode ser por exemplo um predicado.

A maior vantagem do mecanismo descrito é, a nosso ver, a sua robustez à falhas, fruto da redundância das tabelas do LC. Em cada nó existe adicionalmente um controlador local de fechaduras (LLC) responsável pela manutenção de uma cópia local da parte da tabela correspondente aos dados locais. Sempre que uma queda no sistema tornar o LC não disponível, qualquer LLC pode assumir o controle em seu lugar.

Outros sistemas que utilizam modelos discrecionários são o DBS 440, o ASAP, o CODASYL e o SISTEMA R. No DBS 440 (Sal 74b) desenvolvido no RZO (Rechenzentrum Oberpfaffenhofen), Alemanha, o computador de base é um Telefunken e o modo de acesso ao banco de dados é controlado dinamicamente. No ASAP (CMM 72) a matriz de acesso está contida em um dicionário especial, criado pelo usuário e possui descri-

ções de arquivos, relatórios e informação de segurança. No CODASYL (GKS 79) o mecanismo empregado é o de chave-fechadura tradicional em que as fechaduras podem ser definidas em cada um dos seguintes grupos de dados: esquema, área, registro, itens de dados e conjuntos. No SISTEMA R (GrW 76) existe uma regra de autorização para cada tipo de objeto. Por exemplo, para o tipo UPDATE o objeto é uma coluna da tabela. Mais detalhes sobre o SISTEMA R serão vistos na seção VIII.4.

VIII.3 - MODELOS NÃO-DISCRECIONÁRIOS

Como o modelo da matriz de acesso, também o modelo de múltiplos níveis pode ser aplicado aos bancos de dados. Dorothy Denning percebeu isso ao desenvolver seu modelo do fluxo seguro de informações e citou como exemplo o fluxo existente entre uma lista de nomes e uma lista de salários associados (Den 76a). É possível controlar separadamente o fluxo de informação na treliça definida por nomes, salários e os pares (nomes, salários).

Ao aplicar o modelo de níveis de segurança aos bancos de dados, os mesmos conceitos de níveis de classificação e grupos de categorias podem ser utilizados (WFS 80). A cada sujeito é atribuído um nível de autorização e a cada objeto um nível de classificação, como as militares: ULTRA-SECRETO, SECRETO, CONFIDENCIAL e NÃO-CLASSIFICADO. Cada sujeito e objeto também pertencem a um conjunto de categorias, exemplo: NUCLEAR e NATO. Um nível de segurança é definido pelo par:

NÍVEL DE SEGURANÇA = (NÍVEL DE CLASSIFICAÇÃO, CATEGORIA)

Um nível de segurança "domina" outro se e somente se seu nível de autorização ou classificação é maior ou igual ao outro e se o seu conjunto de categorias contém o do outro.

Além disso o modelo considera os estados de um sistema seguro, descritos por:

. acesso corrente, que é um triplet (s, O, t)
. uma matriz de acesso
. o nível de segurança de cada objeto
. os níveis máximo e corrente de cada sujeito

Transações podem causar mudança no estado de segurança do sistema. Regras de validação são empregadas para garantir que a mudança levará a outro estado seguro.

As propriedades simples e estrela dos modelos não-discrecionários são empregadas para garantir que transações não leiam acima do seu próprio nível e não escrevam abaixo, degradando a informação.

O modelo multinível envolve também o controle de fluxos implícitos de informação, que tem imensa aplicabilidade em bancos de dados estatísticos.

Entretanto, os modelos não-discrecionários possuem algumas desvantagens que restringem sua aplicação em bancos de dados. Devido à estrutura rígida que estabelecem para seus componentes, não é possível representar políticas de segurança arbitrárias. Consideremos por exemplo o caso de uma política simples que permite ao sujeito A acessar os objetos O_1 e O_2, ao sujeito B acessar O_2 e O_3 e ao sujeito C acessar O_1 e O_3. Esta política primitiva não pode ser manuseada nos mecanismos multiníveis. Além disso, a criação de novos objetos com novas restrições de segurança pode requerer uma reestruturação completa de toda a treliça de fluxos. Mais

ainda, regras de acesso dependentes de tipo e de conteúdo não podem ser representadas de forma simples pelo modelo multinível. Se variáveis de programa podem mudar sua classe durante a execução, as análises feitas na compilação não valem mais e o controle de fluxo requer cheques durante a execução, que causarão um overhead inaceitável.

Mesmo assim, formas simples de mecanismos multiníveis foram implementadas nos seguintes bancos de dados: I.P. Sharp Associates, SRI International, Systems Development Corporation e Mitre Corporation. Todos os projetos assumem bancos de dados relacionais, mas diferem na escolha do objeto protegido, alguns escolhem domínios, outros relações inteiras.

DMS da I.P. Sharp

O projeto da I.P. Sharp (Woo 79) decompõe cada objeto em 3 elementos: um conjunto de valores, um descritor (descreve o formato dos valores) e uma matriz de permissão que lista os usuários autorizados ao objeto e seus tipos de acesso permitidos. Diretórios e listas de usuários ativos são também protegidos. Cada relação (definição de RELAÇÃO na seção VIII.4) está associada a um nível de segurança e se vários dados diferentes forem reunidos numa mesma relação, é assumido o mais alto nível. O pioneirismo deste sistema decorre do fato de que em vez de utilizar primitivas já existentes em sistemas operacionais, foram inventadas primitivas de núcleo para suportar uma família de bancos de dados. O estudo mostra que o DMS (Data Management System) criado é geral o suficiente para ser aplicado a gerentes de bancos de dados dedicados a sistemas, aplicações em sistemas operacionais seguros e a redes de computadores.

DMS da SDC

O projeto da Systems Development Corporation tinha por objetivo desenvolver um banco de dados que agisse como uma interface protegida para o ambiente multinível do MULTICS (Sal 74a). A equipe da SDC optou por um banco de dados relacional e procurou saber que partes de um banco de dados convencional eram afetadas pelas restrições de segurança do sistema operacional (Woo 79). Foi observado que os mecanismos de proteção normais do MULTICS eram suficientes para prover a segurança necessária, de modo que o DMS (Data Management System) criado não contém nenhuma codificação de segurança. portanto não precisa ser certificado. O DMS foi projetado de acordo com o princípio do mínimo de privilégios, assim, quando roda sob um processo de usuário, não contém privilégios que o processo em si mesmo não contenha. O sistema armazena a informação referente ao banco de dados nos mesmos recipientes de memória, providos pelo MULTICS: segmentos, e utiliza um conjunto de segmentos para cada nível de segurança.

A importância do trabalho desenvolvido na SDC se situa na demonstração de que um banco de dados pode ser desenvolvido para um sistema seguro sem nenhuma codificação adicional de segurança. Entretanto, o DMS não aufere a flexibilidade que poderia ter sido ganha se unidades de informação menores do que o segmento fossem independentemente protegidas.

INGRES DA MITRE

Aproveitando as duas experiências anteriores, a Mitre Corporation desenvolveu uma versão do INGRES (DoP 79) baseada em modificação do pedido (Woo 79). Restrições de segurança foram impostas ao INGRES e o sistema resultante foi integrado ao protótipo do NÚCLEO DA MITRE (Mil 76). Assim, esta foi a primeira implementação real de um Banco de Dados Seguro em um Sistema Operacional Seguro. Não foi necessário o desenvolvimento de primitivas, pois as do núcleo foram aproveitadas. Nenhuma codificação de segurança foi adicionada ao sistema, simplesmente se fez uso dos objetos providos pelo NÚCLEO DA MITRE (seção VII.4), capazes de acomodar as relações do banco de dados INGRES.

O sistema de arquivos do NÚCLEO DA MITRE é hierárquico, consistindo de diretórios que podem conter arquivos de dados e outros diretórios. No INGRES DA MITRE é assumido que todos os arquivos em um diretório estão no mesmo nível de segurança do diretório. Consequentemente o nível de segurança de cada relação do INGRES (interpretada como um arquivo) deve ser o mesmo do seu banco de dados (interpretado como um diretório). Embora tal limitação reduza a conveniência do sistema, essa coordenação foi necessária por motivos de mapeamento. Aceitando essas restrições, operações multiníveis podem ser realizadas em relações do INGRES. Deve ser criado um banco de dados para cada nível. Um usuário pode LER informação de um arquivo a nível inferior ao seu, assim, novas relações podem ser criadas pela combinação das informações contidas nos bancos de dados de nível inferior ao que está sendo criado.

VIII.4 MECANISMOS PARA BANCOS DE DADOS RELACIONAIS

Um banco de dados relacional é um agrupamento de dados organizados por um conjunto finito de relações (tabelas não ordenadas). O modelo relacional é tão agradável ao preguiçoso quanto ao especialista, para o primeiro porque tabelas são um método fácil de manter dados e para o segundo porque uma relação é um conceito matemático simples e familiar (CaB 80).

O conjunto finito de relações de um banco de dados relacional é chamado RELAÇÕES DE BASE. Constituem a forma mais simples e natural de modelar o universo em que existem. A descrição de uma relação de base inclui uma lista de colunas como a relação EMPREGADO abaixo, que contém 4 colunas:

EMPREGADO (NOME, SALÁRIO, GERENTE, DEPARTAMENTO)
VENDAS (DEPARTAMENTO, ÍTEM, VOLUME)
LOCALIZAÇÃO (DEPARTAMENTO, ANDAR)

Essas 3 relações se referem ao banco de dados de uma loja de departamentos e são suficientes para descrever suas operações de venda.

Entretanto, as consultas ao banco de dados nem sempre são feitas obedecendo ao padrão das relações de base. Algumas variações simples podem ser feitas, originando diferentes VISTAS. Vistas podem ser criadas, entre outras formas (CGT 75):

. trocando o nome ou a ordem das colunas
. convertendo unidades ou representação de uma coluna
. selecionando um subconjunto de linhas com um predicado
. projetando algumas colunas de relação de base
. juntando várias relações de base em uma tabela única

Vistas portanto, são as relações virtuais, isto é, derivadas das relações de base que estão realmente construídas no banco de dados. No exemplo acima, poderia ser criada uma vista, unindo as relações de base EMPREGADO e LOCALIZAÇÃO.

EMP - LOC VISTA (NOME, SALÁRIO, GERENTE, DEPARTAMENTO, ANDAR)

Para que servem as vistas? Se apenas um usuário tivesse acesso a um banco de dados, qualquer mecanismo de proteção seria desnecessário, exceto uma autenticação na entrada, mas mesmo assim as vistas seriam úteis para conversão, isolamento, etc. Entretanto se muita gente é esperada para compartilhar seletivamente os dados, as vistas formam um mecanismo simples e flexível de controlar o acesso à informação. Usuários e linguagens de perguntas podem acessar apenas as vistas que lhes são autorizadas e que são estritamente necessárias para realizar suas tarefas.

Vamos examinar a seguir 2 bancos de dados relacionais; o SISTEMA R e o UCLA SECURE INGRES.

SISTEMA R

Na maioria dos sistemas de gerência de banco de dados a habilidade de conferir autorização ao uso de cada objeto está centralizada em uma única pessoa: o administrador do banco de dados. O sistema R (de relacional) da IBM foi desenvolvido para aumentar a flexibilidade dos mecanismos de proteção em banco de dados (GrW 76).

O sistema R é construído sobre uma linguagem de definição e manipulação de dados voltada para solução interativa de problemas por usuários que não são especialistas em computador: SEQUEL (CGT 75). A linguagem SEQUEL postula a existência de um conjunto finito de relações de base, sobre as quais pode ser definido um conjunto finito de vistas ou relações virtuais.

Cada usuário do sistema R tem um catálogo com as suas relações de base e vistas. Sempre que define uma nova relação de base, esta é colocada no seu catálogo. Qualquer acesso ao banco de dados só pode ser feito por essas relações. Entretanto, ao usuário é permitido conceder acesso a outros usuários a qualquer de suas vistas (GRANT), revogar o acesso (REVOKE), inserir elementos nas vista (INSERT), deletar elementos (DELETE), atualizar o valor das colunas (UPDATE) e destruir a vista (DESTROY). Quando uma relação de base é criada, todos os campos estão autorizados para todas as operações, mas o criador pode imediatamente definir uma vista não atualizável sem as autorizações acima. O problema da revogação é importante, visto que o GRANT coloca uma cópia da vista no catálogo de cada usuário beneficiado. Quando a vista é destruída, é também deletada de todos os catálogos dos usuários aos quais foi concedida. Qualquer modificação na autorização de uma vista é também propagada em todas as cópias da mesma.

Cada vista carrega as autorizações para as operações descritas acima. Além disso, cada coluna de uma vista carrega um conjunto de atributos: escopo, compartilhamento, unidades, representação, descrição de papel e autorização de atualização. Vistas podem ser definidas a partir de outras vistas, formando uma hierarquia, na base da qual está a relação de base original (a única que existe realmente na memória).

Quando uma transação é validada se estiver trabalhando sobre uma vista, todas as atualizações são refletidas hierarquia abaixo até a relação de base e hierarquia acima em todos os seus descendentes. Este procedimento é regulamentado por dois princípios, que gerem as atualizações nas vistas:

1. REGRA DA UNICIDADE- qualquer inserção, atualização ou deleção é permitida numa vista apenas se existe uma operação única que pode ser aplicada às relações de base.

2. REGRA DO RETÂNGULO- qualquer inserção, atualização ou deleção só pode afetar a informação que é visível no retângulo da vista.

Dessa forma, é impossível que através da modificação de uma vista, seja subvertido o conteúdo das relações de base que não aparecem na vista. Finalmente, predicados podem ser definidos como um

critério de seleção de acesso dentro de uma vista, especificando restrições de acesso.

Fagin (Fag 78) propôs a seguinte modificação ao mecanismo de proteção do Sistema R: permitir que a tabela de autorização contivesse mais que um GRANT do mesmo usuário para o mesmo objeto com os mesmos privilégios. Isto porque, em certas situações o sistema R pode não autorizar algum usuário a conceder ou exercer um privilégio que a ele tivesse direito. Por exemplo: se o usuário A revogar um direito concedido a B e que B tenha concedido a C, não é sempre necessário que C perca o direito, pois pode tê-lo recebido de D (fig. 8.3). A modificação de Fagin consistiu em alterar o grafo de autorizações onde cada nó corresponde a um usuário e cada caminho corresponde a uma concessão permitindo que hajam múltiplos caminhos de cada tipo de um nó para outro.

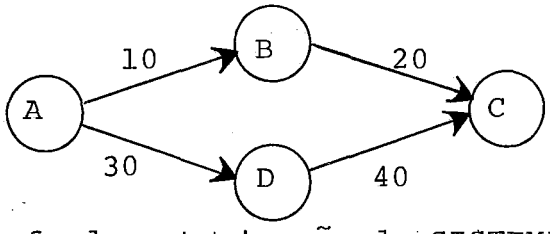


Fig. 8.3 Grafo de autorização do SISTEMA R

UCLA SECURE INGRES

Um núcleo seguro para gerência de banco de dados foi desenvolvido na UCLA utilizando na base o software INGRES (DoP 79). A experiência positiva com o desenvolvimento do UCLA SECURE UNIX (PKK 79) encorajou a utilização das idéias de núcleo para banco de dados. Um conjunto mínimo de funções de DBMS foi selecionado, estruturado, certificado e colocado para rodar sobre o próprio núcleo do UCLA UNIX, constituindo o núcleo do banco de dados.

O sistema é constituído por 4 módulos principais: o KIC (controlador de entrada do núcleo), o núcleo, o software de formatação e o DMM (modelo de gerência de dados). O DMM não necessita ser certificado, porém deve rodar isoladamente no seu próprio ambiente de execução sem possibilidade de retornar informação ao usuário. Contém todas as funções não relacionadas à segurança mas necessárias para a gerência do banco de dados. Poderia rodar sozinho como um DBMS inseguro. Suas funções são: criação e pesquisa em dicionários, catálogos e índices, projeção, compressão, reformulação de dados, etc. O KIC e o núcleo são os dois módulos funcionais do núcleo de segurança, portanto precisam ser certificados. O KIC é a interface segura entre o usuário e o DMM e o núcleo é a interface segura entre o DMM e o banco de dados. O software de formatação roda para cada usuário com seus próprios privilégios e realiza funções específicas do banco de dados como reformatação de dados.
A fig. 8.4 apresenta a interconexão dos 4 módulos numa operação de recuperação.

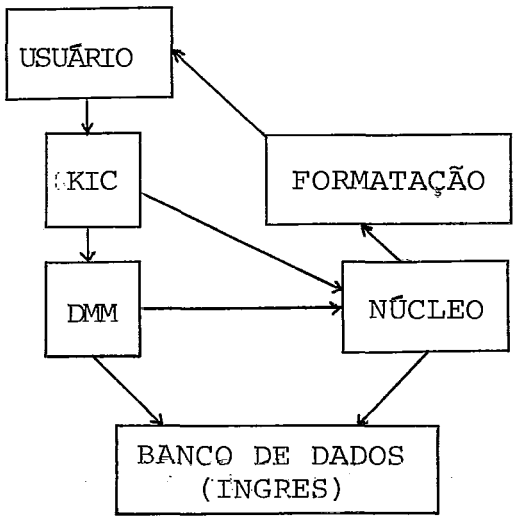


Fig. 8.4 - Operação de recuperação no UCLA INGRES

Quando um pedido de recuperação entra no sistema, o KIC o analisa gramaticamente, retém o tipo de pedido (recuperação) e os nomes lógicos de cada objeto a ser recuperado. Esses "nomes lógicos" não são nada mais que as VISTAS que diferentes usuários possuem do mesmo objeto físico. Para reduzir a multiplicidade de entidades lógicas usadas para identificar o mesmo dado, o KIC analisa gramaticamente as linguagens de definição de dados e constrói as tabelas necessárias para transladar vistas em entidades do sistema.

O KIC realiza também cheques da cláusula "onde" e checa os dados de proteção para determinar se o usuário está autorizado a acessar os objetos que solicitou. Pedidos ilegais são recusados nesta fase. Se o pedido é legal, o KIC passa para o núcleo o tipo de pedido, as entidades do sistema e a identificação do usuário. Abaixo desse nível não existem mais vistas nem outros esquemas de usuários, tudo o que o INGRES recebe são as entidades do sistema.

Os dados que serão retornados para o usuário precisam ser transladados da forma interna usada na memória do banco de dados para o formato do esquema do usuário; este trabalho é realizado pelo módulo de formatação que roda no seu próprio ambiente de execução protegido e usa uma nova cópia em cada invocação, assim, dados de rodadas anteriores não são retidos. Uma vez formatados os dados são liberados para a área de trabalho do usuário.

O INGRES foi escolhido como software básico para a implantação do núcleo por ser um modelo relacional, estruturado e modular, que suporta adaptação sem muita dificuldade. Além disso é um software tipicamente acadêmico, mais de 100 instalações já foram feitas em universidades e pouquíssimas em sistemas comerciais, e rodava na UCLA paralelamente ao UNIX antes de ter sido idealizado o núcleo de segurança.

A conclusão final do projeto foi que o uso de núcleos de segurança para gerência de dados e particularmente de banco de dados é surpreendente e encorajante. Além disso, que a adaptação de um software de banco de dados existente para atender a uma certa política de segurança é não só viável como relativamente fácil.

VIII.5 MECANISMOS PARA BANCOS DE DADOS ESTATÍSTICOS

Bancos de dados estatísticos tem o objetivo de produzir sumários estatísticos de uma população sem revelar dados confidenciais de nenhum indivíduo em particular. O problema é que os sumários contém vestígios da informação original e é possível comprometer a privacidade individual deduzindo-se informações pessoais a partir destes. Isto se chama DEDUÇÃO POR INFERÊNCIA.

Não apenas bancos de dados estatísticos estão sujeitos à exposição por inferência. Sistemas modernos de bancos de dados relacionais possuem poderosas linguagens de perguntas que facilitam a obtenção de estatísticas sobre subgrupos arbitrários de indivíduos.

Ao invocar um programa de respostas, o usuário deve fornecer uma fórmula característica C, cujos termos são compostos por operadores booleanos (AND, OR, NOT). O conjunto dos registros cujo conteúdo satisfaz a fórmula C é chamado "conjunto-resposta para C". A resposta que o usuário solicitou é proveniente desse conjunto - resposta e pode ser obtida por (DeD 79):

- contagem dos registros
- soma dos valores dos registros
- média dos registros
- média dos valores dos registros
- valor máximo ou mínimo
- seleção de um valor
- etc.

Perguntas-padrão são formuladas e levam à obtenção do conjunto-resposta: quantos, qual a média, qual o máximo, etc. Um registro é comprometido se o usuário pode deduzir seu conteúdo confidencial através das respostas aparentemente inofensivas que são retornadas às perguntas-padrão. Existem dezenas de métodos conhecidos para se extrair informação confidencial de um dossiê. Vamos apresentar 5 deles:

MÉTODO 1. Isolar o indivíduo por algumas características que ele possui. Suponhamos por exemplo que X tenha um conjunto de informações sobre Y e queira obter outras. Então faz a seguinte pergunta a um banco de dados médico, onde garantidamente Y está:

P1: "Quantos pacientes do sexo masculino, idade entre 45 e 50 anos, formado em Economia na FGV, casado e com dois filhos é vice-presidente de Banco?"

Se a resposta for 1, o indivíduo Y já está isolado. A partir daí, qualquer informação sobre Y pode ser obtida.

P2: "Quantos pacientes do sexo masculino, idade entre 45 e 50 anos, formado em Economia na FGV, casado e com 2 filhos, vice-presidente de Banco toma drogas para depressão?"

Se a resposta for 1, X fica sabendo que "Y toma drogas para depressão". Se for 0 "Y não toma". Isolar um indivíduo num banco de dados pode ser feito facilmente, bas-

tando que se tenha algumas informações do indivíduo.

A solução mais intuitiva para esse problema é estabelecer um VALOR MÍNIMO para as respostas ou um LIMITE INFERIOR , abaixo do qual a informação é negada (Hof 77).

MÉTODO 2. Isolar o indivíduo por uma característica que ele não possui. Suponhamos que Z seja jornalista e do sexo feminino. Para saber seu salário, X pode excluí-la de um grupo:

P1: "Quantas pessoas nesta empresa são jornalistas?"
R1: 10
P2: "Quantas pessoas nessa empresa são jornalistas e do sexo masculino?
R2: 9

Neste ponto, Z já está excluída do grupo.

P3: "Qual é o total de salários recebidos por jornalistas nessa empresa?"
R3: Cr$ 5.000.000
P4: "Qual é o total de salários recebidos por jornalistas do sexo masculino nessa empresa?"
R4: Cr$ 4.000.000

Neste ponto, X acaba de descobrir que o salário de Z é Cr$ 1.000.000.

Também este problema pode ser intuitivamente solucionado, através da criação de um VALOR MÁXIMO para as respostas ou um LIMITE SUPERIOR, acima do qual toda a informação é bloqueada (DeD 79).

MÉTODO 3. Colocar o indivíduo num grupo com uma determinada característica. Não é necessário isolar Y para saber que ele possui uma característica. Suponhamos que X faça duas perguntas:

P1: "Quantos pacientes do sexo masculino, idade entre 45 e 50 anos, casado e com 2 filhos é vice-presidente de Banco?"
R1: 10
P2: "Quantos pacientes do sexo masculino, idade entre 45 e 50 anos, casado e com 2 filhos, vice-presidente de Banco, já sofreu ataque cardíaco?"
R2: 10

Se R2 for igual a R1 e Y está no conjunto 1, Y está também no conjunto 2, portanto, já sofreu ataque cardíaco.

Uma medida de proteção é fazer o sistema retornar sempre INTERVALOS (exemplo: entre 10 e 20) em vez de números exatos.

MÉTODO 4. Adicionar entradas fantasmas. O interlocutor pode adicionar 1 a 1 entradas fantasmas ao banco de dados, de maneira que quando o intervalo muda, fica sabendo o número exa

to de indivíduos com a informação desejada. Este método é chamado "rastreamento" e tem sido objeto de muita pesquisa. Mecanismos de proteção para encontrar "rastreadores" de bancos de dados estão sendo criados e implementados (DDS 79, DeS 80).

MÉTODO 5. Resolver um sistema de equações simultâneas. Para isso, o interlocutor faz um grande número de perguntas, todas "honestas" e obtém uma grande massa de informação redundante. Resolvendo um sistema de equações lineares pode determinar o valor exato de cada item.

Um exemplo desse caso é a campanha de levantamento de fundos para fins políticos nos USA. $C_1 \ldots C_9$ são os contribuintes específicos, cujos nomes e quantias devem ser confidenciais. Por outro lado, o banco de dados que contém essas informações é estatístico e aceita perguntas do tipo:

P1: "Qual o total de contribuições dadas por empresários do aço?"
P2: "Qual o total de fundos obtidos por democratas?"

O resultado dessas perguntas gera um sistema de equações lineares que, solucionado, fornece a quantia exata com que cada companhia contribuiu.

O mecanismo proposto para resolver êsse problema é o da SOBREPOSIÇÃO ('overlap'). A técnica da sobreposição impede que sejam respondidas perguntas que tenham além de um número máximo de registros em comum com perguntas anteriores. Entretanto, além de ser difícil de implementar, esse mecanismo não é totalmente seguro, pois o sistema de equações pode ser construído por vários usuários diferentes que colaboram entre si de tal forma que os pedidos de cada um não são suspeitos, mas o conjunto todo é (Hof 77).

Peter e Dorothy Denning (DeD 79) analisaram vários mecanismos para resolver esses casos mais complicados:

1. INOCULAÇÃO DE INFORMAÇÕES FALSAS — no banco de dados, de maneira que apenas as estatísticas sejam válidas e nenhum registro individual mereça crédito. Este mecanismo, entretanto, não é aplicável para arquivos de dados longitudinais, onde dados individuais precisam ser mantidos. Outra desvantagem é que a inoculação causa modificação no dado antes que ele seja armazenado no registro, por isso, os dados originais são perdidos.

2. ARMAZENAMENTO DE UM FATOR DE PERTURBAÇÃO — permanentemente no registro, o qual não altera os dados originais na sua fonte, mas é aplicado sempre que for fornecida uma resposta.

3. ARREDONDAMENTO — pela adição de um valor aleatório de média

zero, mas possui o risco de a resposta correta ser deduzida pela média de um número grande de respostas à mesma pergunta. Arredondamento pela adição de um valor aleatório que depende do dado é preferível, visto que a uma dada pergunta sempre retornará a mesma resposta. Bancos de dados que utilizam esse mecanismo também podem ser subvertidos pelo uso de rastreadores.

4. PARTICIONAMENTO — do banco de dados armazenando registros em vários grupos, cada qual contendo um número mínimo, pré-determinado, de registros. Perguntas podem ser aplicadas a qualquer grupo de registros, mas nunca a subconjuntos internos a um grupo. Através desse mecanismo, os ataques baseados em inclusão e exclusão podem, no máximo, isolar um dos grupos, o que não altera nada pois perguntas a grupos isolados são permitidas. Uma variação desse mecanismo é a MICROAGREGAÇÃO: grupos de indivíduos são agregados em "médias individuais" sintéticas e as estatísticas são calculadas para os indivíduos do grupo sintético em vez de para os reais.

5. DISSOCIAÇÃO — ou "mentira" é o mecanismo no qual o banco de dados responde a uma pergunta com o valor exato de um registro, porém não necessariamente o registro selecionado pela pergunta. Técnicas relacionadas são baseadas na troca de valores de dados entre registros de maneira que propriedades estatísticas sejam preservadas, mas, infelizmente não foi ainda descoberto um meio eficiente de trocar esses valores (DeD 79).

6. AMOSTRAGEM ALEATÓRIA DE REGISTROS — é o mecanismo em que cada pergunta é respondida através de uma amostragem aleatória dos registros do conjunto-resposta. Entretanto deve ser aceito um risco, baixo, de erro na resposta, introduzido pela amostragem.
D. Denning escreveu um artigo detalhado sobre esse mecanismo (Den 80).

Kam e Ullman (KaU 77) desenvolveram um modelo de banco de dados estatístico em que cada pergunta é interpretada como uma especificação binária de s valores sobre os k totais das informações contidas no banco de dados. Por exemplo, se as informações forem (nome, idade, sexo, altura, peso, instrução, trabalho, estado civil, salário), temos k=9 e podemos fazer s=2 se fixarmos (sexo, instrução) com a pergunta: "Total de homens com curso universitário". O modelo prova que é impossível obter valores individuais por s perguntas para $s < k$, mas só é aplicável para perguntas tipo SOMA.

Foi demonstrado (Bec 80) que qualquer sistema que produza respostas estatisticamente corretas é passível de penetração. Entretanto, o número de perguntas necessárias para comprometer um banco de dados pode ser tornado arbitrariamente grande se forem aceitos aumentos moderados na variância das respostas. Becquai apresenta um mecanismo de proteção pelo qual o administrador do banco de dados escolhe os parâmetros C (para perguntas TOTAL e MÉDIA) e Cl (para perguntas CONTAGEM e PERCENTUAL) que determinam esse número mínimo de perguntas.

Finalizando, de todos os métodos empregados para controle de bancos de dados estatísticos, o mais barato é o uso de HISTÓRICO DE ATIVIDADES. Históricos devem conter informações sobre perguntas fora do padrão, tentativas não realizadas de acesso, solicitações repetidas do mesmo registro. Um histórico completo de cada transação submetida a processamento deve ser gerado, com os seguintes itens: identificação do usuário e do terminal, data, hora, modo do sistema, aplicação, nível da transação e outras relacionadas ao tipo do dado manipulado. Embora os históricos não tenham o poder de impedir o fluxo de informação entre os programas de respostas, proporcionam uma barreira psicológica que muitas vezes é suficiente para afastar idéias da mente dos curiosos.

IX - SISTEMAS DISTRIBUÍDOS

IX.1 PROTEÇÃO EM REDES

A necessidade de descentralizar sistemas de computadores, bancos de dados e terminais levou ao uso crescente das comunicações de dados. O desenvolvimento dos minicomputadores nos últimos anos tem aumentado as tendências de descentralização. A capacidade de comunicar com diferentes computadores a partir de um terminal remoto é agora possível graças ao uso de micro-processadores flexíveis e baratos, que podem inclusive realizar a interface entre terminais antigos e redes modernas.

Essa descentralização tem proporcionado acesso fácil e rápido a grandes quantidades de dados dispersos fisicamente, mas também tem ocasionado aumento na vulnerabilidade desses dados. Sistemas distribuídos sem a proteção adequada são sujeitos a todos os perigos de acesso não autorizado, modificação e destruição da informação. Os requisitos de proteção de uma rede (Bra 77) podem ser especificados em 5 itens:

1. Dados devem ser protegidos contra exposição e modificação não autorizadas desde a sua origem até o seu destino.

2. Todos os usuários, terminais e computadores devem ser reconhecidos e identificados em cada novo pedido de acesso à rede.

3. Usuários da rede devem ter vários privilégios de acesso, baseados nas credenciais e autorizações pessoais.

4. A segurança da rede deve depender o mínimo de procedimentos manuais e não pode confiar na integridade pessoal de cada usuário.

5. Técnicas de monitoração e auditoria devem ser capazes de detetar e evitar tentativas de penetração ilegais.

Como atingir êsses requisitos? A simples atitude de inserir um componente seguro na rede não a torna automaticamente segura. É necessário que hajam mecanismos de proteção específicos para sistemas distribuídos, como já existem para os sistemas isolados.

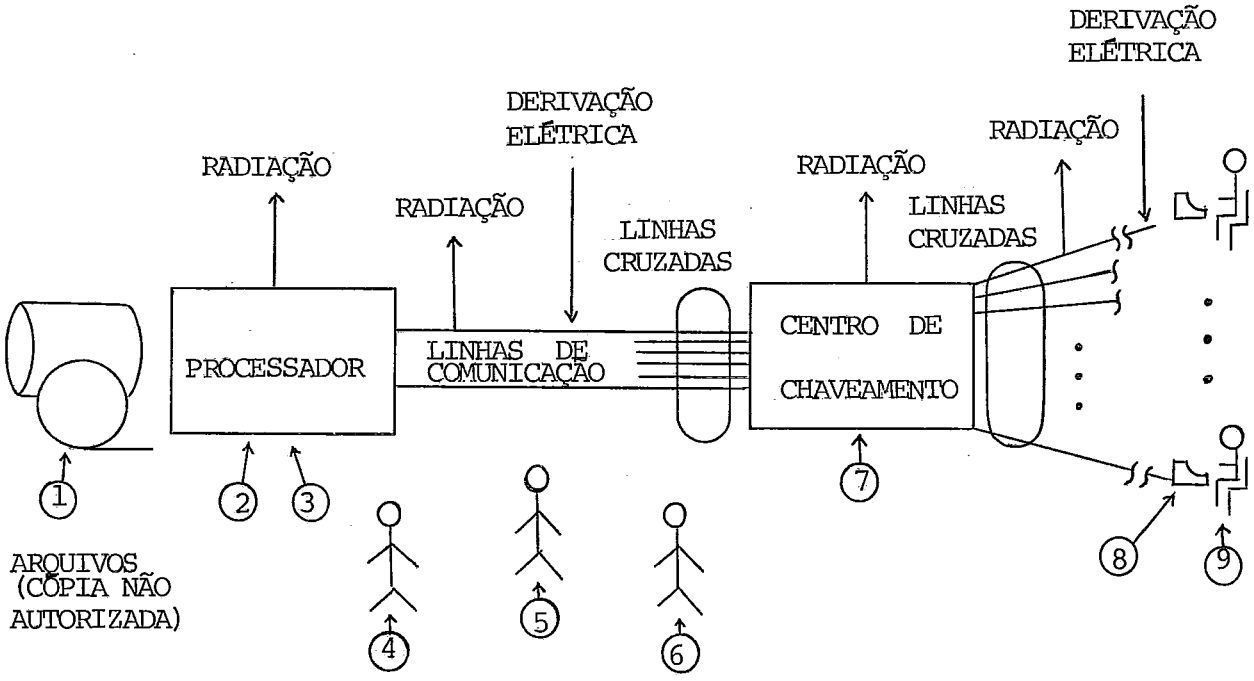
A primeira etapa no estudo dos mecanismos de proteção dos sistemas isolados foi a determinação dos componentes básicos do sistema. É também essa a primeira etapa no caso de redes. Uma rede pode ser definida como consistindo de 3 conjuntos de componentes (WoK 79).

- sistemas de terminais
- sistemas hospedeiros
- equipamento de comunicação de dados

Até onde devem chegar os privilégios de cada conjunto? Ao nível de TERMINAIS devem ser empregados procedimentos de segurança como acesso controlado à sala dos terminais, sinal de identificação interno em cada terminal, além dos mais comuns que se referem à autenticação de usuários. É fundamental para o sistema saber quem é a pessoa que está usando um terminal, para poder determinar seus privilégios e suas limitações. Várias técnicas de autenticação serão descritas na seção IX.2.

Considerando-se o EQUIPAMENTO DE COMUNICAÇÃO, há perigo de escoamento e adulteração da informação por meios eletromagnéticos, e outros problemas, quase sempre resolvidos pela criptografia. Numerosas técnicas de criptografia já foram desenvolvidas, das quais algumas são vistas na seção IX.3.

Finalmente, na segurança dos SISTEMAS HOSPEDEIROS devem ser considerados vários aspectos: a segurança de cada componente isolado, a modificação causada na rede pela adição do componente, o retroajustamento que a ligação com a rede provoca no componente, a compatibilização dos componentes heterogêneos e o posicionamento lógico do controle da rede, que pode ser centralizado ou distribuído. Esses são os assuntos das últimas 3 seções. A fig. 9.1 resume as vulnerabilidades de uma rede.



① ARQUIVOS - cópia não autorizada

② HARDWARE - falha dos circuitos de proteção

③ SOFTWARE - falha dos mecanismos de controle de acesso

④ OPERADOR - revelação das medidas de proteção

⑤ MANUTENÇÃO - incapacitar dispositivos ou componentes do hardware

⑥ PROGRAMADOR DE SISTEMAS - incapacitar mecanismos de proteção, criar janelas

⑦ HARDWARE - conexões incorretas

⑧ ACESSO - conexão incorreta

⑨ USUÁRIO - modificação sutil de software, inclusão de erros

Fig. 9.1 Vulnerabilidades de uma rede de computadores (Hof 77)

IX.2 AUTENTICAÇÃO

Quando um grande número de usuários está potencialmente apto a acessar o conjunto de recursos do computador, devem ser empregadas técnicas que permitam ao sistema realizar uma identificação a nível individual. A utilização de qualquer recurso só deve ser permitida a usuários reconhecidos pelo sistema. Este reconhecimento, envolve duas ações: a primeira do usuário se identificando e a segunda do computador encontrando-o no seu quadro de pessoas conhecidas. A esta sequência é dado o nome de AUTENTICAÇÃO.

A autenticação é parte integrante de qualquer sistema de computador. Mesmo os antigos sistemas de processamento em batch já a utilizavam com propósitos de contabilização e com a chegada dos sistemas de multiprogramação e time-sharing, tornou-se um componente imprescindível nos sistemas operacionais.

As regras que regem a autenticação de usuários dependem das características do sistema e da instalação, podem ser flexíveis ou rígidas e sua severidade é função direta do grau de segurança desejado. Uma vez que em todo sistema há vários níveis de autorização de acesso partindo do simples usuário até o gerente de segurança, a autenticação pode também ser realizada em vários níveis e em qualquer parte da execução de um trabalho, não apenas no início como é mais comum encontrar.

Martin (Mar 73) identificou 3 maneiras de se autenticar um usuário:

1. Por alguma coisa que a pessoa sabe ou memoriza - onde estão incluídos todos os tipos de passwords, métodos de pergunta-resposta, assinaturas, etc.

2. Por alguma característica física da pessoa - exemplos: geometria da mão, timbre de voz, impressões digitais, pupila, etc.

3. Por alguma coisa que a pessoa possui - exemplos: crachá, cartão de autenticação, chave eletrônica, etc.

Vamos a seguir descrever brevemente cada uma dessas técnicas.

1. AUTENTICAÇÃO POR MEMORIZAÇÃO

1.a. PASSWORDS - O método da password consiste em fornecer ao computador uma palavra-chave, conhecida apenas pelo usuário e pelo mecanismo de proteção, através da qual é feito o reconhecimento e o acesso é autorizado. Devido ao seu custo relativamente baixo e facilidade de implementação, passwords são o mecanismo mais comumente empregado para obter a autenticação de usuários. Praticamente todos os sistemas de time-sharing acadêmicos, comerciais e governamentais utilizam passwords há muito tempo, enquanto os outros métodos de autenticação estão começando a ser comercializados.

Entretanto, passwords não representam uma forma totalmente segura de autenticação. A pesquisa exaustiva pode levar à descoberta após um certo tempo, que pode ser abreviado com a utilização de pequenos computadores para fornecer as combinações de caracteres. Hoffman (Hof 77) demonstrou que o tempo esperado para descoberta de uma password é tanto maior quanto maior for seu comprimento e a probabilidade de uma password ser obtida é dada pela fórmula:

$$P \geq \frac{R \times 4.32 \times 10^4 \times M/E}{A^S}$$

onde: R = taxa de transmissão da linha de comunicação (caract./minuto)

E = número de caracteres trocados em uma tentativa de login

M = período de tempo da pesquisa sistemática (meses)

A = tamanho do alfabeto que é universo da password (26,36,etc)

S = comprimento da password

Wood e Kimbleton (WoK 79) afirmam que o problema da password é que elas não são usadas em toda a sua plenitude por humanos. Passwords fáceis de advinhar como nome dos filhos, dos pais, iniciais, datas de nascimento, placas de carro são algumas das selecionadas. Mesmo a utilização de palavras selecionadas de inglês ou português reduz significantemente o universo das passwords. O ideal é tornar as passwords tão longas ou extraí-las de um alfabeto tão amplo, que a pesquisa exaustiva se torne irremediavelmente cara para ser viável.

Variações em torno das passwords são:

. LISTA DE PASSWORDS 'ONE-WAY' - O usuário fornece ao sistema uma lista de passwords que só podem ser utilizadas uma vez cada, em ordem sequencial. Assim, a impressão em relatório não causa exposição do sistema de proteção (exceto nos casos de excessiva falta de criatividade: PASS1, PASS2, PASS3...). Uma desvantagem existente para esse método é que o usuário pode esquecer a lista, ou ainda, anotá-la. Helen Wood (Woo 77) descreve várias implementações de passwords 'one-way'.

. ALGORITMO SELETIVO DE CARACTERES - Um algoritmo interno que escolhe aleatoriamente posições da password cujos caracteres devem ser fornecidos. Por exemlo:

```
LOGIN, SMEDLEY NOP.
     authentication: characters 5 and 7 please?
     RE
     authentication OK. program desired?
```

Finalmente, passwords devem ser alteradas periodicamente, e via-de-regra não devem ser impressas nas seções de terminais nem nos relatórios, ou, nos casos em que for absolutamente impossível, devem ser impressas mascaradas tornando-se ilegíveis.

1.b PERGUNTA-RESPOSTA - No método da pergunta-resposta o sistema fornece ao usuário uma série de perguntas escolhidas aleatoriamente de uma lista que deve ser respondida corretamente para efetuar a autenticação. Devem ser perguntas particulares, de universo restrito a um indivíduo e desconhecidas de outros usuários. Exemplos:

.Sobrenome de solteira da sua mãe
.Escola primária frequentada
.Cidade em que morava em 1964

O método pode ser simplificado se existir um número suficientemente grande de perguntas-padrão no sistema, de onde cada usuário pode extrair um subconjunto, que responde, e se torna seu universo de respostas. Assim, o sistema só precisa prover memória individual para armazenar os números das perguntas e suas respectivas respostas (KrG 79).

1.c ASSINATURA - Basicamente existem 2 formas de autenticar assinaturas: estaticamente, baseado na posição e direção dos caracteres da assinatura e dinamicamente, baseado na velocidade, aceleração e na inclinação do vetor força exercido pela mão durante a assinatura.

O sistema VERIPEN registra de uma maneira tridimensional as componentes do vetor força que atua no movimento da assinatura. Uma codificação digital dessas forças, estabelecendo limites inferior e superior é feita e arquivada como amostra, contra a qual é comparada a assinatura do usuário. Na verdade, a Veripen Inc. admite que 5% das pessoas não tem assinatura coerente (Car 77).

2. AUTENTICAÇÃO POR CARACTERÍSTICA FÍSICA

2.a TIMBRE DE VOZ - O equipamento necessário é apenas um telefone. O usuário fala uma frase pré-determinada, a qual é convertida em uma sequência de bits e comparada a um modelo previamente armazenado que é o padrão deste usuário. Alterações involuntárias de timbre (por exemplo durante um resfriado) devem ser consideradas, estabelecendo-se um limite de variação dentro das mesmas características sonoras. Os laboratórios da Bell Telephone Company desenvolveram um dispositivo em que a probabilidade de falsa rejeição e de falsa aceitação eram ambas 1,2%. Ainda assim existem possibilidades de quebra no mecanismo de proteção. Uma pessoa treinada para imitar perfeitamente outra pode obter uma porcentagem de aceitação igual a 42%. Ou também, o impostor pode gravar numa fita a voz da pessoa autorizada. Usada conjuntamente com outro mecanismo, por exemplo, uma password, o reconhecimento da voz pode resultar em ótima medida de proteção (Mar 73).

2.b GEOMETRIA DA MÃO - Em 1969 o Stanford Research Institute realizou uma pesquisa em mais de 4000 pessoas obtendo o seguinte resultado: se o padrão de reconhecimento for os comprimentos dos 4 dedos de uma mão, a probabilidade de êrro é de 0.5%, para uma tolerância de ± 1.5mm. Se a medida de tolerância for de ± 0.75mm a probabilidade de êrro é de 0.05% o que é um resultado excelente. Entretanto, o crescimento das unhas pode alte

ar o resultado, rejeitando autenticação a usuários honestos. Para evitar isso, o IDENTIMAT 2000 usa uma luz intensa que torna os dedos translucentes e então mede a transmissão de luz proveniente das suas pontas independentemente do comprimento das unhas (Mar 73).

2.c IMPRESSÕES DIGITAIS - Há pelo menos 2 sistemas registrados para análise das impressões digitais. Um deles, chamado FINGERSCAN é fabricado pela Calspan Corpo. O outro, FIS (Fingerprint Identification System) é oferecido pela Rockwell International. São baseados no mapeamento de minúcias que são os términos ou bifurcações das marcas digitais. A Defense Inteligence Agency realizou um teste no FINGERSCAN e obteve as taxas de 6,3% de falsa aceitação e 6,1% de falsa rejeição (Car 77).

3. AUTENTICAÇÃO POR OBJETO POSSUÍDO

3.a CARTÃO DE AUTENTICAÇÃO - Este sistema tem sido usado por bancos em caixas automáticas. O usuário insere um "cartão de recebimento" na máquina da rua e recebe o dinheiro. Obviamente, cartões perdidos deverão ser imediatamente comunicados ao computador, e nesse caso, a sua apresentação deve causar a retenção do cartão pela máquina. A American Bank Association padronisou um modelo de cartão de crédito, magnético, onde os campos: data de expiração, limite de crédito, número da conta e emitente estão em posições fixas. Os dispositivos de leitura desses cartões padronizados, são baratos e simples de acoplar ao computador (CoM 77).

3.b CHAVE ELETRÔNICA - Há centenas de anos o homem aprendeu a usar chaves para guardar os seus bens. Dorothy Denning (Den 78) advoga a idéia de que o usuário não deve confiar no sistema de proteção que lhe é fornecido gratuitamente pelo computador, mas deve ser responsável pela proteção dos seus próprios bens eletrônicos. Para isso, apresenta uma chave eletrônica, um microcomputador LSI, que cada usuário deve possuir, o qual codifica a mensagem e a envia ao sistema. Na volta, decodifica a mensagem e a fornece ao usuário. Neste processo, o chip é elemento indispensável para qualquer interação com o sistema. Maiores detalhes desse mecanismo serão vistos na seção IX.3.

AUTENTICAÇÃO REVERSA é a que se processa no sentido contrário, ou seja: do sistema ao usuário (Hof 77). Em sistemas distribuídos é muito importante, porque havendo vários componentes na rede o usuário precisa ter garantias de que está se comunicando exatamente com o nó que deseja e não com um espião que está se fazendo passar por este. Uma das formas de realizar autenticação reversa é, após o usuário ter sido autenticado, obrigar o computador a imprimir no terminal do usuário uma password secreta conhecida apenas por esse computador e pelo usuário. Outros processos transparentes ao usuário podem ser realizados em software ou em hardware.

Várias técnicas podem ser combinadas para produzir uma autenticação legítima. Qualquer que seja o esquema utilizado, é necessário que seja: fácil para utilização por qualquer usuário do sistema, barato e rápido em termos de utilização de CPU e confiável. Nada é mais ridículo do que um mecanismo de proteção facilmente burlável.

As taxas de falsa rejeição e falsa aceitação devem ser baixas e uma supervisão administrativa deve ser mantida. Isto significa que devem ser gerados históricos e que deve existir alguém responsável pelo seu exame e pela tomada das providências competentes.

IX.3 CRIPTOGRAFIA

Criptografia é um mecanismo de proteção de dados que pode ser efetivamente utilizado em sistemas distribuídos. O grau de proteção proporcionado pela criptografia depende do algoritmo empregado, da implementação do algoritmo e dos procedimentos administrativos que regulam o seu uso. Mecanismos adicionais como autenticação do usuário, autorização de acesso e auditoria devem ser combinados com os de criptografia para se atingir um bom nível de segurança na rede.

A palavra CRIPTOGRAFIA origina-se de um vocábulo grego (escrita camuflada) e significa a transformação dos dados para uma forma camuflada, não compreensível, chamada cifra. DECRIPTOGRAFIA ou decifragem é o processo inverso e retorna a cifra à sua forma original (Bra 77).

Todas as transformações de dados produzidas durante um processo de cifragem são variações de apenas duas transformações básicas: permutação e substituição. A PERMUTAÇÃO muda a ordem dos símbolos no texto de dados. A SUBSTITUIÇÃO troca os símbolos por outros. Durante a permutação os caracteres retém suas identidades mas perdem suas posições. Durante a substituição mantém suas posições, mas perdem suas identidades.

As transformações básicas podem ser combinadas dando origem às transformações complexas. Considerando que toda informação é armazenada no computador sob a forma binária, a criptografia consiste no reposicionamento ou na substituição de um grupo de bits por outro. O grupo de bits substituído pode ser menor, de mesmo tamanho, ou maior que o grupo original.

A transformação de substituição é utilizada tanto na cifragem como na CODIFICAÇÃO. Um código contém um símbolo de substituição para cada símbolo do texto inicial, que pode ser uma letra, uma sílaba, uma palavra, uma frase ou um símbolo especial. Um livro de código é uma lista de todos os símbolos possíveis ou elementos do texto inicial e seus códigos equivalentes. Codificação é o processo de substituição dos símbolos do texto pelos códigos, que constituem a mensagem a ser transmitida numa linha de comunicação. DECODIFICAÇÃO é o processo inverso.

Existem algumas diferenças entre codificação e cifragem, tão sutis que passam imperceptíveis para a maioria das pessoas (Bra 77). A unidade básica de um CÓDIGO é uma palavra ou frase, portanto, a codificação opera em entidades linguísticas. Por outro lado, a unidade básica de uma CIFRA é uma letra, um número fixo de letras ou um grupo de bits, portanto a cifragem opera em entidades sintáticas. Geralmente os códigos utilizam unidades de comprimento variável e as cifras unidades de comprimento fixo. Pode acontecer que um bloco tenha seu comprimento reduzido após a cifragem, mas isso é consequência de um processo de compressão interno ao de cifragem. Na maioria dos casos os dois grupos de bits são do mesmo tamanho. Embora ambos os processos: cifragem e codificação sejam utilizados nas transmissões em sistemas distribuídos, nos ocuparemos apenas do primeiro.

Um processo de cifragem é normalmente construído de maneira iterativa, em que as transformações de substituição e permutação são realizadas várias vezes, originando o nome CIFRAGEM DE RECIRCULAÇÃO. Existem 2 métodos diversos: por um processo secreto ou por um processo conhecido que depende de parâmetro secreto. Este último método foi escolhido para redes de computadores, pois permite a compatibilização dos seus componentes heterogêneos. Assim, um processo conhecido é realizado por um algoritmo utilizando um parâmentro secreto denominado CHAVE, e os dados são criptografados antes de sua transmissão através da rede. No destino os dados são decriptografados e retornam à forma original. Quando a mesma chave é usada para cifrar e decifrar diz-se que a cifragem é simétrica, em caso contrário, é assimétrica.

CIFRAGEM SIMÉTRICA

A cifragem simétrica é utilizada pelo algoritmo do DES (Data Encryption Standard) desenvolvido pela IBM e lançado nacionalmente nos Estados Unidos pelo NBS (NBS 77). O algoritmo do DES utiliza um bloco fixo de 64 bits como entrada e uma chave de 64 bits. É baseado em várias permutações e em um conjunto de tabelas de substituição. A cada iteração o conjunto de 64 bits é dividido em 2 metades de 32 bits. A primeira metade é adicionada (módulo 2) ao resultado da combinação da segunda metade com parte da chave. Em seguida as duas metades são permutadas e a operação é repetida 16 vezes. O processo de decifragem é exatamente o mesmo, bastando entrar com o resultado e a chave na ordem inversa para se obter os bits originais (fig. 9.2). Este algoritmo tem várias características interessantes para utilização em redes de computadores. Inicialmente é fácil de implementar em qualquer sistema. Além disso, a cifragem de cada bloco é um processo totalmente independente da cifragem dos demais, permitindo que blocos simples sejam decifrados sem necessariamente decifrar todas as mensagens (utilizando várias chaves diferentes). Não são necessárias sincronizações de tempo nem de posição nas operações de cifragem/decifragem. O formato de cada bloco de dados pode ser definido para cada aplicação, pois não é necessário que os 64 bits sejam todos preenchidos. Subcampos de cada bloco podem ser definidos para incluir: número de seqüência do bloco, número aleatório para mascarar campos idênticos, informação de deleção e erros, para evitar modificações não autorizadas, identificação de usuário, terminal ou mensagem.

Embora o DES tenha sido implementado e utilizado em larga escala nesses 4 anos de existência, tem sido também severamente criticado. A primeira crítica é que a segurança do algoritmo é totalmente baseada na chave de criptografia. Além de todo o aparato administrativo que deve controlar a geração e distribuição das chaves, Diffie e Hellmann (DiH 79) afirmaram que por 20 milhões de dólares se constrói um computador que obtém a chave em 12 horas e por 4 milhões um que o faz em 24 horas (Hel 78). Hellman afirmou que houve deliberação por parte da National Security Agency em reduzir o tamanho da chave para poder espiar transações comerciais. O tamanho original da chave no projeto da IBM era de 128 bits. Outra crítica se referiu às tabelas de substituição empregadas no algoritmo, supostamente aleatórias, mas que poderiam conter segredos de como identificar facilmente a chave. Essas denúncias foram rebatidas severamente tanto pelo staff da IBM como do NBS (HDT 79). Um Comitê do

Senado foi constituído para investigá-las e não chegou a provar nenhuma evidência de envolvimento deliberado nas transações privativas de cada empresa.

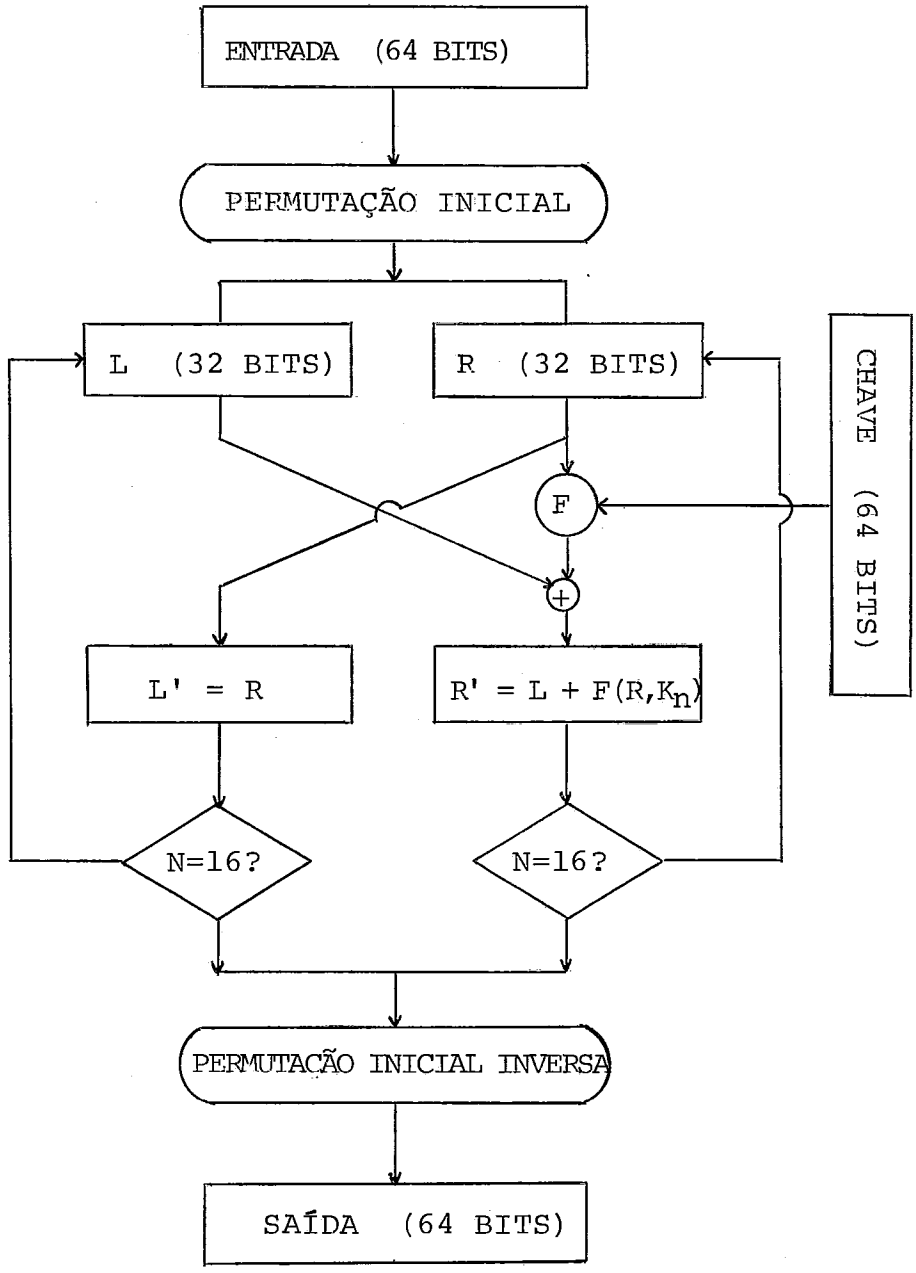


Fig. 9.2 - Algoritmo de criptografia do DES (NBS 77)

CIFRAGEM ASSIMÉTRICA

São algoritmos de criptografia em que duas chaves diferentes são utilizadas: uma para cifragem outra para decifragem. O mecanismo de cifragem assimétrica mais divulgado chama-se CHAVE PÚBLICA. Neste, uma das chaves é secreta e outra é de conhecimento público. Normalmente usa-se a chave pública para a cifragem do texto por qualquer usuário e a secreta para a decifragem por apenas um usuário ou um grupo selecionado, mas o oposto pode acontecer. Pode-se denotar matematicamente esse algoritmo em que M é a mensagem, P a chave pública, S a chave secreta e C a mensagem criptografada por:

$$C = P(M) \qquad M = S(C)$$

Deve-se também assumir que o algoritmo de chave pública produz uma cifra no mesmo espaço da mensagem M e que as chaves P e S são comutativas. Portanto:

$$S(P(M)) = P(S(M)) = M$$

Dorothy Denning (Den 78) propôs um mecanismo de criptografia assimétrica construído em hardware de chips individuais. Cada usuário do sistema deve ser proprietário de um chip de ROM onde as 2 chaves estão gravadas. A 1ª sequência de bits corresponde à chave pública, é informada ao usuário quando adquire o chip. A sequência correspondente à chave secreta é desconhecida mesmo pelo dono do chip, por não ter utilidade o seu conhecimento sem a posse do chip.

O fabricante pode reter registros de chaves para resolver os casos de chaves perdidas ou roubadas, mas devem ficar bem guardados. O risco é o mesmo que em outras áreas: ninguém que compra um carro se preocupa em ser roubado pelo fabricante do mesmo. Uma alternativa é fornecer ao usuário um mecanismo para gerar um par de chaves aleatórias em um chip de memória redigível e depois selar o chip impedindo qualquer alteração.

O algoritmo de chave pública é muito útil na geração de ASSINATURAS ELETRÔNICAS, uma técnica usada para garantir a procedência das mensagens. Suponhamos que um usuário A quer enviar uma mensagem assinada M a outro usuário B. Inicialmente criptografa a mensagem com sua chave secreta S_A gerando $S_A(M)$ e depois com a pública P_B de B gerando $P_B(S_A(M))$. Quando a mensagem chega, B decriptografa-a com sua chave secreta S_B gerando novamente $S_A(M)$ e depois com a pública A gerando M. Como apenas A seria capaz de cifrar uma mensagem com S_A, B fica seguro de que a mensagem proveio de A.

Com o uso das chaves públicas a segurança da rede não é mais condição essencial para a segurança do usuário que se torna um problema particular de cada indivíduo do sistema. O sistema não precisa mais tomar conhecimento da chave e quaisquer pessoas podem enviar mensagens a outras sabendo que apenas essas serão capazes de decifrá-las.

CARACTERÍSTICAS ESPECIAIS DA CRIPTOGRAFIA EM REDES

Existem 2 caminhos básicos pelos quais a criptografia pode ser aplicada às redes, dependendo se a cifragem for responsabilidade da rede ou dos usuários. Na criptografia de LINK, a mensagem é decifrada e recifrada em cada nó, o que permite que inclusive informações de endereçamento sejam cifradas. Na criptografia END-TO-END a mensagem é cifrada na fonte e decifrada apenas no seu destino, o que a torna mais protegida, embora não permita a cifragem dos endereços. Essa restrição pode acarretar um fluxo implícito de dados, como a identificação de 2 nós quaisquer que estejam se comunicando.

Outra característica especial diz respeito a TRANSMISSÃO DAS CHAVES entre os nós de uma rede. Needham e Schroeder propuseram um método para uma troca segura de chaves (DeD 79). A idéia é usar um computador (KG) que gera aleatóriamente uma chave para cada compo-

nente na rede e guarda uma cópia de cada chave gerada numa tabela. Suponhamos que A, cuja chave é K_A queira se comunicar com B cuja chave é K_B (fig. 9.3). Como apenas KG conhece as chaves K_A e K_B, nem A nem B podem usá-las entre si. Inicialmente A escolhe um identificador arbitrário I para a mensagem que quer enviar a B e envia a KG o seguinte texto: (A, K_A(I, B). Como o nome de A não está cifrado, KG vai à sua tabela, obtém K_A e decifra (I, B). Fica então sabendo que A quer enviar I a B. Então KG gera uma chave aleatória K e responde a A com o texto K_A(I, K, K_B(K, A). Como K_A é conhecido por A, este fica conhecendo a chave K e em seguida envia a B o pacote K_B(K, A), que é desconhecido para A, mas que é corretamente decodificado por B, o qual fica também conhecendo a chave K. A partir daí, ambos se comunicam quantas vezes quiserem pela chave K. É claro que seria mais simples se KG fornecesse diretamente K_B a A ou K_A a B, mas nesse caso estes ficariam de posse da chave um do outro e poderiam interceptar qualquer mensagem dos mesmos que circulasse pela rede.

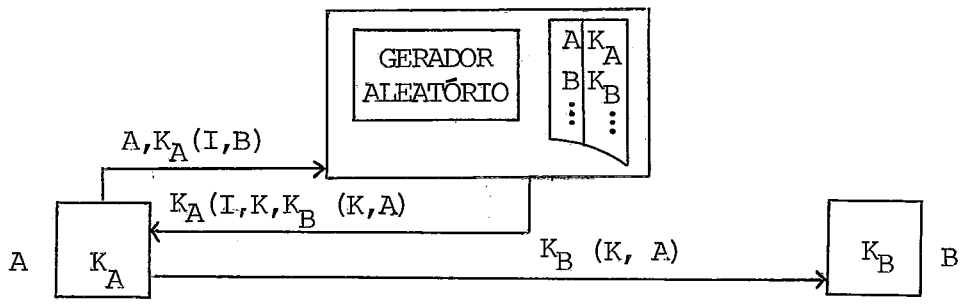


Fig. 9.3 - Transmissão de chaves entre dois nós de uma rede

Concluindo, embora hajam inúmeras outras aplicações para a criptografia (exemplo Seção V.6) sua maior importância se refere ao uso em redes de computadores. Arquivos e sistemas de tempo compartilhado possuem um controle central que se encarrega da maioria das funções de proteção. Mas redes, formadas por componentes geograficamente dispersos e por empresas possivelmente concorrentes, podem não possuir nenhum controle central. É vital para o funcionamento do sistema que os dados sejam transmitidos seguramente, portanto que os elos de comunicação estejam protegidos criptograficamente. Mecanismos de criptografia devem ser certificados e expostos a uma equipe de atacantes tão sagaz quanto possível antes de serem implementados. Existem técnicas de penetração que se baseiam por exemplo na probabilidade das letras e palavras da língua inglesa (letra E=13%, palavra THE=4,5%, sufixo ING=1,2%). Mesmo métodos tradicionais de decifragem arqueológica e militar podem ser empregados para atravessar uma barreira criptográfica. Conhecedores do perigo resta nos agora procurar as salvaguardas de proteção.

IX.4 SISTEMAS OPERACIONAIS PARA REDES

Uma rede de computadores é criada para suprir aos seus usuários serviços melhores e mais confiáveis do que qualquer de seus componentes. Grande disparidade existe frequentemente entre as características fundamentais e as provisões de segurança desses componentes: computadores, terminais, protocolos de comunicação. Padronização de um único tipo de terminal ou computador não é só impossível como indesejável, porque cada usuário tem necessidades específicas, grande quantidade de dinheiro investidas nos componentes atuais e qualquer padronização rapidamente se tornaria obsoleta. Dessa forma, o usuário que deseja fazer pleno uso da rede, precisaria aprender as linguagens de cada sistema em separado e mais os comandos específicos para comunicação da rede.

Um sistema operacional para rede (NOS) é o comumente visto como um mecanismo que mascara as diferenças entre os diversos sistemas acoplados (KiM 76). Seus objetivos fundamentais são dar suporte e simplificar o acesso aos serviços existentes e produzir a construção e o acesso a novos serviços, independentemente da sua estrutura lógica ou localização geográfica (fig. 9.4).

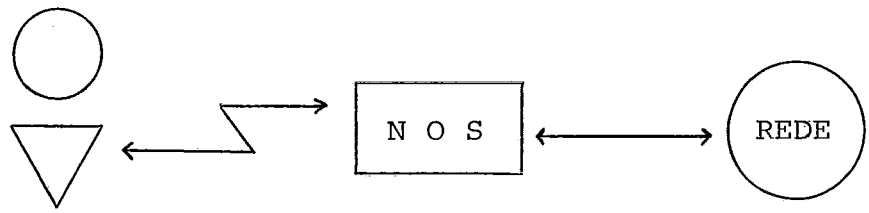


Fig. 9.4 Visão do usuário da rede

Basicamente existem dois problemas a considerar na construção de um sistema operacional para rede:

. COMPATIBILIDADE - se refere à heterogeneidade dos componentes que são conectados. Frequentemente sistemas que antes eram independentes começam a pertencer à rede e os seus softwares e bancos de dados que foram projetados sem considerações de rede devem continuar a existir. Através da rede começam a existir diferenças consideráveis de estruturas de diretórios, primitivas de acesso a arquivos, etc., tornando difícil a usuários de uma máquina acessar recursos de outra.

. DINÂMICA - É irracional esperar que uma grande rede de computadores se torne estática por um longo período de tempo. Além disso, falhas de componentes, conecção e remoção de máquinas, mudanças no equipamento de comunicação e de software, etc., devem ser previstos.

Sendo assim, um sistema operacional para rede deve ser altamen

te modular, gerando respostas localizadas às modificações ocasionais. O sistema deve também prover uma interface que seja objetiva o suficiente para encorajar usuários a acessar recursos remotos da maneira mais semelhante possível à usada para recursos locais.

Kimbleton e Mandell (KiM 76) definiram 4 aspectos globais que necessitam funções primitivas num sistema operacional para redes: comunicação entre usuários, migração de dados, execução de jobs e controle.

COMUNICAÇÃO ENTRE USUÁRIOS

O sistema de comunicação desempenha 2 tarefas: processamento e teleconferência. A primeira consiste em conduzir as mensagens corretamente da sua fonte ao seu destino. A complexidade da sua operação depende da topologia da rede. Redes organizadas em estrela, anel ou completamente conectadas são triviais, porém redes parcialmente conectadas necessitam sofisticados algoritmos de roteamento. O mecanismo de processamento de mensagens pode incluir: criação, coordenação, emissão, aviso e processamento de evento. Teleconferência pode incluir comunicação vocal, visual e até uma coleção de facilidades para comunicação simultânea de vários usuários.

MIGRAÇÃO DE DADOS

Migração de dados é a capacidade de acessar e transmitir dados remotos. Arquivos inteiros ou simples blocos de dados podem ser transmitidos. Só deve ser realizada a migração se o proprietário dos dados tiver autorizado explicitamente. A migração deve ser suportada por 3 funções:

. Seleção de registros - provê o acesso ao dado remoto ao nível de sub-arquivo. Pode ser visto como um processador de transação que atende pedidos de um processador remoto.

. Translação de registros - permite que computadores heterogêneos, com modelos diferentes de bits para codificar a informação, se "compreendam". O formato interno de todos os tipos de dados deve ser conhecido para cada computador da rede.

. Transformação de registros - um conjunto de operações lógicas, aritméticas e de caracteres que reestruturam a forma lógica do dado para atender aos requisitos de proteção da informação sensitiva. Exemplo: o proprietário de um arquivo pode impor a condição de que o campo PARTICULAR não seja transmitido, nesse caso, o registro é transformado antes da transmissão. Na ARPANET, uma pequena transformação é realizada pelo Serviço de Reconfiguração de Dados (DRS).

EXECUÇÃO DE JOBS

A execução de jobs numa rede é diferente da execução de jobs em um computador isolado. Isto porque pode necessitar a execução concorrente de steps paralelos, migração de steps para lados alternativos, sincronização entre steps nos vários hosts e geração de

JCL da rede, além de procedures ou dados remotos, interação com usuários remotos e comunicação entre processos remotos. Essa última função, comunicação entre processos remotos é a característica mais importante de um sistema operacional para rede. Metcalfe disse: "formar uma rede é estabelecer um mecanismo de comunicação entre os processos dos seus componentes" (ABS 74). Uma vez que os processos são as únicas entidades ativas em um sistema de computador, a comunicação interprocessual (IPC) é o alicerce que suporta a comunicação entre computadores.

Como no caso de sistemas isolados, existem duas formas básicas de comunicação interprocessual numa rede: por dados compartilhados e por mensagens. Entretanto, a separação física dos processos sugere mais o mecanismo de mensagens. Mensagens podem transportar a qualquer distância e em qualquer topologia, dados, informações de controle, parâmetros, e uma infinidade de outros objetos, inclusive capabilities, como no sistema de Watson e Fletcher (WaF 80).

No Experimental Network Operating System (KWF 78) o mecanismo de IPC é uma aproximação de nível mínimo de comunicação e requer muito pouco software de cada hospedeiro para seu suporte. Consiste de conjuntos chamada-retorno análogo aos de chamadas de subrotina, em que cada processo entra em estado de "espera" logo após emitir a chamada. Esse esquema simplificado é possível porque os jobs são supostamente interativos e as mensagens podem ser transmitidas do canal controlador do teletipo para a destinação apropriada onde o processo reverso ocorre. Embora seja um mecanismo limitado, com banda de passagem entre processos restrita, é fácil de implementar em qualquer modelo de hospedeiro e de baixo custo. Uma forma de IPC em redes que utiliza criptografia (PoK 79) está esquematizada na figura 9.5.

CONTROLE

Todas as funções descritas acima estão agrupadas em 3 níveis de controle:

- CONTROLE DA SUBREDE - limitado à topologia e à capacidade das linhas. É geralmente estático, a constante de tempo é da ordem de meses, embora esteja emergindo uma necessidade de um controle mais dinâmico a esse nível.
- CONTROLE DO HOSPEDEIRO - consiste das estratégias de scheduling, gerenciamento de software de arquitetura básica, etc., comumente desempenhado por programadores de sistema através de uma "sintonização" com as necessidades da instalação.
- CONTROLE DA REDE - é a soma dos dois controles anteriores e pode ser de duas formas: centralizado ou distribuído. Controle centralizado significa que existe um nó especial que detém a maior parte dos mecanismos de supervisão e proteção da rede. Vários níveis de centralização podem ser encontrados, desde o mais despótico, que controla o uso de cada recurso em cada nó, até o mais democrático, que dá ampla liberdade aos seus controlados. Quando o controle é distribuído todos os nós da rede possuem a mesma autoridade e uma cópia das funções básicas do sistema operacional e o sistema de comunicação deve prover mecanismos que incluam as informações de proteção nos próprios protocolos.

Independentemente de ser centralizado ou distribuído o software que realiza o controle da rede pode estar localizado em um ou mais computadores hospedeiros da rede ou em um ou mais computadores especificamente introduzidos na rede com essa finalidade. O primeiro método se chama CASO NÃO-AUMENTATIVO pois o número de componentes da rede permaneceu o mesmo, tendo apenas "ampliado" suas funções. O segundo método se chama CASO AUMENTATIVO pois foram incluídos na rede outros elementos para realizar as funções de controle e proteção.

Enquanto a pesquisa está apenas se iniciando na área de NOS é importante garantir que os requisitos de segurança e integridade de dados sejam bem especificados e sejam incluídos mecanismos de proteção em cada projeto. Com isso as versões subsequentes de NOS já serão portadoras desses mecanismos e não se terá que enfrentar a desagradável situação de "segurança retrospectiva" como foi para muitos sistemas isolados.

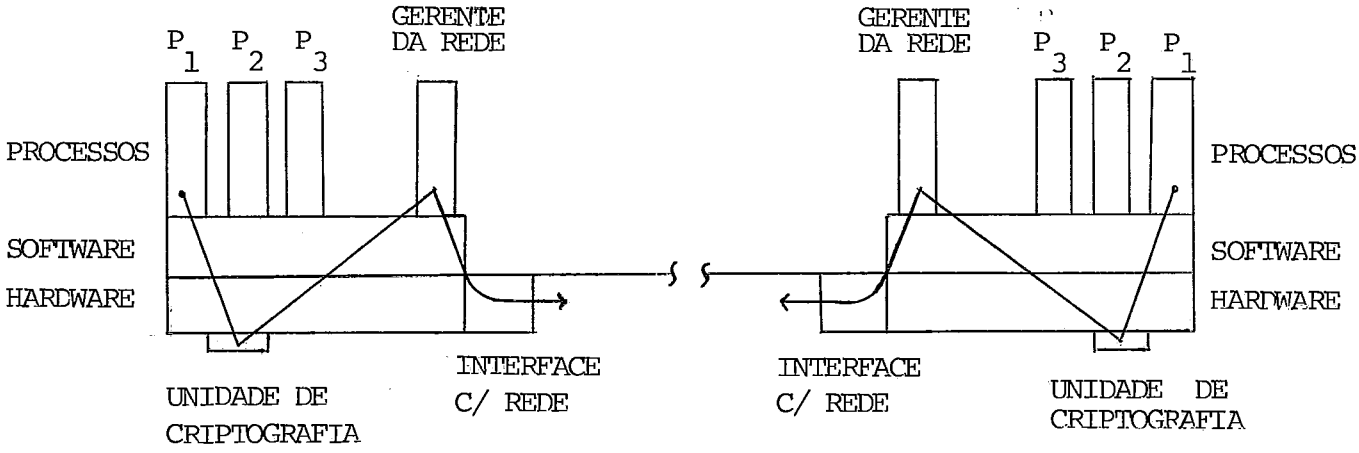


Fig. 9.5 - Fluxo de dados em canais criptografados processo-a-processo

IX.5 CONTROLE CENTRALIZADO

Pequenas redes podem ser controladas centralizadamente por um dos nós. A topologia não importa, entretanto, uma vez que é escolhido um nó para ser o responsável lógico pela rede é útil que se construam caminhos diretos entre este e cada componente. Tratando-se de proteção, mecanismos centralizados são mais simples e mais econômicos pois são apenas uma extensão dos mecanismos normalmente utilizados nos sistemas independentes. Para a maioria deles, a localização física do sujeito e objeto é indiferente, pois esses são conhecidos através de tabelas centrais e depois contactados via algoritmos de routing e protocolos de comunicação (WiD 74). Um mecanismo pode ser do tipo aumentativo, se foram incluídos componentes para realizar as funções de proteção ou não-aumentativo, se a topologia não se alterou.

CASO NÃO-AUMENTATIVO

Um exemplo de caso não-aumentativo é o Distributed Processing Executive (DPPX) da IBM (Kie 79). É um sistema de processamento distribuído projetado para gerenciar uma rede formada por vários mini e microprocessadores, constituindo o Sistema de Informação 8100. A rede pode estar ou não geograficamente dispersa, sendo supostamente formada apenas por componentes do sistema 8100 ou do sistema /370. Existe um gerente lógico que coordena a distribuição das funções e dados, e um software de compatibilização de dados entre os elementos 8100 e /370.

Apenas uma definição de cada recurso é necessária para toda a rede. Um profile de cada usuário do sistema é mantido centralmente. Uma vez que um usuário tenha sido autenticado e o recurso que solicita esteja definido, o controle central consulta suas tabelas para tomar a decisão de acesso. Um dos componentes do software de controle é o Administrador de Catálogo, responsável pelas decisões de criação, modificação e deleção de arquivos de usuários. Esses catálogos são diretórios especiais que realmente contém os arquivos e seus descritores, organizados em árvores e protegidos por chaves e passwords. Cada usuário pode ser proprietário de um catálogo, em que define seus arquivos, insere restrições de acesso e mantém controle rígido sobre estes (fig 7.9). A maior vantagem desse mecanismo é a unicidade das definições de objetos, que sendo geral para toda a rede, permite que estes sejam transportados a qualquer parte do sistema 8100. Por exemplo, o mesmo arquivo pode ser lido em qualquer nó da rede. Isto só é possível através de um controle centralizado. A simplicidade e eficiência desta aproximação se deve principalmente à homogeneidade da rede, constituída apenas por elementos de tecnologia IBM.

CASO AUMENTATIVO

O Centro de Segurança da Rede (NSC) é um esquema proposto pelo NBS que exerce as funções de controle de acesso normalmente desempenhadas pelo sistema operacional da rede, podendo trabalhar em combinação com este. O NSC consiste de um computador, um mini ou um conjunto de minis que tenham probabilidade mínima de erros e acesso completamente restrito. Além de prover a criação de trajetórias se

guras de acesso aos recursos de uma rede, o NSC deve gerar e distribuir a informação apropriada relativa ao uso desses recursos (Bra77).

Um NSC desempenha serviços de segurança para uma rede da mesma maneira que uma agência desempenha serviços para empresas clientes. Numa rede, existem serviços e restrições de acesso a estes. Usuários possuem credenciais e autorizações de acesso. O propósito do NSC é coletar e comparar essa informação centralmente, realizando as decisões de acesso.

As funções do NSC podem ser listadas como segue:

. identificação e autenticação de usuário
. identificação e autenticação de terminal e computador
. manutenção de perfis de usuário
. manutenção de especificações de acesso a serviços e dados
. decisão de acesso
. geração de chaves de criptografia
. geração de históricos de acesso concedido e recusado

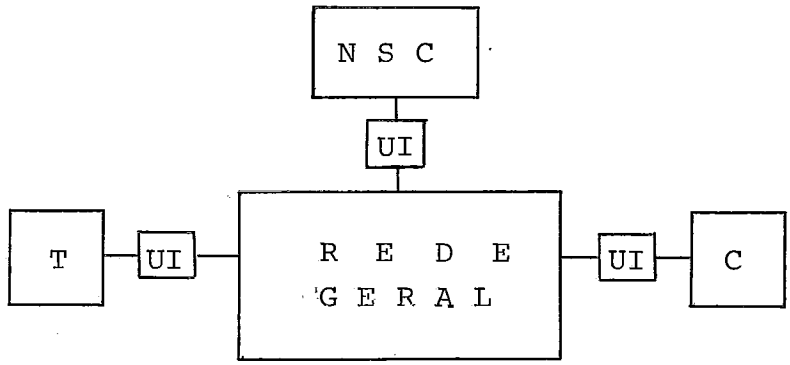


Fig. 9.6 - Configuração de uma rede com NSC (WoK 79)

A fig. 9.6 apresenta a configuração de uma rede em que um terminal T deseja acessar serviços do computador C. As unidades de interface (UI) são dispositivos criptográficos inteligentes que devem intermediar qualquer componente com a rede. O processo de autenticação é ligeiramente diferente do normal. A chave de autenticação do usuário é a própria chave de criptografia e nunca é transmitida pela rede. É suposto que o usuário está autenticado se sua mensagem pode ser corretamente cifrada, transmitida e decifrada pelo NSC. Um usuário autorizado só precisa fornecer a sua identificação, sua chave de acesso e o tipo de serviço que necessita.

A sequência de ações realizada no estabelecimento de uma conexão é a seguinte:

1. Identificação e autenticação do terminal - Assim que o terminal é ligado fisicamente, envia ao NSC seu código de identificação que vai criptografado pela sua chave de codificação. O NSC recebe, decodifica e envia uma mensagem de OK que vai codificada com a mesma chave do terminal.

2. Identificação e autenticação do usuário - Pelo mesmo processo anterior, o usuário se identifica ao NSC e recebe dele um OK.

3. Autorização de acesso - O usuário envia seu pedido ao NSC, que pesquisa nas suas tabelas, toma a decisão de acesso e avisa ao usuário que vai ligá-lo com a rede.

4. Ligação do terminal com a rede - O NSC fornece ao terminal uma chave para comunicação com a rede.

5. Identificação e autenticação do computador - O NSC localiza o computador solicitado (ou um que possua o recurso solicitado) e pede sua identificação, autenticando-o.

6. Ligação do computador com a rede - O NSC põe o computador em contato com a rede fornecendo-lhe a mesma chave que foi enviada ao terminal (se a cifragem for assimétrica, deve ser fornecida a chave de decifragem).

7. Comunicação entre usuário e computador - O usuário acessa o recurso do computador.

O NSC também pode realizar as funções de adicionar informação de auditoria no fim de cada pedido, quer tenha sido atendido, quer recusado. Em uma rede grande e dispersa podem haver vários NSC's funcionando regionalmente e cooperando nas decisões globais, que podem também agir como back-ups uns dos outros.

Finalizando, vamos relacionar algumas vantagens de se utilizar mecanismos de proteção de controle centralizado para redes de computadores:

- permitem balanceamento na utilização dos recursos da rede, pois supervisionam centralmente o trabalho em cada nó.
- aumentam a eficiência de cada componente, liberando-o das funções de mapeamento, routing, etc.
- simplificam a manutenção preventiva e a deteção de erros.
- diminuem o tempo de resposta médio da rede, pois a maioria das respostas são obtidas das tabelas do nó central.
- facilitam a migração de sujeitos e objetos de um nó para outro da rede.
- diminuem o número de pessoas responsáveis pelo funcionamento e proteção na rede.
- simplificam a modificação e expansão das capacidades da rede.
- permitem controle mais acurado de cada componente e no desempenho da rede como um todo, podendo fornecer estatísticas, históricos, etc.

Entretanto, em redes muito grandes o controle central é uma limitação que impede alguns elementos de exercitarem todas as suas flexibilidades. Isto explica a sugestão do NBS de criar NSC's regionais. Mas esta já é uma forma de CONTROLE DESCENTRALIZADO.

IX.6 CONTROLE DESCENTRALIZADO

Controle descentralizado é realizado quando as decisões de acesso são tomadas de forma descentralizada em toda a rede, e normalmente no próprio nó onde reside o objeto solicitado. Em redes muito grandes ou muito dispersas é interessante que o controle seja descentralizado, por razões de desempenho e economia de recursos. Também aqui podemos ter caso aumentativo e não-aumentativo.

CASO NÃO-AUMENTATIVO

Um exemplo de controle descentralizado de caso não-aumentativo é a ligação feita entre o KSOS e uma rede multinível (CaD 79).

Interno ao sistema operacional, logo acima do núcleo, fica o serviço do Transmission Control Protocol (TCP) que traduz em mensagens para a rede multinível as mensagens geradas pelos usuários do sistema. A única restrição é que o software do Network Daemon seja privilegiado, para poder manusear vários níveis para dentro e para fora da rede. O resto do TPC pode ser não-privilegiado, pois atua em um só nível.

Daemon é um processo do sistema usado como buffer para as mensagens que foram enviadas a componentes não conectados, armazenando a "correspondência" e enviando-a quando o nó for ativado (PoK 79). O Network Daemon manuseia o protocolo da rede, separa a cadeia de dados em vários níveis e envia cada nível, separadamente à parte do TCP interna ao Emulador. As duas partes do TCP se comunicam usando os dois mecanismos de IPC providos pelo núcleo: o segmento compartilhado é usado para passagem dos dados e o mecanismo de evento é usado para sincronização e para comandos entre os dois (fig. 9.7).

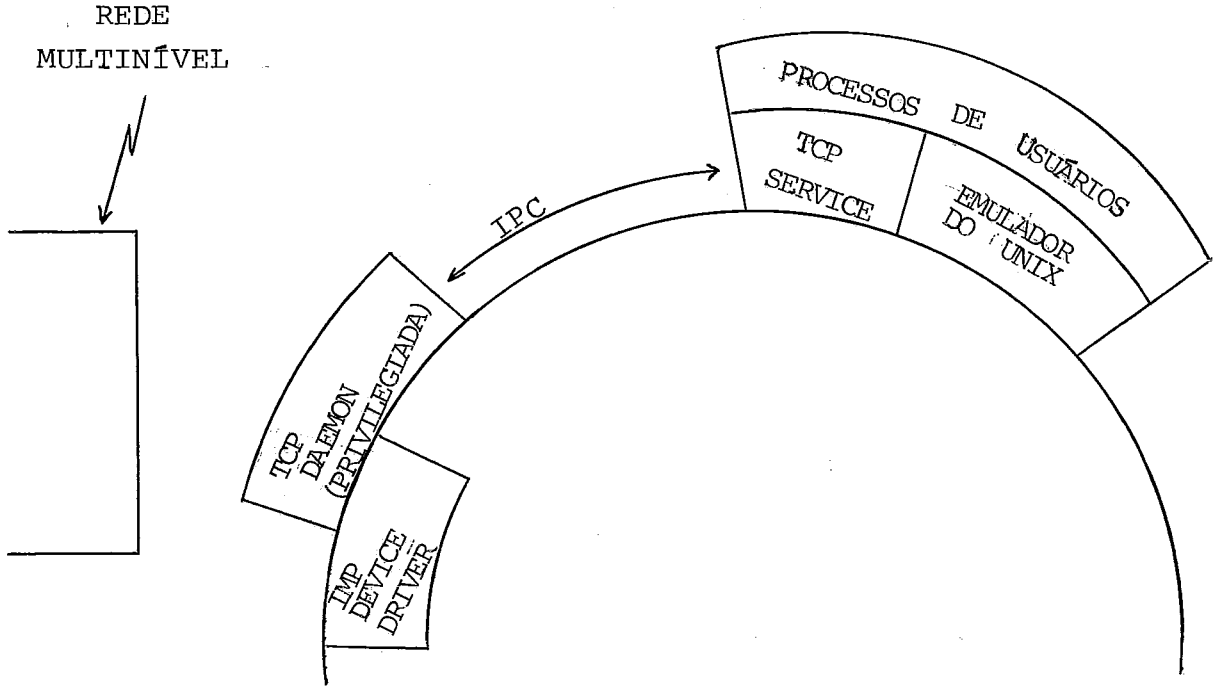


Fig. 9.7 - Conexão entre KSOS e rede multinível (CaD 79)

Como se vê, as decisões de acesso aos recursos internos do nó são tomadas pelo próprio núcleo do KSOS, não importa de que parte da rede venha a solicitação. O KSOS provê também as funções de um sistema operacional de rede, como gerência de protocolos, etc., eliminando a necessidade de qualquer controle central.

CASO AUMENTATIVO

Dezenas de exemplos de controle descentralizado do caso aumentativo podem ser enunciados. Um extenso conjunto de aplicações voltadas para esse caso existe na literatura, embora uma boa parte não tenha sido realmente implementada. Vamos apresentar rapidamente 5 dessas aplicações: front-end, front-door, mini-host, guarda e interfaces protegidas.

FRONT-END (FE) é um computador separado, intimamente acoplado ao hospedeiro, que é interposto entre este e a rede com o objetivo de descarregá-lo tanto quanto possível das funções de controle e comunicação interprocessual com a rede. Front-ends realizam muitas funções diferentes e requerem quantidades variáveis de suporte de segurança dos sistemas seguros. Alguns necessitam muita codificação privilegiada e além disso rodam na interface do núcleo do sistema operacional hospedeiro. Por exemplo, um front-end para um sistema de controle militar que roda sobre o núcleo do KSOS está sendo pesquisado pela Ford Aerospace and Communications Corporation (PBN79).

Também a Honeywell desenvolveu um FE para manusear múltiplos níveis de terminais conectados ao sistema MULTICS. Este FE tinha que multiplexar cadeias de dados entre os vários terminais e o MULTICS e mais tarde foi empregado em outros sistemas (Woo 79).

A estratégia de ligação de um FE pode ser rígida ou flexível. É rígida quando o FE simula um ou mais dispositivos conhecidos pelo hospedeiro, exemplo: terminais como no caso do MULTICS. É flexível quando um protocolo compacto deve ser interpretado no FE e no hospedeiro, uma aproximação melhor porque se aplica mais aos objetivos de uma rede.

A maior vantagem do FE se refere ao CONFINAMENTO. Pode acontecer que um software malicioso do hospedeiro estabeleça um canal coberto de comunicação com algum software malicioso de comunicação da subrede. Entretanto, interpondo-se um FE entre a rede e cada componente inseguro, isto é suficiente para assegurar os propósitos de segurança. Não é necessário que o sistema operacional do hospedeiro seja verificado para que as trajetórias ilícitas de dados sejam bloqueadas. O preço que se paga por esta segurança é uma perda no desempenho geral da rede causada pelo overhead incluído pelos FE's.

FRONT-DOOR (FD) é uma aplicação semelhante ao FE, com uma diferença potencial significante: em vez de descarregar o software da rede com o propósito de realizar essas funções gerais, o FD é interposto entre o hospedeiro e a rede com o único propósito de permitir que uma dada aplicação do hospedeiro seja colocada à disposição de usuários remotos. A aplicação mais imediata do FD se refere aos bancos de dados pré-existentes, como explicaremos a seguir.

É axiomático que segurança não pode ser "adicionada" a sistemas existentes, entretanto as redes futuras tornarão possível a usuários em vários níveis de segurança acessar sistemas existentes que rodam importantes subsistemas de aplicações, particularmente bancos de dados. Portanto, se um FD pode conferir a esses usuários meios de acessar seletivamente a informação sem comprometê-la, será muito mais eficiente do que prover "mudança de cores" nos hospedeiros. Deve ser notado, entretanto, que o grau de confiança é discutível, porque um hospedeiro "preparado" pode mentir ao seu FD sobre que saída é destinada a qual usuário comprometendo a informação por um canal aberto, mais perigoso que o coberto exposto no caso do FE. Pode-se evitar esse problema relegando o hospedeiro a uma atitude simplesmente passiva, no sentido de ser impedido de realizar as conecções lógicas iniciais (PBN 79).

MINI-HOST (MH) é o nome dado a um sistema essencialmente dedicado a suportar usuários de terminal, cujo trabalho computacional real é realizado em algum hospedeiro, localizado em qualquer parte da rede. O conceito do MH é familiar às redes existentes, particularmente à ARPANET, onde o TIP (Terminal Interface Processor) realiza suporte de terminal exatamente como um MH.

A motivação para o MH é essencialmente econômica: é muito mais barato espalhar TIP's pelo país do que PDP10s, H6180s, ou outros computadores. As mesmas considerações se aplicam para redes seguras, mas existe um princípio arquitetural que deve ser considerado, que afeta basicamente as políticas multiníveis: o fato de todos os acessos ativos serem forçados a se juntar num único ponto causando o encontro de múltiplos níveis de segurança. É necessário que haja uma sintonização a uma granularidade bem específica, tanto em relação aos terminais como aos níveis de segurança. Uma alternativa seria reunir apenas os terminais de mesma "cor" criando um MH para cada nível de segurança (PBN 79).

GUARDA é o nome dado a um minicomputador que estabelece a conexão entre dois computadores de diferentes características de segurança, protegendo as informações. É aplicado principalmente em redes multiníveis, e realiza basicamente 2 funções: supergradação e sanitização.

Consideremos, por exemplo, dois computadores, HIGH e LOW de diferentes níveis de segurança. O computador de guarda é colocado entre os dois e realiza a função de supergradação nos dados que vão do LOW para o HIGH e de sanitização nos dados que vão do HIGH para o LOW.

Para a segurança do sistema ser mantida, a função de sanitização é a mais importante. Sanitização é geralmente realizada pela remoção de fontes (sensores, etc.) dos dados e pela redução da sua precisão. Existem correntemente vários métodos para se realizar a sanitização. O mais simples envolve o trabalho de um funcionário de segurança que utiliza um terminal para verificar a informação que será transmitida do HIGH para o LOW. Formas mais automatizadas envolvem uso de tabelas, dígitos de segurança e informações embutidas nos dados especialmente para serem utilizados nessa situação (Woo79).

Finalmente, microcomputadores tem sido usados com grande propriedade como INTERFACES PROTEGIDAS em redes de computadores especialmente para mascarar diferenças de sistemas e de linguagem entre os computadores de uma rede heterogênea. Por exemplo, uma obscura e incompreensível mensagem pode ser reconhecida e traduzida em alguma coisa significativa para um usuário inexperiente. No sentido oposto uma mensagem de um usuário expressando apenas superficialmente o que ele deseja pode ser convertida por um microprograma protegido em uma sequência de comandos de controle requisitados por um sistema operacional particular. Interfaces protegidas em microcomputadores foram desenvolvidas para operar em uma rede complexa de computadores CDC no Imperial College em Londres (JaI 80). A forma dessas interfaces tem muito a ver com as aplicações conhecidas de aprendizado assistido por computador. A estratégia básica é obter informações do usuário através de um conjunto de perguntas que podem ser respondidas em formato livre e não necessitam acesso a nenhuma documentação. O processador local interage com o usuário, determina os pedidos que devem ser feitos à máquina remota e automaticamente gera o protocolo necessário para o serviço solicitado, sem que o usuário necessite saber qualquer coisa sobre o sistema operacional da máquina remota. Estas interfaces podem também ser microprogramadas para tomar decisões de proteção em relação aos componentes da rede, implementando quaisquer políticas de acesso que queira.

X - RESUMO E CONCLUSÕES

X.1 RESUMO

Mecanismos de proteção de sistemas de computador podem ser decompostos em 3 partes fundamentais: OBJETOS, que são os recursos do sistema, SUJEITOS ou entidades ativas que solicitam acesso aos recursos do sistema e REGRAS DE ACESSO que são leis que governam as relações entre sujeitos e objetos.

MECANISMOS DISCRECIONÁRIOS são aqueles em que as regras de acesso são flexíveis, de maneira que o acesso aos objetos pode ser definido à discreção (ao escrutínio) dos seus proprietários. Privilégios podem ser concedidos a outros sujeitos em diferentes modos de acesso e o controle está descentralizado, expandido por todo o sistema. Exemplos de mecanismos discrecionários são as CAPABILITIES ou 'tickets' para objetos, as LISTAS DE ACESSO ou listas de controle de acesso específicas para cada objeto e os mecanismos de CHAVE-FECHADURA, um misto dos dois primeiros.

MECANISMOS NÃO-DISCRECIONÁRIOS não permitem especificação das regras de acesso à discreção dos usuários, mas estas são rígidas e previamente determinadas. Dentre estes, o mais comum é o mecanismo multinível empregado nas áreas militares em que a cada informação do sistema é atribuído um NÍVEL DE SEGURANÇA para os quais os tipos de acesso estão pré-fixados. Embora menos flexíveis, esses mecanismos pressupõem maior segurança pois permitem controle de todos os FLUXOS DE DADOS, não só explícitos como também implícitos.

Dentre os recursos do sistema (objetos), o mais raro é a memória. Do ponto de vista de proteção também um ênfase especial deve ser dado aos OBJETOS DE MEMÓRIA, pois estes são os receptáculos da informação, de sua segurança depende a segurança do conteúdo. Inúmeras técnicas de PROTEÇÃO DE MEMÓRIA tem sido pesquisadas: ARQUITETURAS ETIQUETADAS, MICROPROGRAMAÇÃO, CRIPTOGRAFIA. Construções de alto nível como ARQUIVOS e DIRETÓRIOS visam não só organizar o acesso aos dados armazenados, como também criar mecanismos para sua proteção.

PROCESSOS são, em muitos casos, as únicas entidades ativas do sistema (sujeitos). O campo de ação de um processo, ou seja, o escopo de objetos que pode acessar no sistema é chamado seu DOMÍNIO DE EXECUÇÃO. Mecanismos de proteção são empregados na definição, geração e utilização dos domínios de execução de processos, que podem ser alterados ou substituídos mesmo durante a sua atividade. SEGMENTOS DE DESCRITORES e LISTAS DE CAPABILITIES são os tipos mais comuns de domínios. ANÉIS DE PROTEÇÃO são os múltiplos domínios de execução de um processo, que lhe conferem maiores ou menores privilégios. Além disso, ao executarem, processos precisam se COMUNICAR com outros processos, solicitando ou prestando COLABORAÇÃO, quer para uma divisão eficiente dos recursos físicos da máquina, quer para executar tarefas referentes ao mesmo job. Técnicas de comunicação devem prevenir o escoamento de dados privativos de cada processo, criando um ambiente em que haja CONFINAMENTO.

Construir um sistema operacional seguro é uma tarefa sem dúvida gigantesca. Garantir que em um software com dezenas de milhares de linhas de codificação todas as regras de acesso estão sendo atendidas, não há fluxos de dados incorretos, portas falsas nem armadi-

lhas é praticamente impossível. Para isso foram criados os NÚCLEOS DE SEGURANÇA, uma parte do sistema operacional, tão pequena quanto possível que contenha toda a codificação necessária para a implementação de mecanismos de proteção. Por ser pequena é simples, específica e certificável. Núcleos de segurança podem utilizar mecanismos discrecionários e/ou não-discrecionários e sua evolução histórica caminhou em 2 polos: os NÚCLEOS DISCRECIONÁRIOS que tiveram muita dificuldade em serem realmente pequenos e os NÚCLEOS NÃO-DISCRECIONÁRIOS que implementaram com sucesso a política multinível militar.

BANCOS DE DADOS são grandes quantidades de dados em que o acesso para leitura é muito mais comum que o acesso para alteração. Além disso, os dados possuem maior granularidade, várias trajetórias de acesso e várias restrições diferentes para o mesmo tipo de acesso de leitura, que pode ser dependente do ambiente, do conteúdo ou do histórico de outros acessos. Desta forma, mecanismos de proteção normais do sistema operacional não cobrem todas as suas necessidades e foram desenvolvidos mecanismos específicos para bancos de dados. MODELOS DISCRECIONÁRIOS e NÃO-DISCRECIONÁRIOS que geraram os mecanismos conhecidos foram transportados e ampliados para Bancos de Dados. Foram desenvolvidos mecanismos para BANCOS DE DADOS RELACIONAIS e mecanismos para BANCOS DE DADOS ESTATÍSTICOS, que se afeiçoassem às suas características e correspondessem às expectativas em matéria de proteção.

Finalmente, em SISTEMAS DISTRIBUÍDOS existem 3 conjuntos que necessitam proteção: os sistemas de terminais, as linhas de comunicação e os hospedeiros. Técnicas de AUTENTICAÇÃO dizem respeito ao primeiro conjunto, algoritmos de CRIPTOGRAFIA ao segundo e SISTEMAS OPERACIONAIS PARA REDES ao terceiro. Neste último, mecanismos de proteção podem agir sob CONTROLE CENTRALIZADO ou sob CONTROLE DESCENTRALIZADO, sendo que os parâmetros de escolha giram em tôrno da dimensão da rede, da homogeneidade dos seus componentes e do grau de segurança desejado.

X.2 CONCLUSÕES

Procrustes era o torturador antigo que forçava suas vítimas a se encaixarem numa cama de ferro, esticando-as ou cortando-lhes os pés até que atingissem a medida exata desejada. Muitas vezes sistemas de computadores agem como verdadeiras camas de Procrustes e os usuários tem que se adaptar às limitações que lhes são impostas, expondo informações de conteúdo sensitivo.

Três limitações básicas são encontradas na utilização de mecanismos de proteção: físicas, econômicas e humanas. FÍSICAS se referem às restrições do hardware, dos equipamentos, etc. Dois aspectos podem ser considerados: a vulnerabilidade às falhas probabilísticas dos seus componentes e a morfologia inconsistente entre hardware e software. Técnicas de deteção e localização de erros, redundância, reconfiguração, etc, dizem respeito ao primeiro aspecto. Quanto ao segundo, vimos que há sistemas (PSOS, etc) que nunca foram implementados por falta de hardware adequado ou que construiram hardware especificamente para seus propósitos (SIGMA, etc). Na maioria dos sistemas o hardware não provê a base necessária para a implementação dos mecanismos escolhidos, como os anéis de proteção, que durante anos foram simulados em software aguardando a construção do hardware apropriado. Mudanças radicais devem ser realizadas na arquitetura do computador, para serem a ponte sobre o abismo semântico entre "conceitos de liguagem de programação" e "conceitos de linguagem de máquina". Assim como desceram ao núcleo do sistema operacional, os mecanismos de proteção devem descer ao âmago do hardware.

Limitações ECONÔMICAS são óbvias: a implementação de um sistema de proteção flexível é mais cara do que a de um sistema de proteção rígido e limitado. Um mecanismo baseado em capabilities necessita mais suporte de hardware e software do que todo o necessário para implementar um núcleo multinível. Limitações econômicas levam, por exemplo, à concessão de privilégios excessivos ao sistema operacional. Ou à criação de mecanismos de proteção ineficientes, como os grandes segmentos que empacotam juntos dados de várias qualificações de segurança, alternativa mais barata do que proteger individualmente cada palavra. Problemas financeiros podem também causar a suspensão de importantes fontes de pesquisas como o sistema CAL, depois de 3 anos de investimentos deixou de ser concluído por falta de recursos. O próprio mercado se retrai a certas inovações mais caras pois o trauma que foi a mudança de 2a. para 3a. geração de computadores ainda está na memória de muita gente.

Problemas físicos e econômicos são difíceis de resolver, mas há sempre uma forma de se estabelecer um equilíbrio razoável entre eles. Entretanto, a terceira limitação dos mecanismos de proteção é a maior de todas: DEPENDÊNCIA HUMANA. Estar sujeitos, como todas as máquinas, ao controle humano, significa estar sujeitos à vulnerabilidade humana, justamente o motivo pelo qual foram construídos. Levando-se em conta que vivemos numa sociedade de consumo e que grandes somas de dinheiro são comumente envolvidas em qualquer transação por computador, o problema se torna muito mais sério. É uma questão filosófica, técnica e social, que, infelizmente vem sendo resolvida como na medicina primitiva: primeiro a doença depois o

remédio. É fundamental que se avance no tempo em matéria de proteção, criando vacinas que se antecipem aos virus eliminando a probabilidade da doença ou tornando-a de pouco efeito. Muito tem sido feito na área de profilaxia, mas todos esses mecanismos analisados ainda são por demais dependentes das mãos humanas.

Alimentação de dados pessoais, modificação de informações, alteração de privilégios, estabelecimento de parâmetros, tudo depende do trabalho e da decisão humanos. Pode-se tentar tornar o sistema tão autônomo quanto se queira, mas isso seria viável? Seria econômico? Seria moral?

Acreditamos firmemente que não. Os perigos de um sistema excessivamente automático são maiores do que as vantagens que pode trazer, que nos diga o 'Big Brother'. Um sistema fechado ou autosuficiente também pode ser penetrado por pessoas inescrupulosas, e nesse caso, os danos são incalculáveis. É necessário que haja aberturas para controle em vários níveis, descentralizando a responsabilidade. Mecanismos de proteção são como um bisturi nas mãos de um médico ou uma arma nas mãos de um soldado: seu benefício depende da forma em que são utilizados. Daí a importância do estudo relativo ao perfil psicológico do profissional de computador, que vem sendo realizado com interesse (Beq 78, Ham 72).

Um outro aspecto que envolve as 3 limitações descritas acima diz respeito ao acompanhamento e fiscalização dos mecanismos de proteção, especificando, à sua AUDITORIA. Militarmente se diz " ordem dada, ordem não fiscalizada, ordem não cumprida". Excelentes técnicas de auditoria existem e estão sendo desenvolvidas para fiscalizar os mecanismos e seus responsáveis (BuS 78, Dew 78). Mesmo projetos certificados podem intencional ou inadvertidamente sofrer erros de parametrização e implementação, deixando falhas que os sagazes usuários acabam descobrindo. A auditoria sistemática é indispensável e as procedures de auditagem devem ser tanto quanto possível incluídas no projeto do sistema. Um bom projeto deve antecipar as necessidades de auditoria e controle de irregularidades de modo que violações detetadas tenham efeitos de propagação limitados e sejam dotadas de rápidos meios de recuperação.

A manutenção contínua de um sistema seguro depende de todo o seu hardware e software, incluindo conceitos de arquitetura tolerante a erros e software confiável. Depende também de um ambiente operacional apropriado. Requer planejamento e gerência globais. Um CONTROLE DE RISCO deve ser implementado, para que se tenha consciência das vulnerabilidades do sistema. Técnicas de gerência de risco voltadas para sistemas de computador existem e estão sendo criadas (Sch 78, Pri 78).

Finalmente, acreditamos que segurança é uma meta possível de ser atingida. Uma vez que seja definido o grau de proteção desejado e sejam alocados recursos financeiros e potencial humano, pode-se obtê-la. Risco e perigo existirão sempre, mas a probabilidade de ocorrência pode ser feita tão baixa que permita um sono tranquilo aos gerentes de segurança.

A P Ê N D I C E A

SISTEMAS DE HARDWARE CONCENTRADO

A.1 ADEPT-50

O ADEPT-50 (Wei 69), da Systems Development Corporation, é um sistema de time-sharing implementado em hardware IBM/360-50.

É um sistema pioneiro em vários aspectos. Primeiro, implementa, em 1969, as idéias não-discrecionárias de níveis de segurança, que mais tarde foram adotadas, com algumas modificações, pelo Departamento de Defesa dos Estados Unidos. Segundo, estabeleceu uma separação clara entre CODIFICAÇÃO de segurança e DADOS de segurança, identificando os mecanismos e possibilitando cuidadoso escrutínio quanto à confiabilidade. E terceiro, optou por uma solução simples e prática, por razões tanto de desempenho como de custo.

Quatro tipos de objetos de segurança são considerados: usuário, terminal, job e arquivo. Cada objeto é descrito em um PROFILE de segurança, que é um triplet ordenado das propriedades de segurança referentes a: autoridade, categoria e franquia. AUTORIDADE é o conjunto dos 4 níveis militares: secreto, ultra-secreto, etc. CATEGORIA é um conjunto de 16 compartimentos que diferencia por exemplo projeto e área do usuário. FRANQUIA se aplica aos objetos terminal, job e arquivo e contém o conjunto de usuários que possuem permissão para acessá-los, segundo o princípio 'need-to-know' militar.

Objetos do tipo USUÁRIO só podem ser criados na inicialização do sistema. É composto, no profile, por uma identificação única escolhida pelo usuário (USER: ID), sua autoridade, sua categoria e uma lista dos direitos de acesso aos objetos tipo terminal. Também está incluída uma lista de passwords 'one-way': sempre que a password corrente for utilizada, a próxima da lista se coloca no seu lugar. Ao término da lista, outra é gerada. O sistema está limitado a um máximo de 500 usuários (500 USER: ID) por ativação.

TERMINAIS são identificados ao sistema por seus endereços de hardware e o mesmo ocorre com drives de discos, fitas, impressoras , leitoras, perfuradoras e teclado, todos pertencentes ao objeto TERMINAL. Além da TERMINAL: ID cada um possui uma autoridade e uma categoria.

A identificação de um objeto tipo JOB é feita dinâmicamente através do JOB:ID, parâmetro interno que só existe enquanto o job está ativo no sistema. Cada JOB possui uma área de trabalho protegida, inacessível ao programador, que é salva quando o programa é paginado. A autoridade de um objeto tipo JOB é a menor das autoridades do seu USUÁRIO e seu TERMINAL e a categoria de um objeto tipo JOB é a intersecção das categorias do seu USUÁRIO e seu TERMINAL.

Objetos do tipo ARQUIVO podem ser definidos como temporários ou permanentes. Os temporários só podem ser utilizados pelo programa que os criou. Os permanentes podem ser compartilhados com outros programas e usuários, desde que estejam incluídos na lista de acesso do arquivo. Toda a informação de controle do arquivo como tipo, organização, localização física na memória, data de criação e segurança está no catálogo do arquivo, separada dos dados. A lista de acessos corresponde a um conjunto de pares (u, q) onde u é uma USER:ID e

q o tipo de acesso: ler, escrever, ler-escrever e ler-escrever durante modificação. O último é utilizado para permitir leitura e escritura em arquivos que estão sendo modificados e que por isso estavam fechados a outros acessos.

A inicialização do sistema é realizada através de um pacote, SYSLOG, responsável pelo fornecimento dos dados de proteção, antes da ativação dos terminais. O SYSLOG cria e atualiza um arquivo em disco altamente sensitivo, que constitui o profile dos USER:ID e TERMINAL:ID. Este profile é construído a partir de um deck de cartões consistindo de conjuntos de dados separados para as definições de compartimentos, as classificações de terminais e as autorizações de usuários. Por ser altamente sensitivo, este deck de controle é produzido por procedures especiais, que só podem ser acessadas por pessoal estritamente autorizado.

Um usuário comum só é admitido no sistema se satisfizer as condições da procedure de LOGIN. Deve ser fornecida em cada sessão a identificação e a password corrente. A LOGIN realiza 3 pesquisas na SYSLOG: se existe a USER:ID fornecida, se esta USER:ID está na lista da TERMINAL:ID correspondente ao terminal utilizado e se a password fornecida é a corrente. Se houver alguma resposta negativa, o terminal é trancado por 30 segundos. Se estiver tudo correto, a LOGIN cria a JOB:ID.

Cada job pode executar concorrentemente até 4 programas. LOAD é o componente do ADEPT-50 usado para carregar os programas escolhidos pelo usuário, e todos os programas executam exatamente com a mesma autoridade e categoria do job. Programas são arquivos catalogados, portanto, o LOAD só pode carregar os programas cuja autoridade e categoria não excedam a do job. Criadores do ADEPT-50 utilizaram o termo "Guarda-chuva de execução" para identificar essa proteção que é conferida a todos os arquivos, independente da sua qualificação anterior, quando utilizados pelo mesmo job.

Finalmente, a função AUDIT registra certas transações relativas a arquivos, terminais e usuários, mantendo um histórico de todas as violações de segurança que ocorreram e a extensão da exposição a que os dados foram submetidos. Todos os dados são registrados em disco ou em fita em tempo real, assim, estão salvos se o sistema cair. Um utilitário adicional, AUDIT-LIST, provê a listagem de toda a informação contida no arquivo AUDIT.

O ADEPT-50, pelo exposto, contém as sementes de inúmeras idéias que agora já são árvores grandes e já nos proporcionaram ótimos frutos. Não é a lista de TERMINAL-ID em cada USER-ID uma lista de capabilities? Não é a lista de USER-ID em cada terminal, job e arquivo uma lista de controle de acessos? Não são as idéias de passwords 'one way' trancamento de terminal e identificação de jobs por usuário e terminal válidas e utilizadas amplamente ainda hoje?

O sistema ADEPT-50 foi, há 12 anos atrás, um protótipo de simplicidade, rapidez e exatidão em matéria de mecanismos de proteção. Não se constitui, é claro, um sistema impenetrável, para isso contribuiu o hardware do IBM/360-50, uma máquina de apenas 2 estados (supervisor - problema), e a precocidade das idéias. Seu valor está exatamente em ter sido o precursor na implementação prática de muitas delas.

A.2 BCC-500

O sistema BCC-500, da Berkeley Computer Corporation, é um projeto integrado de hardware e software desenvolvido por Lampson(Lam69) para suportar um grande número (até 500) de usuários de time-sharing. O computador consiste de uma grande memória primária de núcleo e circuito integrado e uma enorme memória secundária de mais de 500×10^6 bytes, onde está incluído um tambor Burroughs de 12×10^6 bytes. Existem 5 processadores microprogramados compartilhando a memória primária, cada qual dedicado a uma função específica. Dois deles executam apenas processos de usuários e os restantes são dedicados respectivamente ao gerenciamento da memória, ao gerenciamento do I/O e ao scheduling de processos.

São implementados 8 tipos de objetos de proteção: processos, domínios, páginas, arquivos, diretórios, chaves de acesso, chamadas de interrupção e terminais. Todos esses objetos são gerenciados por capabilities, sendo portanto capazes de implementar uma política discrecionária de controle de acesso.

PROCESSO é o objeto básico que executa programas e corresponde a uma cópia da máquina virtual do usuário. A qualquer momento um processo só pode executar dentro do seu domínio, isto é, só pode acessar objetos cujas capabilities pertençam ao seu domínio. Cada processo tem aproximadamente 12 domínios associados e pode executar em qualquer deles. Nunca dois processos podem rodar simultâneamente no mesmo domínio. O espaço de endereçamento de um processo está dividido pelo hardware em 3 anéis: monitor, utility e usuário. Cada processo tem uma página de memória bloqueada chamada 'context block' onde seus dados privativos são mantidos, devido a isso, processos são caros e há interesse em minimizar o seu número. Normalmente um processo é suficiente para cada job.

DOMÍNIO é um objeto usado para definir o âmbito de ação de um processo. É um conjunto de capabilities que representam, entre outros direitos, o espaço de endereçamento do processo, até um máximo de 64 páginas de 2 K bytes cada. Cada domínio corresponde a um conjunto de privilégios de acesso. Transferências entre domínios significam mudança de privilégios. Por isso, são realizadas através de pontos de entrada protegidos ou 'gates' que são um tipo de capability para o domínio. 'Gates' especificam para onde vai o controle quando o domínio é acessado e podem ser transmitidas como outras capabilities. Além disso, quando a transferência representa uma troca de anel, o hardware realiza cheques de consistência nos parâmetros que são permutados entre os domínios. Portanto domínios possuem 'gates' para outros domínios. Pode-se dizer que alguns domínios controlam outros, no sentido de que as capabilities do controlado são um subconjunto das capabilities do que controla. Embora os processos sejam caros, os domínios são muito baratos, suas capabilities são representadas por simples cadeias de bits, portanto as transferências são encorajadas.

PÁGINAS são unidades de memória de 2 K bytes identificadas por nome único de 48 bits. O gerenciamento da memória, incluindo pesquisa nas tabelas, paginação, etc., é feito por um dos 5 processadores, rapidíssimo, escrito em ROM. Na memória central são mantidas as tabelas que permitem a localização das páginas nos vários níveis da memória. A primeira vez que a página é referenciada no processo as tabelas são utilizadas, daí para frente o endereçamento na memória central é mantido em registradores rápidos.

ARQUIVOS são sequências de páginas randomicamente endereçáveis. A única forma de acessar dados nos objetos tipo arquivo é colocar as páginas correspondentes na memória virtual do processo. Arquivos possuem nome e informação de proteção associada.

DIRETÓRIOS são objetos que consistem de pares (nome extenso, capability) e de uma operação "abertura", a qual, dado o nome extenso, fornece a capability. Sendo os diretórios eles mesmos acessados por capabilities, fica implícita uma estrutura de árvore. Quando um programa de usuário precisa acessar seus próprios objetos, basta apresentar uma capability para o seu diretório, quando precisa acessar objetos de outro usuário, basta apresentar a capability para o outro diretório. Para isso, cada usuário que deseja compartilhar recursos, deve ter no mínimo 2 diretórios: um privativo no qual trabalha sozinho e um diretório de transferência, para o qual outros usuários também tenham capabilities.

CHAVE DE ACESSO é um tipo especial de capability, privativa para cada usuário do sistema. Enquanto as outras capabilities são expandidas pelo sistema, a chave de acesso é única e só pode ser usada por seu proprietário. Foi criada para facilitar o compartilhamento de objetos entre usuários, de forma que os privilégios concedidos não posssam ser propagados indefinidamente. É também um objeto protegido por capability. Qualquer usuário pode a qualquer momento solicitar ao sistema uma chave de acesso. Ela será formada pelo número de microsegundos que o sistema já existiu e nunca será igual a outra. Sua capability aparecerá no diretório do usuário como qualquer outro objeto. Cada usuário está "matriculado" no sistema por no mínimo uma chave de acesso, que lhe dá o direito de acessar o seu diretório, e através dele, outros de transferência. Um conjunto de usuários pode ser agrupado por uma chave de acesso. Domínios também podem ser acessados por chaves de acesso, por exemplo, o job de um usuário pode ser iniciado com 2 delas: uma para o domínio correspondente às suas próprias capabilities e uma para o domínio contendo as capabilities para todos os funcionários que trabalham com contabilidade.

CHAMADA DE INTERRUPÇÃO é a maneira que o sistema tem de chamar a atenção de um processo em execução. Não é o caminho normal de comunicação entre processos, pois este consiste dos mecanismos básicos de 'block' e 'wake-up'. Por exemplo, é utilizado quando o usuá

rio no terminal deseja interromper um processo que está em loop. A chamada de interrupção age forçando uma situação estranha no processo como overflow, violação de memória, fim de arquivo e transferindo o controle para o usuário no terminal. É considerado um objeto protegido e só pode ser utilizada através de uma capability porque causa troca de domínio, alterando o estado de proteção do processo.

TERMINAIS devem ser objetos protegidos em qualquer sistema de time-sharing. Capabilities para utilização de terminais são armazenadas nos diretórios de usuários e processadas como quaisquer outras.

Utilizando capabilities como uma forma de acessar objetos, o sistema BCC-500 implementa os conceitos de domínios protegidos, diretórios estruturados em árvores e chaves de acesso. Nenhum deles era novo na época mas o enfoque foi interessante. Principalmente o posicionamento que incentiva a troca de domínios em um mesmo processo, consequência do alto custo do processo e do baixo custo dos domínios. A chamada de interrupção é uma característica sui-generis, pois em nenhum outro sistema é considerada um objeto protegido e foi consequência direta do propósito de se obter eficiência num sistema de time-sharing de tal porte.

A.3 MULTICS

O MULTICS é um sistema de propósitos gerais, para múltiplos usuários, desenvolvido sob os auspícios do Projeto MAC no MIT, em esforço conjunto com a Honeywell Information Systems Inc. e, até 1969, com a Bell Telephone Laboratories (S&S 72). Seu projeto, iniciado em 1965, tinha como objetivo principal tirar vantagem de tudo o que era conhecido até a data sobre as técnicas avançadas de projeto e desenvolvimento de sistemas operacionais (Neu 78). As primeiras versões foram implementadas em hardware da General Eletric, o GE635 e o GE645; mais tarde foi utilizado o Honeywell 6180 (Sal74a).

Tendo sido construído com propósitos de segurança, o MULTICS implementa 5 tipos de objetos protegidos: processos, segmentos, filas de mensagens, diretórios e descritores removíveis.

O PROCESSO é um programa em execução, a única entidade ativa no conjunto de objetos. Quando um processo é criado, é associado a ele um identificador principal formado por 3 campos: nome do usuário, projeto e compartimento. A cada tentativa de acessar outro objeto, o identificador principal do processo é comparado com a lista de controle de acesso do objeto para verificar se existe um elemento na lista que combine, permitindo o acesso em caso positivo e negando-o em caso negativo. O campo de compartimento tem várias interpretações: classes de execução, níveis de autorização ou mesmo pode ser usado como uma chave extra para usuários controlarem o acesso a arquivos de sua propriedade. Exemplos de identificadores principais:

```
Jones . Inventory. a
Jones . * . *              (qualquer projeto e compartimento)
*  . Inventory. *          (Todos os usuários de Inventory)
```

Cada processo tem um espaço de endereçamento na memória onde se encontram todas as suas referências (S, W) = (segmento, palavra). Um segmento de descritores, privativo do processo, mapeia números de segmentos em endereços de memória real e uma tabela, também privativa, mapeia nomes simbólicos de segmentos em números de segmentos. Assim, cada processo está hermeticamente isolado dos outros, embora utilizem paralelamente os mesmos recursos do sistema (BCD 72).

SEGMENTO é a unidade de proteção da memória. Cada segmento tem associada uma lista de controle de acessos, contendo os identificadores principais de todos os processos que podem acessá-lo. Cada entrada nesta lista é composta por 4 campos: os 3 primeiros são comparados um a um aos do processo e o quarto especifica os modos de acesso. Exemplos:

```
Jones. * . * . rw          (ler e escrever)
*    . Inventory . *. re   (ler e executar)
```

A informação contida na lista de acesso é uma redundância da contida no segmento de descritores. Na realidade, é copiada neste a partir da lista de acesso, quando o segmento é referenciado pela primeira vez no processo.

DIRETÓRIOS são os objetos responsáveis pela catalogação hierárquica dos nomes de segmentos e seus atributos de acesso. Cada entrada no diretório contém: subnome do segmento, comprimento, endereços na memória (1 para cada página até um máximo de 64 páginas), a lista de controle de acessos: usuários com os respectivos modos de acesso, status (ativo ou inativo), data e hora da criação. O nome completo do segmento reflete a sua posição na hierarquia, pois é o conjunto dos subnomes de todos os diretórios que o antecederam. Por ser único em toda a hierarquia de diretórios, este nome do segmento foi escolhido para ser o nome simbólico através do qual os processos o referenciam. O diretório que está acima de todos na hierarquia é chamado RAIZ, está sempre ativo e conectado a todos os processos. Qualquer operação nos segmentos que contém diretórios só podem ser feitas através do supervisor.

Cada diretório contém, por simplicidade, uma lista de controle de acessos inicial. Sempre que um novo objeto é criado neste diretório, o conteúdo desta lista é copiado para a lista de usuários autorizados, e apenas se o usuário desejar, pode alterá-la. Permissão para modificar entradas em um diretório implicam em permissão para modificar sua lista de acessos inicial.

FILAS DE MENSAGENS são objetos que controlam separadamente o enfileiramente e atendimento de mensagens e DESCRITORES REMOVÍVEIS são objetos que permitem controle separado da leitura, escritura e adição de informação em arquivos de fitas.

Aí estão, descritos, os 5 objetos protegidos do MULTICS. Entretanto, existe um mecanismo de proteção de grande importância que ainda não foi mencionado. Trata-se dos ANÉIS DE PROTEÇÃO. Estes foram criados com o intuito de permitir múltiplos domínios de execução para cada processo do MULTICS. Como vimos, cada processo tem um segmento privativo contendo os descritores de todos os segmentos que pode, potencialmente acessar. Por construção, ó supervisor do MULTICS roda sob os programas do usuário, isto é, processos do supervisor são chamados por processos do usuário e executam sob as suas ordens. Como os processos dos usuários podem executar com diferentes privilégios, os processos do supervisor rodando sob eles devem também ser capazes de modificar seus privilégios em cada execução. Para isso, existem os "números de anel", que determinam, em cada execução, qual é o nível de privilégio que o processo terá. Cada processo possui, no seu segmento de descritores, um número para cada descritor de segmento. Dessa forma, além dos modos de acesso que são diferentes, cada segmento da memória pode ser acessado por processos em até 8 níveis diferentes de privilégio.

Também na AUTENTICAÇÃO de usuários o MULTICS contém medidas de proteção não comuns. Sendo um sistema de propósitos gerais, aceita tanto processamento em batch como em time-sharing, mas dá enfase ao segundo. Nenhum job em batch é iniciado sem que o seu proprietário autorize pelo terminal. Decks de cartões são lidos e guardados num buffer até que o usuário se identifique num terminal e envie ordem para a execução do job. Assim, é impossível rodar jobs com cartões de outros usuários, e outros procedimentos desaconselháveis. O operador do sistema também deve se autenticar como qualquer outro usuário e não desfruta de nenhum privilégio especial.

O MULTICS pode ser considerado o software de computador que utilizou o maior número de pessoas no seu projeto e implantação. Já existiram pelo menos 17 versões, umas aperfeiçoando outras e o resultado é um sistema altamente sofisticado. Uma avaliação foi realizada por Neumann (Neu 78) que concluiu ser o MULTICS um sistema relativamente seguro. Após tantas modificações no projeto inicial, que por si só já era bom, um nível significativo de segurança foi atingido diminuindo bastante a possibilidade de penetração. Embora seu hardware inicial tenha se mostrado vulnerável, as versões correntes removeram as falhas que haviam.

Finalizando, um NÚCLEO DE PROTEÇÃO foi projetado para o MULTICS, num esforço conjunto da Força Aérea, da Mitre Corporation, da Honeywell e do MIT (Sch 75). O objetivo deste núcleo era separar as funções relevantes para a segurança em um software pequeno e certificável. Foi usada a Metodologia de Desenvolvimento Hierárquico da SRI International. Entretanto, os recursos para esse projeto (GUARDIAN) expiraram antes que o núcleo fosse implementado (Neu 78). O maior problema concernente ao núcleo do MULTICS foi a linguagem de implementação. O MULTICS foi construído em PL/I, linguagem de alto nível e apenas o nível 0 possuia 80.000 linhas de codificação. Usar uma linguagem de alto nível para gerar um núcleo de proteção requer que o compilador da linguagem seja certificado, uma tarefa considerável, visto que um compilador é incomparavelmente maior que um núcleo. Este e outros problemas, aliado ao bom desempenho do MULTICS tradicional levaram ao abandono do projeto GUARDIAN.

A.4 CAL-TSS

O CAL-TSS foi projetado em Berkeley para ser um sistema de propósitos gerais, batch e time-sharing, implementado num CDC 6400. Um dos objetivos do projeto é que fosse competitivo em eficiência com o SCOPE, sistema operacional do fabricante e que fosse capaz de similar o SCOPE para evitar a translação de programas e utilitários. Foi escolhido um mecanismo de proteção uniforme baseado em capabilities e uma arquitetura hierárquica de no mínimo 4 camadas de software protegidas umas das outras (LaS 76).

O hardware consiste de uma memória central (CM) com 3 K palavras de 60 bits e uma memória de núcleo expandida (ECS) com 300K palavras de 60 bits, sendo rapidíssima a transferência de palavras entre ambas. O próprio hardware se encarrega de prover mecanismos de proteção dessas memórias, sob a forma de 2 pares de registros, um par controlando o acesso à CM outro à ECS.

Das 4 camadas que compõem a hierarquia do sistema, a primeira constitui o Sistema de Núcleos, que provê o conjunto mínimo de recursos necessários para a construção das camadas seguintes. As 3 últimas constituem o Sistema do Usuário. Outras camadas podem ser adicionadas à estrutura, gerando subsistemas protegidos para usuários. O simulador do SCOPE, por exemplo, foi escrito como um programa de usuário e colocado numa 5ª camada.

São implementadas 2 classes de objetos protegidos. A primeira corresponde aos objetos do núcleo: arquivos do núcleo, canais de eventos, blocos de alocação, C-listas, labels, processos, operações e tipos. A segunda classe corresponde aos objetos do usuário: arquivos de discos, diretórios, chaves de acesso, etiquetas de nomes e descritores de domínios. O universo dos objetos disponíveis ao usuário inclui também os do núcleo, com a restrição de que nenhum objeto do núcleo pode ser armazenado em disco ou ter capabilities em diretório.

ARQUIVOS DO NÚCLEO são objetos de páginas removíveis utilizados para armazenar dados, especialmente de arquivos de usuários que residem a maior parte do tempo em disco. Para acelerar o acesso durante a execução, esses arquivos são autorizados a residir parcialmente na ECS, como se fossem arquivos do núcleo.

CANAIS DE EVENTOS são os objetos utilizados na comunicação interprocessual, sendo "evento" o nome dado a cada mensagem de 1 palavra. Duas operações podem ser realizadas: enviar evento e receber evento. Cada canal possui uma memória que acondiciona um número fixo de eventos e quando está cheio, a operação "enviar evento" retorna com um código de erro. Operações de I/O são realizadas entre dispositivos e usuários através dos canais de eventos.

BLOCOS DE ALOCAÇÃO são objetos que possuem autoridade para alocar recursos do núcleo, como tempo de CPU e espaço na ECS, que serão

contabilizados. Possui também autoridade para alocar elementos para criação de novos objetos. Cada objeto criado sob a responsabilidade de um bloco é seu dependente e a posse de uma capability para esse bloco implica em autoridade para acessar e deletar o objeto. Existe uma árvore de blocos de alocação, em que uma única raiz, criada na inicialização do sistema, é responsável por todos os blocos de alocação, sendo portanto dona de todos os recursos do sistema.

C-LISTA é uma sequência finita de capabilities. Cada capability possui 3 componentes: tipo, conjunto de direitos e valor. O tipo é um inteiro que representa o tipo de objeto que é referenciado pela capability. O conjunto de direitos contém os modos de acesso ou as operações que podem ser realizadas naquele tipo de objeto. O valor depende do conteúdo do tipo, mas geralmente é um par (nome único, índice) onde índice aponta para uma entrada na MOT (Tabela de Objetos Mestre) que retorna um pointer para o objeto. Além do pointer, na MOT existe também o nome único do objeto, que é comparado ao nome único da capability, reforçando o cheque. A existência da MOT é uma das características mais interessantes do CAL-TSS pois a revogação de capabilities é simples: como cada objeto só tem endereço na MOT e as capabilities só carregam direitos de acesso, revogar o acesso a um objeto é simplesmente eliminar o seu endereço da MOT, não importa quantas capabilities existam para ele espalhadas no sistema.

LABEL é um nome global para domínios equivalentes em diferentes processos. Desde que um processo possua uma capability para um label, está autorizado a trocar seu domínio natural de execução para o domínio especificado pelo label.

PROCESSO é um objeto protegido composto de 6 elementos distintos, a maioria dos quais inacessíveis como objetos independentes: uma árvore de domínios, uma pilha de chamadas, uma cadeia de pointers para canais de eventos, um bloco de alocação para contabilizar ciclos do processador, um conjunto de registros de máquinas e alguns timers. Trocas de domínios podem ser realizadas através da árvore ou através do LABEL. Cada programa de usuário entra no sistema como um PROCESSO distinto.

OPERAÇÕES são objetos com autoridade para executar em domínio diferente do corrente. São constituídas por níveis, cada qual com duas partes: a ação a ser realizada, isto é: o novo domínio com seu ponto de entrada e uma lista de parâmetros. Caso a operação não seja uma ação do núcleo, o nome do novo domínio é substituido por um objeto tipo LABEL, o que a torna disponível a quantos processos queiram fazer uso dela.

TIPOS são objetos que autorizam a criação de capabilities para objetos de usuários. Dado um tipo T e uma palavra P, o objeto TIPO

cria uma capability P para T, com todos os direitos de acesso presentes. Uma vez criada, a capability nunca poderá ser modificada , apenas apresentada como autorização para objetos tipo T. Assim, objetos TIPO agem como um selo usado para autenticar uma simples palavra, que se torna um objeto protegido. Devido a isso, são os responsáveis pela criação de EXTENSÕES de objetos.

ARQUIVOS DE DISCO são objetos do usuário que tem a mesma estrutura dos arquivos do núcleo, podendo residir parcialmente no disco e na ECS para agilizar a leitura. Contém dados ou informações.

DIRETÓRIOS são listas de entradas de objetos, cada uma contendo um nome simbólico, uma especificação do objeto e uma lista de fechaduras. O nome simbólico é uma cadeia de caracteres. A especificação do objeto pode ser de 3 tipos: uma entrada contendo um pointer para um objeto próprio, um link contendo um pointer para um objeto não-próprio, ou um link contendo um pointer para outro diretório. Cada fechadura de acesso contém um inteiro, utilizado para variar a chave de acesso.

CHAVE DE ACESSO é um objeto que contém 1 inteiro e age como uma chave ordinária abrindo fechaduras de diretórios. Um domínio só pode acessar as entradas de um diretório para as quais tenha chave, e como existem vários tipos de fechaduras e de chaves para a mesma entrada, os direitos de acesso ao objeto são conferidos pela fechadura que for aberta pela chave apresentada.

ETIQUETAS DE NOMES são objetos que foram inventados para resolver o impasse de as capabilities do núcleo não poderem aparecer em diretórios, construções de usuários. Funcionam como chaves de acesso: há capabilities para cada etiqueta, e na parte de valor da capability há um inteiro de mesmo valor do inteiro contido na etiqueta. O sistema do usuário mantém uma tabela de correspondência entre esses inteiros e as capabilities para objetos do núcleo. Esta tabela é destruída a cada desligamento ou queda do sistema, portanto, no mínimo uma vez por dia. A cada inicialização do sistema é reconstituída a tabela apenas com objetos especiais como arquivos e canais de eventos usados na comunicação com dispositivos de I/O.

DESCRITOR DE DOMÍNIO ou simplesmente DOMÍNIO é o objeto responsável pela descrição do ambiente de execução para um processo de usuário. Por ser uma criação do usuário, o DOMÍNIO é uma entidade estática em que as operações de criação e destruição são muito menos frequentes que as de chamada. Cada domínio contém os seguintes elementos: um label, que dá nome ao domínio, uma C-lista local, uma especificação de mapeamento, um domínio-par e uma lista de erros possíveis de serem recuperados. Se durante a execução do processo surge um erro que não pode ser manuseado pelo domínio vigente, ocorre uma mudança para o domínio superior mais próximo que possa recu-

perá-lo. As capabilities da C-lista podem ser exercidas, mas não copiadas. A construção dos DESCRITORES DE DOMÍNIO é feita por OPERAÇÕES em que o LABEL e a C-LISTA são parâmetros fixos ou "selados", assim, nenhum domínio pode conter privilégios que o seu ancestral não contenha.

O projeto do CAL-TSS durou de junho/68 a novembro/71 e foi encerrado por falta de recursos, pois após 3 anos o sistema não era nem eficiente nem usável. Lampson justifica isso pelo excesso de novas idéias, inexperiência da equipe e erro nas estimativas de tempo e dinheiro. Da experiência provieram algumas sugestões, amplamente utilizadas nos projetos subsequentes:

1. definir apenas 1 classe de objetos para todo o sistema.
2. construir domínios com suporte de hardware.
3. não procurar contabilizar todos os custos diretos de cada job.
4. construir uma interface entre o núcleo e o programador.
5. evitar levar adiante "idéias interessantes" não essenciais para o sistema.

Dentre as características que funcionaram bem encontram-se o bom desempenho das capabilities, do núcleo e dos dispositivos de I/O simulados como processos ordinários.

O sistema funcionou durante os últimos 3 meses por no mínimo 8 horas diárias suportando até 15 usuários e durante esse tempo houve 18 quedas no sistema: 14 devidas aos níveis acima do núcleo, 3 causadas pele hardware e 1 de causa desconhecida. Embora não tendo causado dividendos para si próprio, o CAL-TSS representou um investimento lucrativo, se considerarmos os benefícios que foram legados à comunidade científica.

A.5 UCLA UNIX

O UCLA SECURE UNIX é um sistema operacional em hardware de PDP -11 (45 e 70) desenvolvido como um núcleo de segurança que suporta interface com o UNIX standard. O UCLA UNIX foi projetado para ser um sistema de propósitos gerais, com controles discrecionários de acesso, por isto implementa capabilities como mecanismo básico de proteção (PKK 79).

Um grande número de facilities normalmente encontradas em grandes sistemas é também encontrada no UCLA UNIX, mas propósitos de certificação o tornaram pequeno e de estrutura objetiva. É baseado em apenas 4 tipos de objetos primitivos: processos, páginas, dispositivos e capabilities, não havendo possibilidade de criação de extensões.

PROCESSO é um objeto definido por um conjunto de variáveis de estado e uma página pequena na memória principal. A memória virtual do processo não está incluída, assim, chamadas ao processo são simples, bastando apenas mover dados das tabelas para registros da CPU e vice-versa. Sendo tão pequena a memória bloqueada por processo, é possível que um grande número deles esteja ativo simultâneamente, este número está em torno de 500. Cada processo possui uma C-lista com todas as suas capabilities, que são exercidas nos pedidos de acesso a objetos e na comunicação interprocessual. Esta C-lista constitui o domínio de execução protegido do processo. As operações disponíveis para processos são: Invoke, Inicialize, Zero -relocation-register, Return, Set-interrupt e Send-interrupt, as duas últimas usadas na comunicação entre processos.

Processos podem ser inicializados a partir de programas do usuário ou de tarefas do sistema operacional. Os 2 processos que constituem o sistema de proteção são o POLICY MANAGER, isto é, o processo capaz de alterar dados sobre os quais as decisões de segurança são tomadas e o DIALOGUER, ou seja, o processo que inicialmente possui todos os terminais e é responsável pela autenticação dos usuários. O SCHEDULER embora seja um processo importante para o sistema em geral, está fora do sistema de proteção pois não toma decisões de segurança, apenas aloca os recursos baseado nas decisões que foram tomadas pelos outros dois. Para os processos de usuários várias UNIX INTERFACES são providas, produzindo uma estrutura semelhante à de máquinas virtuais: cada usuário roda num UNIX virtual independente.

PÁGINA é a unidade de memória abstrata suportada pelo núcleo do sistema operacional. Todas as páginas tem uma localização fixa na memória secundária, a qual não é desalocada quando a informação é copiada na memória principal. O scheduler é responsável pelos mecanismos de memória virtual (paginação, despaginação) que é invisível aos outros processos. Para acessar qualquer página, um processo precisa apenas ter uma capability para ela e indicar no seu espaço de endereçamento o local em que ela deverá ser copiada.

Através desse mecanismo, o compartilhamento de páginas fica trivial. As operações possíveis para páginas são: Swap-in, Reflect e Free. Para otimizar as funções do núcleo existem 2 tamanhos diferentes de páginas e cada memória (principal e secundária) está dividida em duas partes, uma para cada tamanho, fixadas na geração do sistema. Uma vez que cada página tem um endereço fixo na memória secundária, a memória principal é simplesmente uma cache, páginas de 'core' são meras cópias das páginas de discos.

Operações de I/O para todos os DISPOSITIVOS, incluindo terminais, são controladas pelo mesmo mecanismo de capabilities que controla a utilização de páginas. Entretanto, terminais e similares são considerados dois dispositivos diferentes: um de entrada e um de saída. Assim, cada dispositivo possui 2 capabilities: uma para leitura e outra para escritura, simplificando bastante o sistema de proteção. Interrupções de término de serviço são manuseadas exatamente como qualquer outra interrupção: todos os processos que possuem ativa a habilidade de ser interrompidos receberão a notificação. As operações permitidas para dispositivos são: Start-i/o, Completion-interrupt e Status. Um disco físico pode alternativamente ser tratado como um dispositivo e um start-i/o ser emitido para ele, mas nesse caso, o disco não poderá também receber páginas.

CAPABILITIES são a representação básica da informação de proteção. Cada processo possui uma C-lista, acessível apenas pelo núcleo. Cada capability possui 4 campos: nome do objeto, direitos de acesso, endereço provável e descritor de arquivo. O endereço provável é um valor estimado utilizado pelo sistema antes de pesquisar a tabela toda, para ganhar tempo. Exemplos: para páginas, o endereço provável é o último local em que elas apareceram na memória principal, embora já possam ter sido removidas, e nesse caso terá que ser feita pesquisa na tabela, mas é raro acontecer. O descritor de arquivo é utilizado pelo POLICY MANAGER para gravar o arquivo ao qual a página ou dispositivo está associado. A única operação possível em uma capability é Grant/Revoke, que adiciona/deleta uma capability em local especificado de uma C-lista. Somente processos autorizados por outra capability podem realizar essa operação. A revogação é realizada colocando-se uma capability nula no lugar da capability correspondente à que deve ser revogada. Processos não podem transmitir capabilities a outros processos. Todas as decisões sobre transferência de direitos, inclusive atribuição dos direitos iniciais de cada capability são tomadas pelo POLICY MANAGER.

No UCLA UNIX processos, páginas e dispositivos nunca são criados ou destruídos. Existem tantas páginas quantas são necessárias na memória secundária e o número de processos é fixado pelo tamanho da tabela de processos. Dispositivos são adicionados na geração do sistema. Esta visão estática não é uma limitação real, uma vez que o POLICY MANAGER reutiliza "corpos" de processos e páginas após inicializá-los. Como resultado, o sistema de proteção é simples e robusto o suficiente para ser utilizado nas atividades mais gerais de um sistema operacional.

Uma característica importante encontrada no UCLA UNIX é a exclusão do POLICY MANAGER do núcleo. Isto foi feito, como em outros sistema de mesma filosofia, para reforçar a diferença existente entre "mecanismo de proteção" e "informação de proteção". Os mecanismos, sim, pertencem ao núcleo, mas a informação fica fora dele. O POLICY MANAGER é um processo privilegiado que roda sobre o núcleo, provendo-lhe as informações de sistemas de arquivos, profiles de usuários, etc. É envolvido sempre que um processo é inicializado, registrando o nome do usuário, concedendo-lhe 2 capabilities e colocando o program-counter e o status em valores padrões.

Finalizando, além do POLICY MANAGER, DIALOGUER, SCHEDULER e UNIX INTERFACES existe mais um processo que interage diretamente com o núcleo, o NETWORK NUCLEUS. O núcleo da rede realiza as funções de interface entre os processos do sistema e uma rede, particularmente, a ARPANET. Funções de criptografia e protocolos de conecção inicial são realizadas de maneira que o software da rede seja tratado simplesmente como uma parte insegura do canal de transmissão.

O UCLA SECURE UNIX é o primeiro sistema operacional de propósitos gerais verificável (PKK 79). Outros sistemas verificáveis não são de propósitos gerais. A verificação formal foi causa e consequência da utilização de um núcleo de segurança. Foi utilizada a linguagem PASCAL e as etapas da verificação formal são descritas por Popek e Kline (POK 78). Além disso, o núcleo provê proteção a uma granularidade bastante boa e pode ser totalmente microprogramado. A performance geral do sistema apresentou-se excelente, não tendo sido afetada pelas restrições de segurança. A conclusão final do projeto é otimista: pelo menos em máquinas de média escala, sistemas operacionais seguros são viáveis e constituem apenas um problema de engenharia.

A.6 KSOS

O KERNELIZED SECURE OPERATING SYSTEM foi projetado e desenvolvido na Ford Aerospace & Communication Corporation, com recursos de várias agências do Departamento de Defesa Americano para atuar em grandes minicomputadores. A implementação inicial deu-se num PDP-11/70, seguida por uma versão modificada para o Honeywell Nível 6.

Produzido em uma instalação habituada ao uso do sistema operacional UNIX, o KSOS foi projetado "com sabor do UNIX", sem ser entretanto dependente deste, assim os programas de aplicação escritos para o UNIX rodam sem modificação ou recompilação no KSOS. As únicas mudanças realizadas no sistema operacional antigo foram a inclusão de cheques de segurança em algumas chamadas ao supervisor e a adição de novos tipos de chamada, modificações estas que certas instalações já haviam adicionado aos seus sistemas (CaD79, BeB 79).

O KSOS possui estruturalmente 3 componentes: KERNEL (núcleo), NKSR (software de segurança não pertencente ao núcleo) e o Emulador do Unix. O núcleo controla o acesso aos recursos do sistema intermediando cada solicitação, o NKSR desempenha as funções básicas de suporte de sistema e o Emulador do Unix transforma comandos do Unix em chamadas para o núcleo.

Tendo sido projetado com o propósito de implementar a política não-discrecionária multinível, o núcleo verifica a observância às duas propriedades não-discrecionárias simples e estrela. Além disso, dispõe de um mecanismo primitivo de controle de acesso discrecionário. Para cada objeto do sistema existem 9 bits sob controle do usuário, que especificam 3 tipos de acesso: ler, escrever e executar/pesquisar para o dono, outros usuários do mesmo grupo e todos os outros usuários. Embora esse mecanismo de acesso discrecionário seja limitado em comparação com esquemas mais sofisticados, é simples, necessita pouco suporte e é extremamente útil.

O sistema KSOS implementa 5 tipos de objetos primitivos: processos, segmentos de processos, arquivos, dispositivos e subtipos de arquivos, este último sendo a fonte para criação de extensões. Todos os objetos, independente de qual seja o tipo, possuem um nome ou SEID (Secure Entity Identifier), e um bloco de informações associado que é verificado pelo núcleo a cada tentativa de acesso. Esse bloco é chamado "informação independente de tipo" e simplifica o trabalho do núcleo na autorização do fluxo de informação entre fonte e destinatário.

PROCESSOS são os únicos objetos ativos do sistema (sujeitos). Cada comando do UNIX roda como um processo separado, portanto, processos devem ser baratos. Cada processo necessita para si próprio poucos recursos do núcleo e pode ter a maioria dos dados em memória paginável pois apenas as tabelas devem estar em memória fixa. Os processos podem ser privilegiados e neste caso algumas funções são permitidas sem a interferência do núcleo, ex: reclassificação de um arquivo. Existe um grande número de tipos de privilégios e a cada processo é concedido o mínimo necessário para a realização das suas funções. Os privilégios são obtidos do arquivo imagem do processo

na inicialização do mesmo e para processos novos (filhos) criados a partir de outros já existentes (pais) os privilégios dos filhos são os mesmos dos pais. Pode-se substituir o arquivo imagem original do processo por um arquivo especificado pelo usuário contendo seus próprios privilégios, mas o núcleo verifica e interage nessa operação delicada. Normalmente processos rodam em um único nível de segurança à excessão dos processos do NKSR privilegiado que modificam o nível de segurança do usuário. Processos podem comunicar-se entre si através de segmentos compartilhados ou de mensagens, forçar interrupções em outros processos e proteger-se durante alguma fase crítica de sua execução contra interrupções provocadas por outros processos.

SEGMENTO DE PROCESSO é uma porção do espaço de endereçamento virtual de um processo. Não está amarrado à memória de nenhuma das máquinas virtuais. O tamanho de cada segmento é variável desde o menor elemento de memória do hardware até o espaço de endereçamento virtual inteiro de um processo. Um processo não necessita ter todos os segmentos presentes no seu espaço de endereçamento, uma vez que os segmentos compartilhados podem estar no espaço de endereçamento de outros processos. Segmentos compartilhados devem ser utilizados através de um protocolo, para garantir que não serão corrompidos inadvertidamente.

ARQUIVOS são suportados pelas máquinas virtuais ficando sua estrutura interna transparente ao núcleo. Mesmo os diretórios não são diferenciados dos arquivos comuns pelo mecanismo de proteção, permitindo que usuários os criem como desejarem. Uma vez autorizado o acesso, o núcleo não interfere no I/O de arquivos, que é feito diretamente no espaço de endereçamento do usuário. Sendo o UNIX a máquina virtual basicamente utilizada pelos usuário do KSOS sua estrutura de diretórios e arquivos é empregada.

DISPOSITIVOS são como um tipo especial de arquivos na ótica do mecanismo de proteção. Terminais tem apenas o suporte de mais baixo nível, as funções mais elevadas são realizadas pela máquina virtual abaixo do núcleo. No KSOS cada terminal é logicamente interpretado como 2 dispositivos: o normal e o de segurança. Apenas o software privilegiado está autorizado a usar o dispositivo de segurança. Quando o usuário solicita um serviço ao sistema de segurança, a trajetória normal é suspensa e é ativado num processo privilegiado que ocasiona a realização do serviço solicitado. Após terminado, a trajetória normal é reatada e o terminal volta a ser normal.

SUBTIPOS DE ARQUIVOS, o último tipo de objeto protegido foi projetado para permitir o agrupamento seletivo de classes de arquivos, já que o objeto tipo arquivo em si mesmo não inclui diferenciações. Cada arquivo é membro de uma classe de subtipos. Podem ser criadas várias estruturas diferentes de diretórios, coexistindo pacificamente. O acesso a cada elemento de um subtipo é restringido

de acordo com regras pré-estabelecidas que se aplicam a todos os elementos do mesmo subtipo. É este objeto que implementa o conceito de extensão no KSOS. O mecanismo de subtipo permite que haja encapsulação e controle de objetos sem ser necessário o conhecimento da sintaxe e semântica dos mesmos pelo sistema de proteção. Quando o núcleo realiza os testes de acesso, se o objeto for subtipo, realiza também os testes definidos especificamente para ele pelo seu proprietário.

Existem duas formas de se usar o KSOS: via Unix ou diretamente. Ambas são atrativas, pois a primeira é baseada num dos melhores sistemas existentes e a segunda, além de obter uma excelente performance, é a base para o desenvolvimento de aplicações multiníveis. A utilização dos subtipos permite a criação de ambientes de proteção encapsulados, que podem também ser usados para minimizar a quantidade de codificação privilegiada em aplicações multiníveis. Além disso, através de um processo privilegiado, NETWORK DAEMON, provê uma interface protegida para redes multiníveis.

O projeto do KSOS utilizou a Metodologia de Desenvolvimento Hierárquico (HDM), a linguagem de especificação formal SPECIAL e foi o primeiro núcleo certificado desenvolvido em um ambiente industrial.

A.7 SYSTEM /38

O S/38 da IBM é uma arquitetura que provê segurança através de um controle de endereçamento e da criação de domínios de execução protegidos no próprio hardware (Ber 80).

A primeira decisão de projeto foi a escolha de um endereçamento baseado em capabilities, denominadas POINTERS no S/38. Estes são objetos do hardware, gerados e manipulados apenas por microprogramas e autenticados através de etiquetas. Existem 4 tipos de pointers: "do sistema", que endereçam objetos protegidos, "de espaço", que endereçam bytes (espaços de memória), "de dados", que endereçam dados e contém seus atributos e "de instruções", que auxiliam nas transferências de controle entre instruções de um programa.

Objetos são construções de alto nível como programas, filas e índices e são armazenados em um ou mais segmentos. Cada segmento tem um cabeçalho de informações legível apenas pela microprogramação, que identifica o tipo de objeto contido e seus atributos de acesso. Se houverem outros segmentos associados ao objeto, estes são também definidos no segmento base, o qual é usado internamente para identificar unicamente o objeto.

Ao todo são implementados 14 tipos de objetos de proteção que, como construções de alto nível, podem ser manipulados diretamente por programas de usuário. A proteção está em conceder capabilities ou POINTERS apenas aos programas que estão autorizados a acessá-los.

ESPAÇO é um objeto protegido que representa uma quantidade de memória endereçável por bytes. É o único objeto no qual são permitidas operações ao nível de bits. Tipicamente é utilizado para armazenamento de variáveis de programa. Pode estar associado a qualquer outro objeto e pode ter tamanho variável.

PROGRAMA é a forma interna de um programa de usuário, consistindo de uma cadeia de instruções e descrições de operandos. Após ser transladado para microprogramação e ter recebido o status de objeto protegido, um programa não pode ser alterado nem executado fora dos seus entry-points. Os operandos de um programa são armazenados com ele se forem constantes ou em ESPAÇOS se forem variáveis automáticas e estáticas. Cada programa é executado em um processo, que não é em si mesmo um objeto protegido, mas utiliza muitos objetos protegidos durante a sua execução.

ESPAÇO DE CONTROLE de processo é a identificação de um processo e seu repositório de informação interna de controle. Cada processo do sistema contém além do ESPAÇO DE CONTROLE, duas pilhas de objetos tipo ESPAÇO onde são alocadas as variáveis tipo estáticas e automáticas de cada rotina executada. Quando uma rotina é invocada pela primeira vez em um processo, as variáveis estáticas são coloca

das num ESPAÇO e inicializadas. As variáveis automáticas são alocadas, inicializadas e desalocadas a cada chamada e retorno da rotina em um processo.

PERFIL DE USUÁRIO é o objeto protegido que contém, para cada usuário, seus direitos de acesso aos outros objetos do sistema. É responsável pela implementação de domínios de proteção dos processos. Quando um usuário liga um terminal, sua autenticação e todo o seu job são realizados dentro do domínio descrito pelo seu perfil. Existem 4 níveis de autorização possíveis para um perfil de usuário incluindo as autorizações específicas para cada objeto. No perfil do usuário cada objeto está explicitado como "próprio" ou não, se for "próprio" são especificados os outros usuários autorizados a utilizá-lo. Quando um objeto é criado o perfil corrente (domínio corrente) se torna seu proprietário e todos os direitos lhe são atribuídos. Objetos podem ser permanentes ou temporários, os últimos são eliminados do perfil a cada IPL. Um usuário pode substituir seu próprio perfil por um de outro usuário que o tenha autorizado, em programas que tenham um atributo especificando a "adoção" de outro perfil.

Os objetos seguintes se referem às funções de paginação, arquivamento de dados, comunicação entre processos e entre diferentes sistemas por linhas de comunicação. Como essas funções representam áreas vulneráveis que podem constituir em invasão do sistema operacional, no S/38 foram todas protegidas por capabilities e só podem ser acessadas através de autorização específica.

GRUPO DE ACESSO é um agrupamento físico de objetos para otimizar funções de paginação.

CONTEXTO é um índice utilizado para a obtenção de pointer de endereçamento para objetos através de nomes. Contém nomes de objetos e seus endereços virtuais. Cada processo deve ter autoridade (capability) para cada contexto que consultar. O contexto fornece o endereçamento do objeto, mas não os direitos de acesso (estes só são obtidos do PERFIL DO USUÁRIO).

ESPAÇO DE DADOS é um arquivo homogêneo de dados.

ÍNDICE DE ESPAÇO DE DADOS é um arquivo lógico de entradas em um ou mais espaços de dados.

CURSOR é uma trajetória de acesso a entradas de espaço de dados.

ÍNDICE é um índice de no máximo 120 registros de dados em sequência ordenada.

FILA é uma fila de mensagens para comunicação e sincronização entre processos.

DESCRIÇÃO DE UNIDADE LÓGICA é a identificação e os atributos de um dispositivo de I/O.

DESCRIÇÃO DE CONTROLADOR é a identificação e os atributos de um controlador de dispositivos de I/O.

DESCRIÇÃO DE REDE é a identificação e os atributos de uma linha de comunicação.

As características notórias do IBM S/38 são a implementação total em microprogramação, a proteção por capabilities etiquetados de todos os objetos do sistema e a existência de domínios protegidos para processos como criação de alto nível, podendo ser utilizados também para isolamento entre várias rotinas de um mesmo processo.

A.8 PSOS

Projetado no SRI International (Stanford Research Institute) o PROVABLY SECURE OPERATING SYSTEM tem como característica principal a certificação, isto é, a verificação formal do atendimento aos requisitos. Outra característica de projeto é a arquitetura hierárquica de vários níveis, em cuja base se encontram capabilities (FeN 79).

Como em todo sistema certificável, foi empregada uma técnica formal de orientação, neste caso optou-se pela Metodologia de Desenvolvimento Hierárquico (HDM) também da SRI International (Neu 78). Na especificação formal foi empregada a linguagem SPECIAL (Specification and Assertion Language).

A arquitetura hierárquica é uma coleção de dados e procedures constituídos de forma hierárquica (fig. A.1).

16.	interpretação de pedidos do usuário
15.	ambiente de usuário e espaço de nomes
14.	I/O de usuário
13.	registros de procedures
12.	processos de usuários e I/O visível
11.	criação e deleção de objetos de usuários
10.	diretórios
9.	gerentes de objetos abstratos
8.	segmentos e janelas
7.	páginas
6.	processos do sistema e I/O do sistema
5.	I/O primitivo
4.	procedures aritméticas e outras básicas
3.	relógios
2.	interrupções
1.	registros e outros objetos de memória
0.	capabilities

Fig. A.1 - Arquitetura do PSOS

Cada nível na hierarquia representa um conjunto de abstrações relativas aos níveis inferiores. Abstrações em níveis mais altos são implementadas usando objetos abstratos introduzidos nos níveis mais baixos. É indiferente se a abstração é implementada em hardware, firmware ou software, embora seja interessante que as dos ní-

veis inferiores o sejam em hardware e firmware e as dos níveis superiores em software. Pode-se interpretar os níveis do PSOS como pertencentes a 6 grandes grupos de abstrações:

14-16	abstrações do usuário
10-13	abstrações da comunidade
9	gerentes de objetos abstratos
6-8	recursos virtuais ou objetos abstratos
1-5	recursos físicos
0	capabilities

Na base do sistema está o mecanismo de capabilities, sobre os quais todos os outros objetos são constituídos. Em seguida estão os recursos físicos do sistema: memória primária e secundária, processadores e dispositivos de I/O. Após estes, vem os recursos virtuais, constituídos a partir dos físicos e que permitem gerenciamento daqueles no sentido de aumentar a sua eficiência. Em seguida vem os gerentes de objetos abstratos, que cuidam dos recursos virtuais, associando capabilities de objetos abstratos à capabilities de sua representação física. Finalmente vem as abstrações da comunidade e as abstrações de cada usuário.

No PSOS, como em outros sistemas, CAPABILITY é um nome protegido para um objeto. Sua criação e transferência a outros usuários ou processos é controlada e uma vez criada nunca pode ser alterada. Possuem um bit de etiqueta que as diferencia dos outros objetos do sistema, através do qual mecanismos de hardware as tornam não-alteráveis e não-falsificáveis.

Cada capability consiste de 2 partes: identificador único e direitos de acesso. O identificador único diferencia cada capability das outras e é repetido nas várias cópias de uma mesma capability. Os direitos de acesso definem as operações que podem ser realizadas no objeto, e pode ser transmitido iguais ou reduzidos na cópias.

Para implementar algumas políticas que restringem a propagação das capabilities, existem capabilities que não podem ser copiadas. Estas são denominadas "limitadas à memória" e não podem ser transferidas para fora do processo que as contém. Cada segmento da memória é designado, em cada autorização de armazenamento, se vai conter capabilities limitadas ou não. O mesmo segmento não pode conter simultaneamente ambos os tipos de capabilities. Essa restrição é manuseada pelo gerente da memória, através de um simples cheque nas operações de modificação de segmento.

Capabilities são usadas tanto na construção dos objetos mais simples (canais de I/O, memória primária) como dos mais complicados (diretórios do sistema, extensões de objetos criadas pelo usuário). Assim, representam uma maneira uniforme de tratar todos os recursos do sistema e evitam a necessidade de facilities especiais. Para ca

da tipo de objeto existe um GERENTE DE TIPO que é um programa responsável pelo suporte às propriedades e operações daquele tipo. Por exemplo, DIRETÓRIO, é um objeto composto por entradas que mapeiam nomes simbólicos de objetos em capabilities para esses objetos. É gerenciado pelo Diretory Manager, que interpreta os direitos de acesso de cada capability que lhe é fornecida. Estas podem indicar: adicionar entradas, remover entradas, exercer a capability contida em uma entrada.

O fato do PSOS ser altamente extensível (suportar EXTENSÕES de objetos) e possuir diferentes gerentes de tipos lhe permite suportar muitas aplicações com necessidades diferentes e até conflitantes. Por exemplo, vários subsistemas podem ser projetados para implementar diferentes políticas de segurança. Uma tarefa particular pode ser restringida a ter acesso apenas a um desses subsistemas, mas várias tarefas podem executar simultaneamente em diferentes subsistemas. Assim, o PSOS representa uma FAMÍLIA DE SISTEMAS OPERACIONAIS, onde cada usuário pode escolher o nível que contenha o conjunto de recursos mais satisfatório para as suas necessidades. Quanto mais baixo o nível escolhido mais primitivo o sistema será. Este nível pode ser aumentado pela inclusão de gerentes de tipo específicos em cada aplicação. Assim, escrever um programa "de sistema" é tão simples como escrever um programa "de usuário" e não são necessários conhecimentos nem privilégios especiais para isso. De certa forma, cada um desses subsistemas pode ser interpretado como um "núcleo" para as tarefas que tem acesso a ele. Isto vem diretamente de encontro à restrição mais séria que se tem aos núcleos de segurança, que é a sua excessiva dependência de uma única política de segurança e consequente falta de flexibilidade.

Finalizando, as 4 características principais do PSOS são: arquitetura hierárquica, mecanismo de proteção baseado em capabilities, especificação formal levando à certificação e flexibilidade para implementação de várias políticas concomitantemente. Estas nos permitem afirmar que este sistema é uma combinação aperfeiçoada de dezenas de sistemas anteriores e pode ser considerado o melhor em termos de sistemas operacionais seguros. Infelizmente, porém, não foi implementado por falta de hardware adequado, embora, segundo seus criadores, as características de hardware requeridas sejam todas encontradas, dispersas, em máquinas conhecidas.

A P Ê N D I C E B

SISTEMA DE HARDWARE DISTRIBUÍDO

B.1 HYDRA

O HYDRA, da Carnegie - Mellon University, foi implementado em um sistema de multi-processadores denominado Cmmp composto por 16 módulos autônomos de PDP-11. Cada processador é realmente um computador independente, com uma pequena memória privativa, memórias secundárias, dispositivos de I/O, etc., conectados a uma memória central compartilhada, através de uma chave transversal. Processadores podem se interromper em 4 níveis de prioridade, são sincronizados por um relógio central e possuem hardware de relocação para mapear seus endereços virtuais nos endereços reais da memória compartilhada (WCC 74, CoJ 75).

O sistema HYDRA é a base para a construção de um número indefinido de sistemas operacionais e diferentes subsistemas protegidos de usuários. Os objetos em cada um desses sistemas podem ser escolhidos entre os primitivos, mas existem também mecanismos para a criação e a proteção de extensões.

Cada objeto tem 3 partes: um nome único, um tipo e uma representação. A parte de tipo serve para agrupar os objetos de mesma características, exemplo PÁGINA. A representação contém toda a informação sobre os recursos do mesmo. Pode conter dados, manipulados livremente pelos sujeitos que tem direito ao objeto, ou capabilities , manipuladas apenas pela invocação das funções do núcleo. Existem 11 tipos de objetos primários, diretamente suportados pelo núcleo: procedure, LNS, página, processo, semáforo, port, dispositivo, política, dados, universal e tipo.

PROCEDURE é uma abstração da noção intuitiva de subprograma ou subrotina, tem codificação, aceita parâmetros e dados e retorna valores. Entretanto tem um campo de ação muito mais amplo que as subrotinas comuns e leva mais tempo executando. Cada procedure tem um nome e uma C-lista a qual representa o domínio de execução da procedure. Existem 3 tipos de capabilities na C-lista: as independentes de chamada, que são especificadas na criação da procedure e existem sempre em qualquer ativação, as dependentes de chamada, desconhecidas pela procedure até o momento da execução que lhe são enviadas como parâmetros pelo sujeito que a chama e as "templates". Templates são um conjunto de parâmetros "esperados" que validam a chamada do procedure, garantindo que o sujeito está autorizado a executá-la. Assim, quando os parâmetros de ativação são enviados, uma parte deles é checada contra as templates, este cheque é considerado o coração do mecanismo de proteção. Cada procedure executa em um módulo independente, mas pode se comunicar com outras e acessar dados remotamente, se possuir capabilities para isso.

LNS (espaço de nomes locais) é o registro do ambiente de execução de uma procedure, agindo como um domínio de execução dinâmica. Existe um LNS para cada chamada da procedure o qual desaparece quando esta é terminada. Cada objeto que aparece no LNS possui os direitos de acesso correspondentes a uma combinação das capabilities in-

dependentes e dependentes de chamada. O objeto LNS é independente do objeto PROCEDURE, embora precise do último para a sua criação, assim, alterações no LNS não implicam em alterações na procedure, o que permite que procedures sejam reentrantes e recursivas. Uma vez que cada LNS é formado por capabilities provenientes de duas fontes diferentes, procedures podem ter maior liberdade que o sujeito que a ativou, a qual pode ser aumentada se os objetos referenciados pela LNS tiverem outras capabilities na sua parte de representação.

PÁGINA é um objeto imagem da página de 4 K palavras do hardware. Cada entrada na C-lista de um LNS contém uma capability para a página correspondente e o conjunto de todas as páginas é o espaço de endereçamento do LNS.

PROCESSO é a unidade de processamento assíncrono do HYDRA. É a menor entidade que pode ser independentemente esquedulada para execução, normalmente correspondendo a um programa de usuário. O domínio de proteção corrente de um processo pode mudar várias vezes durante a execução, tantas quantas forem chamadas a procedures. Cada vez que uma procedure é invocada, um novo LNS é criado, portanto, um processo é na realidade uma pilha de LNS's que representam o estado acumulativo de uma tarefa sequencial. Como o HYDRA trabalha realmente em multiprocessamento, várias procedures do mesmo processo podem ser executadas simultaneamente e vários processos podem executar em paralelo, comunicando-se e sincronizando-se através de semáforos e ports.

SEMÁFORO é um objeto utilizado para a sincronização de processos, com operações P, V e P-condicional definidas ao estilo de Dijkstra (Dij 68).

PORT é um objeto básico do sistema de comunicação de mensagens entre processos. Age como um centro de chaveamento de mensagens, e se comunica com outros PORTS através das conecções de canais que enviam e recebem mensagens.

DISPOSITIVO é a representação do dispositivo físico de I/O. É tratado no HYDRA como uma variação de PORT cujas funções são apenas conectar e desconectar o PORT.

POLÍTICA são objetos que agem como caixas de correio entre o núcleo e os sistemas de política responsáveis pelo schedulling de processos. No HYDRA a separação entre "mecanismos de proteção" e "política de segurança" era tão primordial que chegou a originar um objeto protegido com a função de implementar a política.

DADOS são objetos do usuário usados apenas como portadores de dados que precisam de proteção especial sem que seja necessária a definição de um subsistema específico para isso. Não podem conter C-listas.

UNIVERSAL são objetos como os dados, com a diferença de que contêm C-listas, podendo agir como portadores de dados ou de capabilities.

TIPO é o objeto que provê os mecanismos necessários para a criação e proteção de extensões de objetos. Objetos TIPO podem representar subsistemas completos. Suas C-listas contêm capabilities para todas as operações do sistema. Extensões de objetos são criadas a partir de uma árvore em cuja raiz está um objeto TIPO, criando tipos diferentes ou adicionando objetos aos tipos existentes. Lembrando que cada objeto tem 3 partes: nome, tipo e representação, o ramo de cada tipo é um objeto cujo nome é o tipo dos objetos que lhe são superpostos. Como na estrutura da fig. B.1.

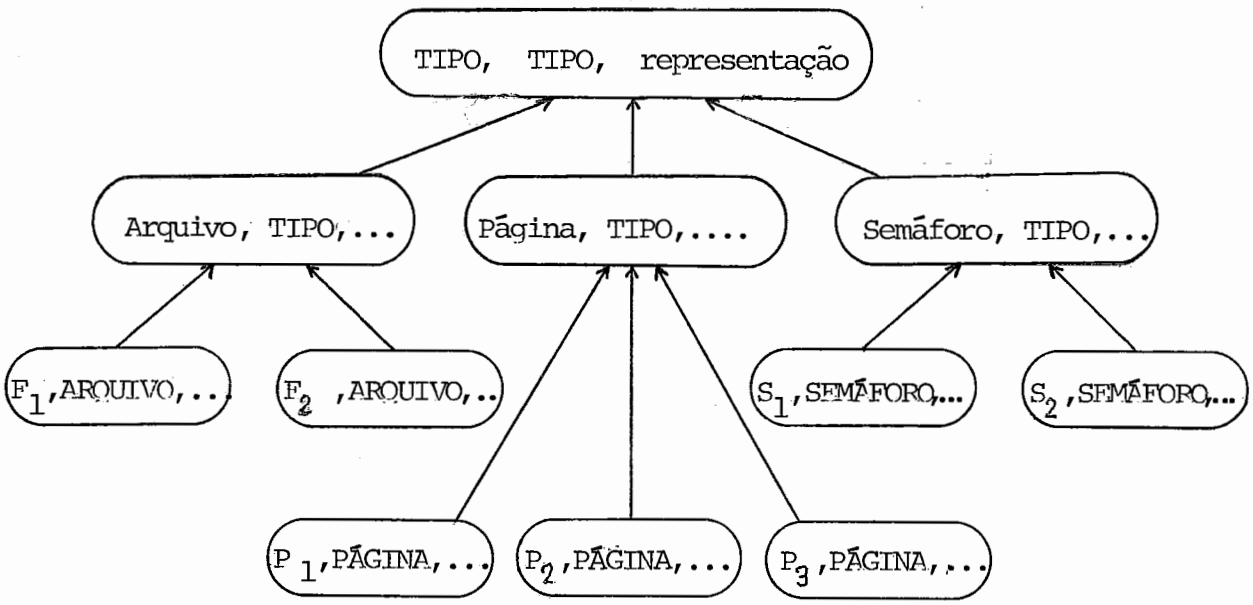


Fig. B.1 Extensões de objetos no HYDRA

Desta forma qualquer objeto ou sistema protegido pode ser criado. Por exemplo, pode-se criar um objeto DIRETÓRIO que contenha uma lista de ARQUIVOS e um SEMÁFORO interno para controlar operações mutuamente exclusivas no diretório.

Essa extensibilidade provida pelo HYDRA é provavelmente a sua melhor característica. Enquanto em outros sistemas os usuários tem que se adaptar às características existentes, o usuário do HYDRA constroi seu sistema operacional, seus subsistemas protegidos, seus objetos segundo as suas próprias políticas de alocação de recursos.

Uma certa dificuldade circundou a utilização do HYDRA por programadores ordinários, incapazes de utilizar os mecanismos do sistema em toda a sua extensão. Sugestões para a montagem de subsistemas protegidos foram dadas (CoJ 75) para incentivar o uso das versatilidades do sistema em toda a sua amplitude.

B.2 MEDUSA

O MEDUSA, da Carnegie - Mellon University, é um sistema operacional para um hardware distribuído. A estrutura desse hardware se assemelha à estrutura física de uma rede: são 50 módulos autônomos, reunidos em 5 clusters, cada qual presidido por um controlador de comunicação Kmap (fig. B.2). Um módulo contém um microprocessador LSI-11, 64 ou 128 K bytes de memória e não necessariamente, alguns dispositivos de I/O (fig. B.3). Cada módulo é conectado à estrutura através de uma chave, Slocal. É a presença da Slocal, em vez de um dispositivo de I/O que distingue o MEDUSA de uma rede. Tabelas na Slocal permitem decidir se as referências à memória estão na memória local ou se devem ser encaminhadas ao Kmap para serem resolvidas.

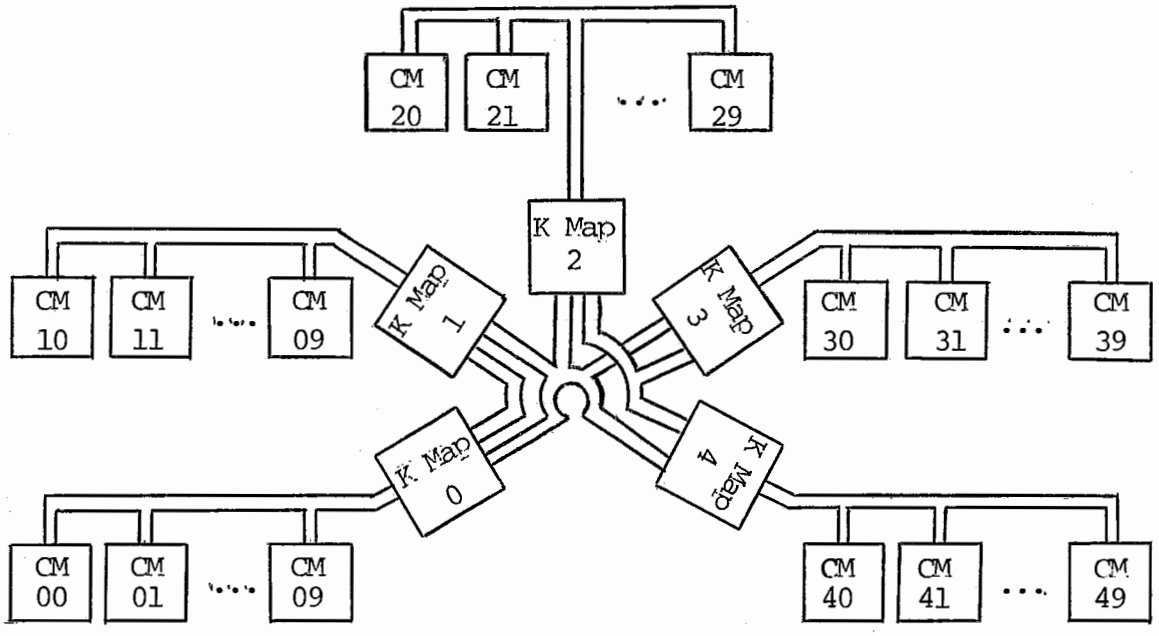


Fig. B.2 Sistema distribuído MEDUSA

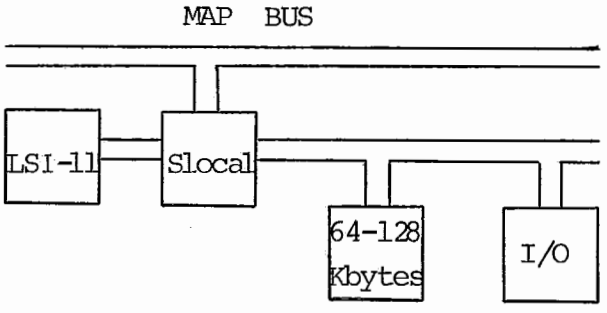


Fig. B.3 Organização de um CM

Os vários Kmaps microprogramados cooperam entre si para prover as funções básicas de comunicação entre processadores. Estas incluem referências às memórias, operações simples em descritores, trans-

ferência de blocos e operações nas mensagens entre processadores. A comunicação interprocessual é feita através de mecanismos que permitem a cada módulo ter seu próprio "ambiente de execução protegido". Os limites entre módulos são rigidamente estabelecidos e a estrutura do controle é totalmente distribuída. Como no HYDRA, não existe hierarquia e nenhum módulo é superior a outro. As funções do sistema operacional estão divididas em um conjunto de utilitários que podem ser duplicados em vários módulos mas não necessariamente em todos.

Existem 3 classes de objetos protegidos: objetos de memória, objetos de comunicação e objetos de utilitários.

PÁGINAS são objetos de memória associados a uma das 16 partes do espaço de endereçamento virtual de 64 K bytes de uma atividade. Podem ser lidas ou escritas utilizando as referências de memória dos processadores LSI-11.

Existem 2 tipos de objetos de comunicação: PIPES e SEMÁFOROS. PIPES são os portadores de mensagens entre os vários módulos. São utilizados da seguinte forma: se um processador necessita uma função de outro, envia-lhe uma mensagem, num PIPE com os parâmetros de invocação da função e uma indicação de outro PIPE, que deverá conter a resposta. A identidade do emitente acompanha a mensagem, mas descritores e outras informações protegidas nunca são transmitidos. Assim, cada vez que um PIPE é recebido, o processador verifica se o processo emitente está autorizado a utilizar os recursos necessários para a complecção da função solicitada e em caso positivo ela é executada e o resultado é enviado no PIPE de retorno. Apenas o Kmap microprogramado tem competência para modificar a estrutura interna de um PIPE. SEMÁFOROS são utilizados para sincronizar atividades.

A terceira classe de objetos protegidos é implementada pelos UTILITÁRIOS do sistema operacional. Um pequeno "núcleo" de funções é replicado em cada módulo para suportar as funções principais de multiplexação e gerência de dispositivos de I/O. Além desse núcleo, existe também nos módulos a codificação dos UTILITÁRIOS, que são distribuídos pela estrutura, mas não necessariamente em cada nó. Os UTILITÁRIOS provem suporte para os seguintes objetos: FORÇAS-TAREFAS, LISTAS DE DESCRITORES, BLOCOS DE CONTROLE DE ARQUIVOS e outros. Programas de usuários podem solicitar aos UTILITÁRIOS que façam modificações nesses objetos.

Cada programa de usuário entra no MEDUSA como uma FORÇA-TAREFA. Os próprios UTILITÁRIOS do sistema podem ser interpretados como FORÇA-TAREFAS. Quando uma FORÇA-TAREFA é criada, suas atividades são alocadas estaticamente em vários processadores individuais (vários módulos) que contém a codificação necessária para executar cada atividade. Se as atividades existentes se tornam sobrecarregadas, elas criam novas atividades para assumir parte do trabalho. Quando as atividades se tornam subaproveitadas, elas entregam seu trabalho a

outras e se deletam a si próprias. Processadores individuais podem ser multiplexados entre várias atividades pertencendo a diferentes FORÇAS-TAREFAS. Cada FORÇA-TAREFA possui um conjunto de objetos protegidos que são manipulados pelas suas várias atividades e controlados pelas LISTAS DE DESCRITORES.

LISTAS DE DESCRITORES são objetos providos por utilitários que controlam o acesso aos objetos de uma FORÇA-TAREFA. Existem 4 tipos de listas de descritores: PDL, SDL, UDL e XDL.

PDL é a lista de descritores privativos de uma atividade em execução num módulo qualquer. Cada PDL está sempre localizada no módulo que executa a atividade. Contém os objetos (programas, pilhas, etc) que aquela atividade pode acessar. Contém também descritores para um ou mais PIPES através dos quais é feita a comunicação com os outros módulos.

SDL é a lista de descritores compartilhados, isto é, a lista dos objetos globais de uma FORÇA-TAREFA que podem ser acessados por cada uma das suas atividades. Objetos de PÁGINA geralmente pertencem à SDL, pois só pode existir 1 descritor para cada página, e estas podem ser compartilhadas por atividades da mesma FORÇA - TAREFA mas não de FORÇAS-TAREFAS diferentes.

UDL é a lista de descritores de utilitário, usada na comunicação entre atividades e utilitários, por exemplo para fornecer parâmetros. Contém descritores de PIPES, já que estes formam o único meio de comunicação interprocessual. Cada módulo contém uma UDL fixa desde o projeto do sistema e dividida em duas partes: os descritores que podem ser utilizados por atividades de qualquer FORÇA-TAREFA e os descritores que só podem ser utilizados por atividades de utilitários.

XDL é a lista de descritores externos usada pelos utilitários do sistema para obter privilégios especiais na forma de acesso às listas PDL e SDL dos objetos dos usuários. Dessa forma, ao serem invocados os utilitários adquirem os direitos de acesso das atividades que os invocam, sem ser necessária a perigosa transferência de descritores dentro dos PIPES de comunicação.

Existem planos para a inplementação de uma quarta classe de objetos protegidos no MEDUSA, gerenciada por programas ao nível do usuário, mas esta ainda não foi definida.

O MEDUSA é um excelente exemplo de como atingir eficiência e proteção com simplicidade. Com poucos objetos de proteção, o mecanismo de segurança é construído basicamente através da distribuição do hardware e de um mecanismo de comunicação que impede a ocorrência de efeitos colaterais (fluxo implícito de informação). Além disso é transparente a falhas de equipamento: nenhum processador é auto-suficiente, mas também não é indispensável. Se um módulo estiver defeituoso ou desconectado, a duplicação dos utilitários permite que suas funções sejam executadas por outros nós. Tentativas intencionais de espionar mensagens são infrutíferas, pois o conteúdo de cada mensagem não é sensitivo e tentativas de alterar propositadamente o conteúdo de uma PIPE são impossíveis, pois apenas os Kmap estão autorizados a fazê-lo. Assim, vários usuários podem utilizar simultaneamente o sistema e até compartilhar o uso de processadores em diferentes programas, pois cada atividade executa num domínio protegido (composto pelas suas listas de descritores) e a comunicação é realizada de uma maneira absolutamente protegida.

B.3 SIGMA

O SIGMA é um sistema de processamento de mensagens desenvolvido pela MITRE CORPORATION com recursos da Marinha americana e implementado num computador PDP-10 com sistema operacional TENEX (AmO78). É um dos vários projetos militares relativos ao processamento de mensagens (MMP) em ambiente multinível militar.

Por MMP (Sistema de Processamento de Mensagens Militares) se entende um sistema acoplado a uma rede de propósitos gerais que utiliza alguns nós exclusivamente para o processamento de tipos prescritos de textos de mensagens segundo padrões definidos. Uma rede pode conter vários computadores do MMP, que estão restritos a comunicarem-se apenas entre si, formando uma subrede segura dentro da rede insegura (PBN 79).

Um sistema de MMP deve ser capaz de compor, distribuir, selecionar e analisar complexos textos de mensagens. Tratando-se da implementação da política não-discrecionária multinível, um MMP deve realizar essas funções obedecendo às propriedades simples e estrela, sem degradar a informação. Para simplificar o controle de fluxos entre os vários níveis de segurança a arquitetura do SIGMA inclui um núcleo de segurança certificado, reduzindo o número dos fluxos que devem ser acompanhados.

Estruturalmente o SIGMA é composto pelo núcleo e mais 5 processos que executam sob o seu controle: um processo privilegiado e 4 processos não-privilegiados correspondentes aos 4 níveis de classificação de mensagens (ultra-secreto, secreto, confidencial e não-classificado). Cada um desses 4 só tem permissão para escrever dados em objetos do seu nível (propriedade estrela) e ler apenas no seu nível ou em inferior (propriedade simples). O processo privilegiado pode atuar em vários níveis, tendo permissão para violar as regras de segurança de uma maneira controlada.

O SIGMA foi projetado com 2 idéias em mente: a primeira é segurança, a segunda é simplicidade de uso. Levando em conta que a maioria dos usuários do sistema não são especialistas em computador, é necessário que a interface com o usuário seja agradável e lhe forneça uma pintura real do que está acontecendo na máquina, de forma o usuário não se sinta coagido a ultrapassar os controles para encontrar um caminho mais fácil.

O TERMINAL utilizado pelo SIGMA foi projetado especialmente para ele. A necessidade de permitir mudança de nível durante a mesma sessão forçou a divisão da tela do terminal em várias janelas lógicas, cada qual correspondente a um terminal lógico e podendo ter diferentes níveis de segurança. Cada "janela" está também subdividida em domínios que possuem vários atributos: receptível, editável, sublinhável, etc. Dois conjuntos de 4 lâmpadas foram adicionados ao terminal, o primeiro montado no teclado, especifica em que nível de

segurança o cursor está no momento, o segundo, montado ao lado da tela, informa o nível máximo da informação exposta em toda a tela (fig B.4).

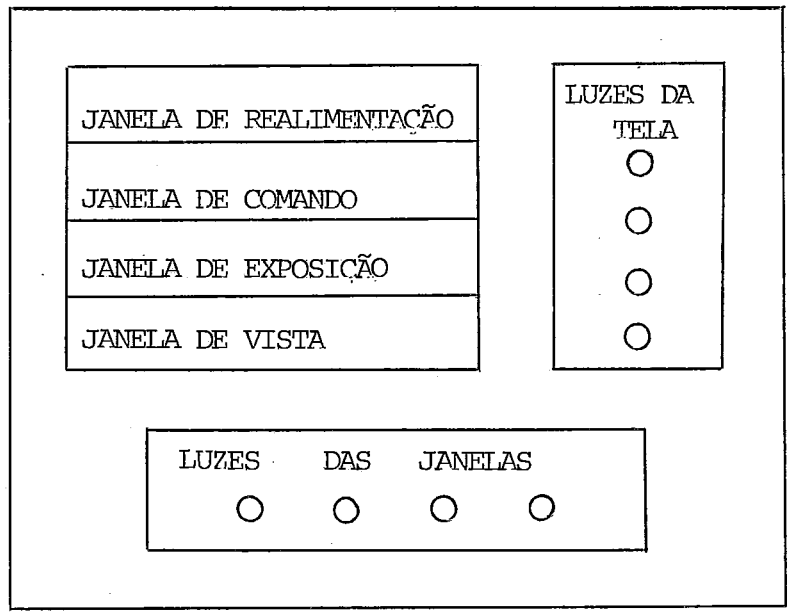


Fig. B.4 Terminal multinível do SIGMA

A característica mais original do SIGMA é a dicotomia de estados que cria durante cada sessão de terminal. A cada usuário são atribuídos 2 estados, o primeiro mantido pelo processo NÃO-CLASSIFICADO para poder fornecer informações a todos os outros processos e o segundo mantido pelo processo correspondente ao nível de classificação do arquivo acessado. O processo NÃO-CLASSIFICADO mantém as informações do estado da sessão, chamado CONTEXTO CORRENTE. Tenta "compreender" os objetivos do usuário e interpretar o seu comportamento em cada situação. Inclui a lista corrente de objetos acessados, funções realizadas e outras informações. O outro processo contém a lista de entradas nas mensagens do arquivo solicitado.

O problema de SUPERCLASSIFICAÇÃO da informação foi resolvido no SIGMA através da autorização que cada usuário possui para modificar o nível das mensagens às quais tem acesso. Para isso, usa o processo privilegiado, o único que tem competência para trabalhar em vários níveis e diminuir a classificação das mensagens. O sistema confia inteiramente nos seus usuários e isto não é uma falha, mas uma representação da situação real existente fora dos domínios do computador.

Além disso, o SIGMA provê 2 formas de comunicação interprocessual: preemptiva (por interrupções) e não-preemptiva (por mensagens), esta última podendo transmitir pequenas mensagens ou grandes mensagens como cadeias inteiras de comandos. O sistema de arquivos é simples, já que não há necessidade de estruturas hierárquicas como di

retórios, um grande arquivo plano de mensagens é suficiente. E apenas 3 tipos de privilégios são suficientes para manter o controle sobre o sistema: o que permite a reclassificação de textos, o que restringe a liberação de mensagens a um grupo selecionado e o privilégio do gerente de segurança para incluir e inicializar novos usuários.

O projeto do SIGMA demonstrou que é possível construir um sistema seguro de processamento de mensagens baseado na aproximação de núcleo de segurança. As técnicas usadas podem ser diretamente aplicadas ao projeto de outros sistemas orientados para transações como os sistemas de banco de dados.

B.4 XNOS

O EXTENDED NETWORK OPERATING SYSTEM foi desenvolvido no Centro de Desenvolvimento da Força Aérea americana e implementado no NBS para atuar na ARPANET. É um sistema operacional de rede de controle descentralizado e do caso aumentativo, pois utiliza máquinas de interface (XNIM) entre cada hospedeiro (ou um cluster de hospedeiros) e a rede (KWF 78).

Cada XNIM é um PDP-11/45 rodando sob UNIX. Está ligado à ARPANET através de uma versão ligeiramente modificada do NCP e age como um ponto focal para a segurança da rede. Algumas funções de proteção normalmente desempenhadas pelos hospedeiros são transferidas para o XNIM, como autenticação, manutenção do espaço de nomes de arquivos e profiles, etc. Pela primeira função, usuários podem ser autenticados a sistemas, sistemas a usuários, sistemas a sistemas, etc. Pela segunda, podem ser empregados nomes únicos em toda a rede, simplificando o mapeamento. Além disso, a maioria das funções de controle de acesso são realizadas pelo XNIM, que pode também funcionar como um nó independente. Isto foi, com efeito, realizado na configuração inicial do XNOS (fig. B.5).

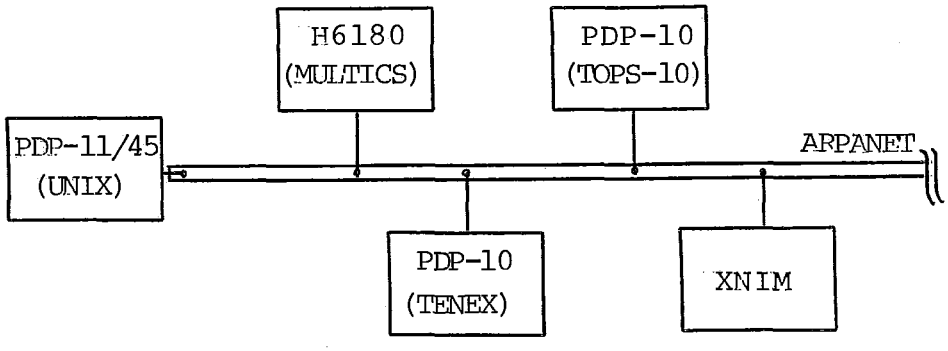


Fig. B.5 - Configuração inicial do XNOS

O XNOS foi projetado para ser capaz de suportar tanto políticas discrecionárias como não-discrecionárias de controle de acesso. Para isso, baseia-se em 3 conjuntos de informações: nível de segurança (de sujeitos), nível de segurança de objetos (recursos da rede) e direitos específicos que usuários possuem para acessar recursos discrecionariamente.

Todos os usuários da rede estão incluídos no PROFILE de níveis de segurança. Como nos sistemas isolados, um nível de segurança é composto por um nível de classificação (ultra-secreto, secreto, confidencial e público) e por uma categoria (contratos do governo, fo-

lha de pagamento, etc). As propriedades simples e estrela são válidas, implementando assim a política multinível, que inclui controle de fluxo.

Para suportar controle de acesso discrecionário cada arquivo possui LISTAS DE ACESSO similares às do MULTICS, contendo o nível de segurança do objeto e os direitos de acesso de cada usuário. Os proprietários dos arquivos podem definir até quatro modos de acesso: ler, escrever, executar e adicionar, para qualquer usuário, inclusive para si próprio (fig. B.6).

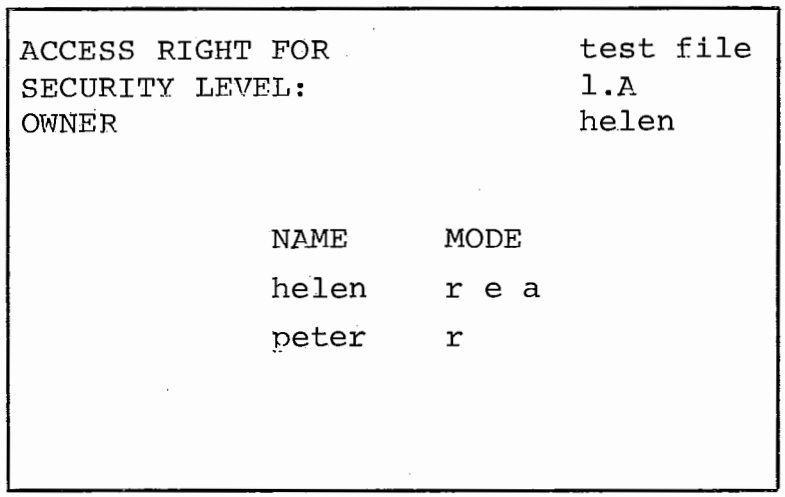


Fig. B.6 - Lista de acesso de um arquivo no XNOS

Como um sistema operacional de rede, o XNOS deve ser capaz de executar jobs de rede. Um JOB DE REDE é uma coleção de steps, cada qual especificando o programa que será executado, o nó da execução, arquivos de entrada e saída e todas as informações que se fizerem necessárias. Um exemplo simples de job de rede é:

```
RUN    PROG∝      ,   HOST A
IN 1 = INPUT 1    ,   IN2 = INPUT 2
OUT 1 = OUTPUT 1  HOST 1   ,  OUT 2 = OUTPUT 2   HOST 2
```

Neste exemplo, o PROG∝ deverá ser rodado no computador HOST A, terá como entrada os arquivos INPUT 1 e INPUT 2 e como saída os arquivos OUTPUT 1 e OUTPUT 2 cuja localização final deverá ser, respectivamente, HOST 1 e HOST 2. Antes de cada step ser iniciado todos os arquivos de entrada são verificados se residem no nó da execução. Caso contrário, é iniciada a MIGRAÇÃO de dados para este nó e reversamente, após o término do step, os arquivos de saída que forem dirigidos para outros nós iniciam a migração para estes.

Um ponto importante do controle de acesso diz respeito ao MOMENTO em que as decisões de acesso são tomadas. A maioria das decisões de acesso são feitas durante a EXECUÇÃO de cada step do job de rede. Entretanto, para minimizar o overhead de processamento e co-

municação, no início do job todos os arquivos de entrada e saída tem os seus níveis de segurança comparados ao do usuário. A partir daí é que as operações de migração são iniciadas.

Entre sistemas heterogêneos (ou seja: hospedeiros heterogêneos), as operações de transferência de dados são realizadas levando em conta 4 características:

. Estrutura do registro e ordem de armazenamento dos seus elementos de dados.

. Tipos de elementos de dados.

. Representação de elementos de dados (dependem do sistema e do compilador).

. Nomes de elementos de dados (podem haver diferenças de nomes entre fonte e destino).

No caso de acesso a arquivos ao nível de ELEMENTO DE DADOS devem haver tanto controles discrecionários como não-discrecionários. Devido ao grande número de informações de acesso que são necessárias neste caso a NBS está desenvolvendo um Gerente de Banco de Dados, o XNDM. Este possui mecanismos discrecionários e não-discrecionários e permite a distribuição da informação de acesso aos sistemas hospedeiros. Portanto, ao processar uma pergunta emitida por um programa fonte S, seus direitos são investigados e transmitidos com a pergunta através da rede e são comparados no nó destino com os requisitos de acesso aos dados solicitados.

O maior problema ao implementar controle de acesso dependente de dados numa grande rede é que não se sabe até que ponto se pode confiar no sistema operacional de cada hospedeiro, que na maioria das vezes é formado por codificação não verificada. Por isso, toda a informação de controle do XNDM é residente na interface XNIM e os argumentos que devem ser enviados para estabelecer o nível de segurança são análogos aos utilizados no acesso simples a arquivos de dados.

O CUSTO adicional decorrente do uso de máquinas de interface em redes (como os XNIM) depende do tamanho de cada máquina e do seu número. Uma estimação inicial sugere que se utilize um computador de interface para cada classe de sistemas operacionais. O aumento na eficiência do sistema ao deslocar funções de autenticação e comunicação com a rede gera uma economia em custo comparável ao custo da introdução da máquina de interface.

O XNOS foi um sistema desenvolvido com propósitos de segurança e contou com o apoio de muitos órgãos governamentais, por isso, muitas características que foram possíveis nele talvéz não sejam custo efetivas num sistema comercial ou acadêmico. Além de prover um alto nível de segurança permite a implementação de vários tipos de po

lítica sem modificações nos sistemas hospedeiros. Essa última característica é a sua vantagem mais notável, pois possibilita a conecção de sistemas completamente heterogêneos sem que estes sofram adaptações no seu funcionamento normal e fornece aos seus usuários meios de acessar remotamente qualquer recurso da rede sem precisar pagar em aumento de overhead.

BIBLIOGRAFIA

A. Artigos

1. ABS 74 - "Interprocess Communication Facilities for Network Operating Systems"
   E.Akkoyunly, A.Bernstein, R.Schantz
   Computer Jun/74, pags. 46-55

2. AmO 78 - "Design of a message processing system for a multi level secure environment"
   S.Ames, D.Oestreicher
   NCC 1978, pags. 765-771

3. AMP 76 - "Penetrating an operating system: a study of VM/370 integrity"
   C.Attanasio, P.Markstein, R.Phillips
   IBM Systems J. Vol. 15 # 1, pags. 102-116

4. Att 79 - "Virtual Control Storage - security measures in VM/370"
   C.Attanasio
   IBM Systems J. Vol. 18 # 1, pags. 93-110

5. BaH 80 - "Data element security and its effects on file segmentation"
   Y.Babad, H.Hoffer
   IEEE Transactions on Software Engineering Sep/80, pags. 402-410

6. BCD 72 - "The Multics Virtual Memory: concepts and design"
   A.Bensoussan, C.Clingen, R.Daley
   CACM May/72, pags. 308-318

7. BeB 79 - "KSOS - Development methodology for a secure operating system"
   T.Berson, G.Barksdale
   NCC 1979, pags. 365-371

8. Bec 80 - "A Security Mechanism for Statistical Databases"
   L.Beck
   ACM Trans. Database Sep/80, pags. 316-338

9. Ber 80 - "Security and protection of data in the IBM System /38"

V.Berstis
Proc. 7th. Symposium on Computer Architecture May /80, pags. 245-252

10. BES 79 - "Design of Distributed Multiprocessor Operating System"

R.Bauman, T.Epinay, G.Schrott
Computer in Industry 1979 # 1, pags. 117-121

11. Bra 77 - "Encryption protection in computer data communications"

B.Branstad
Proc. Computer Security and Integrity Symposium May/77, pags. 9.25-9.31

12. CaB 80 - "A Formal System for Reasoning about Programs Acessing a Relational Database"

M.Casanova, P.Bernstein
ACM Trans. Database Jul/80, pags. 386-414

13. CaD 79 - "KSOS - The design of secure operating system"

E.MacCauley, P.Drongowski
NCC 1979, pags. 345-353

14. CGT 75 - "Views, authorization and locking in a relational data base system"

D.Chamberlin, J.Gray, I.Traiger
NCC 1975, pags. 425-430

15 ChS 76 - "On virtual machine integrity"

C.Chaudersekaran, K.Shankar
IBM Systems J. Vl. 15 # 3, pags. 264-278

16. CMM 72 - "On the implementation of Security Measures in Information System"

R.Conway, W.Maxwell, H.Morgan

17. CoJ 75 - "Protection in the Hydra Operating System"

E.Cohen, D.Jefferson
Proc. 5th. Symposium on Operating Systems
Austin, Texas 1975, pags. 141-160

18. CoM 77 - "Approaches to controlling personal acess to computer terminals"

I.Cotton, P.Meissner
Proc. Computer Security and Integrity Symposium
May/77, pags. 9.12-9.19

19. Coo 78 - "The cost of using the CAP computer's protection facilities"

D.Cook
Operating Systems Review Apr/78, pags. 26-30

20. DDS 79 - "The Tracker: A Threat to Statistical Database Security"

D.Denning, P.Denning, M.Schwartz
ACM Trans. Database Mar/79, pags. 76-96

21. DeD 79 - "Data security"

D.Denning, P.Denning
Computing Surveys Sep/79, pags. 227-249

22. Den 71 - "Third Generation Computer Systems"

P.Denning
Computing Surveys Dec/71, pags. 175-216

23. Den 76a - "A Lattice Model of Secure Information Flow"

D.Denning
CACM May/76, pags. 236-243

24. Den 76b - "Fault Tolerant Operating Systems"

P.Denning
Computing Surveys Dec/76, pags, 359-389

25. Den 78 - "Secure Personal Computing in an Insecure Network"

D.Denning
CACM Aug/79, pags. 476-482

26. Den 80 - "Secure Statistical Databases with Random Sample Queries"

D.Denning
ACM Trans. Database Sep/80, pags, 291-315

27. DeS 80 - "A Fast Procedure for Finding a Tracker in a Statistical Database"

D.Denning, J.Schlörer
ACM Trans. Database Mar/80, pags. 88-102

28. DeV 66 - "Programming Semantics for Multiprogrammed Computations"

J.Dennis, E.Van Horn
CACM Mar/66, pags. 143-155

29. Dew 78 - "Systems auditability and control in an E F T S environment"

R.Dewey
NCC 1978, pags. 185-189

30. DiH 79 - "Privacy and Authentication: An Introduction to Cryptography"

W.Diffie, M.Hellman
Proc. IEEE Mar/79, pags. 397-427

31. Dij 68 - "The Structure of the T H E - Multiprogramming System"

E.Dijkstra
CACM May/68, pags. 341-346

32. DJL 79 - "Secure Databases: Protection Against User Influence"

D.Dobkin, A.Jones, R.Lipton
ACM Trans. Database Mar/79, pags. 97-106

33. DoP 79 - "Data Base Management Systems Security and INGRES"

D.Downs, G.Popek
Proc. 5th. International Conf. on Very Large Data Bases Rio de Janeiro Oct/79, pags. 280-290

34. Fab 73 - "Dynamic Verification of Operating Systems Decisons"

R.Fabry
CACM Nov/73, pags. 659-668

35. Fab 74 - "Capability based addressing"

R.Fabry
CACM Jul/74, pags. 403-412

36. Fag 78 - "On an Authorization Mechanism"
R.Fagin
ACM Trans. Database Sep/78, pags. 310-318

37. Fen 74 - "Memoryless subsistems"
J.Fenton
Computer Journal May/74, pags. 143-147

38. FeN 79 - "The foundations of a provably secure operating system (PSOS)"
R. Feiertag, P.Newmann
NCC 1979, pags. 329-334

39. Feu 72 - "The Rice Research Computer - A tagget architecture"
E.Feustel
SJCC 1972, pags. 369-377

40. FiG 79 - "Data Management for the Distributed Processing Programming Executive (DPPX)"
A.Fitzgerald, B.Goodrich
IBM Systems J. Vol 18 # 4, pags. 547-564

41. GKS 79 - "Security in a multi-level structured model of a data base"
E.Gudes, H.Koch, F.Stahl
Computer Journal Nov/79, pags. 303-306

42. GLP 79 - "A security retrofit of VM/370"
B.Gold, R.Linde, R.Peeler, M.Schaefer, J. Scheid, P.Ward
NCC 1979, pags. 335-344

43. Gra 68 - "Protection in an Information Processing Utility"
R.Graham
CACM May/68. pags. 365-369

44. GrD 72 - "Protection - Principles and practice"
G.Graham, P.Denning
SJCC 1972, pags, 417-429

45. GrW 76 - "An Authorization Mechanism for a Relational Data base System"
P.Griffiths, B.Wade
ACM Trans.Database Sep/76, pags. 242-255

46. Gud 80 - "The Design of a Cryptography Based Secure File System"

E.Gudes
IEEE Trans. Soft. Eng. Sep/80, pags. 411-420

47. GWM 75 - "An access control mechanism for computing resources"

H.Gladney, E.Workey, J. Myers
IBM Systems J. Vol. 14 # 3, pags. 212-225

48. Han 70 - "The nucleus of a multiprogramming System"

B.Hansen
CACM Apr/70, pags, 238-250

49. HDT 79 - "On foiling computer crime"

M.Hellman, G.Davida, W.Tuchman, D.Branstad
IEEE Spectrum Jul/79, pags. 31-41

50. Hel 78 - "Security in communication networks"

M.Hellman
NCC 1978, pags. 1131-1134

51. Her 78 - "A new protection architecture for the Cambridge Capability Computer"

A.Herbert
Operating Systems Review Jan/78, pags. 24-28

52. HRU 76 - "Protection in Operating Systems"

M.Harrison, W.Ruzzo, J.Ullman
CACM Aug/76, pags. 461-471

53. JaI 80 - "Microcomputers as Protective Interfaces in Computing Networks"

E.James, D.Ireland
Software - Practice and Experience Dec/80, pags. 953-958

54. JCD 77 - "Software management of Cm* - A distributed multiprocessor"

A.Jones, R.Chansler, I.Durham, P.Feiler, K.Schwans
NCC 1977, pags. 657-663

55. JoL 75 - "The enforcement of security policies for computation"

A.Jones, R.Lipton
Proc. 5th. Symposium in Operating Systems Austin, Texas 1975, pags. 197-206

56. Jon 80 - "Capability Architecture Revisited"

A.Jones
Operating Systems Review Jul/80, pags. 33-35

57. KaU 77 - "A Model of Statistical Databases and Their Security"

J.Kam, J.Ullman
ACM Trans. Database Mar/77, pags. 1-10

58. Kie 79 - "An operating system for distrituted processing - DPPX"

S.Kiely
IBM Systems J. Vol. 18 # 4, pags. 507-525

59. KiM 76 - "A perspective on network operating systems"

S.Kimbleton, R. Mandell
NCC 1976, pags. 551-559

60. KWF 78 - "Network operating systems - An implementation approach"

S.Kimbleton, H.Wood, M.Fitzgerald
NCC 1978, pags. 773-782

61. Lam 69 - "Dynamic protection structure"

B.Lampson
FJCC 1969, pags. 27-38

62. Lam 73 - "A note on the confinement problem"

B.Lampson
CACM Oct/73. pags. 613-615

63. LaS 76 - "Reflections on an Operating System Design"

B.Lampson, H.Sturgis
CACM May/76, pags. 251-265

64. Lin 76 - "Operating System Structure to Support Security and Reliable Software"

T.Linden
Computing Surveys Dec/76, pags. 409-445

65. Lip 75 - "A comment on the confinement problem"

S.Lipner
Proc. 5th. Symposium on Operating Systems Austin, Texas 1975, pags. 192-196

66 Lis 72 - "The Design of the Venus Operating System"

B.Liskov
CACM Mar/72, pags. 144-149

67. Mil 76 - "Security Kernel Validation in Practice"

J. Millen
CACM May/76, pags. 243-250

68. Mor 73 - "Protection in Programming Languages"

J.Morris
CACM Jan/73, pags. 15-21

69. MPM 80 - "A Locking Protocol for Resource Coordination in Distributed Databases"

D.Menasce, G.Popek, R.Muntz
ACM Trans. Database Jun/80, pags. 103-138

70. NBS 77 - "Data Encryption Standard"

National Bureau of Standards
Federal Information Processing Standards Publication Jan/77, pags. 1-18

71. Nee 72 - "Protection systems and protection implementations"

R.Needham
FJCC 1972, pags. 571-578

72. Nee 79 - "Adding Capability Access to Conventional File Servers"

R.Needham
Operating Systems Review Jan/79, pags. 3-4

73. Neu 78 - "Computer system security evaluation"

P.Neuman
NCC 1978, pags. 1087-1095

74. OSS 80 - "Medusa: An Experiment in Distributed Operating System Structure"

J.Ousterhout. D.Scelza, P.Sindhu
CACM Feb/80, pags. 92-105

75. Par 72 - "On the criteria to be used in decomposing system into modules"

D.Parnas
CACM Dec/72, pags. 1053-1058

76 PBN 79 - "KSOS - Computer network applications"

M.Padlipsky, K.Bibba, R.Neely
NCC 1979, pags. 373-381

77. PKK 79 - "UCLA Secure Unix"

G.Popek, M.Kampe, C.Kline, A.Stoughton, M.Urban, E.Walton
NCC 1979, pags. 355-364

78. PoF 78 - "A Model for Verification of Data Security in Operating Systems"

G.Popek, D.Farber
CACM Set/78, pags. 737-749

79. PoK 74 - "Verifiable secure operating system software"

G.Popek, C.Kline
NCC 1974, pags. 145-151

80. PoK 78 - "Issues on Kernel design"

G.Popek, C.Kline
NCC 1978, pags. 1079-1086

81. PoK 79 - "Encryption and Secure Computer Networks"

G.Popek, C.Kline
Computing Survey Dec/79, pags. 331-356

82. Pop 74 - "Protection structures"

G.Popek
Computer Jun/74, pags. 22-33

83. Ros 69 - "Contemporary Concepts of Microprogramming and Emulation"

R.Rosin
Computing Surveys Dec/69, pags. 198-212

84. SAG 72 - "Project SUE as a learning experience"

K.Sevcik, V.Atwood, M.Grusjocow, R.Holt, J.Horning, D.Tsichritzis
FJCC 1972, pags. 331-338

85. SaG 78 - "A hardware architecture for controlling information flow"

H.Saal, I.Gat
Proc. 5th. Symposium in Computer Architecture, pags. 73-77

86. Sal 74a - "Protection and Control of Information Sharing in Multics"

J.Saltzer
CACM Jul/74, pags, 388-402

87. Sal 74b - "Ongoing Research and Development on Information Protection"

J.Saltzer
Operating Systems Review Jun/74, pags. 8-24

88. SaS 75 - "The Protection of Information in Computer Systems"

J.Saltzer, M.Schroeder
Proc. IEEE Sep/75, pags. 1278-1308

89. Sch 75 - "Engineering a security Kernel for Multics"

M.Schroeder
Proc. 5th. Symp. Operating Systems Austin, Texas 1975, pags. 25-32

90. Sch 78 - "Event and efectiveness models for simulating computer security"

E.Schelonka
IEEE COMPCOM Fail/78, pags.262-268

91. ScS 72 - "A hardware architecture for implementing protection rings"

M.Schroeder, J.Saltzer
CACM Mar/72, pags. 157-170

92. Sha 77 - "The Total Computer Security Problem: An Overview"

K.Shankar
Computer Jun/77, pages. 50-73

93. WaF 80 - "An Architecture for Support of Network Operating System Services"

R.Watson, J.Fletcher
Computer Network Feb/80, pags. 33-49

94. WCC 74 - "HYDRA; The Kernel of a Multiprocessor Operating System"

W.Wulf, E.Cohen, W.Corwin, A.Jones, R.Levin, C. Pierson, F.Pollack
CACM Jun/74, pags. 337-345

95. Wei 69 - "Security controls in the ADEPT-50 time-sharing system "

C.Weissman
FJCC 1969, pags. 119-133

96. WFS 80 - "Data base security: requirements, policies, and models"

C.Wood, E.Fernandez, R.Summers
IBM System J. Vol. 19 # 2, pags. 229-252

97. WiD 74 - "Data Security in the computer Communication Environment"

S.Winkler, L.Danner
Computer Feb/74, pags. 23-31

98. Wil 72 - "Design of the Burroughs B1700"

W.Wilner
FJCC 1972, pags. 489-497

99. WoK 79 - "Access control mechanisms for a network operating system"

H.Wood, S.Kimbleton

100. Woo 77 - "On-line passwords techniques"

H.Wood
Proc. Computer Security and Integrity Symposium
May/1977, pags. 9.20-9.24

101. Woo 79 - "Applications for multilevel secure operating Systems"

J.Woodward
NCC 1979, pags. 319-328

B. Livros

1. Beq 78 - "Computer Crime"

August Bequai
Lexington Books, 1978
D.C. Heath and Company
Lexington, Massachusets

2. BuS 78 - "Computer Control and Audit: A Total Systems Approach"

John G.Burch, Joseph L. Sardinas
John Wiley & Sons, Inc., 1978
Santa Barbara

3. Car 77 - "Segurança do computador"

John M. Carroll
Security World Publishing Co., Inc., 1977
Agents Editores Ltda., 1979
Rio de Janeiro

4. Hof 77 - "Modern methods for Computer Security and Privacy"

Lance J. Hoffman
Prentice - Hall, Inc., 1977
Englewood Cliffs, New Jersey

5. Ham 72 - "Computer Security"

Peter Hamilton
First published in Great Britain 1972
Auerbah Publishers Inc., 1973
Philadelphia, Pa.

6. KrG 79 - "Computer Fraud and Countermeasures"

Leonard I. Krauss, Aileen MacGahan
Prentice - Hall, Inc., 1979
Englewood Cliffs, New Jersey

7. Mar 73 - "Security, Accuracy and Privacy in Computer Systems"

James Martin
Prentice- Hall, Inc., 1973
Englewood Cliffs, New Jersey

8. Org 73 - "Computer System Organization: The B5700/B6700 Series"

E. Organick
Academic Press, 1973
New York

9. Pra 75 - "Programming Languages: Design and Implementation"

Terrence W. Pratt
Prentice - Hall, Inc., 1975
Englewood Cliffs, New Jersey

10. Pri 78 - "Computer security risk management in action"

JAT Pritchard
NCC Publications, 1978
Manchester, England

ÍNDICE DE PALAVRAS-CHAVE

Abstrações, grupos de 191
Abstrações, máquinas 97
Abstrato, construção de um modelo 103
Acesso, 6, 7, 55
— a Banco de Dados 123
— corrente 128
— independente de conteúdo 64
— lógico a arquivos 64
—, árvore de listas de 29
—, atributos de 18, 25, 31
—, autorização de 159
—, chave de 173, 180
—, controle de 9, 38, 74
—, grupos de 189
—, indicador de 23
—, listas de 22, 28, 31, 40, 69, 115, 165, 207
—, listas de controle de 28
—, matriz de 18, 19, 21, 69, 81, 87, 113, 128
—, mecanismo de 19
—, regras de 16, 18, 124, 125, 165
—, tipos de 124
Ações, sequência de 72
Acoplados, sistemas 153
Acordar 18
ADEPT-50 35, 39, 48, 170
Administrador de sistemas 29
Algorítmica, especificação 104
Algorítmicos, erros 14
Algorítmo de segurança 149
Alocação de hardware 9
— de memória 50
— de software 9
— de recursos 9
—, blocos de 178
Alteração da informação 6, 7
Alternativa corretiva 10
— preventiva 10
— corretivo-preventiva 11
American Bank Association 146
Amostragem aleatória de registros 138
Análise combinatorial 33
Anéis de proteção 40, 41, 78, 113, 176
Aproveitamento de hardware 13
Aproximação corretiva 10
— preventiva 10
— corretivo-preventiva 11
Armazenamento de fator de perturbação 137
ARPANET 20, 114, 154, 162, 184
Arquitetura etiquetada 53, 165
— hierárquica 40, 41, 57, 85, 95, 97, 98, 111, 191
— nucleada 95, 106
— plana 40, 85
Arquiteturas distribuídas 88

Arquivo, subtipo de 63, 186
Arquivos 165, 173, 186, 197
— cifrados 68
— de disco 180
— de núcleo 178
— estáticos 67
— voláteis 67
—, acesso lógico a 64
—, blocos de controle de 200
—, sistemas de 18, 30, 60, 121
—, subtipos de 186
Arredondamento 137
Árvore 60
— de criação 87, 88, 106, 107
— de listas de acesso 29
— de processos 107
—, estrutura de 108
ASAP 126
Assembler 56
Assimétrica, cifragem 150, 151
Assinatura 145
— eletrônica 151
Atividades, histórico de 139
ATLAS 50
Átomos de informação 69
Atributo, cópia de 25
—, criação de 20
—, deleção de 20
—, transferência de 20
Atributo de acesso 25
ATTACH 108
AUDIT 170
AUDIT-LIST 170
Auditabilidade 8
Aumentativo, caso 156, 157, 161
Autenticação 122,
— de capabilities 54
— de terminal 158
— de usuário 159
— de computador 159
— por característica física 145
— por memorização 143
— por objeto possuído 146
— reversa 146
—, cartão de 146
Auto-controle 28, 29
Autor 62
Autoridade 170
— hierárquica 37
—, níveis de 12
—, qualificador de 35
Autorização 6, 122
— de acesso 159
—, cheques duplicados de 33
— lista de 61
— nível de 42, 128

—, verificação de 81
Autorizar 20
Auxiliar, memória 50

Bancos de dados 60, 120, 166
— estatísticos 128, 135, 166
— relacionais 126, 129, 131, 166
—, controle de acesso ao 122, 123
—, gerente de 122, 208
Basic Language Machine 24, 53
BCC-500 24, 54, 83, 106, 172
Bell Telephon Company 145
Bell Telephon Laboratory 175
Berkeley Computer Corporation 172
Biblioteca de procedures 97
Binding-time 46
Bit de modificação 29
Bits de permissão 29
Blocos de alocação 178
Buferização automática 87
BURROUGHS 1700 57
BURROUGHS 5500 53
BURROUGHS 5700 53, 76
BURROUGHS 6700 24, 53, 76
BURROUGHS 7700 53
Byte tipo fechadura 53

C-lista 22, 26, 31, 69, 81, 108, 179, 182, 183, 195
Cache, memória de 51
Caixas de correio 87, 107
Caixas fechadas para capabilities 27
CALSPAN Corporation 146
CAL-TSS 26, 30, 31, 32, 62, 108, 167, 178
Camadas hierárquicas 106
Camadas, supervisor de 113
Cambridge, Universidade de 56
Canais 44
— cobertos 42, 44, 90
— de eventos 108, 178
— de I/O 57
— de memória 90
— legítimos 44, 90
Canal, linguagem de 116
CAP SYSTEM 83, 87
Capabilities 40, 51, 54, 76, 81, 106, 107, 111, 113, 165, 183, 191
— dependentes 83
— independentes 83
— próprias 92
—, comparação com listas de acesso 28
—, criação 24
—, deleção periódica 27
—, lista de 22, 81, 83, 165, 170
—, propriedades 24
Capability 16, 22, 23, 28, 31

— criptografada 69
— de entrada 81, 82
— em partição reservada 24
— específica 107
— etiquetada 24
— principal 27
— qualitativa 107
— quantitativa 107
— revogável 27
—, formato de 25
—, revogação de 26
—, transmissão de 26
Carnegie-Mellon University 195, 199
Cartão de autenticação 146
CASE, Sistema 46
Caso aumentativo 156, 157, 161
Casão não-aumentado 156, 157, 160
Categoria 37, 38, 42, 170
—, qualificador de 35
Cavalo de Tróia, problema de 89, 90
CDC-6400 108, 177
CDC-6600 74
CDC-7600 51
Central Memory 108
Centro de Segurança da Rede 157
Cerca de separação 25
Certificação 14, 47, 99, 101
— de núcleo 103, 104
Certificado, núcleo de segurança 116, 203
Chamada de interrupção 173
Chamada de monitor 87
Chave 68
— de acesso 173, 180
— de criptografia 149
— de modo 74
— eletrônica 146
— pública 150
Chaves de proteção 51
Chave-fechadura 22, 31, 165
—, combinação com capabilities 31, 33
Chaveamento de domínios 101
Cheques de sistema operacional 123
Chicago Magic Number Machine 24
Chips individuais 151
Cifra 148
Cifragem 148
— assimétrica 150, 151
— de recirculação 149
— simétrica 149
Classe de segurança 37, 38
Classes de segurança 42, 43, 46
Classes, limite superior das 46
Classificação, nível de 37, 38
CM* 87
CMMP (Sistema de multi-microprocessadores) 195
Cobertos, canais 42, 44, 90
Codificação 148

— de segurança 170
CODASYL 65, 126, 127
Código 148
Colaboração 165
Combinação de classes, operador de 43
Combinatorial, análise 33
Compartilhamento 85
— numérico 13
— seguro 51
Compartilhar recursos 9
Compartimentos de controle 38
Compatibilidade 153
Competência 10
Compilação, ligação durante a 46, 47
Compiladores 54
Completa, hierarquia 95
Completaridade 13
Completo, isolamento 30
Componentes de mecanismo de proteção 15
Computador
— autenticação de 159
— ligação de 159
— 56
Computadores, redes de 88, 149, 153
Comunicação 85, 159, 166
— entre usuários 154
— interprocessual 85, 107, 110, 155, 204
— não-preventiva 86, 204
— preemptiva 86, 204
—, mecanismos de 73, 110
—, protocolos de 153
Conceitos 6
Concentrado, sistemas de hardware 169
Condicional, estrutura 44, 46
Conexão entre processos 9
Confiabilidade 7, 14
Confinamento 89, 90, 91, 165
Conjunto
— de categorias 126
— de valores 129
Construção
— de modelo abstrato 103
— de núcleo 123
Consumidor, problema do produtor/ 85
Conteúdo
—, acesso dependente de 64
—, acesso independente de 64
—, agrupamento pelo 40
Context block 172
Contexto 189
— corrente 204
Control Protocol, Transmission 160
Controlador
— central de fechaduras 126
— de comunicação 199
— local de fechaduras 126
—, descritor de 190

Controle 155
— centralizado 155, 157, 166
— da rede 155
— da subrede 155
— de acesso 9, 18, 103
— de acesso em banco de dados 121
— de acesso, listas de 28, 170
— de arquivos, blocos de 200
— de fluxo 42
— de fluxo por vetores 44
— de incentivo, sistemas de 62
— de mapeamento, tabela de 83
— de progresso de um processo 9
— de risco 168
— descentralizado 156, 160
— distribuído 155
— do hospedeiro 155
— hierárquico 28, 29
—, direito de 20
—, espaço de 188
—, funções de 40
Cooperação 12, 85
Cópia, direito de 20
Cópia de atributo 18
Correio, caixas de 87, 107
Corrente, acesso 128
Corretiva
—, alternativa 10
—, aproximação 10
Criação 20
— de processo 9
—, árvore de 87, 88, 106, 107
Criptografia 56, 68, 142, 148, 150, 155, 158, 165
— controlada pelo usuário 69
— de link 151
— em redes, características especiais 151
— 'end-to-end' 151
Cursor 189

Dado 65, 95
Dados 197
— de projeção 125
— de segurança 170
— estatísticos, bancos de 128, 135, 166
— relacionais, bancos de 126, 129, 131, 166
—, acesso em banco de 123
—, banco de 60, 120, 166
—, compartilhamento de 85
—, comunicação de 141
—, controle de acesso em banco de 121
—, elementos de 208
—, espaço de 189
—, fluxo de 156, 165
—, fluxo implícido de 151
—, gerente de banco de 121, 208
—, índice de espaço de 189

—, migração de 154
—, modelo de gerência de 133
—, segmento de 60
—, serviço de reconfiguração de 154
—, vulnerabilidade de 141
Daemon, Network 119, 160, 187
DASD, 116
Data Encryption Standard 149
Data Management System 129
Data Mark Machine 46, 47, 91
DBSM 65, 121, 124
DBS-440 126
Deadlock 85
Decifragem 148
Decodificação 148
Decriptografia 148
Dedução por inferência 135
Degradação manual da informação 48
Deleção periódica de capabilities 27
Dependência humana 167
Dependente de conteúdo, acesso 64
Dependentes, capabilities 83
Depuração, Sistemas de 55
DES 149, 150
Descentralização 14, 141
Descentralizada, responsabilidade de proteção 22
Descentralizado, controle 160, 166
Descrição
— da rede 190
— de controlador 190
— de unidade lógica 190
Descritor 129
— de domínio 180
— de segmento 60, 75, 79
— removível 176
Descritores
—, lista de 201
—, segmento de 75, 165
Desempenho 13, 102
Desenvolvimento
— hierárquico 118
— modular 115
—, metodologia de 187
Deteção de erros 14
Dialoguer 112, 182
Digitais, Impressões 146
Dinâmica 153
—, hierarquia 40, 99
—, ligação 43, 47, 48
Directory Manager 192
Direito
— de controle 20
— de cópia 20
— de transferência 20
Direitos
—, administração de 81
—, revogação de 26
—, uso de 81

Direito, endereçamento 76
Diretórios 30, 32, 60, 61, 69, 112, 165, 173, 176, 180, 192, 197
Disco
— arquivos de 180
— objetos de 108
Discrecionaridade 7
Discrecionário, modelo 18
Discrecionários
—, chaveamento de domínios
—, mecanismos 17, 18, 22, 44, 106, 165
—, modelos 166
—, núcleos 106
Dispositivos 183, 186, 196
— de I/O 102
Dispositivo virtual 98
Dissociação 138
Distribuídas, arquiteturas 88
Distribuído
—, controle 155
—, hardware 92
—, processamento 157
Distribuídos, sistemas 140, 141, 146, 148, 166
Distributed Processing Executive 157
DMM 133
DMS da I.P. SHARP 129
DMS da SDC 129
Domínio 31, 172, 180
— de execução 16, 165
—, descritor de 180
Domínios 74, 107
— de execução 73, 78, 81, 83
— de proteção 76
— definidos por capabilities 81
— protegidos 27, 108
—, chaveamento de 101
—, pequenos 81
—, múltiplos 81
DOS 98
DPPX 157
DRS 154
Duplicação de informação 33

EAI-8400 53
Economia 13
Econômicas, limitações 107
ECS 108, 178
Eficiência de desempenho 13
Elemento 64, 65, 66
— de dados 208
Eletromagnéticos, meios 142
Eletrônica
—, assinatura 151
—, chave 146
Emissor, identificação de 86
Emulados de UNIX 118, 185
Entry-points 54

Entrada, capability de 81, 82
Entradas fantasmas 60
End-to-end, criptografia 151
Endereço 50
—, pointer com 23
Endereços virtuais 57
Endereçamento 50
— direto 76
— indireto 76
—, espaço de 51, 57
Energia atômica 38
Equipamento de comunicação de dados 141, 142
Equações simultâneas 137
Erros 14
— algorítmicos 14
— probabilísticos 14
—, deteção de 14
—, localização de 14
—, propagação de 97
Esferas de proteção, mecanismos de 24
Espaço 188
— de controle 189
— de endereçamento 51, 57
— de memória 50, 51
— de nomes 50, 51, 58
— de nomes locais 83, 92, 110, 195
— de nomes segmentado 60
Espandida, memória de núcleo 178
Específica, capability 107
Especificação
— algorítmica 104
— formal 14, 38, 103
Estrutura
— condicional 44, 46
— de árvore 62, 108
Estrela, propriedade 37, 48
Estatísticos
—, bancos de dados 128, 166
—, mecanismos para bancos de dados 135
Estática
—, hierarquia 40
—, ligação 46
Estado
— de proteção 20
— de um sistema seguro 36
Estados 72
—, transição entre 114
Escravo
—, modo 74
—, memória de 51
Etiqueta 24, 25, 53
Etiquetas 46
— de nomes 109, 180
Etiquetada, máquina 53
Etiquetadas, arquiteturas 53, 165
Eventos, canais de 108
Experimental Network Operating System 155

Explícito, fluxo 42, 43, 46
Extração de informações confidenciais 135
Extensões de objetos 15, 180, 192
Extended Core Store 108
Extended Network Operating System 206
Excluir 19, 20
Execução 207
— de jobs de rede 154
— protegida 71
—, domínio de 16, 73, 78, 81, 83, 165
—, ligação durante a 46
—, parâmetros de 92
—, programa em 72

Facilities 87, 107
—, security 11
Falta de memória 90
Família
— de mecanismos de proteção 33
— de sistemas operacionais 192
Fantasma, objetos 26
Fator de perpetuação em Banco de Dados 137
FBI 38
FD (front-door)161
FE (front-end)162
Fechadura
—, /chave 22, 31
—, controlador local de 126
Fechamento 89
Filas 190
— de mensagens 87, 176
File
— clustering 66
— segmentation 66
Filtro seletivo 48
Fingerprint Identification System 146
FINGERSCAN 146
Firmware 111
Física, segmentação 66
Físicas, limitações 167
Flexibilidade 14
Fluxo
— explícito 42, 43, 46
— ilícito 42
— implícito 42, 43, 44, 46, 48, 128, 151
— por vetores, controle de 44
— seguro da informação 128
—, controle de 38, 42
—, modelo de 42
—, política de 42
—, regra de permissão de 43
—, relação de 43
—, trelica de 43
Forças-tarefas 200
Ford Aerospace & Communication Corporation 161, 185
Forma não-preemptiva de comunicação 86

Forma preemptiva de comunicação 86
Formal
—, especificação 38, 103
—, modelo 22
—, sistema 15
Formatação, software de 133
Formato de capability 25
Formulação de transação 122
Franquia, qualificador de 37, 170
Front-door 161
Front-end 161
Funcionalidade 12
— de núcleo 102
Função de mapeamento 50
Funções
— de controle 40
— de DBMS 133
— de gerente de objetos 102
— do sistema operacional 9
—, desempenho de 40
Funções-V 115
Funções-O 115
Fundamentos 5

Gates 172
GE-135 175
GE-145 175
Geometria da Mão 145
Geração de chaves 158
Gerência de objetos 102
Gerente
— de políticas 112
— de processo 19
— de recursos 10
Grupo
— de abstrações 191
— de acesso 189
Grupos de usuários 62
Guarda 162
Guarda-chuva de execução 170
Guardian, projeto 114, 177

Hand-shaking 88
Hardware 18, 54
— concentrado 169
— de paginação 74
— distribuído 92
—, alocação de 9
—, aproveitamento máximo de 13
—, confiabilidade de 14
—, erros de 47
HDM 118, 187, 191
Heterogênea, hierarquia 40
Heterogeneidade dos componentes de uma rede 153

Hierarquia
— completa 95
— de máquinas virtuais 107
— dinâmica 99
— heterogênea 40
— homogênea 40
—, sistemas operacionais em 95
Hierarquias 9, 60
—, níveis de 96
Hierárquica, autoridade 37
Hierárquicas
—, arquiteturas 40, 41, 57, 85, 95, 97, 98, 111, 191
—, camadas 106
Hierárquico, controle 28, 29
Hierárquicos, sistemas 32
Histórico de atividades 139
Homogênea, hierarquia 40
Honeywell 114, 161, 175, 177
HONEYWELL-6180 162, 175
HONEYWELL NIVEL-6 185
Hospedeiro
—, controle de 155
—, sistema de 14, 141, 142
House keeping 10
Humana
—, dependência 167
—, interface 14
Humanas, limitações 167
HYDRA 26, 41, 83, 85, 88, 92, 110, 195

I/O
—, dispositivos de 102
—, operações de 183
IBM-S/38 24, 34, 54, 58, 59, 188
IBM-S/360 31, 106
IBM-360/50 170, 171
IBM-360/85 51
IBM-370 51
IBM, IMS da 126
ICS 62
Identificação 158
— de capability 25
— de computador 159
— de emissor 86
— de objeto 15
— de terminal 158
— de usuário 159
Identificador 25
Identification System, Fingerprint 146
IDENTIMAT-2000 146
Ilícito, fluxo 42
IMF 62, 63
Imperial College 163
Implementação 104
— de políticas 12
—, linguagem de 107

Implícito, fluxo 42, 44, 46, 48, 128, 151
Impressões digitais 146
IMS da IBM 126
In-line 59
Inclusão 20
Independentes, capabilities 83
Indicador
— de acesso 23
— de propriedade 23
Indicadores
Índice
— de espaço de dados 189
— numérico 23, 25
Indireto, endereçamento 76
Informação 6
— armazenada, proteção de 49
— nula 92
— privativa 92
—, acesso à 38
—, alteração de 6, 7
—, átomos de 69
—, degradação manual de 48
—, duplicação de 33
—, fluxo de 38
—, fluxo implícito de 42, 44, 46, 48
—, fluxo explícito de 42, 43, 46
—, receptáculos de 42
—, sensitividade de 37
INGRES 133
INGRES da MITRE 130
Inicialização 170
Inoculação de Informações Falsas em BD 137
Instalação, organograma da 40
Instalations Management Facilities 62
Instruções
— de máquina 59
— privilegiadas 74
—, sequências de 72
Integridade 6, 7, 51
— de tipo 103
—, política de 6
—, violação de 6, 7
INTERDATA-3 57
Interface 102
— com UNIX 112
— humana 14
— para redes multiníveis 119
— XNIM 208
Interfaces protegidas 163
Interpretador de mensagens 96
Interprocessual, comunicação 107, 155, 204
Interrupção, chamada de 173
Intervalos 136
Inventário 62
I.P. Sharp 125
Isolamento 51, 74
— completo 30, 90

— de processo 89
—, política de 101

Job
— da rede 207
—, execução de 154
JOB:ID 170

Kernel 106, 116, 118
Key-lock 31
KISS 112
KMAP 199, 200
KSOS 28, 39, 40, 41, 63, 101, 113, 118, 160, 161
KVM/370 116, 117

Label 179
Language 191
Leitura 121
Ligação
— dinâmica 43
— do computador 159
— do terminal 159
— durante a compilação 46, 47
— durante a execução 46
— estática 43, 46
Limitações
— econômicas 167
— físicas 167
— humanas 167
Limite máximo superior de classes 46
Linguagem
— de canal 116
— de implementação 107
Link, criptografia de 151
Lista
— de acesso 22, 28, 31, 40, 69, 115, 165, 207
— de autorização 61
— de capabilities 22, 26, 31, 69, 81, 83, 108, 165, 170, 179, 182, 183
— de controle de acesso 28, 170
— de espera 85
— de passwords 'one-way' 144
—, tamanho da 28
Lixo na memória 26, 29
LNS 83, 92
Load 170
Localização 50
— de erros 14
Lógica
— de proteção 15
—, segmentação 65
Lógico a arquivos, acesso 64
Lógicos, subarquivos 65
LOG-IN 170

Macroprogramação 57, 59
Manager, Directory 192
Mão, geometria da 145
Mapeamento
—, função de 50
—, mecanismo de 51
—, tabela central de 81
Máquina, instrução de 59
Máquinas
— abstratas 97
— etiquetadas 53
— virtuais 41, 88, 97, 106
— virtuais, hierarquias de 107
— Von Neumann 53
Matriz de acesso 18, 19, 21, 69, 81, 87, 113, 124, 128
—, alteração da 20, 21
—, armazenamento da 22
Matriz de permissão 129
Matriz de proteção 21, 22, 26
Mecanismos 6
— de acesso 19
— de comunicação 73, 110
— de espera 24
— de mapeamento 51
— de núcleo 103
— de proteção 6, 9, 12, 18, 19, 56, 72, 83, 96, 141, 148
— de proteção discrecionários 18, 40, 44, 106, 165
— de proteção não-discrecionários 34, 40, 44, 106, 165
— de proteção, características de 12
— de proteção, componentes de 15
— de proteção, família de 33
— de proteção, primitivas de 101
— de proteção, projeto de 12
— de sincronização 73
— para Bancos de Dados Estatísticos 135
— para Bancos de Dados Relacionais 131
MEDUSA 88, 92, 199
Meios eletromagnéticos 142
Memória
— auxiliar 50
— central 108
— de cache 51
— de escravo 51
— de núcleo expandido 178
— de um nível 50
— principal 50
— virtual 50, 52, 57, 96
—, alocação de 52
—, canais de 90
—, capability de 81, 82
—, espaço de 50, 51
—, falta de 90
—, lixo na 26
—, objetos de 165
—, proteção de 50, 161
Memorização, autenticação por 143

Mensagens 86
— militares 203
—, filas de 57
—, interpretador de 96
Metodologia de desenvolvimento 182
Métodos para extrair informações confidenciais 135
Microagregação 138
Microcodificação 56
Microintruções 56, 59
Microprograma 56, 57, 98
Microprogramação 56, 59, 165
Migração de dados na rede 154, 207
Mini-Host 162
Mínimo de privilégios 13, 45
MIT 78, 113, 114, 175, 177
Mitre Corporation 36, 46, 47, 113, 114, 127, 203
MITRE, NÚCLEO da 39, 46
Modelo
— abstrato 103
— da matriz de acesso 18
— da treliça 43
— de fluxo de informação 42
— de gerência de dados 133
— discrecionário 18
— multinível 35
— formal 22
— não-discrecionário 35
Modelos
— discrecionários 124, 166
— não-discrecionários 166
Modificação
— de pedido 125
—, bit de 29
Modo
— mestre 74
— escravo 74
—, chave de 74
—, mudança de 100
Modular, desenvolvimento 115
Momento
— de decisão 207
— de ligação 46
Monitor 18, 19
—, chamada de 87
Monitores 87
MULTICS 27, 30, 40, 60, 63, 72, 76,80, 97, 106, 113, 129, 161, 175, 207
Multinível
— modelo 35
— rede 160
Multiníveis, interfaces para redes 119
Múltiplas tabelas de segmentos 76
Múltiplos
— domínios 79, 81
— níveis, transmissão de 39
— usuários 12
Multiprogramação 9, 57, 58

Mútua, suspeição 89, 92

Não-aumentativo, caso 156, 157, 160
Não-discrecionaridade 7
Não-discrecionário, chaveamento 101
Não-secreto, projeto 12
Não-violação de segurança 7
National Security Agency 149
Naturalidade na interface humana 14
NBS 149, 157, 206
NCP 206
Need-to-know 13, 35, 79
Network
— Daemon 119, 160, 187
— Núcleos 184
— Operating System, Experimental 155
— Operating System, Extended 206
Níveis
— arquitetônicos 124
— de autoridade 12
— de hierarquias 96
— de segurança militar 35
Nível
— de autoridade 37
— de autorização 42, 46
— de classificação 37, 38
— de segurança 37, 38, 39, 40, 43, 16, 165
— do sistema 41
NKCP 116
NKSR 118
Nomeação 103
Nomes
— globais 30
— locais 30
—, espaço de 50, 51
—, etiquetas de 109, 180
Non-Kernel Control Program 116
Non-Kernel Security Related System 119
NOS 153
NOTIFY 87
NSC 157, 158
Nucleadas, arquiteturas 95, 106
Nucleados, sistemas 94, 98
Núcleo 11, 16, 56, 109
— certificado 116
— da rede 112
— de sistemas operacionais 106
— expandido, memória de 178
—, arquivo de 178
—, funcionalidade de 102
—, sistemas de 108, 178
Núcleo-tambor 51
Núcleos 41, 99, 106
— de proteção 59, 177
— de segurança 100, 101, 113, 166
— discrecionários 106, 166

— não discrecionários 113, 166
Nula, informação 92

Objetos 15, 18, 35, 124, 165, 170
— de disco 108
— primitivos 15
— próprios 60
— protegidos 102, 107
— TIPO 110
— tipo usuário 170
— tipo job 170
— tipo arquivos 170
—, extensões de 15, 180, 192
—, tipos de 32
Opcional, função 11
Operação
— ATTACH 108
— P 85
— V 83
Operações 179
— de I/O 183
Operacionais, sistemas 9, 10
Operacional, penetração 10
Operador de combinação de classes 43
Organograma de instalação 40
OS/VS 62
OS/360-MVT 62,74
OS/360-91 62
OS/370 60
Overhead 13, 28
Overlap 137

Paginação, hardware de 74
Páginas 172, 182, 196, 200
Palavras 55
Parâmetros 12
— de execução 92
— secretos 149
—, passagem de 57
Paridade 53
Parnas 103
Partição reservada 24
Particionamento 138
Pascal 103, 109, 111
Password 88, 143
— 'one-way' 144, 170
PDP 102, 162, 203
PDP-11 110, 111, 195
PDP-11/45 182, 206
PDP-11/70 118, 182
Pedido 36
—, modificação de 126
Penetração 179
— operacional 10
Pequenos domínios 81

Perfil de usuário 158, 189
Pergunta/resposta 145
Perguntas
— padrão 135
— tipo soma 138
Periódica, deleção 27
Permissão
—, bits de 29
—, matriz de 129
—, revogação da 81
Pipes 200
Plana, arquitetura 40, 85
Plano, sistema 41
Plessey System-250 24, 77, 83
Pointers 188
— com endereço 23
— retroativos 27
Policy Manager 182, 183, 184
Política 103
— de alocação de memória 50
— de compartilhamento 101
— de fluxo 42
— de integridade 6
— de isolamento 30, 101
— de segurança 6, 101, 125
—, gerente da 112
Políticas 110
— de segurança 112
—, implementação de 12
Pool 97, 99
Port 196
Portas 87
— padronizadas 80
Predicados 124, 133
Preemptiva de comunicação, forma 86
Prescrição 29
Preventiva, alternativa 10
PRIME 14, 88
Primitivas dos mecanismos de proteção 101
Principal
—, capability 27
—, memória 50
Prioridade 96
Privativa, informação 92
Privilegiadas, instruções 74
Privilégios 13
—, mínimo de 45
—, tipos de 79
Probabilísticos, erros 14
Problema
— do Cavalo de Troia 89, 90
— do produtor/consumidor 85
Procedimentos 121
Procedures 57, 195, 196
—, biblioteca de 97
Processador virtual 95
Processamento 154

— de mensagens militares 203
— de múltiplos níveis 39
— distribuído 157
Processo 9, 16, 31, 179, 182, 196
— pronto 73
— rodando 73
— secreto 149
— suspenso 73
— virtual 95
—, criação de 9
—, gerente de 19
—, isolamento de 89
—, remoção de 9
Processos 42, 72, 165, 172, 175, 185
— privilegiados 116
—, árvores de 107
—, conexão entre 9
—, segmentos de 186
Produtor/Consumidor, problema de 85
Profile 206
— de segurança 170
Programa 95, 188
— em execução 72
Progresso de processo, controle de 9
Projeto
— Guardian 114
— não-secreto 12
Propagação de erros 97
Propriedade 20
— estrela 37, 42, 48
— simples de segurança 37, 42
—, implementação de 38
—, indicador de 23
Proprietário 62
Proteção
— da informação armazenada 49
— da memória 50
— em redes 141
—, anéis de 40, 78, 113, 165, 176
—, chaves de 51
—, características de mecanismos de 12
—, componentes de mecanismos de 15
—, dados de 125
—, domínios de 76
—, estado de 20
—, lógica de 15
—, mecanismos de 6, 9, 12, 18, 19, 56, 83, 96, 112, 141, 148
—, núcleos de 59
—, regime de 33
—, responsabilidade de 22
—, salvaguardas de 121
—, sistemas de 18
Protegida, execução 71
Protegidas, interfaces 163
Protegido
—, domínio 108
—, subsistema 16

Protegidos, domínios 27
Protocolos de fechamento 126
Provably Secure Operating System 98, 167, 191
Provas, teoria de 15
PSOS 98, 167, 191
Pública, chave 150

Qualificador 35
Qualitativa, capability 107
Quantitativa, capability 107

Raiz 60
RC-4000 86, 108
Receptáculos de informação 42
Reconfiguração 14, 154
Recuperabilidade 8
Recuperação 14, 134
Recursos 9
—, alocação de 9
—, compartilhar 9
—, scheduller de 10
Rede
—, Centro de Segurança da 157
—, controle da 155
—, descrição da 190
—, jobs da 207
—, núcleos da 112
—, segurança da 151
Redes
— de computadores 39, 88, 149, 153
—, sistemas operacionais para 166
Redundância 14, 69, 126
Registradores 56
Registros
—, relação de 154
—, transformação de 154
—, translação de 154
Regra
— da unicidade 132
— de acesso 16, 18, 124, 125, 165
— de permissão de fluxo 43
— de segurança 43
— de validação 128
— do retângulo 132
Relação 129
— de base 131
— de fluxo 43
— de registros 154
Relacional, Banco de Dados 126, 129, 166
Relações virtuais 131, 132
Relógio
— de tempo real 96
— virtual 91
— remoção do processo 9
Revogação 132
— de direitos 26

— de permissão 81
— seletiva 27
Revogável, capability 27
RICE 24, 51
Rice University 51
Risco, controle de 168
ROM 56
RTOS 56
RZO 127

S-Memory 57
Salvaguardas de proteção 121
Scheduler 10, 112, 182
SCOPE 109, 178
Secreto, processo 149
Secretos, parâmetros 149
Secure Computer Systems 36
Secure Entity Identifier 185
Security Facilities 11
Segmentação 74
— física 66
— lógica 65
Segmento
— de dados 60
— de descritores 75, 165
— de processos 186
—,controlador de 96
—, descritor de 60, 75, 79
Segmentos 74, 177
Segurança 6, 7
— da Rede, Centro de 157
— do algorítmo 149
— militar, níveis de 35
— retrospectiva 156
—, classes de 37, 38, 42, 43, 46
—, codificação de 170
—, dados de 170
—, estágio de 51
—, não-violação de 6, 7
—, nível de 16, 38, 39, 40, 43, 128, 165
—, núcleo de 100, 101, 134, 166
—, política de 6, 101, 112
—, profile de 170
—, propriedades de 37
—, regras de 43
—, violação de 6
SEID 185
Seletivo, filtro 48
Selos de autenticação 48
Semáforo 57, 85, 86, 196, 197, 200
Semi-privilegiados, processos 116
Sensitividade da informação 37
Separação 12
—, cerca de 25
SEQUEL 132

Sequência
— de ações 72
— de instruções 72
Serviços tipo Cavalo de Troia 89, 90
SHARP 129
SIGMA 39, 48, 113, 167, 203
Simétrica, cifragem 149
Simples, propriedade 37
Simplicidade 12
Simulador do SCOPE 178
Simultâneas, equações 137
Sincronização 57
—, mecanismos de 73
Sistema
— de controle 62
— de informação 157
— de núcleo 108, 178
— de usuário 109, 178
— formal 15
— operacional 9, 54
— operacional em hierarquias 95
— operacional em redes 166
— plano 41
— protegido 16
—, administração de 29
—, estado de um 36
—, facilities de 87
SISTEMA R 131, 132
Sistemas
— de arquivos 18, 30, 60, 121
— de depuração
— de hardware concentrado 169
— de hardware distribuído 195
— de Processamento de Mensagens Militares 203
— de proteção 18
— distribuídos 140, 141
— hierárquicos 32, 40
— nucleados 94, 98
SLOCAL 199
Sobreposição 137
Software
—, alocação de 9
—, formatação de 133
SPACES 59
SPECIAL 103, 187, 191
Specification & Assertion 191
SRI International 47, 103, 118, 129, 177, 191
STAROS 88
Subarquivos lógicos 65
Submissão de transação ao DBMS 123
Subrede, controle da 155
Subsistema protegido 16
Substituição 148
Suptipo de arquivo 63, 186
SUE 88, 106, 107
Sujeitos 15, 18, 35, 124, 165
Superclassificação de informações 48, 204

Superior, limite 136
Supervisor em camadas 113
Suporte de dispositivos de I/O 102
Suspeição mútua 89, 92
SYSLOG 170
SYSTEM R 127
SYSTEM DEVELOPMENT CORP 35, 129

Tabela
— de objetos mestre 179
— global 26
Tabelas de segmentos, múltiplos
Tag 53
— bits 53
Tambor, núcleo 51
Task 72
TCP 160
Teleconferência 154
Telefunken 127
TELEFUNKEN TR-4 53
Templates 83, 92, 110
Tempo real 57
Tempo virtual 91
TENEX 203
Teoria de provas 15
Terminais 170, 173, 203
Terminal
—, autenticação do 158
—, identificação do 158
—, ligação do 158
Terminal Interface Processos 162
TERMINAL:ID 170
Teste 98
THE 85, 98, 106
Timbre da voz, autenticação por 145
TIP 162
Tipo
— arquivo, objeto 170
— job, objeto 170
— usuário, objeto 170
Tipos 179
— de acesso 124
— de objetos 32
— de privilégios 79
Topologia da rede 154
Trabalho, área de 57
Tradução da transação 122
Trajetórias 10
Transação 121, 122, 132
—, formulação da 122
—, submissão da 123
—, tradução da 122
Transferência pura 21
Transferir 20
Transformação de registros 154
Transição entre estados 114
Transitividade 90
Translação de registros 154
Transmissão

— de capabilities 26
— de chaves 151, 152
— de múltiplos níveis 39
Transmission Control Protocol 160
Treliça 128
— de fluxo 43
—, modelo da 42
TSO 62

UCC 69
UCLA 47, 126
UCLA SECURE INGRES 131, 133
UCLA SECURE UNIX 87, 111, 133, 182
Unicidade, regra da 132
Unidade lógica, descrição de 190
Universidade de Cambridge 56
Universo virtual 108
UNIX 118, 185, 206
— standard 112
—, emulador do 118, 185
—, interfaces com 112, 182
USER:ID 170
Uso de direitos 81
Usuário 62
— principal 24
—, autenticação do 159
—, criptografia controlada pelo 69
—, identificação do 158
—, perfil de 189
—, sistema de 108
Usuários
—, grupos de 62
—, multiplos 12
Utilitários 200

Validação, regras de 128
Valor
— máximo 136
— mínimo 136
Valores, conjunto de 129
Variáveis de estado 115
VENUS 56, 57, 86, 98
Verificação de autorização 81
VERIPEN 145
Vetores
— de capabilities 44
—, controle de fluxo por 44
Violação de segurança 6, 7
Virtuais
—, endereços 57
—, hierarquias de máquinas 107
—, máquinas 41, 88, 98, 106
—, relações 131, 133
Virtual
—, dispositivo 98

—, memória 50, 52, 57, 95
—, tempo 91
—, universo 108
Vistas 65, 131, 132, 134
VM/370 116
Voláteis, arquivos

Wake-up 173

XNDM 208
XNIM 206
XNOS 39, 206