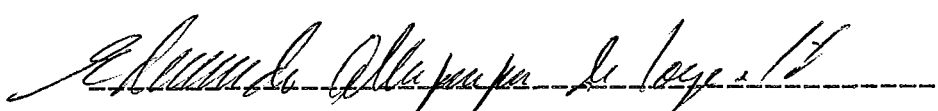
DEFINIÇÃO DE UM SIMULADOR PARA REDES DE COMUNICAÇÃO

Carlos Henrique de Oliveira

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

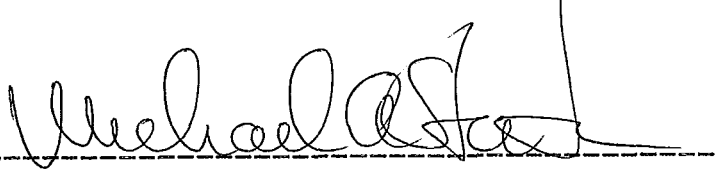
Aprovada por:

Prof. Paulo Henrique Aguiar Rodrigues, PhD.
(Presidente)



Prof. Edmundo Albuquerque de Souza e Silva, PhD.

Prof. Valmir Barbosa, PhD.



Prof. Michael Stanton, PhD.

RIO DE JANEIRO, RJ - BRASIL

ABRIL 1991

OLIVEIRA, CARLOS HENRIQUE
Definição de um Simulador para Redes de Comunicação [Rio de Janeiro], 1991.
IX, 239 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 1991)
Tese - Universidade Federal do Rio de Janeiro, COPPE.
1. Definição de um Simulador para Redes de Comunicação
I. COPPE/UFRJ II. Título (Série)

À Denize, Angela, meus pais e meus sogros

AGRADECIMENTOS

Ao Prof. Paulo Henrique de Aguiar Rodrigues, pela sua constante atenção e incentivo, por sua experiência e sua metodologia como orientador, que foram de grande auxílio para a realização deste trabalho.

Aos meus colegas da AMPLUS INFORMÁTICA S.A., pelo apoio técnico e pela amizade que recebi durante o desenvolvimento do meu mestrado.

A minha esposa Denize e a minha filha Angela, pelo incentivo e pela paciência e carinho que me dedicam.

A meus pais Milton e Maria de Lourdes, que sempre me incentivaram em meus estudos e conquistas profissionais.

A meus sogros Geraldo e Eliete, por sua amizade e pelo seu apoio durante a confecção desse trabalho.

Finalmente, gostaria de agradecer a todos meus familiares e amigos pelo constante incentivo que me dedicaram.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.).

DEFINIÇÃO DE UM SIMULADOR PARA REDES DE COMUNICAÇÃO

Carlos Henrique de Oliveira

Abril de 1991

Orientador : Paulo Henrique de Aguiar Rodrigues

Programa : Engenharia de Sistemas e Computação

RESUMO

Este trabalho apresenta a definição de um Simulador para Protocolos de Redes de Comunicação e a implementação de um protótipo simplificado. Seus principais objetivos são a simulação de protocolos de redes de comunicação de dados de forma distribuída e a execução de código real junto com código simulado.

Para o desenvolvimento deste trabalho foi analisado uma série de temas relevantes ao assunto. Nestes temas estão incluídos: a simulação distribuída e a simulação de redes de comunicação.

O trabalho inclui a definição do Simulador de Comunicação. Ele é construído de forma modular, podendo seus processos ficarem distribuídos na rede real. Seus módulos básicos são: o sincronizador, o simulador de comunicação e o nó de simulação.

Foi realizada a implementação de um modelo simplificado usando o protocolo de enlace CSMA-CD, para se realizarem algumas experiências com o simulador.

Abstract of Thesis presented to COPPE/UFRJ as partial fulfillment of the requirements for the degree of Master of Science (M.Sc.).

DEFINITION OF A SIMULATOR FOR COMMUNICATION NETWORKS

Carlos Henrique de Oliveira

April, 1991

Thesis Supervisor : Paulo Henrique de Aguiar Rodrigues

Department : System Enginnering and Computation

ABSTRACT

This thesis presents the definition of a simulator for communication networks protocols and implementation of a simplified prototype. The simulator provides a simulation of communication networks protocols in a distributed form and permits the execution of real code joined with simulation code at the same time.

The development of this thesis was based on the analysis of distributed communication networks simulation.

This thesis includes the definition of a communication simulator. The simulator architecture is built on a modular form. The processes compoments of the simulator can be run in a distributed form.

The implementation of a simplified model where stations communicate via CSMA-CD link logical protocol was realized in this thesis. Some experiments and tests were realized using this simplified model to demonstrate the eficacy of distributed simulation.

ÍNDICE

ÍNDICE

ÍNDICE

CAPÍTULO 1

INTRODUÇÃO

1.0 Introdução

Normalmente redes de comunicação e sistemas computacionais são de alta complexidade, e sendo assim, com raras exceções, estes sistemas são analiticamente intratáveis e a avaliação numérica proibitiva. Consequentemente temos como solução para avaliação e análise de tais sistemas a simulação. Com a simulação podemos alcançar uma série de objetivos: entendermos o comportamento do sistema, obtermos estimativas das medidas de performance e guiar-nos na seleção dos parâmetros de operação do sistema.

Este trabalho objetiva o desenvolvimento de um simulador para protocolos de comunicação, com a capacidade de operação sob a forma distribuída. Com este tipo de operação tentaremos aumentar a velocidade da simulação, além do que poderemos, com a divisão da simulação em pequenas tarefas, utilizar processadores de pequeno

porte do tipo estações de trabalho ou micro processadores. Este trabalho tenta também explorar a existência de um grande potencial computacional em redes locais, que são ambientes muito comuns em nossas intituições. Objetivamos ainda neste trabalho aprender mais sobre simulação distribuída através do uso de protótipos.

De acordo com o modelo de referência OSI (Open Systems Interconnection) da ISO (International Standards Organization) descrito em [22], uma rede de comunicação é organizada em sete camadas de protocolos. Estas camadas são mostradas a seguir.

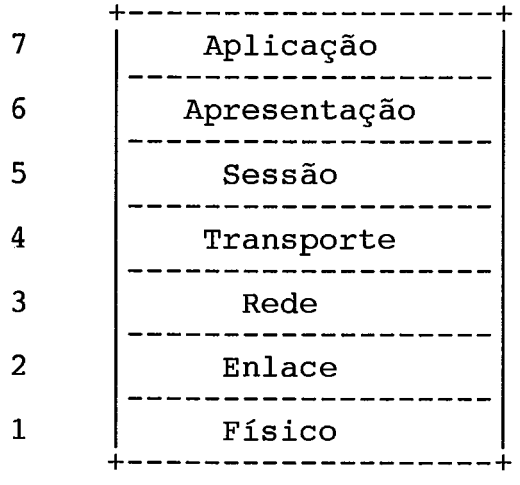


Fig 1. Modelo de Referência OSI

Nosso simulador conterá a camada física num módulo único e as outras camadas (2 a 7) estarão em outro módulo. Simulando a camada física e a camada de enlace poderemos ter topologias que variam de

um anel, passando por uma rede de "broadcast" até uma rede completamente conexa. Nosso simulador pode também em sua operação conter uma parte de código simulado e uma outra parte de código real, podendo usar código desenvolvido pelo usuário (ex: um protocolo da camada de transporte) diretamente no simulador. Desta forma o simulador poderá ter um resultado muito mais realista.

Vamos abordar a seguir uma série de temas relevantes ao desenvolvimento do simulador. Estes temas são os seguintes: simulação de redes, simulação distribuída e sistemas de suporte ao desenvolvimento de aplicações distribuídas.

1.1 Simulação de Redes

Ao analizarmos o tecnologia atual das redes de comunicação de dados podemos notar um grande avanço como por exemplo: aumento das velocidades de comunicação, novas topologias, métodos de acesso complexos, etc. A análise das redes através do modelamento e de metodos matemáticos tem se tornado cada vez mais dificil de ser realizada, devido à complexidade crescente destas redes. Assim uma ferramenta adequada, e muitas vezes única, para avaliação de sistemas complexos é a simulação.

Com a simulação da rede de comunicação podemos estudar, entre outros aspectos, seu comportamento frente a diferentes condições de tráfego e topologia, obter medidas de desempenho, e avaliar sua sensibilidade em relação a parâmetros operacionais. Ao analizarmos a simulação de redes de comunicação identificamos dois tipos de implementações. A primeira forma é uma metodologia que cria uma simulação a partir de uma rede de comunicação específica, não podendo ser alterada facilmente para outras redes de comunicação. Este tipo de implementação é descrito em [1] e [2]. Na segunda forma é construído um simulador genérico, e cada rede a ser simulada é descrita, de acordo com as regras do simulador. Podemos ver isto em [3], [4] e [5].

Temos maior interesse na segunda forma de implementação, pois a partir dela é possível a construção de um simulador de protocolos de comunicação genéricos. Este é o caminho que desejamos seguir no nosso traballho.

A implemetação dos simuladores que foram descritos no grupo dois pode ser realizada de diversas maneiras. Como exemplo de classificação quanto a forma de implementação temos os seguintes tipos: hardware mais software ou apenas software; simulação em processador único ou simulação distribuída; entre outros exemplos.

Em [5], foi construído um hardware que permite simulação em tempo real de protocolos de comunicação, de diferentes arquiteturas de

redes e de sistemas distribuídos. O sistema é constituído de 32 emuladores de nós de comunicação programáveis, de um emulador de canal de comunicação programável e de um sistema de controle.

Cada um dos nós pode emular de forma independente elementos computacionais inteligentes tais como: computadores, servidores, estações de trabalho, "bridges" e "gateways". Cada emulador de nó suporta geração de mensagens, monitoração e nível de enlace lógico.

O emulador de canal de comunicação programável pode simular através do hardware uma grande variedade de funções do nível físico, arquiteturas variáveis de redes e subredes independentes. Este sistema de simulação permite a implementação de diversas topologias, atraso de propagação variado, e permite ainda a geração de deteção de portadora, deteção de colisão, e operação síncrona ou assíncrona.

O sistema de controle é o reponsável pelo controle da simulação, monitoração geral, análise, visualização e as funções iterativas com o usuário. Mostramos a seguir o diagrama do simulador.

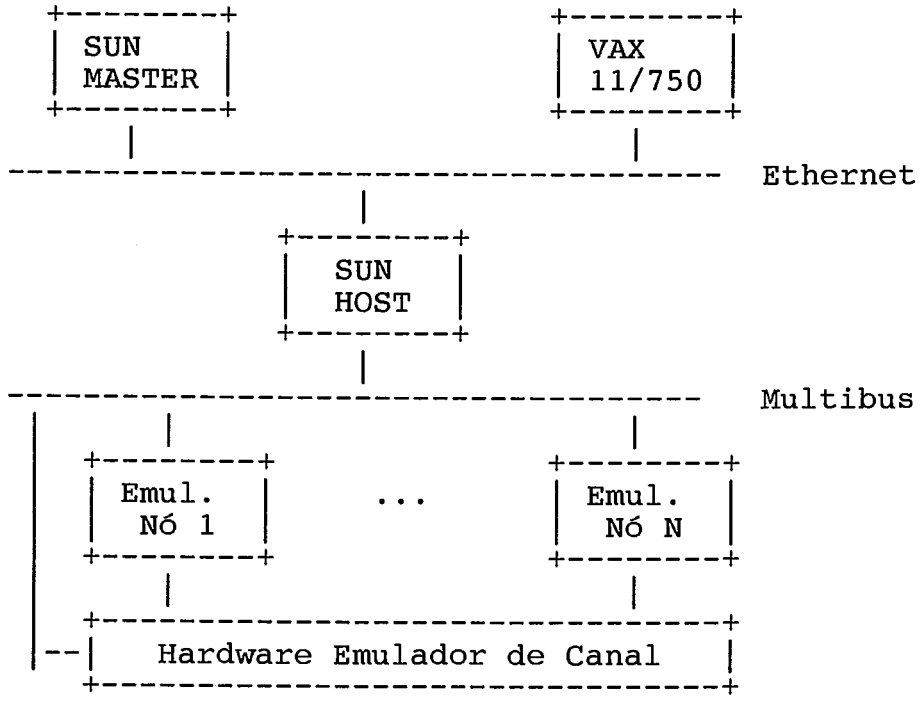


Figura 2. Diagrama do Simulador

Observando o exemplo acima podemos notar que é um simulador implementado com hardware mais software, isto é, foi desenvolvido um harware especial para simular os nós de comunicação e o canal de comunicação. Este sistema possui um software associado para realização e controle da simulacão.

Em [4], temos um simulador que opera de forma distribuída, sendo ele implementado em software, podendo rodar sobre diversos sistemas de harware. Este simulador usa como suporte para sua estrutura distribuída o sistema Jade, que é um sistema que permite o desenvolvimento de software distribuído [6]. Este simulador foi

desenvolvido para simulação de rede de comunicação por rádio, mas sua estrutura é genérica de forma a poder simular outros tipos de redes de comunicação. O simulador possui a estrutura abaixo.

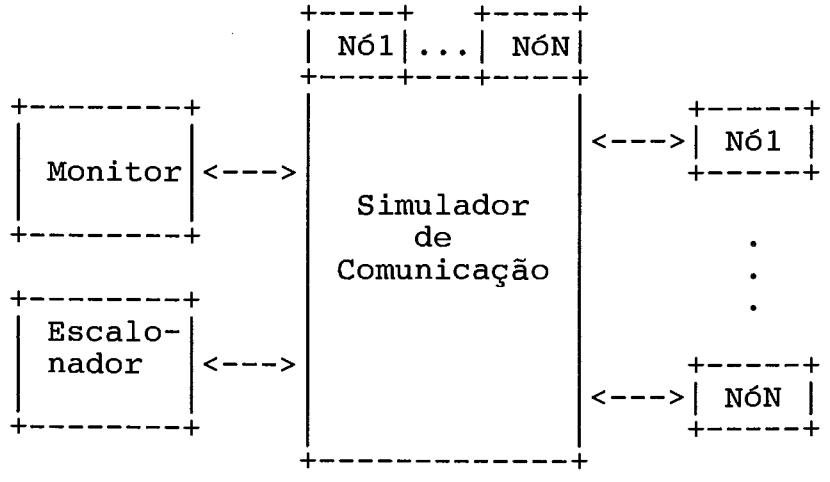


Figura 3. Diagrama do Simulador

O simulador possui os seguintes modulos básicos: simulador de comunicação, nó protótipo, monitor de comunicação e escalonador.O simulador de comunicação é a estrutura básica do sistema, e pode modelar os seguintes efeitos: plataformas móveis, propagação de ondas de rádio, interferência eletromagnética, apontar antena e selecionar a largura de banda, seleção de canal, seleção de codificação, potência do sinal de transmissão, deteção de colisão e deteção de portadora. O simulador de comunicação é responsável pela simulação da troca de mensagens no nível físico.

O nó protótipo é responsável pela simulação dos níveis 2 a 7,

possui uma interface definida com o simulador de comunicação, para troca de mensagens. O nó protótipo pode estar em uma máquina diferente do simulador de comunicação operando assim de forma distribuída.

O monitor de comunicação permite ao usuário visualizar toda a operação do sistema enquanto a simulação está em andamento. Possui basicamente um processo coletor de dados único e um processo monitor para cada variável a ser amostrada. Como exemplo de variáveis a serem amostradas temos: "throughput", atrasos, intervalos de tempo entre chegadas e partidas de mensagens, utilização e taxa efetiva de transmissão.

O escalonador tem como função mudar a estrutura da simulação, criar falhas no sistema que está sendo simulado, a fim de observarmos o comportamento do sistema em simulação.

No exemplo anterior que a implementação da simulacão não possui hardware específico, a simulação é construída em software e usa o computador apenas para executar este software. Observando ainda o exemplo anterior podemos notar que o simulador executa de forma distribuída em uma série de processadores. Ele foi divido em vários processos e estes processos executam de forma distribuída.

Em [3], temos um simulador que roda em um único processador, sendo ele implementado em software. Este simulador possui uma lista

geral de eventos a serem simulados, e assim o simulador continuamente avalia esta lista a fim de executar os eventos ali indicados e poder avançar o tempo de simulação.

Este simulador está logicamente dividido em duas partes. A primeira é independente do protocolo de comunicação a ser simulado. É responsável pelo tratamento da lista de eventos, das filas, da coleta de estatísticas e da geração de mensagens. Na segunda parte temos a descrição do protocolo de comunicação a ser simulado. Aqui é descrito o tratamento do eventos básicos do protocolo de comunicação. Este simulador pode operar na simulação de diversos tipos de protocolos de comunicação, sendo necessário para isso apenas a substituição da parte que descreve o protocolo. Este simulador foi projetado com o intuito de avaliar protocolos de nível de enlace lógico (nível 2).

Este simulador não possui forma iterativa de operação, isto é, recebe uma lista de operações, executa todas elas, e só após isso podemos observar o resultado da simulação.

No exemplo acima podemos notar que a implementação do simulador não possui hardware específico. A simulação é construída em software e usa o computador apenas para executar este software. Vemos ainda que o simulador executa em apenas um processador.

Após a observação do exemplos anteriores existe uma série de

pontos favoráveis e desfavoráveis de cada implementação. Em [5], sua implementação utilizando hardware permite uma resposta em tempo real. Por outro lado a utilização do hardware torna o simulador não portátil e de alto custo financeiro. A implementação de nosso simulador não seguirá a mesma direção de [5], e desta forma não teremos a mesma resposta em tempo real mas ganharemos em portabilidade, tempo de implementação e custo. Já em [3] a implementação por ser em processador único pode não permiter sua execução em máquinas tipo micro processadores, devido ao requisito de memória. Contudo um dos objetivos de nosso simulador e justamente a utilização de micro processadores. Ainda com a operação de forma distribuída poderemos ter um aumento de performance em relação a [3]. Com relação a [4], muitas de suas idéias serão seguidas em nosso implementação. Enquanto que em [4] não é permitido a execução de código real, esta possibilidade é mostrada no modelo de simulação proposto nesta tese.

1.2 Simulação Distribuída

A maioria dos simulações são muito lentas [8], isto é, seu tempo de execução é muito grande. Uma forma que nos parece óbvia de aumentarmos a velocidade da simulação é dedicar a ela mais recursos, e para isso podemos usar uma arquitetura multiprocessada ao invés de um único processador. Descartando a possibilidade de

usar computadores de maior porte, o paralelismo surge como alternativa mais adequada para aumentarmos a performance da simulação. Podemos considerar duas formas de paralelismo. Na primeira delas, de uma forma simples, podemos realizar a tarefa de coleta e processamento das estatísticas em paralelo com a simulação em curso ou, em uma forma mais complexa, podemos dividir nosso modelo em seus componentes básicos e executá-los em paralelo. Em ambos os casos podemos usar uma série de processadores com a mesma simulação para obtermos estatísticas. O uso de vários processadores traz vantagens tais como a redução do tempo de execução da simulação ou a redução do requisito de memória.

1.2.1 Medidas de Desempenho

A partir do uso da simulação distribuída temos que possuir alguma medida de performance em relação ao caso de um processador único. Em [8] é definida a medida "speedup", que é o tempo de execução da simulação em um único processador dividido pelo tempo de execução da mesma simulação num sistema multiprocessado. Fica fácil notar que o "speedup" ideal para N processadores é N. O problema com esta definição é que nem sempre é possível especificar o mesmo processador para execução da simulação quando temos um único processador e a execução da mesma simulação quando num sistema multiprocessado, pois é a partir destes dois valores que obteremos

o "speedup". Outro problema que podemos notar é que quando temos um único processador, ele pode não possuir memória suficiente para executar toda a simulação. Outra medida interressante é a eficiência que é definida como sendo o "speedup" dividido pelo número de processadores usados na execução em sistema multiprocessado. Ela mede a utilização efetiva dos processadores. Temos ainda o "speedup" estatístico que é definido como o tempo para um único processador obter uma estimativa de uma característica do sistema com um determinado erro dividido pelo mesmo tempo obtido usando multiprocessamento. Neste trabalho utilizaremos o "speedup" como sendo nosso medida de performance para a simulação. Nosso objetivo é determinar para quais configurações podemos obter um "speedup" próximo ao ideal. Para alcançarmos tal objetivo devemos ter em mente que o paralelismo nas configurações utilizadas deve ser o maior possível, isto é, em um determinado instante de tempo todos os processadores devem estar processando.

1.2.2 Arquiteturas Distribuídas

No caso de arquiteturas distribuídas podemos distinguir os sistemas multiprocessados baseados no critério do armazenamento dos dados. Temos dois grupos básicos: memória compartilhada e passagem de mensagem.

O grupo memória compartilhada possui arquiteturas com múltiplos processadores mas a memória é comum a todos os processadores. Em alguns sistemas os processadores possuem uma pequena memória local. Temos como exemplo deste tipo de arquitetura: "BBN Butterfly System", "Connection Machine", e "Sequent System" [8]. O problema maior neste tipo de arquitetura é a concorrência ao acesso da memória compartilhada. Em sistemas com um grande número de processadores são necessários protocolos para manutenção consistência dos dados.

O grupo de passagem de mensagem é caracterizado pelos processadores possuirem memória local e as informações são trocadas através de uma rede de comunicação. Podemos citar neste caso: "Hypercubes", "Transputers", e redes com micro processadores ou estações de trabalho [8]. Nesta arquitetura o problema maior é o congestionamento devido ao tráfego alto dentro da rede de comunicação.

Ambas arquiteturas podem ser utilizadas para simulação distribuída, dependendo para isso do tipo de aplicação, do tipo de decomposição, do método de sincronização, entre outros. Podemos ainda utilizar arquiteturas híbridas, dependendo do tipo de problema.

1.2.3 Formas de Decomposição

Vamos agora mostrar as formas de decomposição de uma simulação para ser executada em sistemas multiprocessados. Segundo [8] temos cinco maneiras de realizar esta decomposição:

1) Compiladores paralelos
2) Experimentos distribuídos
3) Distribuição das funções
4) Distribuição dos eventos
5) Distribuição dos componentes do modelo

1) Compiladores paralelos: Neste caso o compilador tenta encontrar no programa sequências de código que possam ser executadas em paralelo em processadores diferentes. Nesta forma de decomposição, o usuário não tem nenhuma participação na hora de escrever o código que irá rodar de forma distribuída. Desta forma é gerado apenas um pequeno paralelismo pois não há conhecimento do problema que está sendo simulado.

2) Experimentos distribuídos: Aqui os N processadores executam simultaneamente cópias da mesma simulação de forma isolada, e no final é realizado a média para obter o resultado da simulação. Podemos notar que nesse caso para N processadores o "speedup" será N. Segundo [9] este tipo de decomposição é a mais eficiente quando

tentamos minimizar o erro médio quadrático de nossa estimativa. Na verdade tal procedimento não caracteriza uma forma de decomposição, pois toda simulação necessita ser executada uma série de vezes para se obter o resultado final. Nesta forma de decomposição temos um problema sério que é a insuficiência de memória dos processadores do sistema para execução de toda a simulação. No caso da decomposição em pequenas tarefas o requisito de memória será menor podendo assim executar em pequenos processadores.

3) Distribuição das funções: Neste caso dedicaremos alguns processadores para realização de algumas das funções a seguir: coleta de estatísticas, entrada e saída de dados, manipulação de arquivos, geração de gráficos, supervisão e etc. A vantagem deste método é que não temos problemas de "deadlock" e é transparente para o usuário. Por outro lado ele não explora todo o paralelismo do problema.

4) Distribuição dos eventos: Aqui é mantida uma lista global de eventos, como na simulação sequencial, e quando um processador se torna disponível ele requisita um novo evento para ser tratado. Esta técnica necessita de protocolos para manutenção da consistência, uma vez que os próximos eventos podem afetar os que já estão sendo executados. Este tipo de decomposição é ideal para arquiteturas com memória compartilhada, pois a lista de eventos pode ser acessada por todos os processadores. Ainda aqui não podemos obter toda a capacidade de paralelismo do sistema pois não

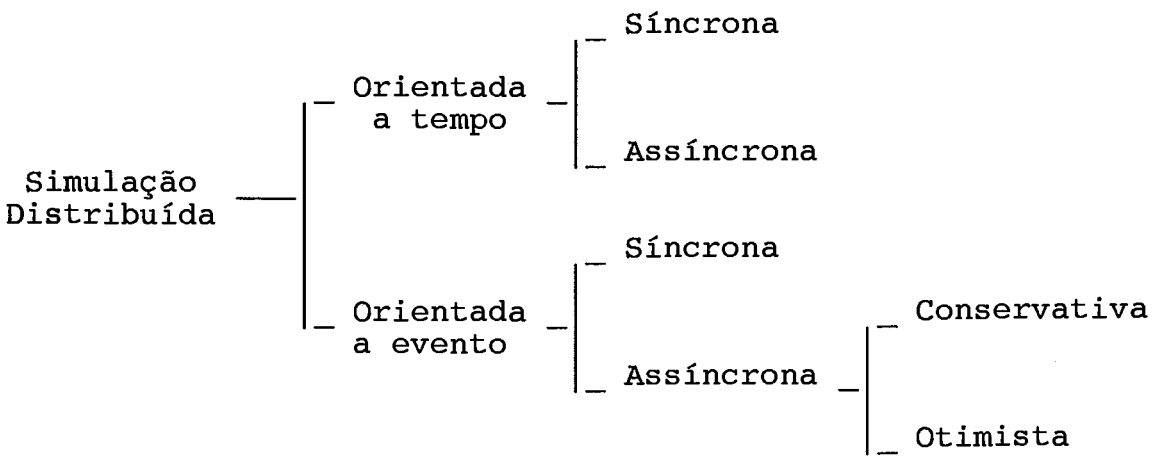
conhecemos o modelo a ser simulado.

5) Distribuição dos componentes do modelo: É feita a decomposição do modelo a ser simulado em partes acopladas e estas partes são distribuídas entre os processadores, como processos independentes, alguns processos podem até rodar num mesmo processador. Esta forma de decomposição permite atingir o máximo da capacidade de paralelismo, pois conhecemos o modelo a ser simulado, mas ela exige formas de sincronização para o sistema.

Existe ainda a possibilidade de podermos reunir num problema mais de uma técnica de decomposição, realizando combinações das técnicas descritas anteriormente podemos obter também bons resultados.

1.2.4 Técnicas de Sincronização

Temos agora que conhecer as formas de sincronização para aplicarmos nos casos de distribuição dos componentes do modelo, a fim de garantir a execução ordenada da simulação e a ausência de "deadlock". Vamos ver a seguir as possíveis técnicas de sincronização.



Simulação orientada a tempo: Neste caso o tempo da simulação avança em incrementos fixos, chamados "ticks", cada processo simula seus componentes durante cada "tick". O "tick" deve ser pequeno para garantir a acuidade, mas um "tick" pequeno implica em uma simulação longa, porque neste caso é provável que nada aconteça durante um "tick". Aqui a simulação pode ser síncrona ou assíncrona.

No caso síncrono todos os processos devem terminar a simulação de um "tick" antes que possa ser iniciada a simulação de um novo "tick". Neste caso, a simulação do "tick" se processa em duas fases, simulação ou fase computacional, e atualização de estado e fase de comunicação. Existem duas formas de gerar o clock global para o sistema: a forma centralizada, onde temos um processo dedicado para agir como sincronizador, e a forma distribuída, através de algoritmos de "broadcast".

O caso assíncrono permite um aumento da concorrência, mas com isso é penalizado com o aumento do custo de comunicação. Aqui um processador para poder avançar o tempo de simulação deve certificar que todos os processadores ligados a ele tenham terminado o processamento do "tick" anterior, assim ele precisa enviar e receber uma série de mensagens.

A simulação orientada a tempo tende a ser menos eficiente que a orientada a evento, porque durante um "tick" pode não ocorrer evento a ser simulado. Este tipo de sincronização se adequa a sistemas que possuem variações de topologias durante o funcionamento (ex: redes de rádio [4]), ou sistemas onde muitas coisas podem estar acontecendo ao mesmo tempo. Esta técnica é usada ainda para tornar discretos sistemas contínuos.

Simulação orientada a evento: Aqui o tempo simulado avança de um evento para o próximo, onde o evento significa a mudança de estado. Neste tipo de simulação temos um potencial de aumento do "speedup" em relação ao caso orientado a tempo. Assim como no caso anterior temos a possibilidade de operação síncrona ou assíncrona.

No caso síncrono temos um clock global que é o tempo mínimo de ocorrência dos próximos eventos para todos os processos. Este clock global pode ser centralizado, dedicando um processo que age como sincronizador, ou distribuído. Existe uma série de formas de

implementação tanto para o caso centralizado quanto para o distribuído [8]. Podemos ver em [32] a proposta de um esquema centralizado no qual o clock global é o tempo do próximo evento de iteração ao invés do próximo evento. Neste caso apenas os eventos de iteração são sincronizados, dando aos processos mais liberdade dentro da simulação.

No caso dos algoritimos distribuídos para serem utilizados em simulação orientada a evento de forma síncrona, vamos citar dois exemplos. O anel virtual que é mostrado em [23], permite um aumento de performance da simulação em relação ao caso concentrado, mas possui um complexidade elevada. O outro algoritmo é utilizado em [33] e é chamado de HM2A ("Hierarchical Multi-Bus Multiprocessor Architecture). Ele opera de forma hierárquica ou em uma arquitetura em árvore, possui uma complexidade inferior a do anel virtual.

A operação assíncrona é a que permite o maior potencial de aumento da performance da simulação, isso porque os processos perdem menos tempo esperando pelos outros. Os eventos que não podem afetar os outros procesos podem ser simulados simultaneamente mesmo que ocorram em tempos de simulação diferentes. Neste caso os processos se comunicam entre si com mensagens do tipo "time stamps", que possuem a informação do tempo de simulação de sua geração. São propostas duas classes para este tipo de implementação: conservativa e otimista.

1.2.4.1 Simulação Orientada a Evento Assícrona Conservativa

A técnica conservativa não permite que o tempo possa voltar, isto é, o tempo se simulação anda apenas para frente. O tempo de simulação de um determinado processo não pode exceder ao tempo de simulação de eventos em seus "links" de entrada, mantendo assim a correta ordem cronológica para todos os eventos. Para exemplificar isso consideremos dois processos P1 e P2, sendo que P2 possui um "link" de entrada vindo de P1. Em um determinado instante, onde não existe mensagem vinda de P1, P2 processa todas as mensagens com o tempo de geração ("time stamps") igual ao seu tempo local, depois fica bloqueado. Isso acontece porque P2 pode receber uma mensagem de P1 com o tempo de geração menor que todos os outros "links" de entrada. Assim para manter a cronologia, P2 é forçado a esperar P1.

Esta técnica pode gerar casos de "deadlock". Vamos a seguir mostrar como isso pode ocorrer, considerando o exemplo da figura 4. O processo P1 envia a mensagem (3,m1) para P2, onde o tempo de geração é 3 e a mensagem é m1, e P3 envia a mensagem (2,m2) para P2. Suponha que P3 não tenha mensagem a processar no "link" de entrada. Agora P2 processa a mensagem (2,m2) produzindo (5,m3) que é enviada para P4. Neste ponto, P2 não pode processar a mensagem (3,m1) porque P2 está bloqueado por P3. P3 também está bloqueado por P2, gerando um "deadlock".

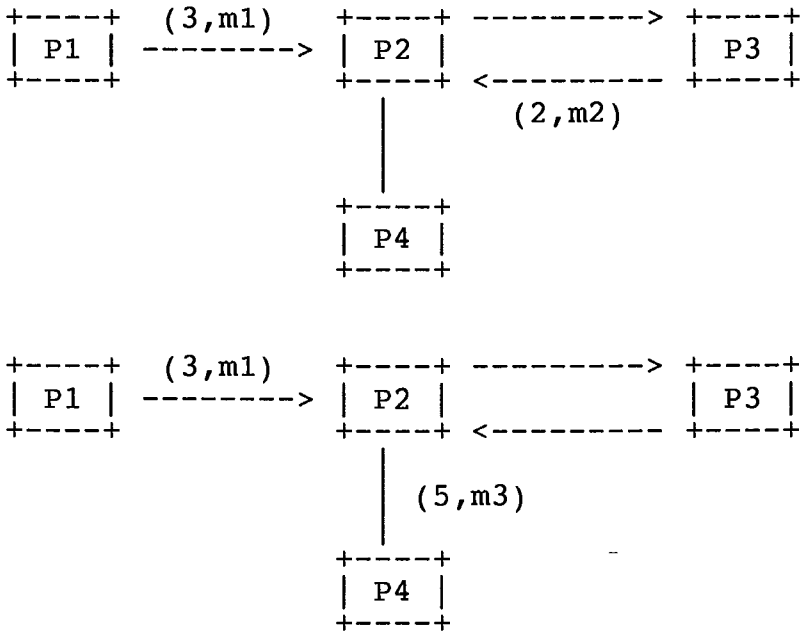


Figura 4. Exemplo de "deadlock"

Existem esquemas auxiliares a fim de evitar o ocorrência de "deadlock", podemos citar como exemplo deste esquemas: uso de mensagens nulas [8]. Este esquema é apresentado em [34]. Mostraremos a seguir, na figura 5, como este mecanismo pode previnir o "deadlock" do exemplo anterior. Neste mecanismo é definida uma variável chamada "lookahead", que é o quanto um determinado processo poderá olhar para o futuro. Definimos que o "lookahead" dos processos P2 e P3 é 2. O número junto ao processos na figura 5, mostra o "clock" local imediatamente antes do envio

da mensagem. P2 envia uma mensagem nula (4,null) para P3. P3 então atualiza seu "clock" local para 4 e envia a mensagem (6,null) para P2. Assim P2 pode agora atualizar seu "clock" para 3 e processar a mensagem (3,m1), produzindo então a mensagem (8,m4) que enviada a P3.

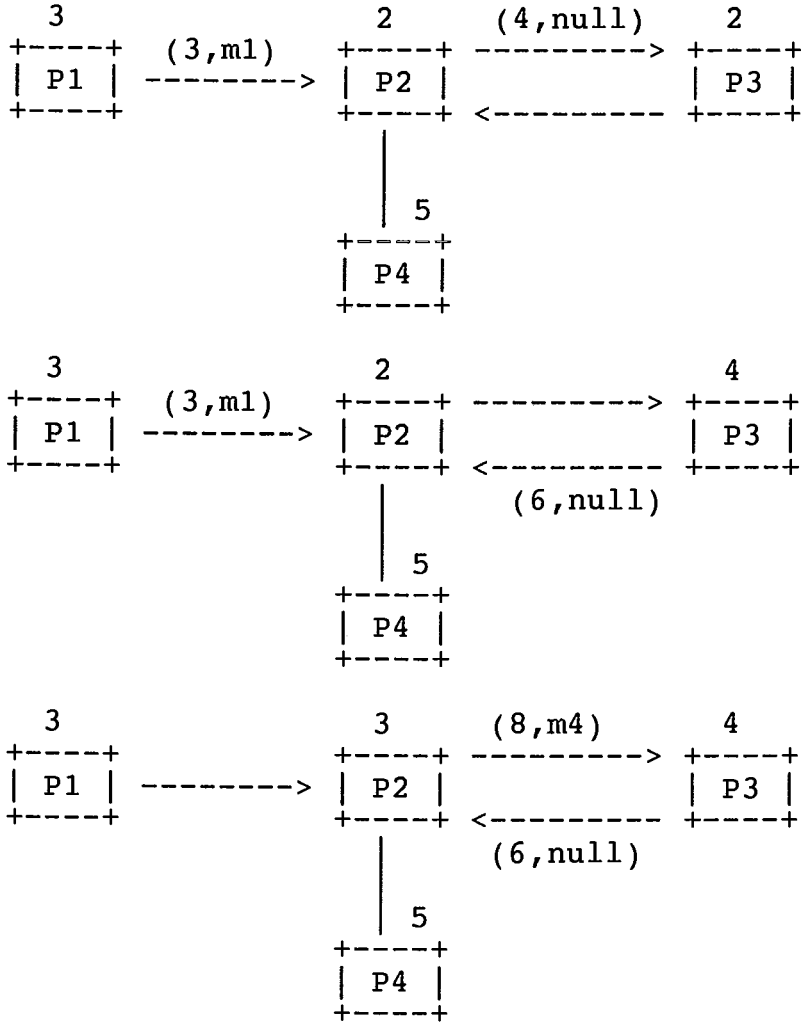


Figura 5. Exemplo de mensagem nula

1.2.4.2 Simulação Orientada a Evento Assícrona Otimista

Na técnica otimista, também conhecida como "time warp", o tempo de simulação de um processo pode exceder ao tempo de simulação dos "links" de entrada de processos ligados a ele, e em caso de erro na cronologia, o tempo de simulação tem que retroceder para corrigir a cronologia ("roll back"). No caso de se realizar um "roll back" o processo correspondente envia anti-mensagens a fim de neutralizar o efeito das mensagens anteriores. Nesta técnica cada processo deve guardar um grande número de informações sobre estados anteriores, que serão necessárias em caso de "roll back". Apesar da técnica "time warp" ser a potencialmente mais rápida, ela é a que mais necessita de memória para sua implementação e possui a maior complexidade.

Nesta técnica as mensagens possuem informação sobre o tempo de sua geração e o tempo de sua chegada. Aqui o "clock" local (que é chamado de "local virtual time" ou LVT) de um processo é setado como o mínimo tempo de recepção dentre as mensagens não tratadas. Temos ainda um tempo chamado de tempo virtual global ou GVT que é o último tempo correto de um processo. Este tempo será usado para permitir o "roll back". Esta técnica não necessita que as mensagens sejam recebidas na ordem em que foram enviadas.

1.3 Visualização no Simulador

Numa simulação de redes é importante para o usuário a visualização do que está acontecendo. Uma série de variáveis podem ser amostradas e visualizadas dentro do simulador.

- Atrasos, média do intervalos de chegadas de mensagens e "throughput", para cada nó do sistema simulado.
- Utilização e "throughput", para cada conecção entre nós ou para toda rede no caso de redes em barra ("broadcast").
- Atraso de mensagens (tempo na fila e tempo no sistema), comprimento, comprimento médio de mensagens e número de mensagens nas filas do nós.
- Utilização ("throughput máximo"), intervalo de chegadas e partidas de mensagens nos transmissores e receptores dos nós.
- "Throughput", atraso da rede.

Dessas variáveis podem ser guardadas estatísticas tais como: número de amostras, soma das amostras, soma do quadrado das amostras, valor máximo, valor mínimo e ultima observação. Existem ainda pacotes já prontos capazes de calcular uma série de valores estatísticos, tais como, média, desvio padrão e variância. Temos como exemplo o pacote mostrado em [35].

Seria importante pordemos visualizar de forma contínua tais variáveis, enquanto a simulação está rodando. Outro ponto importante é a possibilidade também da visualização em forma de histórico de evolução das variáveis de nosso interesse. Poderia existir uma forma de diagramação de visualização que nos permite ver todas as conecções dentro da rede, as possíveis falhas e a passagem dos pacotes com informações de comprimento, origem e destino. Aqui a tela de visualização iria sendo continuamente atualizada de forma a permitir uma animação da rede que está sendo simulada. Este tipo de visualização pode nos dar uma idéia real do funcionamento de uma simulação.

Dentro do programa de visualização poderia existir ainda telas de controle que permitiriam ao usuário do simulador selecionar o tipo e quais variáveis ele deseja observar, podendo ainda selecionar de que forma vai ser visualizada; histograma, curvas, gráficos ou uma lista de dados. Este programa deveria possuir uma série de menus de seleção, podendo chegar a sofisticação de utilizar "mouse" para realizar a seleção dentro destes menus. Podendo ser feita também a utilização de cores para facilitar a visualização.

Deveria ser permitido ao usuário incluir rotinas que façam um tratamento especial para determinada variável em particular, caso haja interesse especial do usuário sobre esta variável.

1.4 Sistemas de apoio ao Desenvolvimento de Simulação Distribuída

Existem sistemas especialmente desenvolvidos para dar suporte a construcão de simulação distribuída, tais sistemas possuem o suporte a comunicação entre processos. Temos como exemplo destes sistemas: o sistema "Jade" [6] e o sistema "Sara" [10]. Ambos os sistemas possuem uma série de regras e funções de suporte para que a aplicação distribuída seja construída. O simulador em [4] usa o sistema "Jade" como base para o seu desenvolvimento. Na figura a seguir temos a estrutura do sistema "Jade".

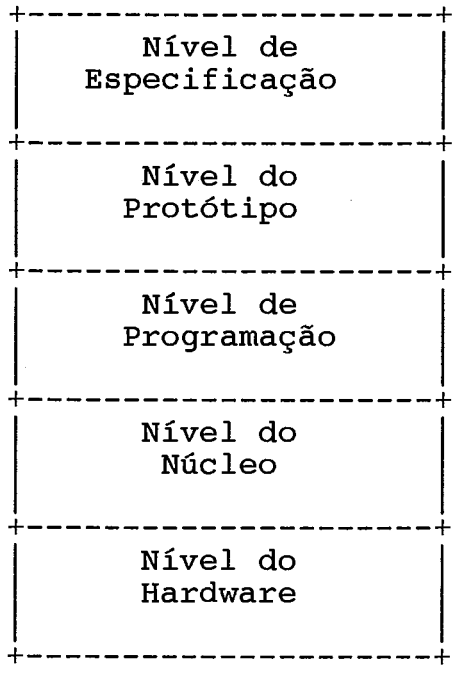


Figura 6. Estrutura do Sistema "Jade"

Temos ainda redes locais de comunicação que dão suporte a construção a aplicações distribuídas, isto é, permitem a troca de mensagens entre processos em máquinas diferentes, podemos apresentar a Amplinet [11] como exemplo. Esta rede local pode rodar em máquinas do tipo micro-computadores da família dos IBM PCs.

Em [12] é mostrado uma forma de projeto de um mecanismo de execução de processos remotos (RPC). Sobre este mecanismo também é possível a construção de aplicações distribuídas.

Nos exemplos [6] e [10] o usuário do sistema que constroi a simulação distribuída tem o seu trabalho reduzido, pois pode usar os mecanismos já disponíveis. Por outro lado nos exemplos [11] e [12] o trabalho do usuário será muito maior.

1.5 Estruturação do Trabalho

Mostramos anteriormente exemplos de simuladores de redes de comunicação. Dentre os simuladores podemos observar tipos diferentes de implementações, suas vantagens e desvantagens. Mostramos ainda como podemos construir uma simulação ditribuída.

Vimos as formas de decomposição, formas de sincronização e possíveis medidas de desempenho. Finalmente mostramos diversos sistemas reais que podem dar suporte a construção de um simulador distribuído.

Com base no nosso objetivo de construir um simulador para protocolos de comunicação e com o conhecimento obtido com o que vimos anteriormente, foi possível definir uma estrutura básica. A estrutura tem como base um simulador distribuído de forma o obter vantagens através da distribuição dos componentes do modelo. Esta estrutura possui algumas vantagens de nosso interesse como por exemplo: permite atingir um grande potencial de paralelismo, possibilita a geração de tarefas resultantes com requisito de memória reduzido e a modularidade dos processos. As técnicas de sincronização vão variar de acordo com o tipo de simulação, isto é, se estivermos com apenas código simulado trabalharemos com a simulação orientada a eventos ou código simulado mais código real onde trabalharemos com a simulação orientada a tempo.

No próximo capítulo mostraremos as topologias básicas a serem simuladas. Faremos a definição do simulador de protocolos de comunicação, mostrando sua estrutura básica. Definiremos cada um dos seus componentes e suas formas de operação. Mostraremos a forma de sincronização do simulador e o fluxo de mensagens dentro dele. Finalmente discutiremos sobre possíveis suportes para implementação.

No capítulo 3 mostraremos a implementação simplificada para uma rede de comunicação com protocolo de enlace CSMA-CD [16]. Apresentaremos as estruturas de dados, formato das mensagens, algoritmos dos processos principais, etc.

No capitulo 4, serão apresentados e discutidos os modelos utilizados em nossas medidas. Descreveremos o ambiente real onde foram realizadas nossas medidas. E finalmente os resultados de nossas medidas são mostrados e analizados.

No Último capítulo, apresentamos nossas conclusões, mostramos nossas dificuldades, limitações e trabalhos futuros.

CAPÍTULO 2

DEFINIÇÃO DO SIMULADOR

2.0 Topologias básicas a serem Simuladas

Vamos selecionar um conjunto de configurações de redes as quais nos interessaremos por simular. Tais configurações são mostradas abaixo.

Redes Síncronas

DQDB (distributed queue dual bus) [15]

CRMA (cyclic-reservation multiple-access) [14]

Redes Assíncronas

CSMA-CD [16]

TOKEN-BUS [17]

TOKEN-RING [18]

Devemos agora conhecer um pouco de cada uma das configurações de redes mostradas anteriormente.

DQDB

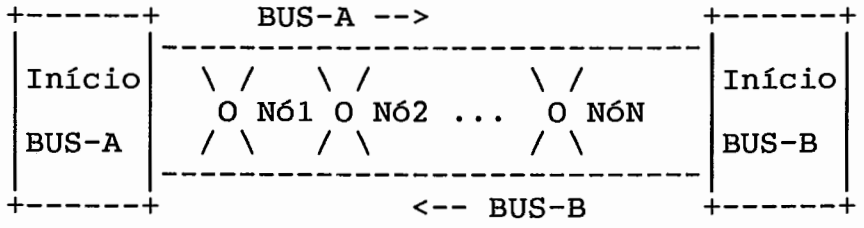


Figura 5. Topologia DQDB

Possui um protocolo síncrono de acesso, isto é, o tempo em cada barramento é dividido em slots e estes slots são gerados pelo início de cada barramento. Cada estação pode acessar qualquer um dos barramentos para transmitir ou receber dados; a escolha do barramento aonde transmitir depende da posição do nó de destino. O acesso é feito através de regras definidas em [15].

CRMA

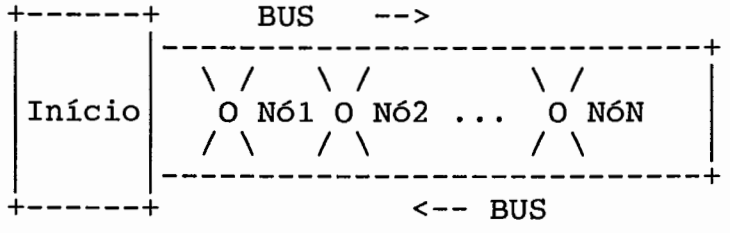


Figura 6. Topologia CRMA

Possui a configuração de um barramento único e unidirecional que passa duas vezes por cada nó. Neste caso o barramento também é dividido em slots. Este protocolo é dividido em dois mecanismos principais: ciclo de reserva e ciclo de acesso. As regras de acesso são mostradas em [14]

CSMA-CD

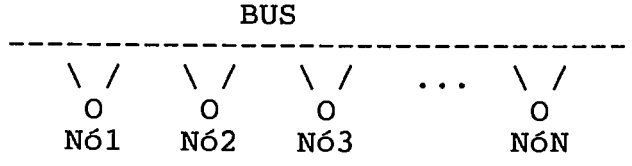


Figura 7. Topologia CSMA-CD

Possui um barramento único e o acesso a este barramento é assíncrono. A estação que deseja transmitir verifica se o meio de comunicação está livre; se estiver ela realiza a transmissão. Este tipo de protocolo está sujeito a colisão. Neste caso existe uma série de regras de acesso e operação descritas em [16].

TOKEN-BUS

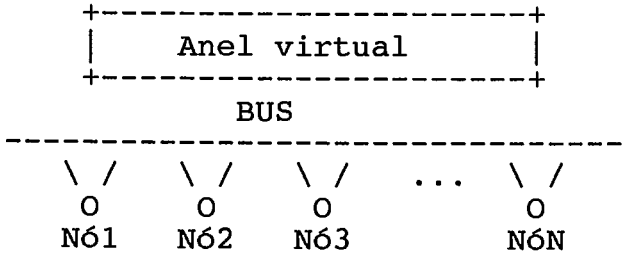


Figura 8. Topoologia TOKEN-BUS

Usa um barramento para fazer a comunicação entre os elementos da rede, mas cria um anel virtual entre estes elementos da rede. Neste anel virtual circula um "token" que controla o acesso ao barramento. Aqui não há o problema de colisão no barramento. As regras de acesso e operação estão descritas em [17].

TOKEN-RING

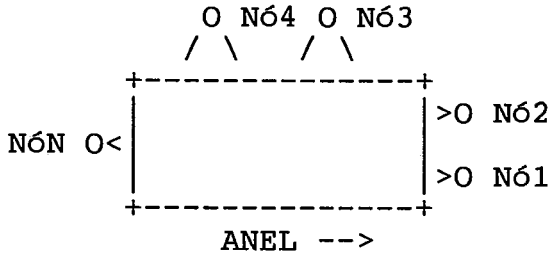


Figura 9. Topologia TOKEN-RING

Usa um anel para fazer a comunicação entre os elemetos da rede. O acesso ao anel é controlado através de um "token" que circula dentro do anel, assim aqui também não existe o problema de colisão dentro do anel. As regras de operação estão descritas em [18].

Podemos ainda incluir uma topologia conexa ponto a ponto que é mostrada abaixo.

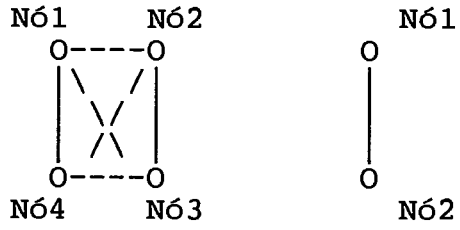


Figura 10. Topologia Conexa

De posse destes exemplos podemos fazer uma análise para saber o mínimo de suporte e funcionalidade que o simulador terá que possuir para poder simular tais configurações. Temos as seguintes conclusões:

- Cada nó deverá possuir no mínimo dois conjuntos transmissor-receptor;
- Devem existir pelo menos dois canais de comunicaçao;
- Não serão considerados parâmetros ligados à potência de sinal da transmissão, isto é, a mensagem transmitida por um determinado nó alcança todos os nós ligados ligados a ele; exceto se for forçada uma falha.
- Não há necessidade à priori de ser criada nenhuma função específica para uma determinada configuração de rede.
- O canal de comunicação deverá ter a capacidade de simular todas as configurações de rede descritas anteriormente.

Ainda antes de apresentarmos a definição do simulador temos um outro ponto que devemos estar atentos. Neste simulador nos interressa poder incluir na simulação código real, isto é, o

código desenvolvido pelo usuário do simulador poderá ser executado dentro do simulador, permitindo assim que o programa (uma camada de protocolo, um serviço, etc) possa ser testado da forma com que irá ser executado em condições reais. Desta forma este simulador se tornará uma grande ferramenta no teste de protocolos e aplicações para redes de comunicação. Pontos importantes para este tipo de característica são: a escolha adequada da técnica de sincronização, o sistema de suporte ao simulador, etc. No caso do sistema de suporte ao simulador deverá ser permitido ter controle sobre o código do usuário.

2.1 Definição Básica do Simulador

A partir de nossos objetivos já descritos e com base no conhecimento apresentado no capítulo anterior, definimos a arquitetura mostrada a seguir para nosso simulador de redes de comunicação. Esta arquitetura permite que todos os processos mostrados a seguir possam estar em processadores diferentes. Sendo que, se for desejado, podem ser executados num mesmo processador.

Ao optarmos por uma arquitetura distribuída para o simulador, temos por objetivo ganhar velocidade na simulação, com o aumento

da performance do sistema; mas devemos desde já estarmos cientes de que apenas configurações que permitam paralelismo vão nos possibilitar tais resultados. Esta arquitetura permite ainda uma facilidade na execução da simulação, pois os processos fazem um uso reduzido de memória permitindo a sua colocação em processadores com menor recurso de memória, tais como micro-computadores. Em [8] são mostradas técnicas de operação em sistemas distribuídos para simulação. Técnicas de sincronização assícronas normalmente são mais eficientes do que as síncronas, mas sua implementação muito mais complexa além de um uso de memória muito maior.

No caso da arquitetura básica do simulador usamos várias das referências como base, dentre elas destacamos [4],[3],[20]. A partir das referências, analizamos seus pontos fortes e suas deficiências. Optamos por uma arquitetura modular com o objetivo de facilitar o uso do simulador em máquinas tipo estações de trabalho e micro-computadores. As formas de sincronização escolhidas por nós tentam também aumentar a performance do simulador, sendo que devem possibilitar ainda a execução de código real dentro do simulador.

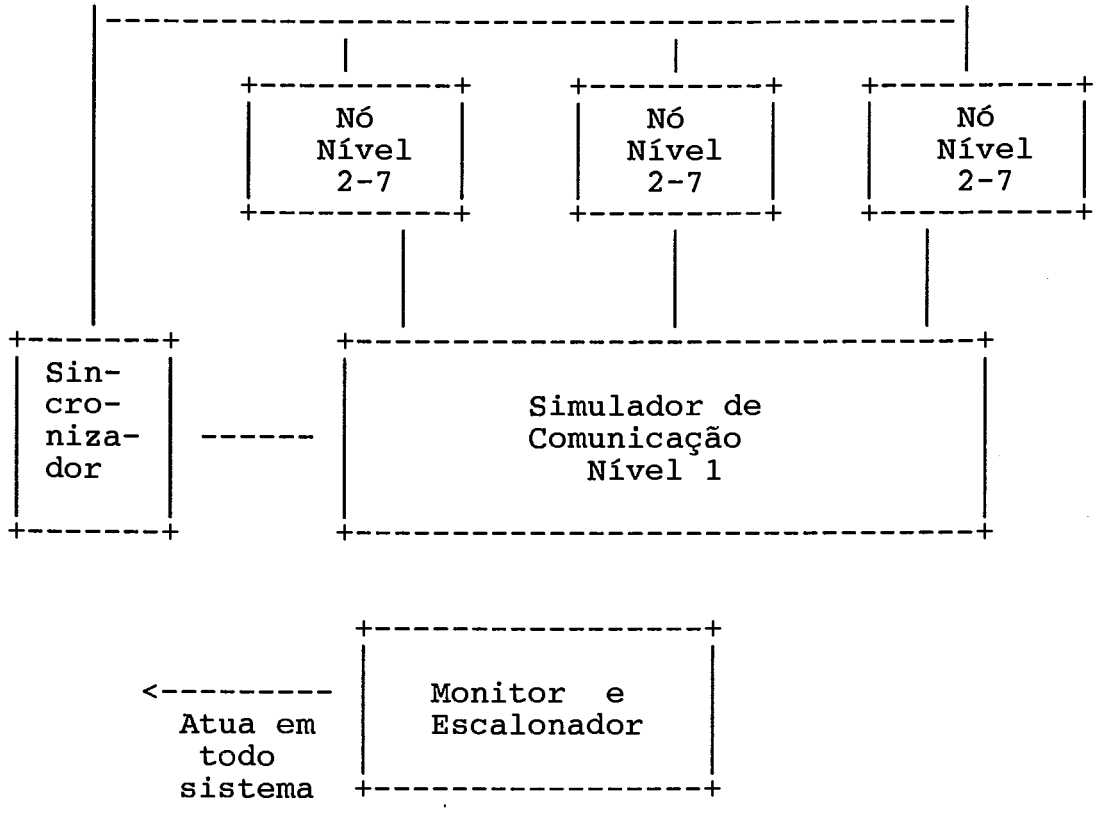


Figura 11. Estrutura do Simulador

PROCESSOS

1) Simulador de Comunicação (SC)

Responsável pela simulação do nível 1, neste processo são simulados todas as conecções entre os nós da rede real. Pode simular falhas e outros problemas do nível físico. Simula ainda todos os atrasos de propagação entre os nós da rede, e colisões que por acaso ocorram. Mantém ainda cada nó de simulação informado

sobre o estado do nível 1. Este processo é único e indivisível.

2) Nó de Simulação (NO)

Responsável pela simulação dos níveis 2 a 7, neste processo os níveis são simulados ou podemos executar o código real que queremos testar. Deve conter um processo deste tipo para cada nó da rede a ser simulada.
Os processos deste tipo podem estar na mesma máquina que o SC ou em máquinas diferentes.

3) Sincronizador (SINC)

É responsável pelo sicronismo do sistema de simulação. Recebe informações dos outros processos do sistema e gera as mudanças no tempo de simulação do sistema. Usa avanço de tempo por eventos, isto é, usamos a técnica de simulação orientada a evento. Mas possui uma variação para possibilitar a execução de código real. Este procedimento é descrito na seção sobre a sincronização dentro do simulador.

4) Monitor/Escalonador

O Monitor é o responsável por observar e a anotar o desenvolvimento da simulação dentro do sistema.
O Escalonador tem como função dar o andamento as eventos externos da simulação em curso.

Vamos agora analizar os elementos que compõe o simulador. Estudaremos como eles são constituídos e como se interconectam com os outros elementos.

2.2 Simulador de Comunicação

Este módulo é o ponto central do nosso simulador, aqui é realizada a simulação do nível 1, camada física. Com base nas arquiteturas de redes que desejamos simular, foi realizada a definição a seguir para o simulador de comunicação.

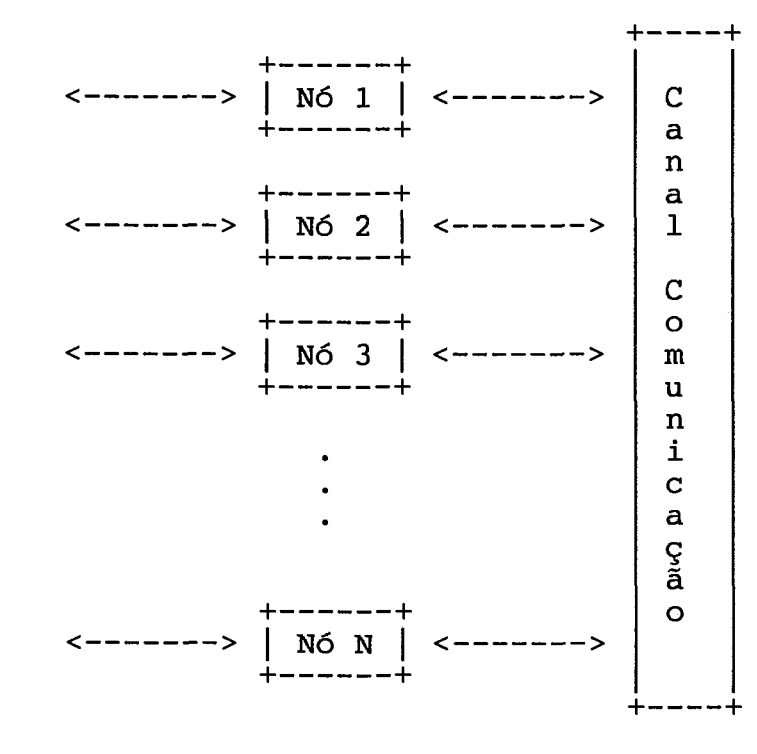


Figura 12. Simulador de Comunicação

Nesse módulo é realizada a simulação de efeitos tais como: transferência de mensagens entre as estações (transmissão e recepção), topologias das redes, atrasos diferentes na entrega das mensagens devido às distâncias diferentes entre estações, interferência entre estações (colisões), presença de mensagem no canal (carry sense), e outros efeitos relativos ao nível 1. A simulação das falhas no nível físico também é realizada neste módulo, apesar de ser comandada pelo módulo monitor/escalonador.

Este módulo possui dois elementos principais que o constitui: os nós e o canal de comunicação. Os nós são elementos responsáveis pela simulação do nó físico e o canal de comunicação simula o canal real. Teremos um nó para cada nó real a ser simulado.

Nesta implementação vemos que dentro do simulador de comunicacão temos o nó físico. Fizemos desta forma objetivando a redução da troca de mensagens entre o simulador de comunicação e os nós de simulação. Esta redução ocorre porque o simulador de comunicação envia mensagens para os nós de simulação apenas quando da ocorrência de eventos significativos, tais como o recebimennto de uma mensagem, ocorrência de uma colisão entre outros eventos. Caso não fosse assim a cada início e término de um pacote o simulador de comunicação teria que enviar uma mensagem para o nó de simulação correspondente. Por outro lado, este tipo de implementação exige uma alteração no simulador de comunicação para simulação de um novo protocolo de acesso. A seguir temos a

estrutura do nó.

2.2.1 NÓ

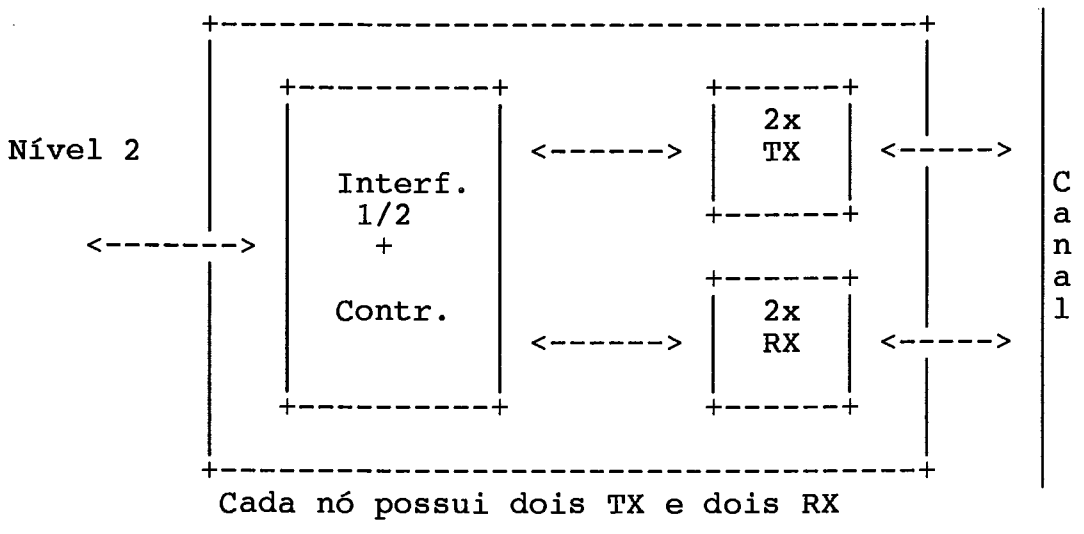


Figura 13. Nó

O nó possui internamente uma unidade responsável pelo controle e pelo interfaceamento com o nível 2 (interface 1/2 + controlador), e onde é mantido o estado do transmissor e do receptor. Este estado pode ser lido e alterado através da interface 1/2 pelo nível superior.

As unidades transmissora e receptora são responsáveis respectivamente pelas operações ligadas à transmissão e à recepção de mensagens no nó. Cada nó possui dois transmissores e dois receptores a fim de poder simular as arquiteturas de barramento duplo.

Vamos ver agora as funções realizadas pelos nós:

. transmissão de mensagens
. recepção de mensagens
. transmissão de "token"
. recepção de "token"
. deteção de portadora ("carry sense")
. deteção de interferência (colisão)
. transmissão de preambulo
. indicação de erro de recepção

Podemos agora descrever a interface funcional entre o nível 2 e o nível 1. Esta interface disponibiliza os serviços fornecidos pelo nó.

Transmissor

Entrada (2 -> 1)	Saída (1 -> 2)
- iniciar transmissão	- final de transmissão
- abortar transmissão	- transmissão abortada
- ligar transmissor	
- desligar transmissor	

Receptor

Entrada (2 -> 1)	Saída (1 -> 2)
- ligar receptor	- mensagen recebida
- desligar receptor	- ativação de indicação
- resetar indicação	

No caso das funções ligadas a transmissão e a recepção, como por exemplo iniciar transmissão, podemos estar nos referindo tanto a uma mensagem como também a um "token".

No receptor é possível verificar se existem indicações ativadas, estas indicações mostram a existência de por exemplo interferência (colisão), ou se o canal está ocupado (existência de "carry sense").

Os blocos transmissor e receptor são máquinas que possuem vários estados e transições, vamos a seguir mostrar estes estados e suas transições.

Transmissor

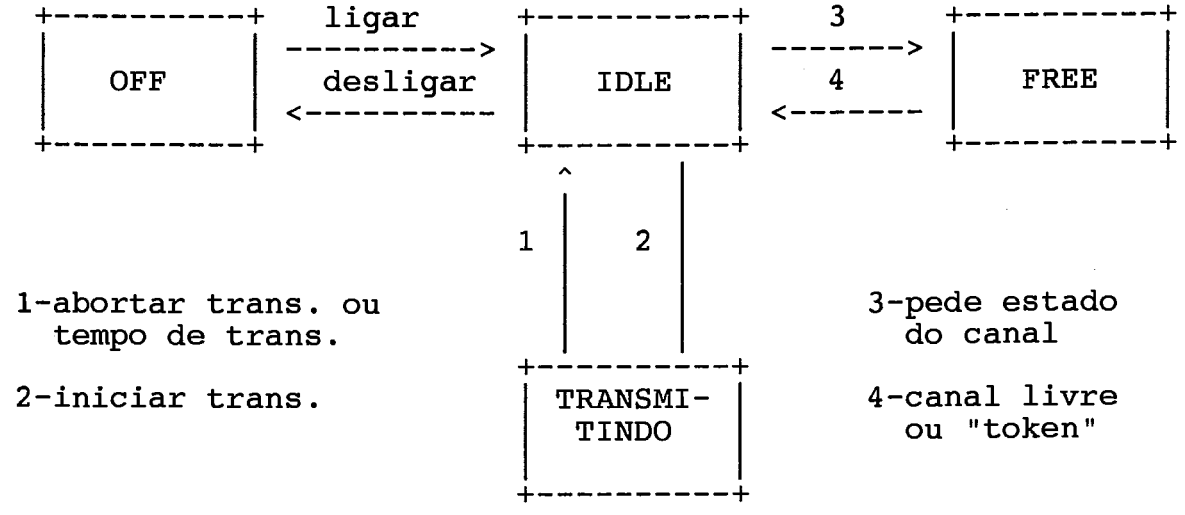


Figura 14. Estados do Transmissor

No primeiro estado OFF, o transmissor não realiza nenhuma função, ele está desligado. A partir do comando de LIGAR ele passa ao estado de operação IDLE, neste estado o transmissor do nó está ativo e pronto para receber pedidos para iniciar transmissão. Antes de fazer o pedido de iniciar a transmissão o nó verifica o estado do canal, através do comando ESTADO DO CANAL, para saber se pode dar inicio a transmissão. No caso de um protocolo CSMA canal livre é não haver portadora, já num protocolo TOKEN-RING é obter o "token". Ao receber o comando ESTADO DO CANAL o transmissor passa ao estado FREE, onde espera a condição de liberação do canal, voltando ao estado IDLE e avisando ao nó sobre a liberação do canal. Ao receber o comando de INICIAR TRANSMISSÃO, o transmissor passa ao estado TRANSMITINDO e avisa ao canal de comunicação um início de mensagem. Após o tempo de transmissão ou se a mensagem for abortada, o transmissor avisa ao canal de comunicação um final

de mensagem e passa ao estado IDLE. Estando o transmissor no estado IDLE através do comando DESLIGAR ele passa ao estado OFF.

Ao término da transmissão de mensagem, por condições normais o transmissor envia através da interface 1/2 a indicação de fim de transmissão para o nível 2. Se for pedido para transmissão ser abortada, da mesma forma ao término da operação, é enviada através da interface 1/2 a indicação de transmissão abortada.

A escolha desta estrutura para o transmissor, uma estrutura bem simples, foi realizada com o intuito de permitir a construção de qualquer método de acesso no nível 2, já que não nos prendemos a características de nenhum protocolo em particular.

Receptor

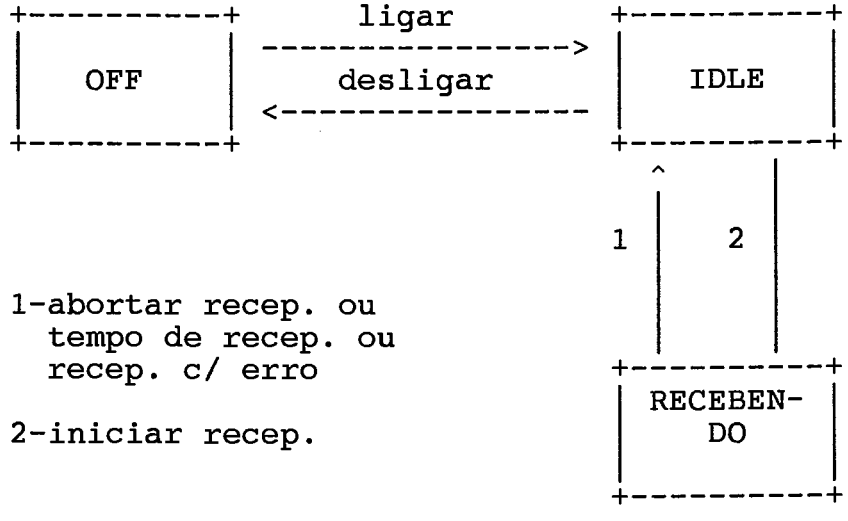


Figura 15. Estados do Receptor

No estado OFF o receptor está desligado, não realizando nenhuma função. Ao receber o comando LIGAR o receptor passa ao estado IDLE, neste estado o receptor está ativo e apto a receber mensagens do canal de comunicação. Quando o canal precisa transferir uma mensagem para o nó, ele avisa ao receptor o início de uma mensagem e o receptor passa ao estado RECEBENDO. Após o término da mensagem o canal avisa ao receptor que passa para o estado de IDLE, avisando através da interface 1/2 ao nível 2 o recebimento de uma mensagem. Se houver erro de recepção na mensagem o receptor avisa através da interface 1/2 ao nível 2 o recebimento de uma mensagem com erro. Usando a função de DESLIGAR o receptor passa do estado IDLE para OFF.

Existe uma função de ativação de indicação, que o receptor avisa ao nível 2 através da interface 1/2, quando há alguma mudança nos indicadores de condições. Estes indicadores mostram a existência de condições tais como: colisão, portadora (carry) e etc.

Quanto a arquitetura do receptor podemos considerar o mesmo comentário apresentado para o transmissor, com o objetivo de podermos ter um simulador versátil.

2.2.2 Canal de Comunicação

Como já dissemos anteriormente o canal de comunicação é o elemento

do simulador de comunicação responsável pela simulação do canal de comunicação real. Ele transfere mensagens de um nó para outro nó dentro da rede, e tem as seguintes funções:

. Gera atrasos na entrega das mensagens, correspondentes ao atraso da rede real.
. Cria a arquitetura física da rede em simulação.
. Gera as falhas na topologia em teste.
. Gera erro nas mensagens transferidas por ele.

O canal de comunicação pode ser visto como por exemplo um segmento de cabo coaxial ou fibra ótica que são os elementos reais que levam as mensagens entre os nós de uma rede de comunicação.

Existe uma tabela associada a cada nó, dentro do canal de comunicação, que contém informações suficientes para que o canal possa entregar uma cópia da mensagem transmitida por um determinado nó, para cada um dos nós conectados a ele com o atraso correspondente. A tabela possui todas as conecções de um determinado nó com o resto da rede, as distâncias, os atrasos associados e etc.

O canal de comunicação possui uma interface com o nó, através da qual são realizadas a transmissão e a recepção de mensagens. A partir desta interface podemos realizar as seguintes funções:

Nó -> Canal de comunicação

. início de mensagem
. término de mensagem

Canal de comunicação -> Nó

. início de recepção
. término de recepção

Existe ainda uma interface entre o canal de comunicação e o escalonador, pela qual são simuladas as falhas na topologia e as mensagens recebidas com erro.

No simulador podemos tem dois canais de comunicação independentes com a finalidade de podermos simular redes duplas ou redes com dois canais de comunicação para aumentar a tolerância a falhas.

2.2.3 Comunicação dentro do Simulador

Podemos agora analisar como se processa a passagem de mesagens dentro do simulador. As mensagens são geradas nos nós de simulação e chegam ao simulador de comunicação através da interface 1/2. A seguir é mostrado o encaminhamento das mensagens.

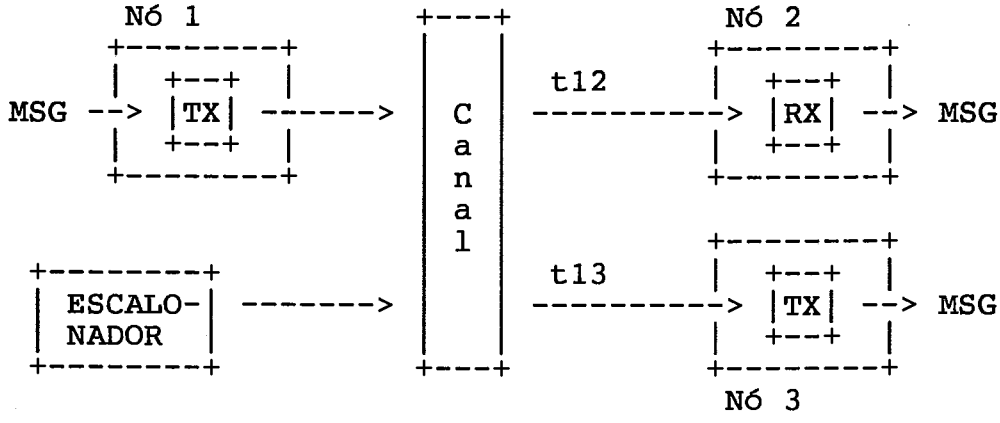


Figura 16. Comunicação no Simulador

As mensagens chegam ao nó 1 através do módulo controlador-interface 1/2, estando o transmissor livre a mensagem é passada para ele, que muda seu estado e atualiza algumas variáveis da mensagem tais como: qual o canal e a identificação do transmissor. Depois a mensagem é entregue ao canal de comunicação correspondente para ser transmitida. O canal cria cópias da mensagem para entregar a todos os nós conectados ao nó transmissor, no nosso exemplo os nós 2 e 3 estão conectados ao nó 1. O canal entrega aos nós correspondentes repeitando o atraso definido para cada conecção. Na hora da entrega das mensagens o canal pode através de um comando do escalonador romper uma conecção, isto é, não entregar uma das mensagens, ou criar outras condições anormais de operação. Ao chegar no nó de destino a mensagem entra pelo receptor do canal corespondente,

posteriormente via a interface 1/2 chega aos níveis superiores.

Para entendermos melhor o processamento de uma transmissão dentro de nosso simulador, temos que conhecer seus eventos básicos. No início de uma transmissão é programado um evento INICIO_TX. Ao ser executado este evento, ele programa um novo evento FIM_TX com o tempo correspondente a transmissão da mensagem. Na execução do evento FIM_TX a mensagem e copiada para seu destino. Todos os nós entre o nó transmissor e o nó receptor também terão a indicação de incio e final de transmissão deste pacote. Ao efetuarmos a transmissão entre dos nós de simulação o caminho da mensagens é mostrado a seguir. A mensagem é enviada do nó de simulação para o simulador de comunicação através da rede real que suporta o simulador. Ao chegar no simulador de comunicações a mensagem é transferida dentro do simulador do nó origem para o nó destino através do uso de ponteiros para mensagem. Novamente a mensagem e transferida através da rede real para o nó de simulação de destino.

2.3 Nó de Simulação

O nó de simulação contém os níveis 2 a 7 e se comunica com o simulador de comunicação através da interface 1/2 descrita no simulador de comunicação. É no nó de simulação que o usuário pode executar código real além do código simulado. Consideremos que o usuário deseje simular um protocolo de nível de transporte (nível

4), neste simulador não será necessário ele construir um modelo de simulação para seu protocolo, poderá usar seu código real. Desta forma a simulação se torna mais versátil e estará sujeita a um erro muito menor do que se fosse usado apenas um modelo de simulação.

Deverá existir um nó de simulação correspondente a cada nó da rede que está sendo simulada. A seguir temos um diagrama do nó de simulação.

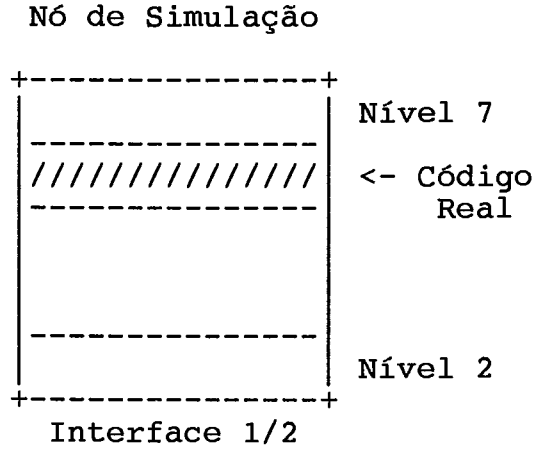


Figura 17. Nó de Simulação

A inclusão de código real junto com o código de simulação exige alterações nas técnicas de sincronização. Isso é mostrado mais adiante quando falarmos sobre o sincronizador.

O código real que roda dentro do nó de simulação possui algumas características, baseadas em regras especiais, a fim de se adaptar

ao funcionamento do simulador. Na implementação será realizada a definição das regras para utilização de código real. Por exemplo, o código real não poderá conter acessos a nenhum sistema operacional ou desabilitar/habilitar interrupções. Para o caso de protocolos estes procedimentos descritos acima não são realizados, pois não são necessários. Temos ainda que nos preocupar na implementação com o sistema de suporte ao simulador que permita manter controle sobre um código em execução, isto é, poderemos ativar ou parar a execução do código real sobre o controle do simulador. Deve ainda existir uma correlação entre o tempo real e o tempo virtual de simulação. Vamos citar um exemplo para isso, originalmente o código real executa numa máquina muito rápida. Mas o simulador é executado numa máquina lenta. Por isso devemos ter a correlação mostrada anteriomente.

2.4 Sincronizador

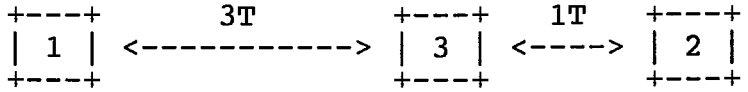
É o modulo responsável pela sincronização dentro do simulador. Foi usado para a parte apenas simulada a forma de simulação orientada a eventos de forma síncrona [8]. Foi escolhida esta forma de operação pelo aumento de performance em relação a simulação orientada a tempo, e a forma síncrona é a mais simples de ser implementada. Por outro lado a forma assíncrona não causaria grande aumento de performance pois o simulador de comunicação é um

elemento centralizador dentro de nossa implementação, não permitindo grande assincronismo no sistema. Já no caso de execução de código real passamos a operar na simulação orientada a tempo, pois é a que mais se adequa a discretização de eventos contínuos [8]. Aqui continuamos a operação de forma síncrona. Criamos um mecanismo capaz de chavear de maneira automática as duas formas de operação e é descrito quando mostramos a sincronização do simulador executando com código real.

Para analizarmos o funcionamento e a sincronização do simulador vamos dividi-lo em duas partes, apenas simulado e simulado mais executado.

Vamos abaixo exemplificar o funcionamento de um sistema orientado a evento.

Temos a seguinte rede exemplo, três nós (Nó 1, Nó 2 e Nó 3), que possui o seguinte diagrama de formação. Onde T é a unidade mínima de tempo do nosso sistema.



Inicialmente temos os seguintes eventos que irão ocorrer no sistema.

t = 1, envio de mensagem pelo Nó 2 para o Nó 3

t = 3, envio de mensagem pelo Nó 1 para o Nó 3

t = 5, envio de mensagem pelo Nó 1 para o Nó 3

t = 6, envio de mensagem pelo Nó 2 para o Nó 3

O Nó 3 é um nó de serviço para os outros nós da rede, ao recebe uma mensagem ele devolve depois de 2T uma mensagem de resposta para o nó origem do pedido.

Assim a partir da tabela inicial de eventos externos, o sincronizador irá ativar o sistema para rodar até um ponto seguro. Agora vamos mostrar a sequência de eventos no sistema.

t = 1, Nó 2 envia mensagem para Nó 3; é programado no sincronizador recepção no Nó 3 para t = 2

t = 2, Recepção no Nó 3; é programado no sincronizador o envio de mensagem do Nó 3 para o Nó 2 para t = 4 (resposta do serviço requisitado)

t = 3, Nó 1 envia mensagem para Nó 3; é programado no sincronizador recepção no Nó 3 para t = 6

t = 4, Nó 3 envia mensagem para Nó 2; é programado no sincronizador recepção no Nó 2 para t = 5

t = 5, Recepção no Nó 2; Nó 1 envia mensagem para o Nó 3; é programado no sincronizador recepção no Nó 3 para t = 8

t = 6, Nó 2 envia mensagem para Nó 3; é programado no sincronizador recepção no Nó 3 para t = 7; Recepção no Nó 3; é programado no sincronizador o envio de mensagem do Nó 3 para o Nó 1 para t = 8 (resposta do serviço requisitado)

t = 7, Recepção no Nó 3; é programado no sincronizador o envio de mensagem do Nó 3 para o Nó 2 para t = 9 (resposta do serviço requisitado)

t = 8, Nó 3 envia mensagem para Nó 1; é programado no sincronizador recepção no Nó 1 para t = 11; Recepção no Nó 3; é programado no sincronizador o envio de mensagem do Nó 3 para o Nó 1 para t = 10 (reposta do serviço requisitado)

t = 9, Nó 3 envia mensagem para Nó 2; é programado no sincronizador recepção no Nó 2 para t = 10

t = 10, Nó 3 envia mensagem para Nó 1; é programado no sincronizador recepção no Nó 1 para t = 13; Recepção no Nó 2

t = 11, Recepção no Nó 1

t = 13, Recepção no Nó 1

Sistema apenas simulado

Assim consideraremos inicialmente um sistema apenas simulado. Cada nó de simulação possui uma fila de eventos, com dois tipos distintos. Os que chamamos internos (locais), são eventos que não podem influenciar o estado dos outros elementos do simulador. Sendo assim os nós de simulação poderiam executar eventos internos independentes dos outros nós. Os outros são os eventos externos, entendidos como iterações entre nós (mensagens transmitidas e recebidas), chegadas externas (pacotes do usuário) e eventos de falhas que podem ser programados.

De uma forma simples podemos realizar a sincronização fazendo com que todos os eventos, internos ou externos, das filas dos nós sejam sincronizados. O processo sincronizador recebe a lista dos próximos eventos de todos os nós e determina o avanço do tempo de simulação como sendo o tempo mínimo entre todos os eventos do sistema. Em [23] é mostrada está forma de operação e de acordo com a referência produz um resultado satifatório quanto a performance em relação a técnicas mais sofisticadas.

Para tentarmos melhorar o desempenho do simulador, podemos usar uma variação na técnica de sincronização descrita anteriormente.

Aqui apenas os eventos externos tem de ser progamados no sincronizador, de forma a fazer parte de uma fila de eventos do sistema. O sincronizador recebe os eventos externos de todos os elementos do sistema e monta uma fila geral de eventos, e informa ao elementos do sistema o tempo até o qual podem avançar com segurança. Este tempo de segurança é o tempo mínimo de todos os eventos externos do sistema. Esta forma de implementação é mais complexa que a anterior, mas por outro lado permite que os nós de simulação possam realizar seu processamento de forma mais livre.

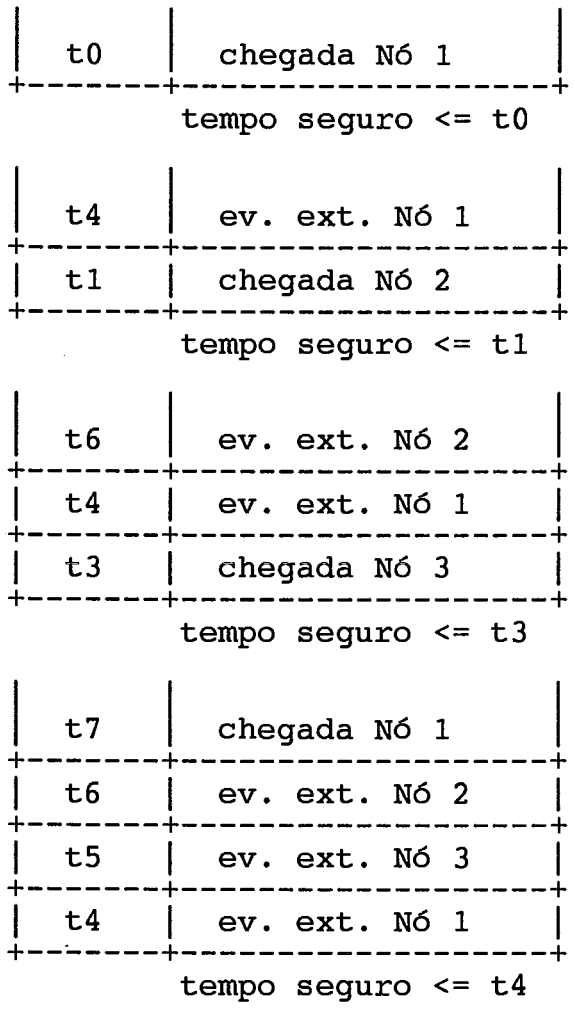
O simulador de comunicação age da mesma forma que os nós de simulação, tanto no primeiro caso, como no segundo.

Vamos exemplificar agora a segunda técnica proposta para sincronização de nosso simulador.

Lista de eventos externos

. t0 -> chegada de mensagem do usuário no Nó 1 (programa t4)
. t1 -> chegada de mensagem do usuário no Nó 2 (programa t6)
. t3 -> chegada de mensagem do usuário no Nó 3 (programa t5)
. t4 -> evento externo no canal originado no Nó 1
. t5 -> evento externo no canal originado no Nó 3
. t6 -> evento externo no canal originado no Nó 2
. t7 -> chegada de mensagem do usuário no Nó 1

Fila de eventos externos



Sistema simulado mais código real

Agora temos que considerar também a existência de nós especiais que podem executar código real. Neste caso vamos considerar duas

possíveis implementações e tentar optar pela melhor das duas.

Implementação 1

O avanço do tempo virtual ficará condicionado ao avanço do tempo nos nós que executam código em tempo real. A partir de um ponto inicial de execução, o tempo virtual será adiantado de unidade em unidade de acordo com o passar do tempo real (código real). Esta operação será realizada por um processo chamado de discretizador, que programará eventos fantasmas, a cada unidade de tempo virtual, no sincronizador em quanto o código real estiver executando. Desta forma passamos a operar a simulação orientada a tempo, isto é, ela é sincronizada por "ticks". Os eventos gerados dentro de um "tick" são sempre considerados como ocorridos no final deste "tick", assim temos erro mínimo inerente a este tipo de procedimento. Devemos ainda aplicar um fator de conversão entre o tempo real dentro do simulador e este tempo no ambiente operacional real (ex: código real simulado rodando numa estação de trabalho SUN e ambiente real rodando em um micro-computador tipo IBM PC).

Implementação 2

O avanço do tempo virtual será independente do avanço do tempo nos nós que executam em tempo real, mas existirá um ponto de sincronização. A partir do ponto de sincronização os nós de simulação são liberados dentro de uma janela de segurança, e os nós executando código real também o são dentro da mesma janela.

Esta janela é baseada na fila geral de eventos do sincronizador. Se dentro desta janela algum nó executando código real gerar um evento externo, a execução do código real é suspensa (para isso precisamos de recursos de hardware) e o sincronizador para os nós que estão rodando o código simulado. Os nós executando código real ficam bloqueados e os nós simulados voltam ao ponto de sincronização para executar até o ponto onde foi gerado o evento externo. A partir daí este será o novo ponto de sincronização. Devemos considerar que vai existir um atraso até que todos os nós estejam bloqueados, só a partir daí podemos voltar a executar a simulação. Este procedimento será realizado enquanto algum nó estiver executando código real. Como foi comentado na implementação anterior devemos aqui também aplicar um fator de conversão entre o tempo real dentro do simulador e este tempo no ambiente real. Para não ser necessário o uso de recursos de hardware neste tipo de implementação seria necessário utilizarmos técnicas assíncronas, conservativas ou otimistas, alta complexidade para sincronização do simulador.

Após analizarmos ambas alternativas podemos observar que a primeira alternativa possui a implementação mais simples que a segunda já que não exige guardar nenhum estado global do sistema. Por outro lado torna a simulação extremamente sincronizada diminuindo seu paralelismo. Quando a performance das duas alternativas temos que fazer algumas considerações sobre o sistema.

a) Se considerarmos que o tempo de execução das tarefas simuladas é muito menor que o das tarefas de execução real, temos como consequência que a segunda implementação será mais rápida, já que ela permite a execução da tarefa real de forma contínua até que seja gerado um evento externo ou que termine sua execução. Mas as tarefas simuladas podem ter que ser executadas com repetições, passando várias vezes pelo mesmo trecho.

b) Caso a situação do item a não seja verdade, isto é, o tempo de execução das tarefas simuladas não for despresível em relação ao das tarefas em tempo real, a primeira será a melhor. Já que nesta implementação os dois tipos de tarefas executam ao mesmo tempo e são executadas apenas uma vez. Devemos considerar aqui também o tempo gasto na geração da discretização do tempo.

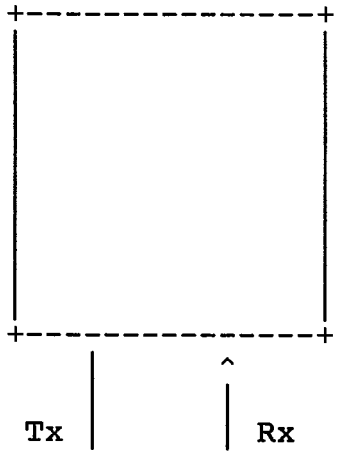
Assim por uma questão de facilidade de implementação e assumindo que os tempos de execução das tarefas simuladas e a das tarefas reais são semelhantes para nosso sistema de simulação optamos pela utilização da primeira implementação. Neste tipo de implementação o tempo de execução da tarefa de código real será a soma de todos os "slots" de tempo dado a ela para executar. O elemento chamado de discretizador é responsável pela totalização destes tempos.

A existência do discretizador permite o chaveamento automático entre a técnica de simulação por eventos quando temos apenas código simulado para a técnica de simulação por tempo quando temos código simulado mais código real. Isso porque ele gera eventos a

cada "tick", fazendo com que a simulação de eventos original seja igual a uma simulação por tempo ao colocarmos código real dentro do simulador.

Agora vamos fazer uma apresentação dos elementos principais do sistema e suas relações com o processo de sincronização dentro do simulador.

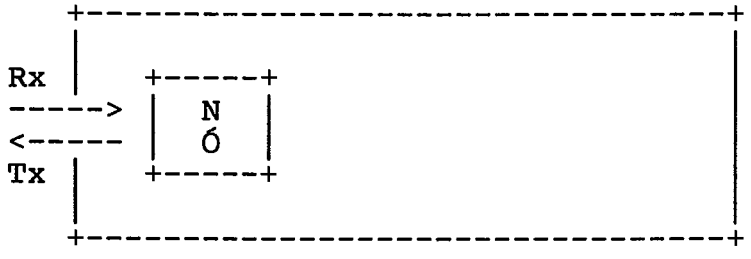
1) Nó de Simulação



Os nós possuem filas
de eventos locais

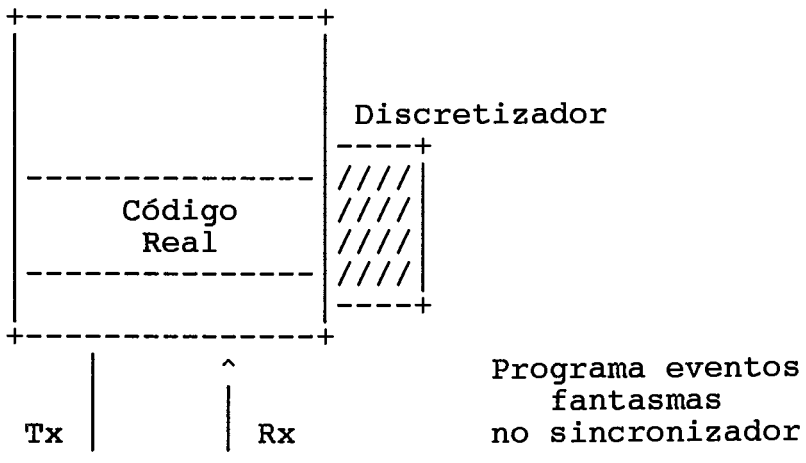
Os nós programam eventos
externos no sicronizador

2) Simulador



O simulador programa
eventos externos
no sincronizador

3) Nó de Simulação (executando código real)



4) Sincronizador

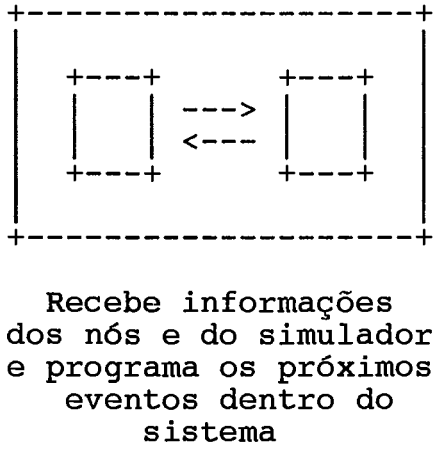


Figura 18. Elementos da Sincronização

Na implementação do protótipo vamos considerar a técnica de sincronizar a partir dos eventos externos de todos os módulos do

sistema. Ficando os eventos internos apenas com sincronização interna a cada módulo do sistema.

O sincronizador é constituido de um processo que possui dois estados principais: avaliação do tempo de simulação e avanço do tempo de simulação. Durante a avaliação do tempo de simulação o sincronizador recebe informações de todos os outros módulos do sistema (simulador de comunicação, nós de simulação, etc) sobre seus próximos eventos. A partir destas informações o sincronizador avança o tempo de simulação para o próximo evento, que é o evento de menor tempo entre todos os módulos do sistema. O sincronizador passa ao estado de avanço do tempo de simulação, avisando a todos os outros módulos do sistema o novo tempo de simulação. Passando novamente ao estado de avaliação do tempo de simulação.

Estados do Sincronizador

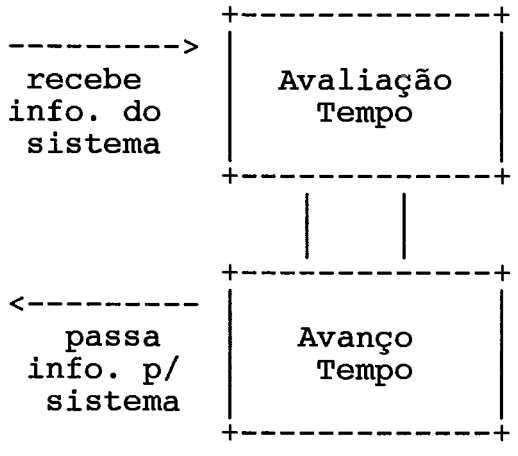


Figura 19. Estados do Sincronizador

Temos aqui que realizar considerações sobre o sistema que suporta o simulador. Nele a troca de mensagens deve ocorrer sem erros ou perdas. Nós assumimos que a rede real que suporta o simulador é um sistema confiável. Existem técnicas tais como "Two-phase commit protocol" ou "send-receive-replay" que só permitem o prosseguimento do processo após o recebimento da confirmação da mensagem.

2.5 Monitor/Escalonador

2.5.1 Monitor

O monitor possui interface com os outros módulos do sistema de forma a permitir a coleta de dados estatísticos sobre a simulação em andamento. A estrutura básica do monitor é um processo coletor de dados e vários processos monitores, um para cada variável a ser analisada. No sentido de simplificar o processo monitor, nele é feito apenas um tratamento inicial dos dados antes de guardá-los dentro de um arquivo do sistema, que é nossa base de dados principal.

2.5.2 Escalonador

Faz o controle do andamento da simulação em curso, pode alterar a configuração da rede, gerar falhas ou mudar a taxa de geração de mensagens. O escalonador possui interface com o simulador de comunicação e com os nós de simulação. Através desta interface ele realiza as mudanças requisitadas pelo usuário dentro do sistema de simulação. Poderiam ser geradas falhas pré-programadas ou falhas aleatórias.

Numa forma de operação as informações sobre o andamento da simulação estão contidas num arquivo a ser lido pelo escalonador, e as operações ali indicadas são realizadas de forma passo a passo. Outra forma de operação é a em linha, isto é, a modificação na simulação é realizada no instante desejado pelo usuário.

Estes resursos de mudança de configuração da rede em simulação e geração de falhas são importantes quando desejamos verificar o funcionamento ou a capacidade de recuperação dos protocolos da rede real em caso de falhas na operação normal. Seria importante podermos ainda gerar mudanças na taxa de geração de mensagens em determinado nó ou em toda rede e verificar o resultado em relação a rede como um todo. Ainda seria possível gerar mensagens recebidas com erro e verificar o procedimento de recuperação de falhas do protocolo em teste dentro do simulador.

2.6 Opções para implementação

Antes de partirmos para implementação do simulador, devemos fazer uma série de escolhas ou opções. Estas escolhas estão relacionadas com: qual suporte de hardware será utilizado para construção do simulador; que sistema operacional utilizaremos; qual a linguagem de programação que nos servirá como base para codificação do programa do simulador; que subsistema de comunicação utilizaremos para troca de mensagens; se será possível utilizarmos algum sistema de suporte ao desenvolvimento de aplicações distribuídas.

Para realizarmos a escolha, fizemos uma pesquisa das diversas possibilidades de cada um dos itens descritos acima. No capítulo anterior fazemos referência aos sistemas de suporte ao desenvolvimento de aplicações distribuídas. A decisão foi tomada usando critérios de disponibilidade, facilidade de utilização, documentação, eficiência, possibilidade de expansão, portabilidade e custo. Assim muitas vezes a que seria a melhor solução apresenta um critério que não é satisfeito, como por exemplo o custo ou a disponibilidade.

Baseado em nossas pesquisas, o estado da arte de sistemas para implementação de protótipos em simulação distribuída tem o perfil

que mostaremos a seguir. Ele é o seguinte: estações SUN, rodando sistema operacional UNIX, interligadas por uma rede local Ethernet ou Token-Ring, com o sistema JADE de suporte a aplicações distribuídas e os programas do simulador escritos na liguagem de programação SIMULA. Contudo esta configuração não estava disponível para nosso uso, sendo assim optamos por uma configuração realista, isto é, disponível para nosso uso. Esta configuração é a seguinte: estações PCs e ATs, rodando o sistema operacional MSDOS, interligadas por uma rede local Ethernet, com o software AMPLIWARE que permite a comunicação entre processos dentro da rede. Utilizaremos a linguagem de programação C por sua portabilidade, mas temos que utilizar rotina em assembler para comunicação com o AMPLIWARE.

A partir desta configuração, de posse de sua documentação completa, procedimentos de uso, etc, iniciamos a implementação do simulador.

CAPÍTULO 3

IMPLEMENTAÇÃO DE MODELO SIMPLIFICADO

3.0 Definição do Modelo

Vamos realizar a implementação do simulador de comunicação para um protocolo de nível de enlace do tipo CSMA-CD [16]. Nosso exemplo terá os seguintes módulos: um sincronizador, um simulador de comunicação e nós de simulação. Estes nós de simulação criam um ambiente tipo servidor-estação de trabalho. Vamos usar este tipo de aplicação para mostrar a validade da utilização de nós de simulação independentes podendo processar em paralelo. Neste tipo de aplicação servidor-estação de trabalho temos um grande número de eventos ocorrendo tanto nas estações de trabalho quanto no servidor. Nesse ambiente o servidor não toma iniciativa de enviar mensagens, apenas responde às requisições realizadas pelas estações de trabalhos. Usaremos como referência uma descrição do protocolo CSMA-CD contida em [19]. Mostramos a seguir um exemplo do diagrama de uma rede que poderá ser usada como modelo.

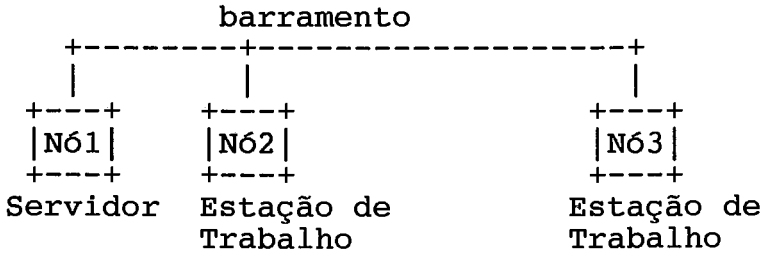


Figura 20. Rede Modelo

3.1 Definição do Sistema de Desenvolvimento

No capitulo anterior já haviamos definido o conjunto de suporte ao desenvolvimento do nosso simulador de comunicação. Agora vamos apresentar o conjunto de ferramentas usadas na nossa implementação.

O hardware utilizado foram micro-computadores da linha IBM PC-XT com a interface AC106 para rede local padrão IEEE 802.3 [16] (protocolo CSMA-CD), fabricada pela Amplus Informática.

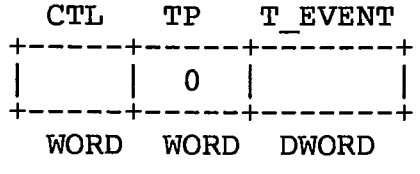
Já o ambiente utilizado foi o sistema operacional MSDOS 4.01 mais o sistema operacional AMPLIWARE AWDOS/106 2.05 que controla a comunicação com a rede. O AMPLIWARE possui primitivas de comunicação entre processos em máquinas diferentes dentro da rede e possui ainda primitivas de sincronização entre processos do tipo

WAIT e SIGNAL. Estas primitivas do AMPLIWARE que foram utilizadas em nosso simulador, estão descritas no anexo I. Para construção do programa foi utilizada a liguagem de programação C e a linguagem Assembler (para acessar diretamente as primitivas do sistema). Os compiladores utilizados foram o MicroSoft C 4.0 e MicroSoft MacroAssembler 4.0, conjuntamente com o linkeditor MicroSoft Link 3.50. Como ferramenta de depuração utilizamos o MicroSoft CodeView 1.0 e o software Periscope 2.02.

3.2 Mensagens Trocadas no Simulador

O simulador é organizado em processos: processo sincronizador, processo simulador de comunicação e os processos nós de simulação. Dentro do simulador são trocados diversos tipos de mensagens entre estes processos. Estas mensagens podem ser mensagens de dados ou de controle. Antes de apresentarmos uma descrição detalhada dos processos envolvidos, faremos a seguir uma descrição dos diferentes tipos de mensagens.

a) Mensagem de informação de tempo virtual



CTL - tipo do processo que enviou
a mensagem (0 = sincronizador,
1 = simulador do canal,
2 = nó de simulação)

TP - tipo da mensagem
(0 = informação do tempo virtual)

T_EVENT - tempo virtual do próximo
evento

b) Mensagem de requisição/informação dos estados dos processos do simulador e transmissão/recepção de mensagens

CTL	TP	FUNC	EST	DADOS
	1			
WORD	WORD	WORD	WORD	BYTE

CTL - tipo do processo que enviou
a mensagem (0 = sincronizador,
1 = simulador do canal,
2 = nó de simulação)

TP - tipo da mensagem
(1 = req./inf. dos estados dos
processos do simulador e trans./
recep. de mensagens)

FUNC - número da req./inf.
(6 = INIC_TX(transmissão de pacote),
10 = BUS_LIVRE(inf. de final de "wait"),
11 = RESP_REQ_TRANS(req./inf. do estado do canal),
12 = COLISAO_DET(inf. de ocorrência de colisão),
13 = INICIO_RECEP(recepção de pacote),
15 = ERRO_RECEP(inf. de erro de recepção),
16 = INFO_TIME(req. de tempo virtual))

EST(AÇÃO) - nó de origem da mensagem

DADOS - campo de dados no caso de
transmissão e recepção de
pacotes

Ao descrevermos os processos integrantes do simulador de comunicação será mostrada cada mensagem utilizada por tais processos e os valores que devem assumir cada campo das mensagens.

3.3 Processo Sincronizador

É instalado em "background" dentro do AMPLIWARE, isto é, será um processo interno ao sistema. Ele possui dois estados: no primeiro espera o recebimento de informações sobre os próximos eventos de cada elemento do sistema. Ao ter recebido a informação de todos, passa ao estado de avaliação de novo tempo de simulação e informa a todos os elementos o novo tempo. A sincronização implementada neste modelo é a forma mais complexa que descrevemos no capítulo anterior, onde apenas os eventos externos são sincronizados de forma centralizada, gerando assim um maior assincronismo dentro do simulador.

3.3.1 Estruturas de Dados

3.3.1.1 Variáveis Principais

. **t_event** - Indica o próximo tempo de simulação do sistema, pode assumir valores que vão de zero ao tempo final de simulação. É uma variável do tipo real.

. **estr_proc[N_PROC_MAX].nx_time** - Guarda os próximos tempos dos eventos de cada um dos processos do simulador e pode assumir valores que vão de zero ao tempo final de simulação.

. **exc_sinc.b1, exc_sinc.b2** - Usados para sincronização entre processos, são reservados para uso do sistema.

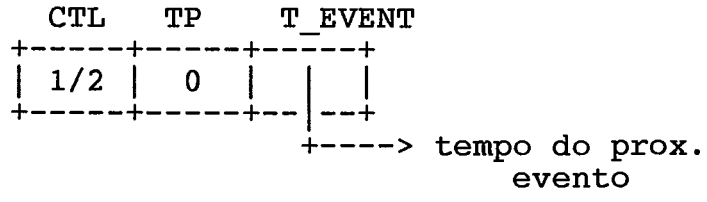
. **exc_info.b1, exc_info.b2** - Usados para sincronização entre processos, são reservados para uso do sistema.

. **ecb_tx, ecb_rx** - estruturas usadas para transmissão e recepção de mensagens respectivamente.

3.3.1.2 Mensagens Trocadas

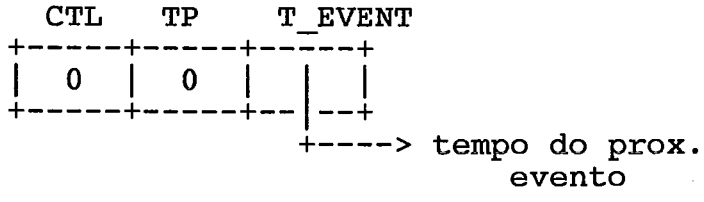
Recebidas

a) Novo tempo virtual do processo



Transmitidas

a) Novo tempo virtual do sistema



b) Informação sobre o tempo virtual

CTL	TP	FUNC
0	1	16

3.3.2 Algoritmo do Sincronizador

O sincronizador possui um processo único o qual está num loop infinito. Inicialmente o processo espera bloqueado (utiliza uma primitiva local de sincronização do AMPLIWARE chamada wait) o recebimento da informação sobre o tempo de simulação de todos os nós de simulação, porteriormente pede ao simulador de comunicação a informação sobre seu tempo de simulação. Espera bloqueado (utiliza a primitiva wait) o recebimento da resposta e passa a avaliar o próximo tempo de simulação como sendo o mínimo entre todos os tempos dos processos do sistema. A partir daí envia o

novo tempo de simulação para todos os processos do sistema, voltando a aguardar o recebimento de informação dos nós de simulação. O sincronizador possui ainda duas rotinas responsáveis pelo recebimento das mensagens com informação do tempo de simulação dos nós de simulação e do simulador de comunicação.

```
programa_principal              /* corpo principal do */
                                /*  sincronizador     */

    inicializa variáveis;

    põe estruturas em recepção;   /* recebem as informações */
                                  /*   de sincronismo       */

    FOREVER

       wait(exc_sinc);          /* espera recebimento de todos */
                                /*    os nós de simulação      */

       pede_temp_simul;         /* envia mensagem com msg.func = */
                                /* INFO_TIME para o simulador do */
                                /*   canal de comunicação        */

       wait(exc_info);          /* espera resposta do simulador do */
                                /*     canal de comunicação        */

       n_proc = 0;

       t_event = T_MAX;         /* faz tempo do próximo evento =   */
                                /* tempo máximo para cálculo do    */
                                /* tempo mínimo entre os processos */

       WHILE ( i < N_MAX_PROC )

   /* próximo tempo de simulação = mínimo de todos os processos */

            IF (estr_proc[i].nx_time < t_event)

              t_event = estr_proc[i].nx_time;

            i = i + 1;

       send_msg_time(t_event);  /* atualiza tempo de simulação */
                                /*    de todos os processos    */
```

```
recebe_mensagem_no     /* trata o recebimento de msg */
                       /*   dos nós de simulação     */

   /* retira a informação de tempo da mensagem que chegou */

    estr_proc[n_proc].nx_time = msg.t_event;

    n_proc = n_proc + 1;

    IF (n_proc = N_MAX_PROC - 1)

      signal (exc_sinc);      /* libera processo do */
                              /*   sincronizador    */


recebe_mensagem_simul    /* trata recebimento de msg */
                         /* do simulador do canal de */
                         /*     comunicação          */

    /* retira a informação de tempo da mensagem */

    estr_proc[n_proc].nx_time = msg.t_event;

    signal (exc_info);        /* libera processo do */
                              /*   sincronizador    */
```

3.4 Processo Simulador de Comunicação

É um processo instalado em "background" dentro do AMPLIWARE. Ele possui um corpo de processo onde são tratados os eventos contidos na fila de eventos do simulador de comunicação. O simulador de comunicação possui ainda elementos internos que são os transmissores e receptores dos nós. Tais elementos possuem uma série de estados, que mostraremos a seguir. Ainda associado a cada

nó existem tabelas contendo informações de endereço, tempo de propagação, e outras informações sobre o nó e suas conecções.

A cada novo tempo de simulação informado pelo sincronizador o simulador de comunicação avalia a existência ou não de algum evento a ser tratado. Caso haja tal evento ele será executado podendo programar um novo evento ou gerar um evento externo. Depois disso o simulador de comunicação avisa ao sincronizador seu próximo tempo de simulação, e passa a esperar a resposta do sincronizador.

O processo simulador de comunicação possui ainda uma rotina de tratamento de recebimento de mensagens tais como: pedido de informação sobre o estado do barramento, transmissão de uma mensagem pelo nó de simulação, etc. Existe ainda uma série de rotinas de apoio, podemos citar como exemplo: colocação de um evento na fila de eventos, cancelamento de um evento, transmissão de mensagens, etc.

As falhas do canal de comunicação podem ser simuladas neste processo. Os eventos básicos deste processo são o início e o final de um pacote. Para obtermos uma falha simplesmente temos que suprimir um destes eventos. Por exemplo, o evento que indica início de um pacote é suprimido no nó de destino gerando assim uma falha no canal de comunicação. O comando de supressão do evento pode ser determinístico ou gerado de forma aleatória através de um gerador randômico.

3.4.1 Estruturas de Dados

3.4.1.1 Variáveis Principais

. **cur_time** - Tempo de simulação atual, variável do tipo real.

. **bus_state[N_MAX_STATION]** - estado do barramento em cada nó.

. **tx_station_state[N_MAX_STATION]** - estado do transmissor de cada nó dentro da rede.

. **rx_station_state[N_MAX_STATION]** - estado do receptor de cada nó dentro da rede.

. **end_est[N_MAX_STATION]** - endereço do nó físico onde está o nó de simulação.

. **end_sock[N_MAX_STATION]** - identificação de processo dentro do AMPLIWARE, relativa ao nós de simulação.

. **n_est_conect[N_MAX_STATION]** - número de nós a que um determinado nó está conectado.

. **temp_prop[N_MAX_STATION][N_MAX_STATION - 1]** - tempo de propagação a partir de um determinado nó para todos os outros conectados a ele.

. **est_end[N_MAX_STATION][N_MAX_STATION - 1]** - endereço dos nós conectados a um determinado nó.

. **event_list** - aponta para o início da fila de eventos.

. **estr-evento[N_MAX_EVENT]** - estrutura descritoras do evento.

. **exc_sinc.b1, exc_sinc.b2** - Usados para sincronização entre processos, são reservados para uso do sistema.

. **ecb_tx, ecb_rx** - estruturas usadas para transmissão e recepção de mensagens respectivamente.

3.4.1.2 Estados e Eventos

Eventos

- PKT_FRONT - inicio de um pacote, em um determinado nó.
- PKT_BACK - final de um pacote, em um determinado nó.

Estados do Transmissor

. IDLE - sem nenhuma operação ativada.

. TRANSMITINDO - transmitindo pacote.

. WAIT_EOC - esperando pela liberação da barramento.

Estados do Receptor

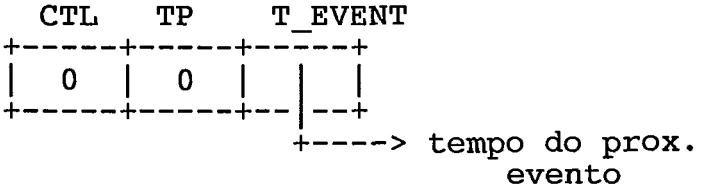
. IDLE - sem nenhuma operação ativada.

. RECEBENDO - recebendo pacote.

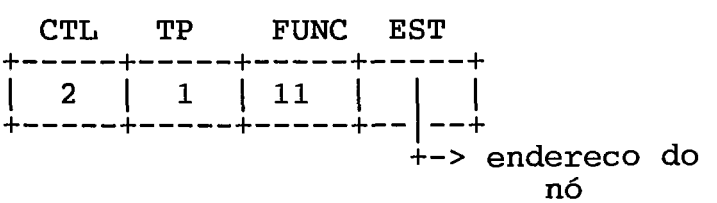
3.4.1.3 Mensagens Trocadas

Recebidas

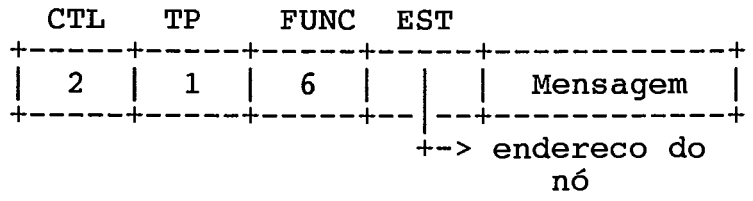
a) Novo tempo virtual do sistema



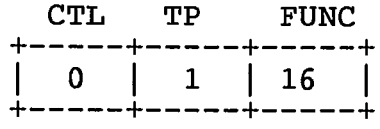
b) Informação sobre o estado do barramento



c) Transmissão de um pacote pelo nó de simulação

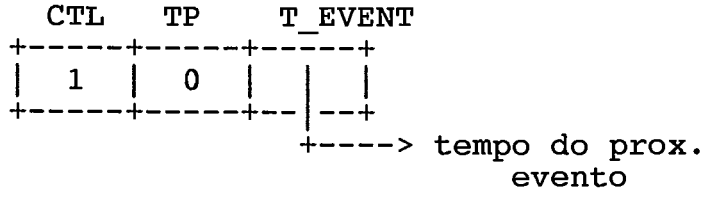


d) Informação sobre o tempo virtual

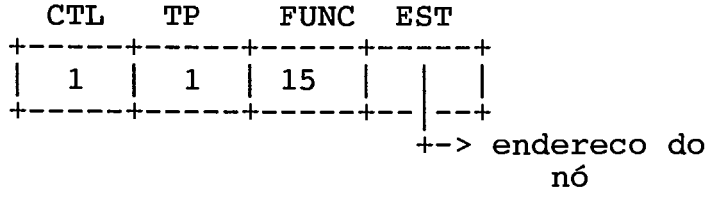


Transmitidas

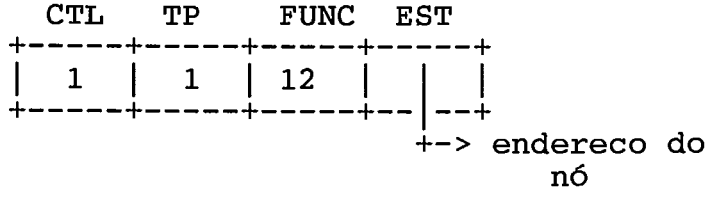
a) Novo tempo virtual do simulador



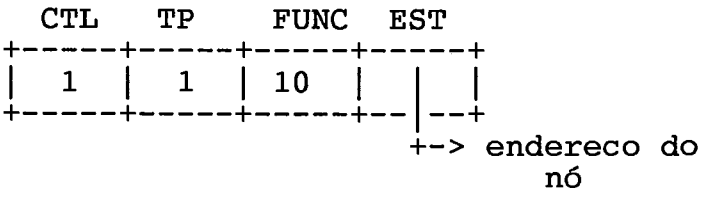
b) Informa erro de recepção



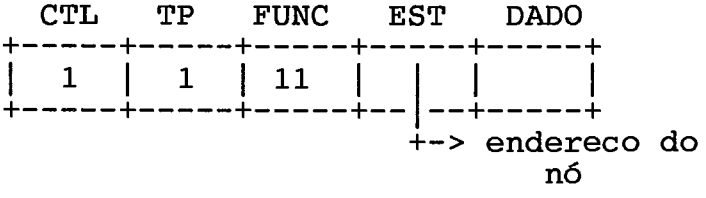
c) Informa ocorrência de colisão



d) Informa final de "wait" para WAIT_EOC

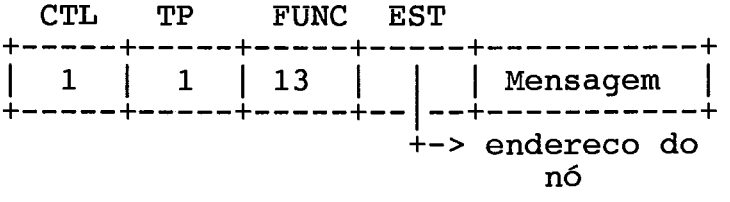


e) Informa estado do barramento



DADO - estado do barramento

f) Informa transmissão de mensagem para nó



3.4.2 Algoritmo do Simulador de Comunicação

Este processo possui um loop infinito que trata a fila de eventos do simulador de comunicação. Ele espera o recebimento da informação sobre o tempo de simulação do sistema. A partir daí avalia a fila de eventos para fazer o tratamento de todos os eventos até o tempo atual. Estas eventos são indexados pelas estações que estão sendo simuladas. Após o tratamento de todos os

eventos possíveis volta aguardar novo tempo de simulação. O sincronizador pergunta ao simulador de comunicação seu novo tempo de simulação, ele envia como resposta o tempo de simulação do próximo evento externo. No simulador de comunicações são considerados eventos externos apenas os inícios e finais de pacotes que geram mudanças no estado de recepção ou transmissão de um determinado nó. Neste caso temos um início de um pacote que ao chegar em um nó que está transmitindo gera uma colisão ou tem como destino este nó gerando uma recepção. O simulador de comunicação possui ainda uma rotina para tratamento das mensagens recebidas de outros processos.

Os eventos básicos dentro do simulador de comunicação são o início e o final de um pacote. Ao ser programado dentro do simulador de comunicação o início de uma transmissão, ele gera a programação de um evento PKT_FRONT em todos os nós conectados ao nó transmissor, de acordo com o tempo de propagação entre os nós. O simulador de comunicação gera também um evento PKT_BACK no nó que gerou a transmissão e em todos os nós conectados a ele, de acordo com o tamanho da mensagem e o tempo de propagação entre os nós.

```
programa_principal          /*    corpo principal do    */
                            /* simulador de comunicação */

    inicializa variáveis;

    põe estruturas em recepção;   /* recebem as informações */
                                  /*   de comunicação       */
```

```
FOREVER

   wait(exc_sinc);          /* espera recebimento de novo */
                            /*    tempo de simulação      */

   IF (event_list.time = cur_time)

     atualiza variáveis de eventos;  /* fila de eventos */
                                     /*  indexada pela  */
                                     /*     estação     */

     CASE (evento)

          PKT_FRONT:

          bus_state[estação] = BUSY;

          CASE (estado do receptor)

              IDLE:
              IF msg.dst_end = estação  /* recebe mensagem */
                atualiza estado (estado = RECEBENDO);
                inicia recepção;
                avisa ao nó correspondente;

              RECEBENDO:
              aborta recepção (estado = IDLE);
              avisa ao nó correspondente;

          IF (estado do transmissor = TRANSMITINDO)
            aborta transmissão (estado = IDLE);
            avisa ao nó correspondente;

          PKT_BACK:

          bus_state[estação] = IDLE;

          IF (estado do receptor = RECEBENDO)
            termina recepção (estado = IDLE);

          CASE (estado do transmissor)

              TRANSMITINDO:
              termina a transmissão (estado = IDLE);

              WAIT_EOC:
              avisa ao nó correspondente (estado = IDLE);


recebe_mensagem      /* trata o recebimento de msg */
                     /*    de outros processos     */
```

```
CASE (msg.tp)

    MSG_SINC:
    atualiza tempo de simulação;
    signal(exc_sinc);      /* libera programa principal */

    MSG_INFO:

    CASE (msg.func)

        RESP_REQ_TRANS:
        envia estado do barramento ao nó correspondente;
           IF (bus_state[estação] = BUSY)
              estado do transmissor = WAIT_EOC;

        INICIO_TX:
        inicia transmissão;
        programa eventos de PKT_FRONT;
        programa eventos de PKT_BACK;

        INFO_TIME:
        envia novo tempo ao sincronizador;
        /* tempo = tempo do menor evento externo) */
```

3.5 Nó de Simulação

Pode ser um processo instalado em "background" dentro do AMPLIWARE, ou não. Ele possui um corpo de processo onde são tratados os eventos contidos na fila de eventos do nó. O nó de simulação possui ainda elementos internos que são os transmissores e receptores dos nós e o nó propriamente dito. Tais elementos possuem uma série de estados, que mostraremos a seguir. Existem dois tipos de nós de simulação. O primeiro deles é o nó que simula

uma estação de trabalho. Ele possui a capacidade de gerar mensagens e se comunica com o segundo tipo que é o nó que simula o servidor. O nó do tipo servidor pode apenas responder às requisições feitas a ele.

A cada novo tempo de simulação informado pelo sincronizador o nó avalia a existência ou não de algum evento a ser tratado. Caso haja tal evento ele será executado podendo programar um novo evento ou gerar um evento externo. Depois disso o nó avisa ao sincronizador seu próximo tempo de simulação, e passa a esperar a resposta do sincronizador.

O nó de simulação possui ainda uma rotina de tratamento de recebimento de mensagens tais como: informação sobre o estado do barramento, transmissão de uma mensagem pelo nó de simulação, etc. Existe ainda uma série de rotinas de apoio, podemos citar como exemplo: colocação de um evento na fila de eventos, cancelamento de um evento, transmissão de mensagens, etc.

3.5.1 Estruturas de Dados

3.5.1.1 Variáveis Principais

. **cur_time** - Tempo de simulação atual, variável do tipo real.

. **cur_time_sis** - Tempo de simulação do sistema, variável do tipo real.

. **bus_state** - estado do barramento visto pelo nó.

. **tx_station_state** - estado do transmissor do nó.

. **rx_station_state** - estado do receptor do nó.

. **event_list** - aponta para o início da fila de eventos.

. **estr-evento[N_MAX_EVENT]** - estrutura descritora do evento.

. **put_tx, put_rx** - apontadores para fila de transmissão.

. **fila_tx[N_MAX_ENTRY]** - fila de transmissão.

. **exc_sinc.b1, exc_sinc.b2** - Usados para sincronização entre processos, são reservados para uso do sistema.

. **exc_info.b1, exc_info.b2** - Usados para sincronização entre processos, sào reservados para uso do sistema.

. **ecb_tx, ecb_rx** - estruturas usadas para transmissão e recepção de mensagens respectivamente.

3.5.1.2 Estados e Eventos

Eventos no Nó

- PK_USU - pacote gerado pelo usuário.
- PK_TRANSP - pacote a ser trans. chega ao nível de transporte.
- PK_REDE - pacote a ser transmitido chega ao nível de rede.
- RESP_USU - pacote recebido chega ao usuário.
- RESP_TRANSP - pacote recebido chega ao nível de transporte.
- RESP_REDE - pacote recebido chega ao nível de rede.
- INICIO_TX - é dado início a uma transmissão de pacote.

Eventos no Transmissor

- INICIO_TX - é dado início a uma transmissão de pacote.
- FIM_TX - a transmissão do pacote chega ao fim normalmente.
- FIM_WAIT - transmissor termina tempo de randomização.
- TRANSM_IDLE - transmissor volta ao estado inicial (IDLE)
- BUS_LIVRE - barramento livre para transmissão
- COLISAO_DET - colisão detectada durante a transmissão

Eventos no Receptor

- INICIO_RECEP - é dado início a uma recepção de pacote.
- FIM_RECEP - a recepção do pacote chega ao fim normalmente.

- ERRO_RECEP - recepção com erro.

Estados do Nó

. IDLE - sem nenhuma operação ativada.

. PROC_TX - processando transmissão.

. PROC_RX - processando recepção.

Estados do Transmissor

. IDLE - sem nenhuma operação ativada.

. TRANSMITINDO - transmitindo pacote.

. WAIT_EOC - esperando pela liberação da barramento.

. HOLD - esperando para realizar nova transmissão.

. WAIT_RANDOM - randomizando após colisão.

Estados do Receptor

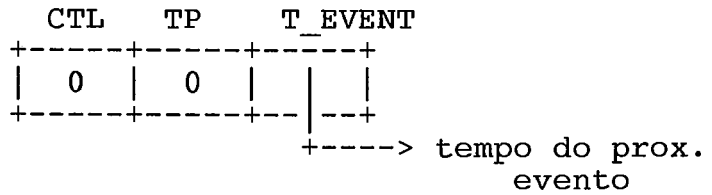
. IDLE - sem nenhuma operação ativada.

. RECEBENDO - recebendo pacote.

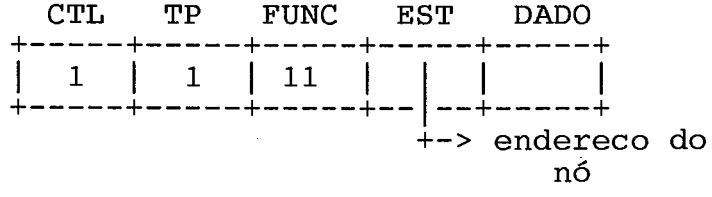
3.5.1.3 Mensagens Trocadas

Recebidas

a) Novo tempo virtual do sistema

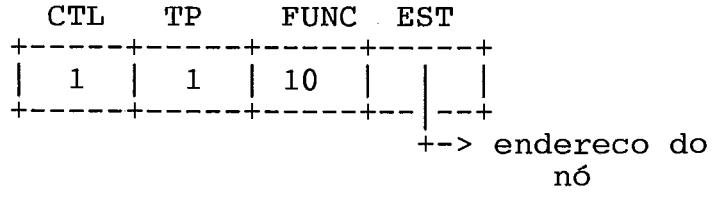


b) Informação sobre o estado do barramento

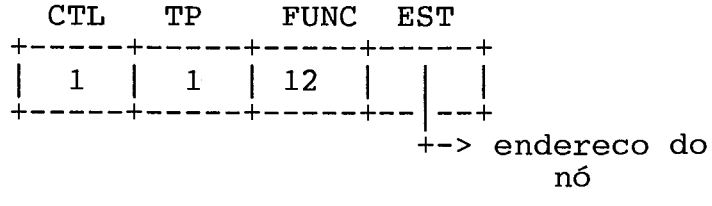


DADO - estado do barramento

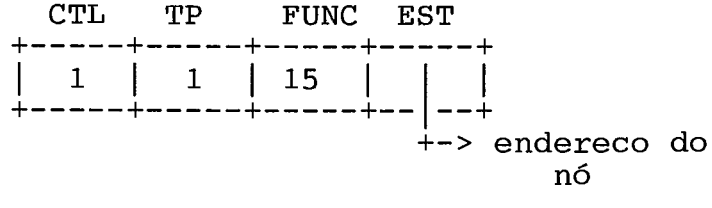
c) Informação final de "wait" para WAIT_EOC



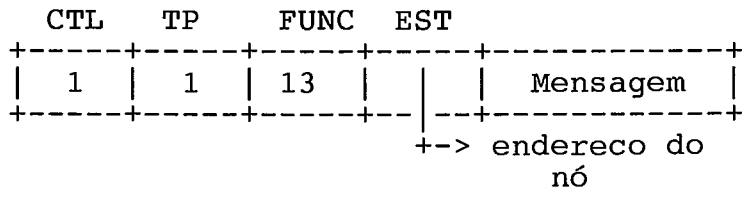
d) Informação sobre colisão detectada



e) Informação sobre erro de recepção

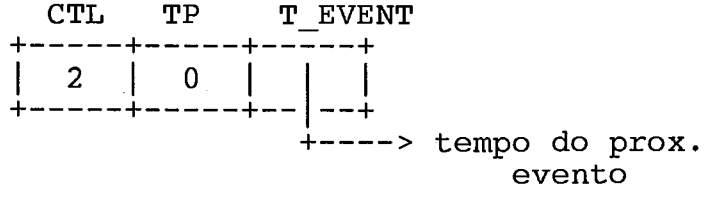


f) Recepção de um pacote para o nó de simulação

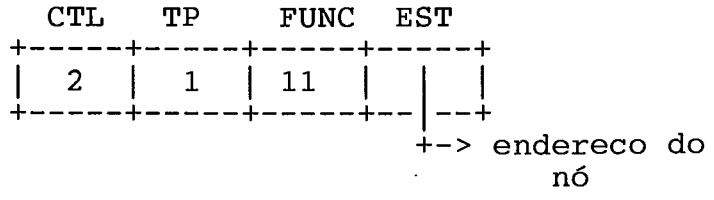


Transmitidas

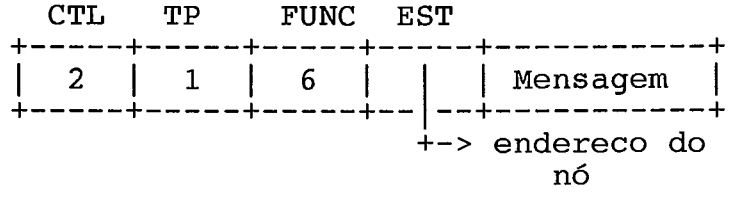
a) Novo tempo virtual do nó de simulação



b) Pede informação sobre estado do barramento



c) Informa transmissão de mensagem pelo nó



3.5.2 Algoritmo do Nó de Simulação

Este processo possui um loop infinito que trata a fila de eventos do nó de simulação. Ele espera o recebimento da informação sobre o tempo de simulação do sistema. A partir daí avalia a fila de eventos para fazer o tratamento de todos os eventos até o tempo atual. Após o tratamento de todos os eventos possíveis, envia ao sincronizador seu novo tempo de simulação e volta aguardar novo tempo de simulação. O nó de simulação possui ainda uma rotina para tratamento das mensagens recebidas de outros processos.

Neste algoritimo mostramos o caso particular de nossa aplicação, em o recebimento de uma mensagem pelo usuário gera uma nova mensagem a ser transmitida pelo nó. Assim o tratamento do evento RESP_USU é particular para nossa aplicação.

```
programa_principal              /* corpo principal do */
                                /*  nó de simulação   */

    inicializa variáveis;

    põe estruturas em recepção;   /* recebem as informações */
                                  /*    de comunicação      */

    FOREVER

       wait(exc_sinc);          /* espera recebimento de novo */
                                /*    tempo de simulação      */

       WHILE (event_list.time <= cur_time_sis)

         atualiza variáveis de eventos;

         CASE (evento)

              PK_USU:
              IF (state = IDLE)
                state = PROC_TX;
                gera mensagem;
                programa PK_TRANSP;
```

```
PK_TRANSP:
IF (state = PROC_TX)
  programa PK_REDE;

PKT_REDE:
IF (state = PROC_TX)
  programa INICIO_TX;

RESP_USU:
IF (state = PROC_RX)
  state = IDLE;
  programa PK_USU;

RESP_TRANS:
IF (state = PROC_RX)
  programa RESP_USU;

RESP_REDE:
  IF (state = IDLE)
    state = PROC_RX;
    programa RESP_TRANS;

INICIO_TX:
  IF (state = PROC_TX)
    state = IDLE;
    põe mensagem na fila_tx;
    IF (state_tx = IDLE)
      IF (bus_idle)
        state_tx = TRANSMITINDO;
        inicia transmissão;
      ELSE
        state_tx = WAIT_EOC;

FIM_TX:
  IF (state_tx = TRANSMITINDO)
    state_tx = HOLD;
    programa TRANSM_IDLE;

FIM_WAIT:
  IF (state_tx = WAIT_RANDOM)
    IF (bus_idle)
      state_tx = TRANSMITINDO;
      inicia_transmissão;
    ELSE
      state_tx = WAIT_EOC;

TRANSM_IDLE:
  IF (state_tx = HOLD)
    IF (fila_tx = vazia)
      state_tx = IDLE;
    ELSE
      IF (bus_idle)
```

```
                    state_tx = TRANSMITINDO;
                    inicia_transmissão;
                  ELSE
                    state_tx = WAIT_EOC;

            FIM_RECEP:
              IF (state_rx = RECEBENDO)
                state_rx = IDLE;
                IF (state = IDLE)
                  state = PROC_RX;
                  programa RESP_REDE;

    env_msg_sinc(novo tempo simulação);
/* tempo = tempo do menor evento externo */


recebe_mensagem      /* trata o recebimento de msg */
                     /*    de outros processos     */

    CASE (msg.tp)

        MSG_SINC:
        atualiza tempo de simulação;
        signal(exc_sinc);      /* libera programa principal */

        MSG_INFO:

        CASE (msg.func)

            RESP_REQ_TRANS:
            bus_state = msg.dado;
            libera transmissor;

            BUS_LIVRE:
            IF (state_tx = WAIT_EOC)
              state_tx = TRANSMITINDO;
              inicia transmissão;

            COLISAO_DET:
            IF (state_tx = TRANSMITINDO)
              cancela FIM_TX;
              state_tx = WAIT_RANDOM;
              programa FIM_WAIT;

            INICIO_RECEP:
            IF (state_rx = IDLE)
              state_rx = RECEBENDO;
              programa FIM_RECEP;

            ERRO_RECEP:
            IF (state_rx = RECEBENDO)
              cancela FIM_RECEP;
              state_rx = IDLE;
```

3.6 Codificação do Sistema

No anexo II apresentamos todo o código fonte dos três módulos do sistema e os procedimentos de geração dos programas. Cada módulo do sistema possui programas codificados em linguagem C e em linguagem Assembler.

3.7 Procedimentos de Depuração e Testes

O processos foram construídos de forma independentes e na ordem com que foram apresentados. Inicialmente para facilitar a depuração destes programas eles foram construídos para executar em "foreground" e posteriomente quando já estavam funcionando adequadamente passamos para seu estado definitivo. Os programas foram depurados usando máquinas diferentes facilitando assim nossos teste.

Um primeiro modelo com apenas o sincronizador, o simulador de comunicação e dois nós de simulação (um servidor e uma estação de trabalho) foi utilizado em nossos teste. A partir do funcionamento adequado deste modelo passamos às configurações para avaliação da

performance do simulador.

Apesar da alta complexidade do sistema sua depuração foi facilitada pela utilização de ferramentas tais como: CodeView e o Periscope. O CodeView permite que seja realizada a depuração do programa escrito na linguagem C diretamente sobre o código fonte. Já o Periscope permite a depuração de tarefas em "background" podendo fazer o sistema parar em qualquer ponto e depois voltando a executar normalmente.

No próximo capítulo apresentamos os modelos utilizados em nossas medidas, os resultados obtidos, e sua análise no sentido de verificar a validade da simulação distribuída com relação ao aumento de performance.

CAPÍTULO 4

RESULTADOS DA SIMULAÇÃO

4.0 Objetivos das Medidas e Modelos Utilizados

Após a implementação do simulador para o protocolo CSMA-CD realizamos medidas de performance a fim de identificar de forma pragmática sob que condições a simulação de forma distribuída apresenta resultados mais favoráveis do que a simulação concentrada.

Neste sentido propomos um conjunto de modelos que serão utilizados para realizarmos comparações entre a forma de operação distribuída e concentrada. Fica fácil notar que quanto maior for o paralelismo contido em nosso modelo, melhor será o resultado a favor da operação de forma distribuída. Sendo assim, modelos que permitam que seus elementos possam processar, de forma independente, um grande número de eventos devem gerar bons resultados.

Como já mostramos no capítulo anterior, um modelo servidor-estação

de trabalho pode gerar um grande número de eventos independentes tanto no servidor como na estação de trabalho. Com base neste tipo de modelo podemos obter uma grande variedade de combinações. Vamos a seguir verificar quais destas combinações podem nos ajudar em nosso objetivo.

4.0.1 Servidor-Estação de Trabalho

Vamos imaginar inicialmente um servidor e uma estação de trabalho. Para este tipo de modelo podemos ter dois tipos de comportamento em relação a geração de mensagens. No primeiro uma nova requisição só é gerada após a recepção da resposta da requisição anterior. Neste tipo de aplicação o paralelismo é muito pequeno pois em um determinado instante, apenas um dos elementos processa. A estação de trabalho faz a requisição ao servidor e espera a resposta para fazer nova requisição. Podemos prever para este tipo de aplicação uma resultado provavelmente não favorável à simulação de forma distribuída. No segundo a nova mensagem é gerada de forma independente, isto é, a geração de uma determinada requisição programa a geração de uma nova requisição, não ficando condionada ao recebimento da resposta. Já neste tipo de aplicação é esperado um aumento da performance da operação na forma distribuída. Para qualquer configuração baseada no modelo descrito anteriormente é possível utilizar uma das duas formas de geração de mensagens. Mostramos a seguir o modelo descrito.

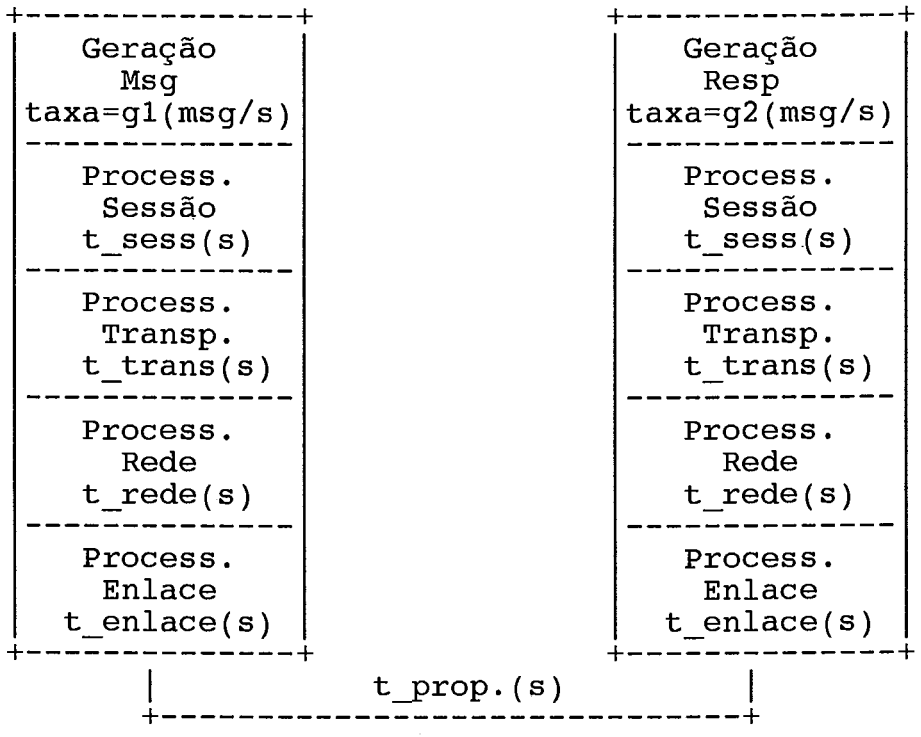


Figura 21. Modelo Servidor/Estação de Trabalho

Ainda utilizando este tipo de aplicação podemos gerar em cada nó de simulação números de eventos diferentes, isto é, de acordo com o número de estados de nossa simulação podemos ter um número maior ou menor de eventos em cada nó de simulação. Chamaremos estes eventos nos nós de eventos paralelos. O aumento do número destes eventos pode aumentar a performance da operação distribuída, caso o modelo permita paralelismo na execução dos nós. Devemos também variar a taxa de geração mensagens a fim de verificar a sensibilidade ou não do simulador de comunicação a esta variação.

Devemos esperar que a variação da taxa não influencie na performance do simulador de comunicação desde que o volume de mensagens geradas dentro da rede real não gere congestionamento na rede. Caso o número de mensagens seja suficiente para gerar um congestionamento a performance do simulador de comunicação será afetada pela performance da rede real. É necessário também determinarmos os tamanhos dos pacotes transferidos entre os nós para evitarmos que nosso simulador faça uma utilização muito elevada da rede real, causando também um congestionamento na rede real. Apresentamos a seguir como representaremos este modelo descrito.

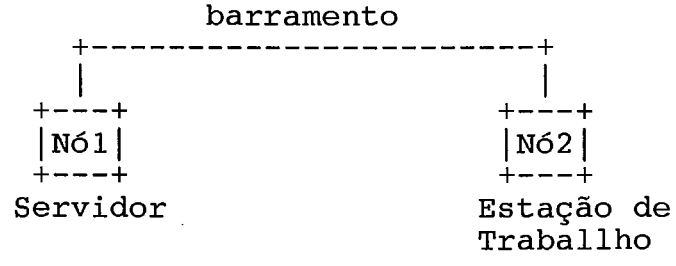


Figura 22. Representação do Modelo

Uma variação do modelo apresentado anteriormente, é a colocação de dois servidores e duas estações de trabalho em uma mesma rede. Neste tipo de configuração é possível notar que o paralelismo no sistema tende a aumentar, e desta forma esperamos obter um resultado melhor que no caso anterior. Aqui cada par servidor e estação de trabalho apresenta estrutura semelhante ao caso anterior. Mostramos a seguir a representação deste modelo.

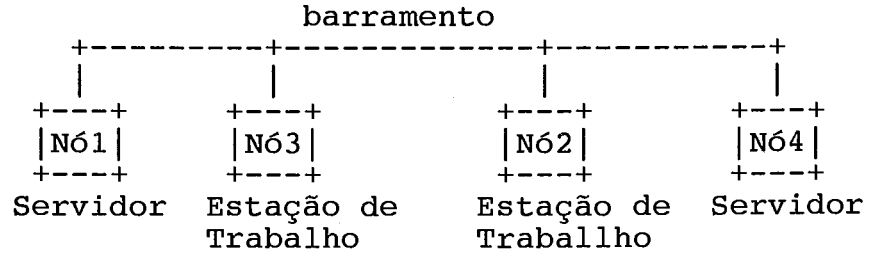


Figura 23. Representação do Modelo

Neste exemplo o Nó1 e o Nó2 formam um par e só se comunição entre si, e o outro par e formado pelo Nó3 e o Nó4. Os dois pares de nós são praticamente independentes, exceto pelo fato de utilizarem o mesmo meio físico. Desta forma o nível de paralelismo entre os dois pares deve ser muito elevado, fazendo com que a operação distribuída neste caso apresente resultado melhor que a do caso anterior. Neste caso devemos esperar ainda que aumentando o número de estados e, consequentemente, o número de eventos paralelos, tenhamos uma melhora na performance distribuída.

Ainda utilizando o mesmo modelo básico temos a configuração com três conjuntos servidor-estação de trabalho numa mesma rede. Aqui, comparando com os modelos anteriores deveremos obter o melhor resultado a favor da operação distribuída.
Cada par servidor-estação de trabalho opera de forma independente, apenas concorrendo por um único meio físico. Temos a seguir a representação deste modelo.

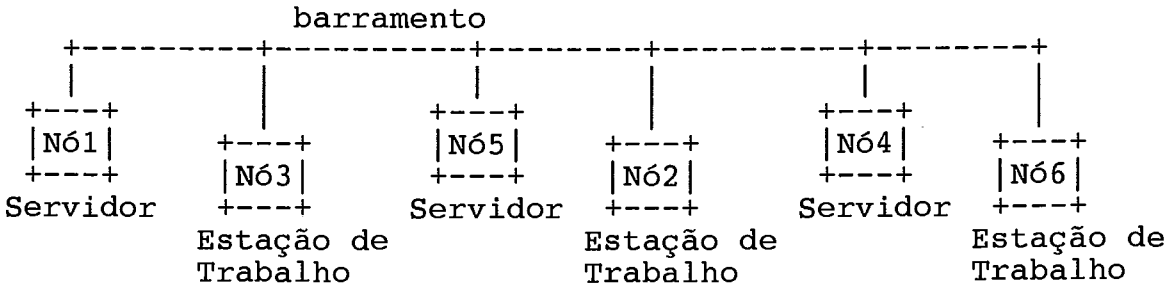


Figura 24. Representação do Modelo

Fica fácil notar que a utilização da geração de mensagens dependente do recebimento da resposta da mensagem anterior, gera uma "speedup" ideal igual a N/2, onde N é igual ao número total de estações de trabalho mais servidores. Isto ocorre porque em cada par estação de trabalho-servidor, em um determinado instante de tempo, apenas um deles está processando. Para o caso de geração independente de mensagens, o "speedup" ideal gerado é igual a N, onde N é igual ao número total de estações de trabalho mais servidores. Neste caso estação de trabalho e servidor processam ao mesmo tempo.

4.0.2 Multi-Servidor/Estação de Trabalho

Utilizamos um modelo um que o servidor pode atender simultaneamente a mais de uma estação de trabalho. O servidor opera praticamente com uma capacidade N vezes maior do que o servidor do caso anterior, onde N é o número de estações de

trabalho que se comunição com o servidor. A capacidade do servidor é menor do que N porque o transmissor/receptor é único é todas as mensagens são enfileiradas para transmissão/recepção, reduzindo assim sua performance. Para este tipo de aplicação utilizaremos o modelo a seguir.

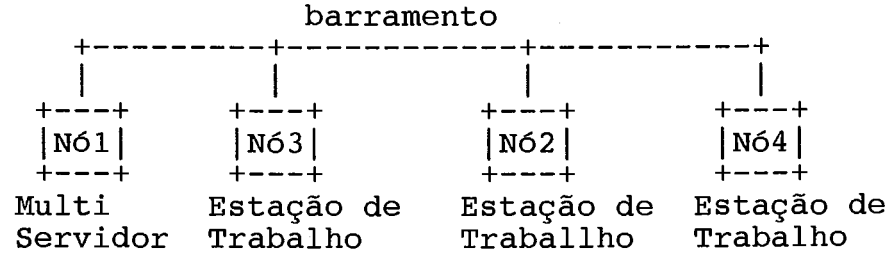


Figura 25. Representação do Modelo

Como vemos neste caso o servidor está servindo a três estações de trabalho simultaneamente. Para este tipo de aplicação devemos esperar uma boa performance na operação sob a forma distribuída, já que existe um alto grau de paralelismo no modelo. Caso comparemos este modelo com o modelo onde temos três pares de servidor-estação de trabalho, devemos esperar um resultado menos favorável para o caso multi-servidor. No caso multi-servidor o paralelismo será menor que o caso dos três pares, já que no multi-servidor as mensagens dentro do servidor vão concorrer por um único par transmissor/receptor. Também é válido notar que o aumento do número de estados e eventos paralelos contribui para um melhor desempenho da simulação distribuída, pois aumenta o número de eventos que podem ser tratados em paralelo.

4.0.3 Definição das Medidas

Faremos nossas medidas para os diversos modelos descritos anteriormente da seguinte forma. Nossa simulação será realizada para um determinado número de mensagens geradas nas estações de trabalho. Para cada configuração realizaremos dez medidas diferentes para obtermos médias com um intervalos de confiança de 95%. Variaremos ainda a taxa de geração de mensagens nas estações de trabalho e nos servidores e finalmente o número de eventos paralelos.

Devemos ainda realizar uma avaliação sobre a ocupação máxima da rede real que estamos utilizando. Esta avaliação considera a taxa de chegada independente, a fim de saber se estamos ou não próximos de uma situação de congestionamento. Assim devemos considerar a condição de maior carga em nossa avaliação. Consideremos que os nós geram as taxas definidas (g1 e g2), e vamos utilizar a configuração onde temos seis nós. Vamos então considerar a figura 24 e temos a seguinte equação:

Carga máxima na rede = Util_n12 + Util_n34 + Util_n56 (bits/s)

Util_n12 = 2.g1(msg/s).tamanho mensagens(bytes/msg).8(bits/byte)
Util_n34 = 2.g1(msg/s).tamanho mensagens(bytes/msg).8(bits/byte)
Util_n56 = 2.g1(msg/s).tamanho mensagens(bytes/msg).8(bits/byte)

Substituindo valores obtemos uma carga máxima na rede de aproximadamente 0.74 Mbits/seg, que é bem inferior a taxa de nossa rede real utilizada (2Mbits/seg). Então vemos que com estes tamanhos de pacotes e estas taxas de geração de mensagens não teremos problemas quanto à taxa de utilização da rede real.

4.1 Ambiente Real de Teste

Para a realização das medidas utilizamos os modelos descritos na sessão anterior. No caso da operação sob forma distribuída alocamos o processo sincronizador e o processo simulador de comunicação em um micro-processador e para cada nó de simulação utilizamos mais um micro-processador. Na operação concentrada todos os processos foram colocados no micro-processador utilizado anteriomente para o sicronizador e o simulador de comunicação. Para obtenção de todas as figurações foi necessário o uso de até sete micro-processadores, cujas características são descritas a seguir.

3 - Micro-processador compatível com IBM-XT, cpu 8088, 8MHz de clock, 704K de memória, Monydata modelo **NYDA 100.**

1 - Micro-processador compatível com IBM-XT, cpu 8088, 8MHz de

clock, 704k de memória, Monydata modelo **NYDA 200 Plus.**

1 - Micro-processador compatível com IBM-XT, cpu 8088, 8MHz de clock, 704k de memória, Cobra **X-PC.**

1 - Micro-processador compatível com IBM-XT, cpu 8088, 8MHz de clock, 640k de memória, Microtec modelo **XT-2002 Master.**

1 - Micro-processador compatível com IBM-XT, cpu 8088, 8MHz de clock, 256k de memória, Itautec modelo **I7000-PCXT.**

Todos os micro-processadores utilizavam placas CSMA-CD, 2Mbits/s, Amplus modelo Amplicard AC106.

Vamos agora precisar a configuração usada em cada teste realizado.

1) Um par servidor-estação de trabalho

a) distribuído:

. sincronizador + simulador de comunicação - NYDA 100

. nó de simulação servidor(Nó1) - NYDA 100

. nó de simulação estação de trabalho(Nó2) - NYDA 100

b) concentrado:

. todos os processos - NYDA 100

2) Dois pares servidor-estação de trabalho

a) distribuído:

. sincronizador + simulador de comunicação - NYDA 100
. nó de simulação servidor(Nó1) - NYDA 100
. nó de simulação estação de trabalho(Nó2) - NYDA 100
. nó de simulação servidor(Nó4) - NYDA 200
. nó de simulação estação de trabalho(Nó3) - X-PC

b) concentrado:

. todos os processos - NYDA 100

3) Três pares servidor-estação de trabalho

a) distribuído:

. sincronizador + simulador de comunicação - NYDA 100
. nó de simulação servidor(Nó1) - NYDA 100
. nó de simulação estação de trabalho(Nó2) - NYDA 100
. nó de simulação servidor(Nó4) - NYDA 200
. nó de simulação estação de trabalho(Nó3) - X-PC
. nó de simulação servidor(Nó5) - XT2002 Master
. nó de simulação estação de trabalho(Nó6) - I7000-PCXT

b) concentrado:

. todos os processos - NYDA 100

4) Um multi-servidor e três estações de trabalho

a) distribuído:

. sincronizador + simulador de comunicação - NYDA 100

. nó de simulação servidor(Nó1) - NYDA 100

. nó de simulação estação de trabalho(Nó2) - NYDA 100

. nó de simulação estação de trabalho(Nó3) - X-PC

. nó de simulação estação de trabalho(Nó4) - I7000-PCXT

b) concentrado:

. todos os processos - NYDA 100

4.2 Resultados dos Testes Realizados

4.2.1 Geração de Mensagens Dependente

Nossos testes tiveram início com a configuração mais simples que contém apenas um par estação de trabalho-servidor, e é mostrada na figura 22. Consideramos em nossa simulação processadores com o tempo médio de execução de uma instruções igual a 0.5 us, assim em 100 us teremos em média 200 instruções. Para esta configuração

foram utilizados os valores apresentados a seguir para as variáveis mostradas na figura 21.

. t_sess = 500 us (tempo médio na camada de sessão)
. t_trans. = 300 us (tempo médio na camada de transporte)
. t_rede = 120 us (tempo médio na camada de rede)
. t_enlace = 150 us (tempo médio na camada de enlace)
. t_prop.
  Nó 1 <-> Nó 2 = 5 us (tempo de prop. - cabo coaxial = 1.0 Km)
. geração de mensagens dependente
  do recebimento da resposta
. taxa de geração de mensagem
  g1 = 140 msg/s
  g2 = 140 msg/s
. tamanho dos pacotes
  requisição para o servidor - 10 bytes
  resposta do servidor - 100 bytes
. número de eventos paralelos na simulação, variável
  (2,7,15,100,1000)

Nesta configuração fizemos testes com a variação dos valores da taxa de geração de mensagens na estação de trabalho e no servidor, os valores de g1 e g2 variaram entre algumas mensagens por segundo simulando uma aplicação via console até milhares de mensagens por segundo simulando uma aplicação de transferência de dados. Dentro desta faixa de variação não percebemos nenhuma alteração

significativa no tempo de execução da simulação, assim concluimos que em nossa modelagem a variaçào da taxa de geração de mensagens não afeta o tempo de simulação, para o caso em que operamos com geração de mensagens dependente. Assumimos então um valor de 140 mensagens por segundo para as taxas g1 e g2.

Para os valores apresentados acima temos os seguintes resultados:

Eventos paralelos	Simul. concentrada (tempo médio)	Simul. distribuída (tempo médio)	Relação (Con./Dis.)
2	2.75 s (<.5%)	3.73 s (1.0%)	0.73
7	24.94 s (<.5%)	25.93 s (1.0%)	0.96
15	47.10 s (<.5%)	48.02 s (1.5%)	0.98
100	4:02.94 s (<.5%)	4:03.12 s (2.0%)	0.99
1000	39:54.23 s (1.0%)	39:54.02 s (4.5%)	1.00

A partir da análise destes resultados podemos concluir que ao aumentarmos o número de eventos paralelos dentro do simulador, os resultados da operação sob forma distribuída tendem a alcançar seu valor máximo ideal.

Proseguindo nosso testes passamos a configuração mostrada na figura 23, que possui dois pares estação de trabalho-servidor. Os valores das variáveis apresentadas na figura 21 são mostrados a seguir, sendo que as variáveis internas dos nós de simulação são idênticas para cada par estação de trabalho-servidor.

. t_sess = 500 us (tempo médio na camada de sessão)

. t_trans. = 300 us (tempo médio na camada de transporte)

. t_rede = 120 us (tempo médio na camada de rede)

. t_enlace = 150 us (tempo médio na camada de enlace)

. t_prop.

Nó 1 <-> Nó 2 = 5 us (tempo de prop. - cabo coaxial = 1.0 Km)

Nó 1 <-> Nó 3 = 3 us (tempo de prop. - cabo coaxial = 0.6 Km)

Nó 1 <-> Nó 4 = 9 us (tempo de prop. - cabo coaxial = 1.8 Km)

Nó 2 <-> Nó 3 = 2 us (tempo de prop. - cabo coaxial = 0.4 Km)

Nó 2 <-> Nó 4 = 4 us (tempo de prop. - cabo coaxial = 0.8 Km)

Nó 3 <-> Nó 4 = 6 us (tempo de prop. - cabo coaxial = 1.2 Km)

. geração de mensagens dependente
do recebimento da resposta

. taxa de geração de mensagem

g1 = 140 msg/s

g2 = 140 msg/s

. tamanho dos pacotes

requisição para o servidor - 10 bytes

resposta do servidor - 100 bytes

. número de eventos paralelos na simulação, variável
(2,7,15,100)

Os valores obtidos nesta configuração são os mostrados a seguir:

Eventos paralelos	Simul. concentrada (tempo médio)	Simul. distribuída (tempo médio)	Relação (Con./Dis.)
2	8.47 s (<.5%)	11.58 s (1.0%)	0.74
7	1:06.98 s (<.5%)	1:07.66 s (1.5%)	0.99
15	1:54.92 s (<.5%)	1:52.63 s (1.5%)	1.02
100	8:07.90 s (<.5%)	5:34.02 s (2.0%)	1.46

Também para este caso podemos concluir que ao aumentarmos o número de eventos paralelos dentro do simulador, os resultados da operação sob forma distribuída tendem a alcançar seu valor máximo ideal. Neste caso no entanto não foi realizada a medida do tempo para 1000 eventos paralelos, pois seu tempo era muito grande impossibilitando sua realização.

Ainda para esse caso fizemos as medidas relativa a valores da simulação da rede modelo de teste, estes valores e suas definições são apresentados a seguir:

Atraso fim a fim - mede o tempo médio entre a geração de uma requisição e o recebimento da resposta. (variável medida na estação de trabalho)

Tempo no nó - mede o tempo médio entre a geração de uma mensagem até o final de sua transmissão bem sucedida. (variável medida em todos os nós)

Tempo no transmissor - mede o tempo médio entre o início e o final

bem sucedido de uma transmissão. (variável medida em todos os nós)

Taxa de mensagens - mede a taxa média de mensagens geradas em cada nó (variável medida em todos os nós)

Variável	Nó1(Serv)	Nó2(Est)	Nó3(Est)	Nó4(Serv)	Unidade
Atraso fim a fim	-	5.96(<.5)	6.02(<.5)	-	ms
Tempo no nó	1.81(<.5)	1.79(<.5)	1.82(<.5)	1.84(<.5)	ms
Tempo no transmissor	379(<.5)	379(<.5)	380(<.5)	381(<.5)	us
Taxa de mensagens	140(<.5)	139(<.5)	138(<.5)	137(<.5)	msg/s

Podemos notar observando as medidas realizadas a precisão de nosso simulador, pois temos um pequeno intervalo de confiança e os valores obtidos estão coerentes com os valores do modelo que foi simulado. Por exemplo, temos a taxa de mensagens em cada nó de simulação que programamos para um valor de 140 mensagens por segundo e obtivemos valores na simulação muito próximos do valor programado. Com relação ao tempo no trasmissor podemos notar que para os quatro nós estes tempos são muito próximos e se compararmos o tempo da transmissão de uma mensagem (tempo no nó) com os tempos definidos para cada nível veremos que o tempo no transmissor é muito próximo do tempo mínimo possível para uma transmissão caracterizando assim a baixa taxa de colisão na rede em simulação.

Passamos então a configuração mostrada na figura 24, que possui três pares estação de trabalho-servidor. Os valores das variáveis apresentadas na figura 21 são mostrados a seguir, sendo que as variáveis internas dos nós de simulação são identicas para cada par estação de trabalho-servidor.

. t_sess = 500 us (tempo médio na camada de sessão)
. t_trans. = 300 us (tempo médio na camada de transporte)
. t_rede = 120 us (tempo médio na camada de rede)
. t_enlace = 150 us (tempo médio na camada de enlace)
. t_prop.

Nó 1 <-> Nó 2 = 5 us (tempo de prop. - cabo coaxial 1.0 Km)
Nó 1 <-> Nó 3 = 3 us (tempo de prop. - cabo coaxial 0.6 Km)
Nó 1 <-> Nó 4 = 9 us (tempo de prop. - cabo coaxial 1.8 Km)
Nó 1 <-> Nó 5 = 4 us (tempo de prop. - cabo coaxial 0.8 Km)
Nó 1 <-> Nó 6 = 10 us (tempo de prop. - cabo coaxial 2.0 Km)
Nó 2 <-> Nó 3 = 2 us (tempo de prop. - cabo coaxial 0.4 Km)
Nó 2 <-> Nó 4 = 4 us (tempo de prop. - cabo coaxial 0.8 Km)
Nó 2 <-> Nó 5 = 1 us (tempo de prop. - cabo coaxial 0.2 Km)
Nó 2 <-> Nó 6 = 5 us (tempo de prop. - cabo coaxial 1.0 Km)
Nó 3 <-> Nó 4 = 6 us (tempo de prop. - cabo coaxial 1.2 Km)
Nó 3 <-> Nó 5 = 1 us (tempo de prop. - cabo coaxial 0.2 Km)
Nó 3 <-> Nó 6 = 7 us (tempo de prop. - cabo coaxial 1.4 Km)
Nó 4 <-> Nó 5 = 5 us (tempo de prop. - cabo coaxial 1.0 Km)
Nó 4 <-> Nó 6 = 1 us (tempo de prop. - cabo coaxial 0.2 Km)
Nó 5 <-> Nó 6 = 6 us (tempo de prop. - cabo coaxial 1.2 Km)

. geração de mensagens dependente
  do recebimento da resposta
. taxa de geração de mensagem
  g1 = 140 msg/s
  g2 = 140 msg/s
. tamanho dos pacotes
  requisição para o servidor - 10 bytes
  resposta do servidor - 100 bytes
. número de eventos paralelos na simulação, variável
  (2,7,15,100)

Os valores obtidos nesta configuração são os mostrados a seguir:

Eventos paralelos	Simul. concentrada (tempo médio)	Simul. distribuída (tempo médio)	Relação (Con./Dis.)
2	15.82 s (<.5%)	20.12 s (1.5%)	0.78
7	1:22.33 s (<.5%)	1:18.38 s (1.5%)	1.05
15	2:33.13 s (<.5%)	2:20.25 s (2.0%)	1.09
100	13:46.32 s (<.5%)	6:59.45 s (2.0%)	1.97

Também para este caso podemos concluir que ao aumentarmos o número de eventos paralelos dentro do simulador, os resultados da operação sob forma distribuída tendem a alcançar seu valor máximo ideal. Vemos também que a partir de sete eventos a tempo para o caso distribuído já é menor que o caso concentrado, isto mostra o maior paralelismo desta configuração em relação as anteriores.

Neste caso no entanto também não foi realizada a medida do tempo para 1000 eventos paralelos, pois seu tempo era muito grande impossibilitando sua realização.

Seguindo em nosso testes utilizamos a configuração multi-servidor que é apresentada na figura 25 e possui três estações de trabalho e um servidor. Os valores das variáveis apresentadas na figura 21 são mostrados a seguir, sendo que as variáveis internas dos nós de simulação são idênticas para cada estação de trabalho e para o servidor a taxa de geração de mensagens é três vezes maior que as estações de trabalho.

. t_sess = 500 us (tempo médio na camada de sessão)
.. t_trans. = 300 us (tempo médio na camada de transporte)
.. t_rede = 120 us (tempo médio na camada de rede)
.. t_enlace = 150 us (tempo médio na camada de enlace)
. t_prop.
  Nó 1 <-> Nó 2 = 5 us (tempo de prop. - cabo coaxial 1.0 Km)
  Nó 1 <-> Nó 3 = 3 us (tempo de prop. - cabo coaxial 0.6 Km)
  Nó 1 <-> Nó 4 = 9 us (tempo de prop. - cabo coaxial 1.8 Km)
  Nó 2 <-> Nó 3 = 2 us (tempo de prop. - cabo coaxial 0.4 Km)
  Nó 2 <-> Nó 4 = 4 us (tempo de prop. - cabo coaxial 0.8 Km)
  Nó 3 <-> Nó 4 = 6 us (tempo de prop. - cabo coaxial 1.2 Km)
. geração de mensagens dependente
  do recebimento da resposta
. taxa de geração de mensagem
  g1 = 140 msg/s

g2 = 420 msg/s

. tamanho dos pacotes

requisição para o servidor - 10 bytes

resposta do servidor - 100 bytes

. número de eventos paralelos na simulação, variável (7,15)

Os valores obtidos nesta configuração são os mostrados a seguir:

Eventos paralelos	Simul. concentrada (tempo médio)	Simul. distribuída (tempo médio)	Relação (Con./Dis.)
7	1:17.32 s (<.5%)	1:16.19 s (1.5%)	1.02
15	2:24.46 s (<.5%)	2:15.01 s (2.0%)	1.07

Também aqui ao aumentarmos o número de eventos paralelos dentro do simulador, os resultados da operação sob forma distribuída tendem a melhorar em relação ao caso concentrado. Vemos também que se compararmos este caso ao caso onde temos três pares estação de trabalho-servidor, podemos notar que o paralelismo contido neste configuração é menor do que na de três pares independentes. Isto porque para um mesmo número de eventos temos valores mais favoráveis para a configuração com os três pares independentes.

4.2.2 Geração de Mensagens Independente

Finalmente realizamos a seguinte teste. Utilizamos as configurações de um par, dois pares e três pares de estação de trabalho-servidor exatamente como havíamos descrito anteriomente alterando apenas a forma de geração de mensagens, isto é utilizamos agora a geração de mensagens independente do recebimento da resposta da mensagem anterior. Com isso esperamos obter resultados muito favoráveis à operação sob forma distribuída. Para este caso realizamos medidas de tempo para 100 eventos paralelos nos três casos e para 1000 eventos paralelos no casos de um par estação de trabalho-servidor. Vemos a seguir estes resultados.

100 Eventos paralelos:

N. de nós	Simul. concentrada (tempo médio)	Simul. distribuída (tempo médio)	Relação (Con./Dis.)
2	5:43.32 s (<.5%)	3:19.60 s (2.0%)	1.72
4	11:20.18 s (<.5%)	3:52.94 s (2.0%)	2.92
6	17:12.48 s (<.5%)	4:05.25 s (2.0%)	4.21

1000 Eventos paralelos:

N. de nós	Simul. concentrada (tempo médio)	Simul. distribuída (tempo médio)	Relação (Con./Dis.)
2	48:38.85 s (1.0%)	24:34.16 s (4.5%)	1.98

Neste caso podemos notar o aumento do "speedup" com o aumento do número de eventos. Vemos que para um número muito grande de

eventos temos praticamente o valor ideal de "seedup". Aqui é onde foi conseguido a maior performance para o caso distribuído em relação ao caso concentrado.

4.2.3 Comparação dos Resultados

Para melhor visualização de nossos resultados apresentados a seguir um gráfico contendo os resultados de nossas medidas.

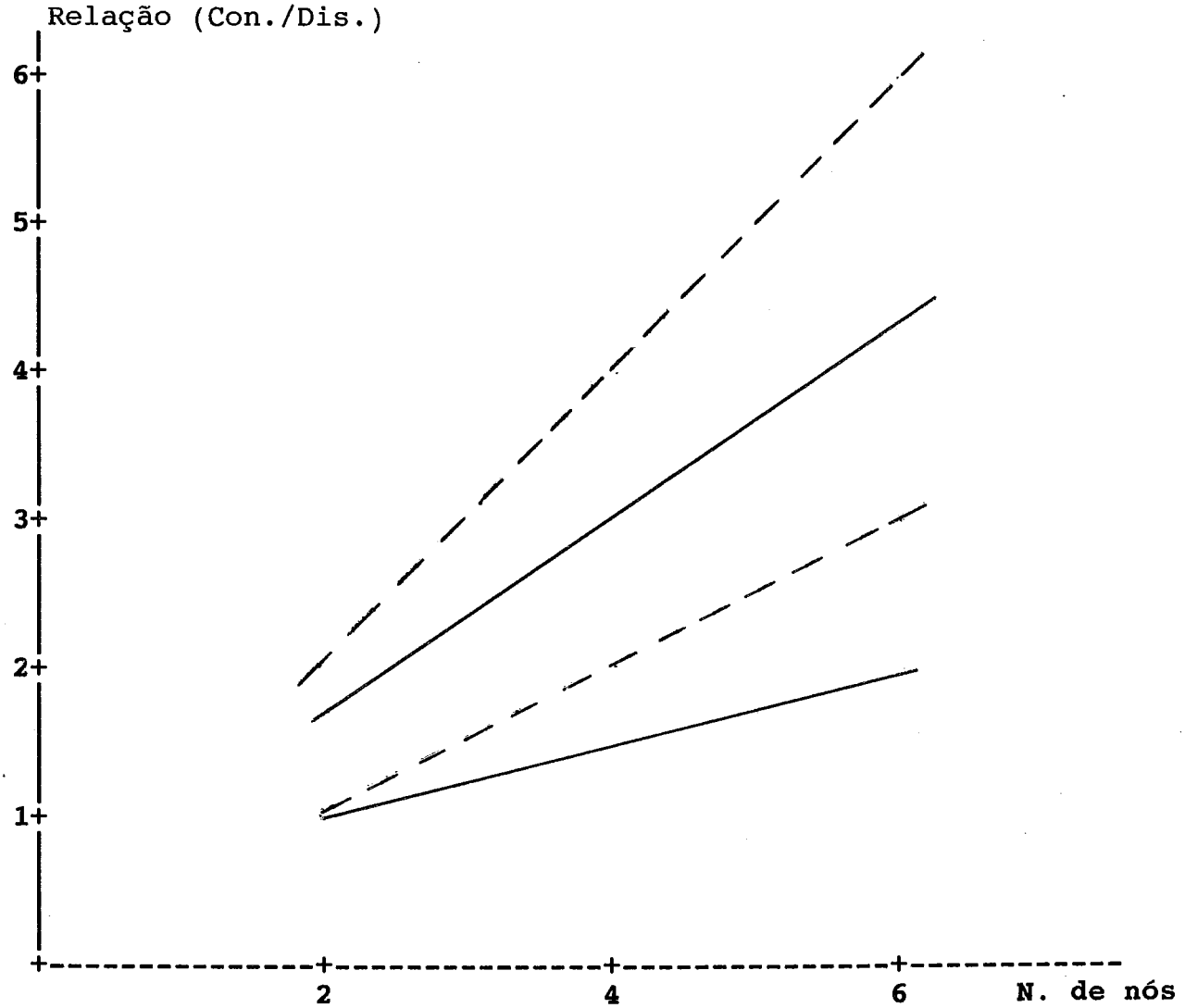


Figura 26. Representação gráfica

Em nosso gráfico as linhas tracejadas representam os resultados ideais para os modelos estaçào de trabalho-servidor nos casos de geração de mensagem dependente e independente da resposta da mensagem anterior. Fica fácil notar que o aumento de estados e eventos é um elemento determinante do aumento da performance do caso distribuído. Isto porque o aumento do número de estados e eventos aumenta o processamento em cada nó de simulação e como cada nó de simulação processa de forma distribuída a performance aumenta. No caso da geração de mensagens independente vemos que ele nos dá o maior "speedup", como já haviamos comentado, nesse caso todos os elementos do sistema estão processando em um determinado instante favorecendo a operação sob forma distribuída.

Podemos então concluir a partir de nossos teste e análise de seus resultados que o "speedup" ideal só será alcançado nos casos em que o paralelismo for realmente efetivo. Para isso os elementos que compõe a simulação distribuída devem estar sempre processando, isto é, para se obter "speedup" máximo não pode haver nenhum instante em que haja elementos ociosos dentro da simulação. Assim o modelo e o simulador (implementação real) tem que propiciar paralelismo caso contrário o "speedup" teórico nunca será alcançado.

CAPÍTULO 5

CONCLUSÕES

5.0 Introdução

Neste trabalho fizemos um estudo sobre técnicas de simulação distribuída que serviu como base para realizarmos a definição e o desenvolvimento de nosso simulador para protocolos de comunicação. Após a definição do simulador de comunicação que opera de forma distribuída realizamos a implementação de um protótipo simplificado para fazermos testes em relação a performance quando comparado a um simulador concentrado. Assim foram realizadas medidas para diversos modelos e seus resultados analizados.

5.1 Limitações do Trabalho

O nosso modelo implementado possui uma série de limitações se

comparado com o modelo de simulador de comunicação proposto inicialmente. Isso se deve ao fato de que primeiramente o modelo implementado se propõe a mostrar a viabilidade da simulação distribuída em relação a simulação concentrada. Assim com este objetivo ele foi desenvolvido apenas para um tipo de protocolo de enlace (CSMA-CD) e somente a aplicação estação de trabalho-servidor foi implementada.

Uma série de mecanismos descritos e de funcionalidades apresentadas por nós na definição do simulador de comunicações não foram implementadas neste modelo. Podemos citar como exemplo: o mecanismo de execução de código simulado mais código real dentro do simulador, a simulação de falhas na simulação, o arquivo que contém as estatísticas das variáveis da simulação e os mecanismos necessários para a simulação de "token".

Algumas destas limitações foram assumidas por nós a fim de simplificar a implementação do modelo de simulador de comunicação, porque queríamos alcançar o objetivo principal de nosso trabalho. Mas por outro lado existiram limitações impostas pelo sistema de suporte a construção de aplicaçoes distribuídas. Podemos citar o mecanismo de execução de código simulado mais código real como sendo uma delas.

Foi necessário reduzir drasticamente a portabilidade da implementação de nosso simulador de comunicação, pois tivemos que construir seu código fonte em linguagem de programação C e

linguagem de programação Assembler. Tal limitação nos foi imposta pelo sistema de suporte a construção de aplicações distribuídas. Originalmente pensamos em utilizar apenas linguagem de programação C.

5.2 Dificuldades Enfrentadas

A parte deste trabalho que consumiu mais tempo foi sem dúvida nenhuma a implementação do modelo de simulador de comunicação utilizado em nossos testes. Isto se deve ao fato de que o sistema de suporte a construção de aplicações distribuídas não ser o mais adequado a nosso trabalho, apesar de ser o único disponível.

Para realização das medidas de tempo dos modelos simulados, foram enfrentadas algumas dificuldades. A necessidade de um grande número de micro-computadores para a realização das medidas, tornou necessário a realização de tais medidas em horários alternativos, tais como durante a noite ou em finais de semana. Existiram ainda algumas medidas que possuíam um tempo muito elevado, dificultando ainda mais sua realização.

Apesar do uso de simulação distribuída ser muito difundido, encontramos poucas referências sobre o tipo de aplicação que desejávamos contruir. Especialmente em relação a utilização de

micro-computadores como estações de processamento da simulação distrubuída. Assim por este pioneirismo tivemos um pouco de dificuldade em sua implementação.

5.3 Trabalhos Futuros

Com base em nossas definições do simulador de comunicação podemos incluir em nossa implementação algums items a mais para que nosso simulador possa ser utilizado em simulações de modelos mais complexos do que o utilizados para as medidas de tempo em nossa trabalho. Tais items são: inclusão do mecanismo de controle do andamento da simulação, inclusão do mecanismo de geração de relatórios sobre a simulação, implementação de uma forma de visualização da simulação em curso, implementação de todas as primitivas propostas em nossa definição (exemplo: transmissão/recepção de "token") e inclusão do mecanismo de geração de falhas na simulação. Além do que já foi dito podemos também modificar o mecanismo de sincronização dentro do simulador, passando a uma técnica de simulação orientada a evento assíncrona para melhorar ainda mais a performance do simulador. Todas estas alterações poderão ser realizadas sem a alteração do sistema de suporte a construção de aplicações distribuídas.

Outro ponto importante é a inclusão do mecanismo capaz de permitir

a execução de código simulado mais código real dentro do simulador. Mas para isso teremos que modificar o sistema de suporte a construção de aplicações distribuídas, passando a utilizar um sistema mais sofisticado que permita controle total sobre os processos que estão executando dentro do sistema.

5.4 Considerações Finais

Inicialmente nossa idéia era contruir um simulador para protocolos de comunicação utilizando uma arquitetura de forma distribuída, a fim permitimos o aumento da performance do simulador em relação a operação sob forma concentrada. Este tipo de aquitetura nos daria ainda a possibilidade de utilização de micro-computadores como elementos para o processamento da simulação. Após a realização de uma implementação protótipo do simulador realizamos uma série de medidas e foi possível confirmar nossas idéias iniciais. De fato com a simulação distribuída podemos aumentar a performance em relação a simulação concentrada, mas devemos para isso possuir uma sistema que permita um elevado grau de paralelismo caso contrário os resultados podem ser desastrosos.

Outros pontos positivos do trabalho foram: o estudo de diversas técnicas de sincronização para sistemas distribuídos, que podem servir para trabalhos futuros neste simulador; a definição da

possibilidade do uso de código simulado mais código real, cria uma grande abertura no sentido de facilitar a construção de protocolos de comunicação complexos, pois neste caso esse simulador seria a ferramenta ideal para testes e avaliação destes protocolos.

Neste trabalho foi realizada uma grande pesquisa no sentido de se conhecer o máximo possível sobre simulação distribuída, definir um simulador de acordo com nossos objetivos e realizarmos uma implementação protótipo a fim de verificar se os objetivos foram alcançados.

ANEXO I

INTERFACE COM O SISTEMA OPERACIONAL DE REDE LOCAL

Para utilização no nosso simulador, baseado nas primitivas do sistema operacional da rede local AMPLIWARE, poderemos dispor das seguintes primitivas para comunicação e sincronização de processos.

Comunicação

- Send_msg (@dst,@msg,ctl,st)
- Receive_msg (@src,@msg.ctl,st)

Sincronização de Processos

- Include (@estr)
- Wait (@exc,tmp)
- Signal (@exc)
- Next ()

Vamos agora definir cada primitiva e seus parametros.

1) Send_msg (@dst,@msg,ctl,st)

Esta função envia um pacote de dados através da rede para um outra estação, possui os seguintes parâmetros:

. dst - endereço de destino
. msg - pacote de dados a ser enviado
. ctl - controle (ex: bloqueada ou não)
. st - status da operação

2) Receive_msg (@src,@msg,ctl,st)

Esta função recebe um pacote de dados vindo através da rede, possui os seguintes parâmetros:

. src - endereço de origem
. msg - pacote de dados a ser enviado
. ctl - controle (ex: bloqueada ou não)
. st - status da operação

3) Include (@estr)

Esta função inclui um novo processo na lista do escalador do sistema. Após a inclusão o processo fica em "background", caso o processo seja "foreground" não há necessidade de inclui-lo. Tem como parâmetro uma estrutura que identifica o processo (estr), definida em [11].

4) Wait (@exc,temp)

Esta função retira o processo que a chamou da fila do escalador. O processo só será recolocado na fila do escalador se ocorrer um Signal, enviado por outro processo ou por uma rotina de serviço de hardware, ou um "time_out" se o Wait for temporizado. Possui os seguintes parâmetros:

. exc - exchange (para uso interno do sistema)
. temp - temporização do Wait

5) Signal (@exc)

Esta função sinaliza uma exchange, a fim de recolocar o processo que estava em Wait nesta exchange na fila do escalador. Se não houver Wait pendente o Signal ficará armazenado, mas os Signals não são enfileirados. Se esta operação de enfileiramento for

necessária deverá ser feita de forma externa ao sistema. Possui o seguinte parâmetro:

. exc - exchange (para uso interno do sistema)

6) Next ()

Esta função retira o processo de execução e o coloca no final da fila do escalador. Não possui parâmetros.

ANEXO II

Este anexo contém o código fonte dos programas desenvolvidos na confecção do protótipo do simulador de comunicação. Existem três programas: o sincronizador, o simulador de comunicação e o nó de simulação.

Estes programas possuem módulos comuns que só serão apresentados uma vez.

SINC.MAK

```
M1.OBJ:  sincn.c
    MSC /Zp/Zi/Od/Gs sincn,M1;

M2.OBJ: rxtx.c
    MSC /Zp/Zi/Od/Gs rxtx,M2;

M3.OBJ:      ssr.asm
         masm ssr,M3;

M4.OBJ:   install.asm
         masm install,M4;

M5.OBJ:   ampbasic.asm
         masm ampbasic,M5;

M6.OBJ:      libcgf.asm
         masm libcgf,M6;

sinc.exe: m1.obj m2.obj m3.obj m4.obj m5.obj m6.obj
   msl /CO /MAP m1 m2 m3 m4 m5 m6 ,sinc.exe;
```

SINC.C

```
/*
*************************************************************************
*                                                                       *
*                              SINC.C                                   *
*                                                                       *
*  Descricao :                                                          *
*              Este arquivo contem as funcoes do processo               *
*              principal do Sincronizador                               *
*  Historico :                                                          *
*                                                                       *
*                                                                       *
*  Autor :                                                              *
*         Carlos Henrique de Oliveira                                   *
*                                                                       *
*                                                                       *
*************************************************************************
*/

#include        <dos.h>
#include        <fcntl.h>
#include<stdio.h>


#include"safunc.h"
#include        "queue.h"
#include        "varglob.h"
#include        "func.h"

#define         T_MAX           0xffffffff
#define         N_MAX_PROC      5         /* 1 simulador + 4 nos */


_setenvp   () {}
_nullcheck () {}


int fh;

unsigned int n_proc;             /* numero de procesos ativos na simulacao */

struct estr_proc {       /* contem as informacoes do processos */
  unsigned long nx_time;
} estr_proc[N_MAX_PROC];

struct exc_sinc {        /* estrutura de sincronizacao entre processos */
  unsigned char b1;
  unsigned char b2;
} exc_sinc;

struct exc_info {        /* estrutura de sincronizacao entre processos */
  unsigned char b1;
  unsigned char b2;
} exc_info;
```

```
/* --- MAIN ------------------------------------------------------------

@ MAIN Funcao de entrada do Sincronizador

  Descricao:

  Entradas :
               int   argc      numero de argumentos na linha de comando
               char  *argv[]   argumentos na linha de comando
                                   argv[1] = numero de ecbs (n. de estacoes)

  Saida :
               Nemhuma

------------------------------------------------------------------------ */

main (argc,argv)

 int argc;
 char *argv[];

{
  unsigned int x;

  /* trata argumentos da linha de comando */

  for (x = 1; x < argc; x++)
  {
    if (*argv [x] != '/')
    {
      puts ("\x0d\x0aParametro invalido\x0d\x0a\x24");
      exit (1);
    }
    else
    {
      switch (*(++argv [x]))
      {
        case 'e':
        case 'E':  num_ecb = atoi (++argv [x]);
                   if (num_ecb > 40 || num_ecb < 5)
                   {
                     puts ("\x0d\x0aNumero invalido de ECBs\x0d\x0a\x24");
                     exit (1);
                   }
        break;
        default: puts ("\x0d\x0aParametro invalido\x0d\x0a\x24");
                 exit (1);
        break;
      }
    }
  }

  /* Aloca area das ECBs */

  first_ecb = (struct ECB_DESC *) aloca_area (num_ecb * sizeof(struct ECB_DESC));

  /* Aloca area das mensagens das ECBs */
```

```
  first_msg = (char *)aloca_area (num_ecb * TAM_BUF_REQ);

  memset ((char *)first_ecb, 0x00, num_ecb * sizeof(struct ECB_DESC));

  for (x = 0; x < num_ecb; x++)
     (first_ecb + x)->src_socket = SOCK_GF;

  if (open_sok ((char *) first_ecb) != OK)
  {
        /* Esqueceram de instalar o Ampliware */

    puts("Ampliware nao instalado\x0d\x0a\x24");
    exit (1);
  }
  else
  {
     if (first_ecb->retcod == 0x03)
     {
        puts ("Nao ha' soquete disponivel\x0d\x0a\x24");
        exit (1);
     }
  }


  proc_principal ();

} /* FIM MAIN */



/* --- PROC_PRINCIPAL ------------------------------------------------------

@ PROC_PRINCIPAL  Loop central do SINC

  Descricao :
               Loop principal do SINC.

  Entradas :
               Nenhuma

  Saida :
               Nenhuma

------------------------------------------------------------------------- */

char *gf_addr_aux;

void proc_principal ()

{
  unsigned int i;

  /* Pega endereco da estacao local */

  gf_addr_aux = &gf_addr [0];

  get_ad_loc ();

  get_my_psp ();                          /* Obtencao da psp do processo */
```

```
  /* Inicializa a fila de ECBs com mensagens a serem tratadas */

  first_msg_ecb = (struct ECB *) 0;
  last_msg_ecb  = (struct ECB *) 0;


  if (backg == 1)
  {
     /* deve ser instalado em background */

     _atopsp = _atopsp - 300;

     if (install (pp, 0x220, (char) 0x31))
     {
        puts ("SINC ja' instalado\x0d\x0a\x24");
        close_sok (first_ecb);
        exit (1);
     }

    _atopsp = _atopsp + 300;

    residente ();
  }
  else

    /* Vou instalar em foreground */

    pp ();

} /* FIM PROC_PRINCIPAL */

/* --- PP ------------------------------------------------------------------

@ PP  Loop infinito do processo principal

  Descricao :
               Loop infinito para controle dos eventos ocorridos no SINC

  Entradas :
               Nenhuma

  Saida :
               Nenhuma

  Obs :

------------------------------------------------------------------------- */

void pp ()
{
  unsigned int i;
  unsigned long t_event;
  int sock;

  n_proc = 0;                          /* nenhuma mensagem recebida */

  exc_sinc.b1 = 0;
  exc_sinc.b2 = 0;
```

```
  exc_info.b1 = 0;
  exc_info.b2 = 0;

  sporx();                          /* dispara ECBs de recepcao */

  for (EVER)                        /* agora espera acontecer alguma coisa */
  {

    /* se nao tiver nada para fazer vai dormir para nao atrapalhar os outros */

    await ((unsigned *)&exc_sinc,0) ;          /* espera resposta de todos os nos */

    env_msg_info() ;

    await ((unsigned *)&exc_info,0) ;          /* espera resposta do simulador */

    n_proc = 0 ;                               /* reinicaliza p/ prox vez */

    /* passa ao estado de avaliacao do proximo tempo de simulacao */

    t_event = T_MAX ;
    i = 0 ;

    while (i < N_MAX_PROC) {
      if (estr_proc[i].nx_time < t_event)
        t_event = estr_proc[i].nx_time ;
      i++ ;
    }
    /* proximo tempo de simulacao = minimo de todos processos */

    if (t_event != T_MAX)
    {
      sock = 0x221;
      for (i = 0; i < N_MAX_PROC; i++)
      {
      send_msg_time (t_event,sock) ;
      sock++;
      }
    }

  }

} /* FIM PP */


/* --- SERV_SINC ------------------------------------------------------

@ SERV_SINC trata o recebimento de mensagens

  Descricao :
              Recebe uma mensagem com a informacao de tempo do evento

  Entradas :
              Ponteiro para mensagem

  Saida :
              Nenhuma
```

```
  Obs :

------------------------------------------------------------------------ */

void serv_sinc (msg_apont)

  int *msg_apont;

{

  /* retira dados da mensagem que chegou */

  *(int *)&estr_proc[n_proc].nx_time = *msg_apont++;
  *(int *)((int *)&estr_proc[n_proc].nx_time + 1) = *msg_apont;

  n_proc++;

  if (n_proc == (N_MAX_PROC - 1)) {
    asignal ((unsigned *)&exc_sinc) ;
  }


} /* FIM SERV_SINC */

void serv_info (msg_apont)

  int *msg_apont;

{
  /* retira dados da mensagem que chegou */

  *(int *)&estr_proc[n_proc].nx_time = *msg_apont++;
  *(int *)((int *)&estr_proc[n_proc].nx_time + 1) = *msg_apont;

  asignal ((unsigned *)&exc_info) ;

}
```

SIMUL.MAK

```
M1.OBJ:  simuln.c
    MSC /Zp/Zi/Od/Gs simuln,M1;

M2.OBJ: rxtx.c
    MSC /Zp/Zi/Od/Gs rxtx,M2;

M3.OBJ: schedn.c
    MSC /Zp/Zi/Od/Gs schedn,M3;

M4.OBJ:      ssr.asm
         masm ssr,M4;

M5.OBJ:   install.asm
         masm install,M5;

M6.OBJ:   ampbasic.asm
         masm ampbasic,M6;

M7.OBJ:      libcgf.asm
         masm libcgf,M7;

simul.exe: m1.obj m2.obj m3.obj m4.obj m5.obj m6.obj m7.obj
   msl /CO /MAP m1 m2 m3 m4 m5 m6 m7 ,simul.exe;
```

SIMUL.C

```
/*
*************************************************************************
*                                                                       *
*                       SIMUL.C                                         *
*                                                                       *
*  Descricao :                                                          *
*               Este arquivo contem as funcoes do processo              *
*               principal do Simulador de Comunicacao                   *
*  Historico :                                                          *
*                                                                       *
*                                                                       *
*  Autor :                                                              *
*          Carlos Henrique de Oliveira                                  *
*                                                                       *
*                                                                       *
*************************************************************************
*/

#include        <dos.h>
#include        <fcntl.h>
#include<stdio.h>


#include"safunc.h"
#include        "queue.h"
#include        "varglob.h"
#include        "func.h"

#define         T_MAX           0xffffffff
#define         N_MAX_EST       3         /* 3 nos */


_setenvp   () {}
_nullcheck () {}


int fh;

struct exc_sinc {         /* estrutura de sincronizacao entre processos */
  unsigned char b1;
  unsigned char b2;
} exc_sinc ;

struct exc_wait {         /* estrutura de sincronizacao entre processos */
  unsigned char b1;
  unsigned char b2;
} exc_wait ;

int station_wait;
int flag_wait;


/* --- MAIN ----------------------------------------------------------
```

```
@ MAIN Funcao de entrada do No de Simulacao

  Descricao:

  Entradas :
               int   argc      numero de argumentos na linha de comando
               char  *argv[]   argumentos na linha de comando
                                   argv[1] = numero de ecbs (n. de estacoes)

  Saida :
               Nemhuma

------------------------------------------------------------------------ */

main (argc,argv)

 int argc;
 char *argv[];

{
  unsigned int x;

  /* trata argumentos da linha de comando */

  for (x = 1; x < argc; x++)
  {
    if (*argv [x] != '/')
    {
      puts ("\x0d\x0aParametro invalido\x0d\x0a\x24");
      exit (1);
    }
    else
    {
      switch (*(++argv [x]))
      {
        case 'e':
        case 'E':  num_ecb = atoi (++argv [x]);
                   if (num_ecb > 40 || num_ecb < 5)
                   {
                     puts ("\x0d\x0aNumero invalido de ECBs\x0d\x0a\x24");
                     exit (1);
                   }
        break;
        default: puts ("\x0d\x0aParametro invalido\x0d\x0a\x24");
                 exit (1);
        break;
      }
    }
  }

  /* Aloca area das ECBs */

  first_ecb = (struct ECB_DESC *) aloca_area (num_ecb * sizeof(struct ECB_DESC));

  /* Aloca area das mensagens das ECBs */

  first_msg = (char *)aloca_area (num_ecb * TAM_BUF_REQ);

  memset ((char *)first_ecb, 0x00, num_ecb * sizeof(struct ECB_DESC));
```

```
  for (x = 0; x < num_ecb; x++)
     (first_ecb + x)->src_socket = 0x221;

  if (open_sok ((char *) first_ecb) != OK)
  {
        /* Esqueceram de instalar o Ampliware */

    puts("Ampliware nao instalado\x0d\x0a\x24");
    exit (1);
  }
  else
  {
     if (first_ecb->retcod == 0x03)
     {
        puts ("Nao ha' soquete disponivel\x0d\x0a\x24");
        exit (1);
     }
  }


  proc_principal ();

} /* FIM MAIN */



/* --- PROC_PRINCIPAL ------------------------------------------------------

@ PROC_PRINCIPAL  Loop central do No de Simulacao

  Descricao :
               Loop principal do No.

  Entradas :
               Nenhuma

  Saida :
               Nenhuma

------------------------------------------------------------------------- */

char *gf_addr_aux;

void proc_principal ()

{
  unsigned int i;

  /* Pega endereco da estacao local */

  gf_addr_aux = &gf_addr [0];

  get_ad_loc ();

  get_my_psp ();                              /* Obtencao da psp do processo */

  /* Inicializa a fila de ECBs com mensagens a serem tratadas */
```

```
  first_msg_ecb = (struct ECB *) 0;
  last_msg_ecb  = (struct ECB *) 0;


  if (backg == 1)
  {
     /* deve ser instalado em background */

     _atopsp = _atopsp - 300;

     if (install (pp, 0x221, (char) 0x31))
     {
        puts ("SIMULADOR ja' instalado\x0d\x0a\x24");
        close_sok (first_ecb);
        exit (1);
     }

    _atopsp = _atopsp + 300;

    residente ();
  }
  else

    /* Vou instalar em foreground */

    pp ();

} /* FIM PROC_PRINCIPAL */

/* --- PP ------------------------------------------------------------------

@ PP  Loop infinito do processo principal

  Descricao :
               Loop infinito para controle dos eventos ocorridos no No

  Entradas :
               Nenhuma

  Saida :
               Nenhuma

  Obs :

------------------------------------------------------------------------- */

void pp ()
{
  unsigned int i;
  struct event_entry *aux_apont;

  sporx();                          /* dispara ECBs de recepcao */

  exc_sinc.b1 = 0;
  exc_sinc.b2 = 0;

  exc_wait.b1 = 0;
  exc_wait.b2 = 0;
  flag_wait = FALSE;
```

```
ini_var();                          /* inicializa var. e programa evento */

for (EVER)                          /* agora espera acontecer alguma coisa */
{

  /* se nao tiver nada para fazer vai dormir para nao atrapalhar os outros */

  await ((unsigned *)&exc_sinc,0) ;          /* espera resposta de todos os processos */

  if (event_list->time == cur_time)
  {
    event = event_list->event;
    station = event_list->station;
    dst_no = event_list->dst_no;
    msg_apont = event_list->msg_apont;
    state_tx = tx_station_state[station];
    state_rx = rx_station_state[station];

    /* guarda evento, tempo e estados */

    switch (event)

    {
    case PKT_FRONT:
    bus_state[station] = BUSY;

    switch (state_rx)
    {
    case IDLE:
    if ((dst_no == 0xffff) || (dst_no == station))
      {
      previous_rx_state[station] = state_rx;
      rx_station_state[station] = RECEBENDO;
      inicia_rx();
      }
    break;

    case RECEBENDO:
    previous_rx_state[station] = state_rx;
    rx_station_state[station] = IDLE;
    env_msg_erro();
    break;
    }

    if (state_tx == TRANSMITINDO)
    {
      previous_tx_state[station] = state_tx;
      tx_station_state[station] = IDLE;
      cancel_pkt_back();
      env_msg_col();
    }
    break;

    case PKT_BACK:
    bus_state[station] = IDLE;

    if (state_rx == RECEBENDO)
    {
```

```
        previous_rx_state[station] = state_rx;
        rx_station_state[station] = IDLE;
        ecb_end_lib = ecb_station[station];
        libera_msg();
      }

      switch (state_tx)
      {
      case TRANSMITINDO:
      previous_tx_state[station] = state_tx;
      tx_station_state[station] = IDLE;
      break;

      case WAIT_EOC:
      previous_tx_state[station] = state_tx;
      tx_station_state[station] =IDLE;
      env_msg_wait();
      flag_wait = TRUE;
      station_wait = station;
      await ((unsigned *)&exc_wait,0);
      break;
      }
      break;

      }

      aux_apont = event_list;
      event_list = event_list->next;
      libera(aux_apont);

    }

  }

} /* FIM PP */


/* --- SERV_NO ------------------------------------------------------------

@ SERV_SINC trata o recebimento de mensagens

  Descricao :
               Recebe uma mensagem com a informacao,dados ou controle

  Entradas :
               Ponteiro para mensagem

  Saida :
               Nenhuma

  Obs :

------------------------------------------------------------------------- */

void serv_no (msg_apont)

  int *msg_apont;

{
```

```
unsigned int msg_ctl;
unsigned int msg_tp;
unsigned int msg_func;
unsigned int station;
struct buf_msg *aux_apont;

recoloc = TRUE;
aux_apont = msg_apont;

msg_ctl = *msg_apont++;
msg_tp = *msg_apont++;

switch (msg_tp)
{

case MSG_SINC:
*(int *)&cur_time = *msg_apont++;
*(int *)((int *)&cur_time + 1) = *msg_apont;
asignal ((unsigned *)&exc_sinc);
break;

case MSG_INFO:
msg_func = *msg_apont++;
  switch (msg_func)
  {

  case RESP_REQ_TRANS:
  station = *msg_apont;
  if (bus_state[station] == BUSY)
  {
    previous_tx_state[station] = tx_station_state[station];
    tx_station_state[station] = WAIT_EOC;
  }
  env_msg_state(station);
  break;

  case INICIO_TX:
  station = *msg_apont++;
  recoloc = FALSE;
  ecb_station[aux_apont->dst] = ecb_end;
  previous_tx_state[station] = tx_station_state[station];
  tx_station_state[station] = TRANSMITINDO;
  prog_pkt_front(station,aux_apont);
  prog_pkt_back(station,aux_apont);
  if (flag_wait)
  {
    if (station == station_wait)
    {
       asignal ((unsigned *)&exc_wait);
       flag_wait = FALSE;
    }
  }
  break;

  case INFO_TIME:
  if (event_list != NIL)
  {
    env_msg_sinc(event_list->time);
  }
```

```
    else
    {
      env_msg_sinc(0x80000000);
    }
    break;

    }
  break;
  }

} /* FIM SERV_NO */


void ini_var()
{

int i;

cur_time = 0;                /* inicio da simulacao */

recoloc = TRUE;              /* flag das estruturas de recepcao */

for (i = 0; i < N_MAX_STATION; i++)
{
tx_station_state[i],rx_station_state[i] = IDLE;
bus_state[i] = IDLE;
}
/* inicializacao variaveis da rede */

end_est[0] = 0x26;             /* 0x26; */
end_sock[0] = 0x222;
n_est_conect[0] = 1;
temp_prop[0][0] = 5;
est_end[0][0] = 1;

end_est[1] = 0xf5;             /* 0xf5; */
end_sock[1] = 0x223;
n_est_conect[1] = 1;
temp_prop[1][0] = 5;
est_end[1][0] = 0;

end_est[2] = 0x10;             /* 0x10; */
end_sock[2] = 0x224;
n_est_conect[2] = 1;
temp_prop[2][0] = 6;
est_end[2][0] = 3;

end_est[3] = 0x11;             /* 0x11; */
end_sock[3] = 0x225;
n_est_conect[3] = 1;
temp_prop[3][0] = 6;
est_end[3][0] = 2;

/*end_est[4] = 0x26;             0x5;
end_sock[4] = 0x226;
n_est_conect[4] = 1;
temp_prop[4][0] = 3;
est_end[4][0] = 5;
```

```
end_est[5] = 0x10;              0x6;
end_sock[5] = 0x227;
n_est_conect[5] = 1;
temp_prop[5][0] = 3;
est_end[5][0] = 4;*/

event_list = NIL;

apont_free = &estr_evento[0];
estr_evento[N_MAX_EVENT - 1].next = NIL;

for (i = 0; i < (N_MAX_EVENT - 1); i++)
{
estr_evento[i].next = &estr_evento[i + 1];
}

}
```

SCHED.C

```
/*
*************************************************************************
*                                                                       *
*                              SCHED.C                                  *
*                                                                       *
*    Descricao :                                                        *
*                Funcoes auxiliares de simulacao                        *
*                                                                       *
*                                                                       *
*************************************************************************
*/

#include        "dos.h"
#include        "safunc.h"
#include"queue.h"
#include"varext.h"

#include        "func.h"



void env_msg_sinc (t_event)

unsigned long t_event;

{

ecbtx.frag_cnt = 1;
ecbtx.frags [0] = 8;                    /* tamanho do dados */
getofseg ((char *) &msg_tx, &ecbtx.frags [1], &ecbtx.frags [2]);
ecbtx.src_socket = 0x221;
ecbtx.dst_net = 0;
ecbtx.dst_node = SINC_ADDR;
ecbtx.dst_socket = 0x220;
ecbtx.esr_off = ecbtx.esr_seg = 0;
msg_tx.ctl = 1;
msg_tx.tp = 0;
msg_tx.dado1 = *(int *)&t_event;
msg_tx.dado2 = *(int *)((int *)&t_event + 1);

txmsg(&ecbtx);                      /* envia mensagem as outras estacoes */
}

void env_msg_erro ()

{

ecbtx.frag_cnt = 1;
ecbtx.frags [0] = 8;                    /* tamanho do dados */
getofseg ((char *) &msg_tx, &ecbtx.frags [1], &ecbtx.frags [2]);
ecbtx.src_socket = 0x221;
ecbtx.dst_net = 0;
ecbtx.dst_node = end_est[station];
ecbtx.dst_socket = end_sock[station];
```

```
ecbtx.esr_off = ecbtx.esr_seg = 0;
msg_tx.ctl = 1;
msg_tx.tp = 1;
msg_tx.dado1 = ERRO_RECEP;
msg_tx.dado2 = 0;

txmsg(&ecbtx);                    /* envia mensagem as outras estacoes */
}

void env_msg_col ()

{

ecbtx.frag_cnt = 1;
ecbtx.frags [0] = 8;                /* tamanho do dados */
getofseg ((char *) &msg_tx, &ecbtx.frags [1], &ecbtx.frags [2]);
ecbtx.src_socket = 0x221;
ecbtx.dst_net = 0;
ecbtx.dst_node = end_est[station];
ecbtx.dst_socket = end_sock[station];
ecbtx.esr_off = ecbtx.esr_seg = 0;
msg_tx.ctl = 1;
msg_tx.tp = 1;
msg_tx.dado1 = COLISAO_DET;
msg_tx.dado2 = 0;

txmsg(&ecbtx);                    /* envia mensagem as outras estacoes */
}

void env_msg_wait ()

{

ecbtx.frag_cnt = 1;
ecbtx.frags [0] = 8;                /* tamanho do dados */
getofseg ((char *) &msg_tx, &ecbtx.frags [1], &ecbtx.frags [2]);
ecbtx.src_socket = 0x221;
ecbtx.dst_net = 0;
ecbtx.dst_node = end_est[station];
ecbtx.dst_socket = end_sock[station];
ecbtx.esr_off = ecbtx.esr_seg = 0;
msg_tx.ctl = 1;
msg_tx.tp = 1;
msg_tx.dado1 = BUS_LIVRE;
msg_tx.dado2 = 0;

txmsg(&ecbtx);                    /* envia mensagem as outras estacoes */
}

void env_msg_state (station)

int station;
{

ecbtx.frag_cnt = 1;
ecbtx.frags [0] = 8;                /* tamanho do dados */
getofseg ((char *) &msg_tx, &ecbtx.frags [1], &ecbtx.frags [2]);
ecbtx.src_socket = 0x221;
ecbtx.dst_net = 0;
```

```
ecbtx.dst_node = end_est[station];
ecbtx.dst_socket = end_sock[station];
ecbtx.esr_off = ecbtx.esr_seg = 0;
msg_tx.ctl = 1;
msg_tx.tp = 1;
msg_tx.dado1 = RESP_REQ_TRANS;
msg_tx.dado2 = bus_state[station];

txmsg(&ecbtx);                    /* envia mensagem as outras estacoes */
}

void inicia_rx ()

{

ecbtx.frag_cnt = 1;
ecbtx.frags [0] = msg_apont->len + HEAD_TAM ;       /* tamanho do dados */
getofseg ((char *) msg_apont, &ecbtx.frags [1], &ecbtx.frags [2]);
ecbtx.src_socket = 0x221;
ecbtx.dst_net = 0;
ecbtx.dst_node = end_est[station];
ecbtx.dst_socket = end_sock[station];
ecbtx.esr_off = ecbtx.esr_seg = 0;
msg_apont->msg_ctl = 1;
msg_apont->msg_tp = 1;
msg_apont->msg_info = INICIO_RECEP;

txmsg(&ecbtx);                    /* envia mensagem as outras estacoes */
}


void prog_pkt_front (station,msg_apont)

int station;
struct buf_msg *msg_apont;
{
int i;

for (i = 0; i < n_est_conect[station]; i++)
{
  sched_event(PKT_FRONT,cur_time + temp_prop[station][i],est_end[station][i],msg_apont-
>dst,msg_apont);
}
}

void prog_pkt_back (station,msg_apont)

int station;
struct buf_msg *msg_apont;
{
int i;
unsigned long t_prop_msg;

t_prop_msg = ((unsigned long)(((msg_apont->len + HEAD_TAM + PREAM_LEN)* 8) / BUS_RATE));

for (i = 0; i < n_est_conect[station]; i++)
{
  sched_event(PKT_BACK,cur_time + temp_prop[station][i] + t_prop_msg,est_end[station][i],msg_apont-
>dst,msg_apont);
```

```
}
sched_event(PKT_BACK,cur_time + t_prop_msg,station,msg_apont->dst,msg_apont);
}


void cancel_pkt_back()

{
int i;

for (i = 0; i < n_est_conect[station]; i++)
{
  cancel_event(PKT_BACK,est_end[station][i]);
}
cancel_event(PKT_BACK,station);
}


void sched_event(event,event_time,station,dst_no,msg_apont)
int event;
unsigned long event_time;
int station;
int dst_no;
struct buf_msg *msg_apont;

{
struct event_entry *apont_list_event;
struct event_entry *aux_apont,*ant_apont;

if (apont_free != NIL)
  apont_list_event = apont_free;
  apont_free = apont_free->next;

aux_apont = event_list;
ant_apont = NIL;
while (aux_apont != NIL && aux_apont->time<=event_time)
{
 ant_apont = aux_apont;
 aux_apont = aux_apont->next;
}

if (aux_apont != NIL)
{
  if (ant_apont == NIL)
  {
    event_list = apont_list_event;
    event_list->next = aux_apont;
    aux_apont = event_list;
  }
  else
  {
    ant_apont->next = apont_list_event;
    ant_apont->next->next = aux_apont;
    aux_apont=ant_apont->next;
  }
}
else
{
  if (ant_apont == NIL)
```

```
  {
    event_list = apont_list_event;
    aux_apont = event_list;
    aux_apont->next = NIL;
  }
  else
  {
    ant_apont->next = apont_list_event;
    aux_apont=ant_apont->next;
    aux_apont->next = NIL;
  }
}
aux_apont->event=event;
aux_apont->time=event_time;
aux_apont->station=station;
aux_apont->dst_no=dst_no;
aux_apont->msg_apont=msg_apont;
}

void cancel_event(event,station)
int event;
int station;

{
struct event_entry *aux_apont,*ant_apont;

aux_apont = event_list;
ant_apont = NIL;
while (aux_apont != NIL && (aux_apont->event != event || aux_apont->station != station))
{
 ant_apont = aux_apont;
 aux_apont = aux_apont->next;
}

if (aux_apont != NIL)
{
  if (ant_apont != NIL)
    ant_apont->next=aux_apont->next;
  else
    event_list=aux_apont->next;
  libera(aux_apont);
}

}

void libera(apont_event)

struct event_entry *apont_event;
{
 struct event_entry *aux_apont;
 struct event_entry *ant_apont;
 struct event_entry *sal_apont;

 sal_apont = apont_event;

 aux_apont = apont_free;
 if (aux_apont == NIL)
 {
   apont_free = sal_apont;
```

```
    sal_apont->next = NIL;
  }
  else
  {
    ant_apont = aux_apont;
    aux_apont = aux_apont->next;
    while (aux_apont != NIL)
    {
     ant_apont = aux_apont;
     aux_apont = aux_apont->next;
    }
    ant_apont->next = sal_apont;
    sal_apont->next = NIL;
  }
}
```

NO.MAK

```
M1.OBJ:  non.c
    MSC /Zp/Zi/Od/Gs non,M1;

M2.OBJ: rxtx.c
    MSC /Zp/Zi/Od/Gs rxtx,M2;

M3.OBJ: schedn.c
    MSC /Zp/Zi/Od/Gs schedn,M3;

M4.OBJ:      ssr.asm
         masm ssr,M4;

M5.OBJ:   install.asm
         masm install,M5;

M6.OBJ:   ampbasic.asm
         masm ampbasic,M6;

M7.OBJ:      libcgf.asm
         masm libcgf,M7;

non.exe: m1.obj m2.obj m3.obj m4.obj m5.obj m6.obj m7.obj
   msl /CO /MAP m1 m2 m3 m4 m5 m6 m7 ,no.exe;
```

NO.C

```
/*
*************************************************************************
*                                                                       *
*                              NO.C                                     *
*                                                                       *
*  Descricao :                                                          *
*              Este arquivo contem as funcoes do processo               *
*              principal do No de Simulacao                             *
*  Historico :                                                          *
*                                                                       *
*                                                                       *
*  Autor :                                                              *
*         Carlos Henrique de Oliveira                                   *
*                                                                       *
*                                                                       *
*************************************************************************
*/

#include        <dos.h>
#include        <fcntl.h>
#include<stdio.h>


#include"safunc.h"
#include        "queue.h"
#include        "varglob.h"
#include        "func.h"

#define         T_MAX           0xffffffff
#define         N_MAX_PROC      4         /* 1 simulador + 3 nos */

#define          N_ITERAC 10
#define         N_EVENT         6

_setenvp   () {}
_nullcheck () {}


int fh;

struct exc_sinc {         /* estrutura de sincronizacao entre processos */
  unsigned char b1;
  unsigned char b2;
} exc_sinc ;

int it_count;
int count_ev;

unsigned long thr_est;
unsigned long t_medio_tx;
unsigned long t_medio_g;
unsigned long atraso_f;

struct time_t {
```

```
  unsigned char m;
  unsigned char s;
  unsigned char c;
};

struct time_t time_i;
struct time_t time_f;


/* --- MAIN ------------------------------------------------------------

@ MAIN Funcao de entrada do No de Simulacao

  Descricao:

  Entradas :
               int   argc      numero de argumentos na linha de comando
               char  *argv[]   argumentos na linha de comando
                                   argv[1] = numero de ecbs (n. de estacoes)

  Saida :
               Nemhuma

------------------------------------------------------------------------ */

main (argc,argv)

 int argc;
 char *argv[];

{
  unsigned int x;

  /* trata argumentos da linha de comando */

  for (x = 1; x < argc; x++)
  {
    if (*argv [x] != '/')
    {
      puts ("\x0d\x0aParametro invalido\x0d\x0a\x24");
      exit (1);
    }
    else
    {
      switch (*(++argv [x]))
      {
        case 'e':
        case 'E':  num_ecb = atoi (++argv [x]);
                   if (num_ecb > 40 || num_ecb < 5)
                   {
                     puts ("\x0d\x0aNumero invalido de ECBs\x0d\x0a\x24");
                     exit (1);
                   }
        break;
        default: puts ("\x0d\x0aParametro invalido\x0d\x0a\x24");
                 exit (1);
        break;
      }
    }
```

```
  }

  /* Aloca area das ECBs */

  first_ecb = (struct ECB_DESC *) aloca_area (num_ecb * sizeof(struct ECB_DESC));

  /* Aloca area das mensagens das ECBs */

  first_msg = (char *)aloca_area (num_ecb * TAM_BUF_REQ);

  memset ((char *)first_ecb, 0x00, num_ecb * sizeof(struct ECB_DESC));

  for (x = 0; x < num_ecb; x++)
     (first_ecb + x)->src_socket = MEU_SOCK;

  if (open_sok ((char *) first_ecb) != OK)
  {
        /* Esqueceram de instalar o Ampliware */

    puts("Ampliware nao instalado\x0d\x0a\x24");
    exit (1);
  }
  else
  {
     if (first_ecb->retcod == 0x03)
     {
        puts ("Nao ha' soquete disponivel\x0d\x0a\x24");
        exit (1);
     }
  }


  proc_principal ();

} /* FIM MAIN */



/* --- PROC_PRINCIPAL ------------------------------------------------------

@ PROC_PRINCIPAL  Loop central do No de Simulacao

  Descricao :
               Loop principal do No.

  Entradas :
               Nenhuma

  Saida :
               Nenhuma

------------------------------------------------------------------------- */

char *gf_addr_aux;

void proc_principal ()

{
  unsigned int i;
```

```
  /* Pega endereco da estacao local */

  gf_addr_aux = &gf_addr [0];

  get_ad_loc ();

  get_my_psp ();                              /* Obtencao da psp do processo */

  /* Inicializa a fila de ECBs com mensagens a serem tratadas */

  first_msg_ecb = (struct ECB *) 0;
  last_msg_ecb  = (struct ECB *) 0;


  if (backg == 1)
  {
     /* deve ser instalado em background */

     _atopsp = _atopsp - 300;

     if (install (pp, MEU_SOCK, (char) 0x31))
     {
        puts ("NO ja' instalado\x0d\x0a\x24");
        close_sok (first_ecb);
        exit (1);
     }

    _atopsp = _atopsp + 300;

    residente ();
  }
  else

    /* Vou instalar em foreground */

    pp ();

} /* FIM PROC_PRINCIPAL */

/* --- PP ------------------------------------------------------------------

@ PP  Loop infinito do processo principal

  Descricao :
               Loop infinito para controle dos eventos ocorridos no No

  Entradas :
               Nenhuma

  Saida :
               Nenhuma

  Obs :

------------------------------------------------------------------------ */

void pp ()
{
```

```
unsigned int i;
int nada;
struct event_entry *aux_apont;

sporx();                              /* dispara ECBs de recepcao */

exc_sinc.b1 = 0;
exc_sinc.b2 = 0;

it_count = 0;
count_ev = 0;

srand(1);

puts("Inicio das Iteracoes\x0d\x0a\x24");
get_hora_i();

ini_var();                            /* inicializa var. e programa evento */

for (EVER)                           /* agora espera acontecer alguma coisa */
{

  /* se nao tiver nada para fazer vai dormir para nao atrapalhar os outros */

  await ((unsigned *)&exc_sinc,0) ;            /* espera resposta de todos os processos */

  while ((event_list->time <= cur_time_sis) && (event_list != NIL))
  {
    event = event_list->event;
    state = station_state;
    state_tx = tx_station_state;
    state_rx = rx_station_state;
    cur_time = event_list->time;

    /* guarda evento, tempo e estados */

    switch (event)

    {
    case PK_USU:
    if (state == IDLE)
    {
      it_count++;
      tx_temp[it_count - 1].t_g = cur_time;
      if (it_count == N_ITERAC)
      {
        get_hora_f();
        puts("Final das Iteracoes\x0d\x0a\x24");
        puts("\x0d\x0a\x24");
        printf("Tempo Inicial = %hu:%hu:%hu",time_i.m,time_i.s,time_i.c);
        puts("\x0d\x0a\x24");
        printf("Tempo Final   = %hu:%hu:%hu",time_f.m,time_f.s,time_f.c);
        puts("\x0d\x0a\x24");
/*        puts("t_geracao   t_inic_tx     t_fim_tx\x0d\x0a\x24");
        for (i = 0; i < N_ITERAC; i++)
        {
         printf("%lu    %lu     %lu",tx_temp[i].t_g,tx_temp[i].t_tx,tx_temp[i].t_f_tx);
         puts("\x0d\x0a\x24");
        }
```

```
        puts("\x0d\x0a\x24");
        puts("t_inic_rx   t_fim_rx    t_resp_usu\x0d\x0a\x24");
        for (i = 0; i < N_ITERAC; i++)
        {
         printf("%lu    %lu     %lu",rx_temp[i].t_rx,rx_temp[i].t_f_rx,rx_temp[i].t_r);
         puts("\x0d\x0a\x24");
        }*/
        puts("\x0d\x0a\x24");

        /* atraso fim a fim */

        atraso_f = 0;
        for (i = 0; i < N_ITERAC - 1; i++)
        {
        atraso_f = atraso_f + (rx_temp[i].t_r - tx_temp[i].t_g);
        }
        atraso_f = atraso_f / (N_ITERAC - 1);
        printf("Atraso fim a fim = %lu microseg",atraso_f);
        puts("\x0d\x0a\x24");

        /* tempo medio no no'*/

        t_medio_g = 0;
        for (i = 0; i < N_ITERAC - 1; i++)
        {
        t_medio_g = t_medio_g + (tx_temp[i].t_f_tx - tx_temp[i].t_g);
        }
        t_medio_g = t_medio_g / (N_ITERAC - 1);
        printf("Tempo medio no no' = %lu microseg",t_medio_g);
        puts("\x0d\x0a\x24");

        t_medio_tx = 0;
        for (i = 0; i < N_ITERAC - 1; i++)
        {
        t_medio_tx = t_medio_tx + (tx_temp[i].t_f_tx - tx_temp[i].t_tx);
        }
        t_medio_tx = t_medio_tx / (N_ITERAC - 1);
        printf("Tempo medio no transmissor = %lu microseg",t_medio_tx);
        puts("\x0d\x0a\x24");

        /* throughput da estacao */
        thr_est = (N_ITERAC * 1000000) / (tx_temp[N_ITERAC - 1].t_g - tx_temp[0].t_g);
        printf("Numero de mensagens por segundo = %lu",thr_est);
        puts("\x0d\x0a\x24");

        sai();
      }
      previous_state = station_state;
      station_state = PROC_TX;
      gera_msg();
      sched_event(PK_TRANSP,cur_time+((unsigned long)(t_proc_usu1())));
      sched_event_ext(INICIO_TX,cur_time+((unsigned
long)(t_proc_usu1()))+t_proc_transp+t_proc_rede);
    }
    break;

    case PK_TRANSP:
    if (state == PROC_TX)
      if (count_ev < N_EVENT -1)
```

```
    {
    count_ev++;
    sched_event(PK_TRANSP,cur_time+(t_proc_transp/N_EVENT));
    for ( i = 0; i < 50000; i++)
     {
       nada = 0;
     }
    }
    else
    {
      sched_event(PK_REDE,cur_time+(t_proc_transp/N_EVENT));
      count_ev = 0;
    }
break;

case PK_REDE:
if (state == PROC_TX)
  for ( i = 0; i < 20000; i++)
  {
     nada = 0;
  }
  sched_event(INICIO_TX,cur_time+t_proc_rede);
break;

case RESP_USU:
if (state == PROC_RX)
{
  rx_temp[it_count - 1].t_r = cur_time;
  previous_state = station_state;
  station_state = IDLE;
  sched_event(PK_USU,cur_time+((unsigned long)(t_proc_usu2())));
  sched_event_ext(PK_USU,cur_time+((unsigned long)(t_proc_usu2())));
  libera_msg();
}
break;

case RESP_TRANSP:
if (state == PROC_RX)
  if (count_ev < N_EVENT -1)
    {
    count_ev++;
    sched_event(RESP_TRANSP,cur_time+(t_proc_transp/N_EVENT));
    for ( i = 0; i < 50000; i++)
     {
       nada = 0;
     }
    }
    else
    {
      sched_event(RESP_USU,cur_time+(t_proc_transp/N_EVENT));
      count_ev = 0;
    }
break;

case RESP_REDE:
if (state == PROC_RX)
  for ( i = 0; i < 20000; i++)
  {
     nada = 0;
```

```
  }
  sched_event(RESP_TRANSP,cur_time+t_proc_rede);
break;

case INICIO_TX:
if (state == PROC_TX)
{
  tx_temp[it_count - 1].t_tx = cur_time;
  previous_state = station_state;
  station_state = IDLE;
  put_fila_tx();
  if (state_tx == IDLE)
  {
    collision_count = 0 ;
    if (bus_idle())
    {
      previous_tx_state = tx_station_state;
      tx_station_state = TRANSMITINDO;
      start_tx();
    }
    else
    {
      previous_tx_state = tx_station_state;
      tx_station_state = WAIT_EOC;
    }
  }
}
break;

case FIM_TX:
get_fila_tx();
if (state_tx == TRANSMITINDO)
{
  tx_temp[it_count - 1].t_f_tx = cur_time;
  previous_tx_state = tx_station_state;
  tx_station_state = HOLD;
  sched_event(TRANSM_IDLE,cur_time+holding_time);
  sched_event_ext(TRANSM_IDLE,cur_time+holding_time);
}
break;

case FIM_WAIT:
if (state_tx == WAIT_RANDOM)
{
  if (bus_idle())
  {
    previous_tx_state = tx_station_state;
    tx_station_state = TRANSMITINDO;
    start_tx();
  }
  else
  {
    previous_tx_state = tx_station_state;
    tx_station_state = WAIT_EOC;
  }
}
break;

case TRANSM_IDLE:
```

```
  if (state_tx == HOLD)
  {
    if (get_tx == put_tx)
    {
      previous_tx_state = tx_station_state;
      tx_station_state = IDLE;
    }
    else
    {
      collision_count = 0 ;
      if (bus_idle())
      {
        previous_tx_state = tx_station_state;
        tx_station_state = TRANSMITINDO;
        start_tx();
      }
      else
      {
        previous_tx_state = tx_station_state;
        tx_station_state = WAIT_EOC;
      }
    }
  }
  break;

  case FIM_RECEP:
  if (state_rx == RECEBENDO)
  {
    previous_rx_state = rx_station_state;
    rx_station_state = IDLE;
    if (state == IDLE)
    {
      rx_temp[it_count - 1].t_f_rx = cur_time;
      previous_state = station_state;
      station_state = PROC_RX;
      sched_event(RESP_REDE,cur_time+t_proc_enlace);
      sched_event_ext(RESP_USU,cur_time+t_proc_enlace+t_proc_rede+t_proc_transp);
    }
  }
  break;

  }

  aux_apont = event_list;
  event_list = event_list->next;
  libera(aux_apont);

}
if (event_list_ext != NIL)
{
  if (event_list_ext->time == cur_time_sis)
  {
    aux_apont = event_list_ext;
    event_list_ext = event_list_ext->next;
    libera_ext(aux_apont);
  }
}

if (event_list_ext != NIL)
```

```
    {
      cur_time = cur_time_sis;
      env_msg_sinc(event_list_ext->time);
    }
    else
    {
      env_msg_sinc(0x80000000);
    }

  }

} /* FIM PP */


/* --- SERV_NO ------------------------------------------------------------

@ SERV_SINC trata o recebimento de mensagens

  Descricao :
               Recebe uma mensagem com a informacao,dados ou controle

  Entradas :
               Ponteiro para mensagem

  Saida :
               Nenhuma

  Obs :

------------------------------------------------------------------------- */

void serv_no (msg_apont)

  int *msg_apont;

{
  unsigned int msg_ctl;
  unsigned int msg_tp;
  unsigned int msg_func;
  struct buf_msg *aux_apont;

  recoloc = TRUE;
  aux_apont = msg_apont;

  msg_ctl = *msg_apont++;
  msg_tp = *msg_apont++;

  switch (msg_tp)
  {

  case MSG_SINC:
  *(int *)&cur_time_sis = *msg_apont++;
  *(int *)((int *)&cur_time_sis + 1) = *msg_apont;
  asignal ((unsigned *)&exc_sinc);
  break;

  case MSG_INFO:
  msg_func = *msg_apont++;
    switch (msg_func)
```

```
    {

    case RESP_REQ_TRANS:
    bus_state = *msg_apont;
    asignal((unsigned *)&exc_info);
    break;

    case BUS_LIVRE:
    if (tx_station_state == WAIT_EOC)
    {
      previous_tx_state = tx_station_state;
      tx_station_state = TRANSMITINDO;
      start_tx1();
    }
    break;

    case COLISAO_DET:
    if (tx_station_state == TRANSMITINDO)
    {
      cancel_event(FIM_TX);
      cancel_event_ext(FIM_TX);
      ++collision_count;
      previous_tx_state = tx_station_state;
      tx_station_state = WAIT_RANDOM;
      sched_event(FIM_WAIT,cur_time_sis+((unsigned long)(t_wait())));
      sched_event_ext(FIM_WAIT,cur_time_sis+((unsigned long)(t_wait())));
    }
    break;

    case INICIO_RECEP:
    if (rx_station_state == IDLE)
    {
      rx_temp[it_count - 1].t_rx = cur_time_sis;
      recoloc = FALSE;
      ecb_end_lib = ecb_end;
      msg_rx_addr = aux_apont;
      previous_rx_state = rx_station_state;
      rx_station_state = RECEBENDO;
      sched_event(FIM_RECEP,cur_time_sis+((unsigned long)(t_rec_msg())));
      sched_event_ext(FIM_RECEP,cur_time_sis+((unsigned long)(t_rec_msg())));
    }
    break;

    case ERRO_RECEP:
    if (rx_station_state == RECEBENDO)
    {
      previous_rx_state = rx_station_state;
      rx_station_state = IDLE;
      libera_msg();
      cancel_event(FIM_RECEP);
      cancel_event_ext(FIM_RECEP);
    }
    break;

    }
  break;
  }

} /* FIM SERV_NO */
```

```
void ini_var()
{

int i;

exc_info.b1 = 0;
exc_info.b2 = 0;

var_len = 0;

recoloc = TRUE;

bus_state = TRUE;

collision_count = 0;

cur_time = 0;                   /* inicio da simulacao */
cur_time_sis = 0;

station_state,tx_station_state,rx_station_state = IDLE;

event_list = NIL;
event_list_ext = NIL;

put_tx,get_tx = 0;

apont_free = &estr_evento[0];
estr_evento[N_MAX_EVENT - 1].next = NIL;
apont_free_ext = &estr_evento_ext[0];
estr_evento_ext[N_MAX_EVENT - 1].next = NIL;

for (i = 0; i < (N_MAX_EVENT - 1); i++)
{
estr_evento[i].next = &estr_evento[i + 1];
estr_evento_ext[i].next = &estr_evento_ext[i + 1];
}

sched_event(PK_USU,2);
sched_event_ext(PK_USU,2);

env_msg_sinc(event_list_ext->time);

}
```

SCHEDNO.C

```
/*
*************************************************************************
*                                                                       *
*                               SCHED.C                                 *
*                                                                       *
*    Descricao :                                                        *
*                Funcoes auxiliares de simulacao                        *
*                                                                       *
*                                                                       *
*************************************************************************
*/

#include        "dos.h"
#include        "safunc.h"
#include"queue.h"
#include"varext.h"

#include        "func.h"



void env_msg_sinc (t_event)

unsigned long t_event;

{

ecbtx.frag_cnt = 1;
ecbtx.frags [0] = 8;                    /* tamanho do dados */
getofseg ((char *) &msg_tx, &ecbtx.frags [1], &ecbtx.frags [2]);
ecbtx.src_socket = MEU_SOCK;
ecbtx.dst_net = 0;
ecbtx.dst_node = SINC_ADDR;
ecbtx.dst_socket = 0x220;
ecbtx.esr_off = ecbtx.esr_seg = 0;
msg_tx.ctl = 2;
msg_tx.tp = 0;
msg_tx.dado1 = *(int *)&t_event;
msg_tx.dado2 = *(int *)((int *)&t_event + 1);

txmsg(&ecbtx);                      /* envia mensagem as outras estacoes */
}

int t_wait()
{

 return (wait_time*(int)(pow((double)2,(double)collision_count)*((float)rand()/2147483647)));
}

int t_rec_msg()
{

 return (((msg_rx_addr->len + HEAD_TAM + PREAM_LEN)* 8) / BUS_RATE);
}
```

```
int t_proc_usu1()
{

  return ((t_gera*log((double)rand()/2147483647)));
}

int t_proc_usu2()
{

  return ((t_resp*log((double)rand()/2147483647)));
}

void put_fila_tx()
{
if (((put_tx + 1) & 0x1f) != get_tx)
{
  fila_tx[put_tx] = &buf_tx;
  put_tx = put_tx++ & 0x1f;
}
}


void get_fila_tx()
{
get_tx = get_tx++ & 0x1f;
}


int bus_idle()
{

ecbtx.frag_cnt = 1;
ecbtx.frags [0] = 8;                    /* tamanho do dados */
getofseg ((char *) &msg_tx, &ecbtx.frags [1], &ecbtx.frags [2]);
ecbtx.src_socket = MEU_SOCK;
ecbtx.dst_net = 0;
ecbtx.dst_node = SIMUL_ADDR;
ecbtx.dst_socket = 0x221;
ecbtx.esr_off = ecbtx.esr_seg = 0;
msg_tx.ctl = 2;
msg_tx.tp = 1;
msg_tx.dado1 = RESP_REQ_TRANS;
msg_tx.dado2 = MEU_NO;

txmsg(&ecbtx);                      /* envia mensagem as outras estacoes */

await ((unsigned *)&exc_info,0);

if (bus_state == IDLE)
  return (TRUE);
else return (FALSE);

}

void sched_event(event,event_time)
int event;
unsigned long event_time;
```

```
{
struct event_entry *apont_list_event;
struct event_entry *aux_apont,*ant_apont;

if (apont_free != NIL)
  apont_list_event = apont_free;
  apont_free = apont_free->next;

aux_apont = event_list;
ant_apont = NIL;
while (aux_apont != NIL && aux_apont->time<=event_time)
{
 ant_apont = aux_apont;
 aux_apont = aux_apont->next;
}

if (aux_apont != NIL)
{
  if (ant_apont == NIL)
  {
    event_list = apont_list_event;
    event_list->next = aux_apont;
    aux_apont = event_list;
  }
  else
  {
    ant_apont->next = apont_list_event;
    ant_apont->next->next = aux_apont;
    aux_apont=ant_apont->next;
  }
}
else
{
  if (ant_apont == NIL)
  {
    event_list = apont_list_event;
    aux_apont = event_list;
    aux_apont->next = NIL;
  }
  else
  {
    ant_apont->next = apont_list_event;
    aux_apont=ant_apont->next;
    aux_apont->next = NIL;
  }
}
aux_apont->event=event;
aux_apont->time=event_time;
}

void cancel_event(event)
int event;

{
struct event_entry *aux_apont,*ant_apont;

aux_apont = event_list;
ant_apont = NIL;
while (aux_apont != NIL && aux_apont->event != event)
```

```
{
 ant_apont = aux_apont;
 aux_apont = aux_apont->next;
}

if (aux_apont != NIL)
{
  if (ant_apont != NIL)
    ant_apont->next=aux_apont->next;
  else
    event_list=aux_apont->next;
  libera(aux_apont);
}

}

void libera(apont_event)

struct event_entry *apont_event;
{
 struct event_entry *aux_apont;
 struct event_entry *ant_apont;
 struct event_entry *sal_apont;

 sal_apont = apont_event;

 aux_apont = apont_free;
 if (aux_apont == NIL)
 {
   apont_free = sal_apont;
   sal_apont->next = NIL;
 }
 else
 {
   ant_apont = aux_apont;
   aux_apont = aux_apont->next;
   while (aux_apont != NIL)
   {
    ant_apont = aux_apont;
    aux_apont = aux_apont->next;
   }
   ant_apont->next = sal_apont;
   sal_apont->next = NIL;
 }
}

void sched_event_ext(event,event_time)
int event;
unsigned long event_time;

{
struct event_entry *apont_list_event;
struct event_entry *aux_apont,*ant_apont;

if (apont_free_ext != NIL)
  apont_list_event = apont_free_ext;
  apont_free_ext = apont_free_ext->next;

aux_apont = event_list_ext;
```

```
ant_apont = NIL;
while (aux_apont != NIL && aux_apont->time<=event_time)
{
 ant_apont = aux_apont;
 aux_apont = aux_apont->next;
}

if (aux_apont != NIL)
{
  if (ant_apont == NIL)
  {
    event_list_ext = apont_list_event;
    event_list_ext->next = aux_apont;
    aux_apont = event_list_ext;
  }
  else
  {
    ant_apont->next = apont_list_event;
    ant_apont->next->next = aux_apont;
    aux_apont=ant_apont->next;
  }
}
else
{
  if (ant_apont == NIL)
  {
    event_list_ext = apont_list_event;
    aux_apont = event_list_ext;
    aux_apont->next = NIL;
  }
  else
  {
    ant_apont->next = apont_list_event;
    aux_apont=ant_apont->next;
    aux_apont->next = NIL;
  }
}
aux_apont->event=event;
aux_apont->time=event_time;
}

void cancel_event_ext(event)
int event;

{
struct event_entry *aux_apont,*ant_apont;

aux_apont = event_list_ext;
ant_apont = NIL;
while (aux_apont != NIL && aux_apont->event != event)
{
 ant_apont = aux_apont;
 aux_apont = aux_apont->next;
}

if (aux_apont != NIL)
{
  if (ant_apont != NIL)
    ant_apont->next=aux_apont->next;
```

```
  else
    event_list_ext=aux_apont->next;
  libera_ext(aux_apont);
}

}

void libera_ext(apont_event)

struct event_entry *apont_event;
{
 struct event_entry *aux_apont;
 struct event_entry *ant_apont;
 struct event_entry *sal_apont;

 sal_apont = apont_event;

 aux_apont = apont_free_ext;
 if (aux_apont == NIL)
 {
   apont_free_ext = sal_apont;
   sal_apont->next = NIL;
 }
 else
 {
   ant_apont = aux_apont;
   aux_apont = aux_apont->next;
   while (aux_apont != NIL)
   {
    ant_apont = aux_apont;
    aux_apont = aux_apont->next;
   }
   ant_apont->next = sal_apont;
   sal_apont->next = NIL;
 }
}

void gera_msg()
{

buf_tx.tp = 1;           /* msg de request para servidor */
buf_tx.dst = NO_SERV;    /* end do no do servidor */
buf_tx.src = MEU_NO;     /* end do no usuario local */
if (var_len == 1)
{
buf_tx.len = 1000;
memset((char *)buf_tx.dado,0x41,1000);
var_len = 0;
}
else
{
buf_tx.len = 10;
memset((char *)buf_tx.dado,0x31,10);
var_len = 1;
}

}

void start_tx()
```

```
{

ecbtx.frag_cnt = 1;
ecbtx.frags [0] = buf_tx.len + HEAD_TAM;                /* tamanho do dados */
getofseg ((char *) &buf_tx, &ecbtx.frags [1], &ecbtx.frags [2]);
ecbtx.src_socket = MEU_SOCK;
ecbtx.dst_net = 0;
ecbtx.dst_node = SIMUL_ADDR;
ecbtx.dst_socket = 0x221;
ecbtx.esr_off = ecbtx.esr_seg = 0;
buf_tx.msg_ctl = 2;
buf_tx.msg_tp = 1;
buf_tx.msg_info = INICIO_TX;
buf_tx.station = MEU_NO;

txmsg(&ecbtx);                    /* envia mensagem as outras estacoes */

sched_event(FIM_TX,cur_time+(((buf_tx.len + HEAD_TAM + PREAM_LEN)* 8) / BUS_RATE));
sched_event_ext(FIM_TX,cur_time+(((buf_tx.len + HEAD_TAM + PREAM_LEN)* 8) / BUS_RATE));


}

void start_tx1()
{

ecbtx.frag_cnt = 1;
ecbtx.frags [0] = buf_tx.len + HEAD_TAM;                /* tamanho do dados */
getofseg ((char *) &buf_tx, &ecbtx.frags [1], &ecbtx.frags [2]);
ecbtx.src_socket = MEU_SOCK;
ecbtx.dst_net = 0;
ecbtx.dst_node = SIMUL_ADDR;
ecbtx.dst_socket = 0x221;
ecbtx.esr_off = ecbtx.esr_seg = 0;
buf_tx.msg_ctl = 2;
buf_tx.msg_tp = 1;
buf_tx.msg_info = INICIO_TX;
buf_tx.station = MEU_NO;

txmsg(&ecbtx);                    /* envia mensagem as outras estacoes */

sched_event(FIM_TX,cur_time_sis+(((buf_tx.len + HEAD_TAM + PREAM_LEN)* 8) / BUS_RATE));
sched_event_ext(FIM_TX,cur_time_sis+(((buf_tx.len + HEAD_TAM + PREAM_LEN)* 8) / BUS_RATE));


}
```

RXTX.C

```
/*
************************************************************************
*                                                                      *
*                              RXTX.C                                  *
*                                                                      *
*    Descricao :                                                       *
*                Funcoes para envio e recepcao de mensagens do GF      *
*                                                                      *
*                                                                      *
************************************************************************
*/

#include        "dos.h"
#include        "safunc.h"
#include"queue.h"
#include"varext.h"

#include        "func.h"



/* --- SPORX ------------------------------------------------------------

@ SPORX  Coloca uma ECB em recepcao

  Descricao :
               Prepara e dispara uma ECB p/ recepcao de uma mensagem prove-
               niente da rede. O numero de "socket fonte" utilizado sera o
               do Processo Principal

  Entradas :
               Nenhuma

  Saida :
               Nenhuma

  Obs :
               O ECB e' preparado para receber mensagens de qualquer pro-
               cesso.

------------------------------------------------------------------------ */

sporx ()

{
  int i;
  struct ECB_DESC *aux_ecb_ptr;

  aux_ecb_ptr = first_ecb;

  for (i = 0; i < num_ecb; i++)
     sporx1 (aux_ecb_ptr++, i);

} /* FIM SPORX */
```

```
/* --- SPORX1 ------------------------------------------------------------

@ SPORX1  Coloca uma ECB em recepcao

  Descricao :
                Prepara e dispara uma ECB p/ recepcao de uma mensagem prove-
                niente da rede. O numero de "socket fonte" utilizado sera o
                do Processo Principal

  Entradas :
                Nenhuma

  Saida :
                Nenhuma

  Obs :
                O ECB e' preparado para receber mensagens de qualquer pro-
                cesso.

------------------------------------------------------------------------- */

sporx1 (ecb, i)

struct ECB_DESC *ecb;
int i;

{
        /* monta endereco da rotina de servico */

  getofseg ((char *) rx_ssr, &ecb->esr_off, &ecb->esr_seg);

        /* numero maximo de bytes esperados */

  ecb->len = TAM_BUF_REQ;

  ecb->dst_net    = 0x0000;
  ecb->dst_node   = 0x0000;
  ecb->dst_socket = 0x0000;

        /* dados sobre a area de recepcao de mensagem */

  ecb->frag_cnt = 1;
  ecb->frags[0] = TAM_BUF_REQ;

  /* Obtem segmento da area de dados */
  getofseg ((char *) &first_ecb, &ecb->frags [1], &ecb->frags [2]);

  /* Obtem offset da area de dados */
  ecb->frags [1] = (unsigned int)(first_msg) + i * TAM_BUF_REQ;

        /* dispara recepcao */

  rxmsg (ecb);

} /* FIM SPORX1 */
```

```
/* --- LIBERA_ECB ----------------------------------------------------------

@ LIBERA_ECB  Libera a ECB em uso

  Descricao : Tira a ECB que esta' sendo tratada da fila das ECBs a serem
              tratadas e a coloca no inicio da fila das ECBs livres (dispo-
              niveis ao uso)

  Entradas :
                Nenhuma

  Saida :
                Nenhuma

  Obs :
                Este tratamento tambem e' feito na rotina de servico de
                transmissao.  Temos esta rotina para rotinas que nao tem
                tx_resp no final das mesmas.

------------------------------------------------------------------------- */

libera_ecb ()

{
  int i;


  struct ECB_DESC *aux_ecb_ptr;

  /* Tira da fila de ECBs a serem tratadas */

  aux_ecb_ptr             = first_msg_ecb;
  (int) first_msg_ecb     = * (int *) aux_ecb_ptr;

  if (aux_ecb_ptr == last_msg_ecb)  /* Se so' houver 1 msg, first e last */
     last_msg_ecb = first_msg_ecb;  /* devem continuar apontando para o  */
                                    /* mesmo lugar...                    */

  /* Calcula o indice da ECB */

  i = ((int) aux_ecb_ptr - (int) first_ecb)/(sizeof (struct ECB_DESC));
  sporx1 (aux_ecb_ptr, i);

} /* FIM LIBERA_ECB */



/* --- TX_RESP -------------------------------------------------------------

@ TX_RESP Transmite resposta do SI

  Descricao :
                Finaliza os detalhes de uma resposta a um comando recebido
                e despacha a mensagem

  Entradas :
                int     i       Indice do ECB a ser enviado
                char    cod     Codigo da resposta
                int     q       Numero de caracteres no buffer de resposta
```

```
  Saida :
                Nenhuma

  Obs :

------------------------------------------------------------------------ */

tx_resp(msg, cod, q)

   char *msg ;
   char cod ;
   int  q ;

{
   struct ECB_DESC *aux_ecb_ptr;


        /* Tira da fila de ECBs a serem tratadas - vamos coloca'-la em tx */

   aux_ecb_ptr         = first_msg_ecb;
   if (first_msg_ecb == last_msg_ecb)
      (int) last_msg_ecb = * (int *) aux_ecb_ptr;
   (int) first_msg_ecb = * (int *) aux_ecb_ptr;

        /* assinala final do buffer de transmissao */

   *msg = cod;
   *(msg + q) = FIM_MSG;
   *(msg + q + 1) = '\x00';

        /* Monta endereco da rotina de servico de tx */

   getofseg ((char *) tx_ssr, &aux_ecb_ptr->esr_off, &aux_ecb_ptr->esr_seg);

        /* assinala o tamanho da mensagem transportada pelo ECB */

   aux_ecb_ptr->frags[0] = q + 2;

        /* Envia a mensagem */

   txmsg (aux_ecb_ptr);

}  /* FIM TX_RESP */




/* --- TX_SI ------------------------------------------------------------

@ TX_SI   Transmite pedido ao SI

  Descricao :
               Emite pedido que deve ser atendido para o SI que trabalha
              no papel exclusivo de consumidor

  Entradas :
               struct ENDEST  *end            Endereco do SI consumidor
```

```
                    char  *msg            Pedido
                unsigned int    tam        tamanho do buffer do pedido

  Saida :


  Obs :         Tratamento de excecao a ser feito na transmissao de um
                pedido para o consumidor SI:

                Se NAO receber confirmacao da chegada do pedido ao SI (apos
                um periodo de time-out a ser estabelecido), o pedido que
                esta' na fila deve ser remarcado como livre (a ser atendido)
                e o SI consumidor como inoperante (de repente podemos exclui-
                lo.

------------------------------------------------------------------------ */

unsigned char tx_si (end, msg, tam)

   struct ENDEST *end;
   char *msg;
   unsigned int tam;

{
   /* Coloca ecb em recepcao para ler a resposta do SI */

   signal_resp_si = 0;

   getofseg( (char *) rot_resp_si, &ecbrx.esr_off, &ecbrx.esr_seg);
   ecbrx.len = 3;
   ecbrx.src_socket = 0x221;
   memcpy ((char *) &ecbrx.dst_net, (char *) &end->dst_net, 6);
   ecbrx.frag_cnt = 1;
   ecbrx.frags[0] = 3;
   getofseg( (char *) resp_si, &ecbrx.frags[1], &ecbrx.frags[2]);

   /* dispara recepcao */

   rxmsg(&ecbrx);

   /* Assinala o tamanho da mensagem transportada pela ECB */

   ecbtx.frag_cnt   = 1;
   ecbtx.frags [0]  = tam;
   getofseg ((char *) msg, &ecbtx.frags [1], &ecbtx.frags [2]);
   memcpy ((char *) &ecbtx.dst_net, (char *) &end->dst_net, 6);
   ecbtx.esr_off = ecbtx.esr_seg = 0;
   ecbtx.src_socket = 0x221;

   txmsg (&ecbtx);

   /* espero a resposta ou o time-out da temporizacao */

   await (&signal_resp_si, 100);

   if (ecbrx.in_use)
   {
       /* Time-out: SI nao responde ==> aborta a recepcao */
```

```
      abort_rx ();
      return (X_TIME_OUT);
    }
    else
      return (resp_si [0]);

}  /* FIM TX_SI */



/* --- GETOFSEG ------------------------------------------------------------

@ GETOFSEG Pega Offset e Segmento

  Descricao :
               Obtem offset e segmento de um dado na memoria e armazena em
               ENDOFF e ENDSEG respectivamente

  Entradas :
               char far   *p           variavel a ser desmembrada
               unsigned   *endoff      area p/ colocar offset
               unsigned   *endseg      area p/ colocar segmento

  Saida :
               Nenhuma

  Obs :
               P deve ser um "far pointer" para char. Desta maneira a funcao
               deve ser declarada da seguinte forma:

               void getofseg(char far *, unsigned, unsigned);

------------------------------------------------------------------------- */

void getofseg(p,endoff,endseg)

  char far *p;
  unsigned *endoff;
  unsigned *endseg;

{
  *endoff = FP_OFF(p);
  *endseg = FP_SEG(p);

} /* FIM GETOFSEG */



void send_msg_time (t_event,sock)

unsigned long t_event;
int sock;

{

ecbtx.frag_cnt = 1;
ecbtx.frags [0] = 8;                    /* tamanho do dados */
getofseg ((char *) &msg_tx, &ecbtx.frags [1], &ecbtx.frags [2]);
ecbtx.src_socket = 0x220;
```

```
ecbtx.dst_net = 0;
ecbtx.dst_node = 0xffff;              /* envia broadcast */
ecbtx.dst_socket = sock;
ecbtx.esr_off = ecbtx.esr_seg = 0;
msg_tx.ctl = 0;
msg_tx.tp = 0;
msg_tx.dado1 = *(int *)&t_event;
msg_tx.dado2 = *(int *)((int *)&t_event + 1);

txmsg(&ecbtx);                        /* envia mensagem as outras estacoes */

}

void env_msg_info ()

{

ecbtx.frag_cnt = 1;
ecbtx.frags [0] = 8;                  /* tamanho do dados */
getofseg ((char *) &msg_tx, &ecbtx.frags [1], &ecbtx.frags [2]);
ecbtx.src_socket = 0x220;
ecbtx.dst_net = 0;
ecbtx.dst_node = SIMUL_ADDR;
ecbtx.dst_socket = 0x221;
ecbtx.esr_off = ecbtx.esr_seg = 0;
msg_tx.ctl = 0;
msg_tx.tp = 1;
msg_tx.dado1 = INFO_TIME;
msg_tx.dado2 = 0;

txmsg(&ecbtx);                        /* envia mensagem as outras estacoes */

}
```

FUNC.H

```
/* AMPBASIC.C */

      void tst_await      (int);
       int n_autentic       (char *, char *);
       int n_desassoc       ();
       int n_dupassoc       ();
       int tst_ret          (union RXDAT *);
       int monta_rx         (char *, unsigned, union RXDAT *);
       int monta_tx         (char *, unsigned, union RXDAT *);
       int can_ecb          ();
      void gera_ocorrencia  (unsigned int);

/* SCHED.C */

      void env_msg_sinc(unsigned long );
       int t_wait(void);
       int t_rec_msg(void);
       int t_proc_usu1(void);
       int t_proc_usu2(void);
      void put_fila_tx(void);
      void get_fila_tx(void);
       int bus_idle(void);
      void sched_event(int ,unsigned long );
      void cancel_event(int );
      void libera(struct event_entry * );
      void sched_event_ext(int ,unsigned long );
      void cancel_event_ext(int );
      void libera_ext(struct event_entry * );
      void gera_msg(void);
      void start_tx(void);
      void start_tx1(void);

/* SPOINFO.C */

       int  main           (int, char *[]);
       int  le_conf        ();
       void proc_principal ();
       void pp             ();
       void serv_no        (int *);
       void ini_var        ();

/* RXTX.C */

       int sporx          ();
       int sporx1         (struct ECB_DESC *, int);
       int libera_ecb     ();
       int tx_resp        (char *, char, int);
unsigned char tx_si       (struct ENDEST *, char *, unsigned int);
      void getofseg       (char far *, unsigned * , unsigned *);
      void send_msg_time  (int);


/* SSR.ASM */
```

```
       int rx_ssr          ();
       int tx_ssr          ();
       int esp_ssr         ();
       int get_ds          ();
       int get_cs          ();
       int rot_resp_si     ();
       int get_ad_loc      ();
       int int_2a          ();
       int libera_msg      ();
       int sai             ();


/* INSTALL.ASM */

      char install         (void (*) (), unsigned int, char);
       int residente       ();
       int aloca_area      (int);
       int get_dos         ();
       int free_dos        ();
       int get_my_psp      ();

/* AMPBASIC.ASM */

         int drive_local    ();
         int open_sok       (char *);
        void close_sok      ();
        void txmsg          (struct ECB_DESC *);
        void rxmsg          (struct ECB_DESC *);
         int name           (char *, char *);
        void await          (unsigned *, unsigned);
        void signal         (unsigned *);
        void abort_rx       ();
         int com_ocorr      (unsigned int);
         int inst_par       (char *);
         int vrf_sub        (int, int);

/* LIBCGF.ASM */

         int int_21h        ();
         int get_int_dos    ();
         int creat   ();
         int open    (char *, unsigned int);
         int lseek   (int, unsigned int, unsigned int);
         int read    (int, char *, unsigned int);
         int write   (int, char *, unsigned int);
         int dup_handle (int);
         int close   (int);
         int strlen  (char *);
         int strchr  (char *, char);
         int strcpy  (char *, char *);
        void get_dos_date();
         int memset  (char *, char , int);
         int memcpy  (char *, char *, int);
         int puts    (char *);
         int atoi    (char *);
         int memcmp  (char *, char *, int);
        void get_hora_i();
        void get_hora_f();
```

```
/* Arquivo "EVENTOS.C" */

         int tst_net_req        ();
        void det_req_cmd        (char *);
        void trata_pedido_after ();
        void c_inclui           (char *);
        void c_status           (char *);
        void c_exc_bloq_desbloq (char *);
        void c_reordena         (char *);
        void c_altparms         (char *);
        void c_blq_lib_fila     (char *);
        void c_assocnsmr        (char *);
struct CNSMR *le_tabela_si      (char);
        void c_statuscnsmr      (char *);
        void c_desatcnsmr       (char *);
        void c_polling          (char *);
        void c_arqcol           (char *);
        void c_impexcl          (char *);
        void c_grupos           (char *);
        void c_printer_control  (char *);

/* Arquivo "FILA.C" */

struct FILA *poe_na_fila        ();
        void retira_pedido      (struct FILA *, struct FILA *);
        char tira_da_fila       ();
        char bloq_desbloq       (char);
        char reordena           ();
        char altera_param       (char *, int *);
        void libera_pedido_ant  (char, struct FILA *);
struct FILA  *aloca_sol         (struct FILA *);
unsigned int conta_fila         ();
struct FILA  *req_search        (struct CNSMR *, struct FILA *);
struct CNSMR *cnsmr_search      (char, struct CNSMR *);
        void aborta_impres      (struct CNSMR *, struct FILA *);
        void tenta_disparar_1   (struct FILA *);
        void tenta_disparar_2   (struct CNSMR *);
         int dispara            (struct CNSMR *, struct FILA *);


/* Arquivo "ARQFILA.C" */

         int le_back          ();
        void atualiza_header  ();
        void seek_buffer      (struct FILA *);
        void atualiza_buffer  ();
        void le_buffer        (unsigned int, char *);
        void pega_data_hora   ();

/* Arquivo "FGERAIS.C" */

         void obtem_usario     (char *);
unsigned char calcula_cksum    ();
 unsigned int comp_exec_key    (struct FILA *);
 unsigned int comp_grupo_si    ();
 unsigned int comp_usr_id      ();
 unsigned int comp_n_ord       (struct FILA *);
```

```
 unsigned int cnsmr_req_match  (char, char);
struct CNSMR *tira_si          (struct CNSMR *);
         void marca_si_ruim    (struct CNSMR *);
         char prep_buf_pedido  (char *, unsigned int *);
          void prep_header     (char *);
struct CNSMR *procura_si       (struct CNSMR *,char);
         char prep_buf_si      (char *, struct CNSMR *, unsigned int *);
         char freeze_lib_si    (struct CNSMR *, char, unsigned char *);
         char desat_reat_si    (struct CNSMR *, char);
         char mostra_grupos    (char *, unsigned int *);
          int get_exec_key     (char, char *);
         char procura_exec_key (char *);
         char prep_msg_redir   (char *);
          int assoc_rasc       ();
unsigned long conv_data        ();
```

VARGLOB.H

```
/*
************************************************************************
*                                                                      *
*                              VARGLOB.H                               *
*                                                                      *
*                                                                      *
*                 Contem as variaveis globais do SI                    *
*                                                                      *
*                                                                      *
************************************************************************
*/

/*
 *    Descritor da RAT (Request Allocation Table) da fila gerenciada pelo GF
 */

 struct RAT rat;

 struct FILA *fim_fila;

/* estrutura auxiliar do pedido -> o que esta' sendo */
/* manipulado                                        */

/*  struct MSG_REQ pedido_raw [MAXMSG]; */
 struct REQUEST pedido;

/* variaveis extraidas do buffer pedido_raw */

 unsigned char del_prior_status;
 unsigned char exec_key;
 unsigned long tam_arq;
 unsigned long hora_limite;
 unsigned int  num_bytes_buf;
 unsigned long n_ordem;
 unsigned char privilegio;

 struct DESCR_USUARIO usuario;

 char nome_grupo [9];          /* nome_grupo e formulario definem a exec_key */
 char formulario;
 unsigned char data_corrente [6];

 unsigned char n_ped_ord;      /* Variaveis a serem utilizadas na funcao     */
 unsigned char posicao;        /* c_reordena.                                */

 unsigned int  num_grupos;     /* Variavel que contem o numero atual de      */
                               /* grupos cadastrados                         */
 unsigned int  contador_fila;  /* Variavel que contem o numero atual de      */
                               /* pedidos na fila                            */
 unsigned long req_id;         /* Variavel que identificara' a requisicao    */
                               /* dentro da fila                             */

 /* Menor hora dos pedidos PRINT_AFTER       */
 unsigned long hora_print_after = SEM_HORA_LIMITE;
```

```
/*   Variaveis de manipulacao do arquivo-fila gerenciado pelo GF          */

char *arquivo_fila = "FILA.GF";
int  handle_gf;
char gf_addr [4];

/*   Tabela dos SIs consumidores na rede                                  */

struct CNSMR *first_free_si;
struct CNSMR *first_si;
struct CNSMR *last_si;

unsigned int num_si;
unsigned int num_si_livres;

unsigned int num_pool_si;

/*   Tabela dos grupos de SI consumidores na rede                         */

struct GRUPOS_SI *pool;

struct TRABALHO pool_trab;

/*    Variaveis do GET_DOS e FREE_DOS                                     */

  int psp_antigo;
  int break_antigo;

/*    Variaveis do SALVA E RECUPERA CONTEXTO                              */

   int my_psp;

int RAX;
int RBX;
int RCX;
char *RDX;
char *RSI;
char *RDI;
char *RES;

/* +++  variaveis para troca de "signals" entre os processos  +++ */

int exchgr = 0;                         /* entre o PP e a rede */
int sigcnt = 0;                         /* Numero de signals pendentes */

/* +++  area de controle de mensagens transmitidas e recebidas  +++ */

struct ECB_DESC *first_ecb;             /* Ponteiro para o inicio das ECBs */
char *first_msg;            /* buffers */
struct ECB_DESC *first_msg_ecb;
struct ECB_DESC *last_msg_ecb;

/* Variaveis utilizadas na autenticacao da area de rascunho */

struct ASSOCIACAO  old_assoc, assoc_dup;

/* +++ area de controle de mensagens de resposta dos SIs +++       */
```

```
unsigned int esp_exc;              /* exchange para recepcao de msg do SA    */
struct   ECB_DESC ecbrx;           /* ecb de comunicacao com o SA e SI       */
struct   ECB_DESC ecbtx;           /* ecb de comunicacao com o SA e SI       */
int signal_resp_si;                /* signal entre GF e SI                   */
char resp_si [3];                  /* buffer de rx vindo do SI = 3 bytes     */

/* +++ variaveis para controle da instalacao do processo em background +++ */

extern unsigned _asizds;                  /* tamanho do programa em bytes */
       unsigned numpar;                /* tamanho do programa em paragrafos */
       unsigned _atopsp;


/* Indica se ja foi gerada a ocorrencia de erro nos arquivos de config. */
unsigned char enviou_erro_config = FALSE;

int backg = 0;                /* Flag para instalacao em foreground (0) ou
                                                        background (1)    */
int sa_instalado = 1;         /* flag que indica se o SA esta' (1) ou
                                                        nao (0) instalado */

int polling = 1;              /* Flag para realizacao de polling (0) nao faz
                                                              (1) faz   */
int num_ecb = 5;

int max_cnsmr = 10;

unsigned long hora;           /* Variaveis de controle da hora do polling */
unsigned long hora_inicial;

/* tabela usada pala set date */
unsigned tab_dia[12] = { 0,31,59,90,120,151,181,212,243,273,304,334 } ;


#define         NIL             0xffff

#define         SINC_ADDR       0xf
#define         SIMUL_ADDR      0xf

#define         HEAD_TAM16
#define         PREAM_LEN       8

#define         BUS_RATE10          /* Mbps */

#define         MEU_SOCK0x223
#define         MEU_NO          1

#define         NO_SERV         0

#define         PK_USU          0
#define         PK_TRANSP       1
#define         PK_REDE         2
#define         RESP_USU3
#define         RESP_TRANSP     4
#define         RESP_REDE       5
#define         INICIO_TX       6
#define         FIM_TX          7
#define         FIM_WAIT8
```

```
#define         TRANSM_IDLE     9
#define         BUS_LIVRE       10
#define         RESP_REQ_TRANS  11
#define         COLISAO_DET     12
#define         INICIO_RECEP    13
#define         FIM_RECEP       14
#define         ERRO_RECEP      15

#define         IDLE            0
#define         PROC_TX         1
#define         PROC_RX         2
#define         TRANSMITINDO    3
#define         WAIT_EOC 4
#define         HOLD            5
#define         WAIT_RANDOM     6
#define         RECEBENDO       7

#define         MSG_SINC 0
#define         MSG_INFO 1

#define         N_MAX_EVENT     10
#define         N_TOTAL_EV      12

#define         t_gera          1
#define         t_resp          2
#define         t_proc_transp   300
#define         t_proc_rede     120
#define         t_proc_enlace   150

#define         holding_time    10

#define         wait_time       55

struct  exc_info {
 unsigned char b1;
 unsigned char b2;
} exc_info;

int recoloc;

int ecb_end_lib,ecb_end;

int bus_state;

int collision_count;

unsigned long cur_time;

unsigned long cur_time_sis;

int event;

int var_len;

int state,state_tx,state_rx;

int station_state,tx_station_state,rx_station_state;

int previous_state,previous_tx_state,previous_rx_state;
```

```
struct event_entry {
  int              event;
  unsigned long    time;
  struct event_entry *next;
};

struct event_entry *event_list;

struct event_entry *apont_free;

struct event_entry estr_evento[N_MAX_EVENT];

struct event_entry *event_list_ext;

struct event_entry *apont_free_ext;

struct event_entry estr_evento_ext[N_MAX_EVENT];

int fila_tx [16];

int put_tx;
int get_tx;

struct buf_tx {
  int msg_ctl;
  int msg_tp;
  int msg_info;
  int station;
  int tp;
  int dst;
  int src;
  int len;
  char dado[1000];
};

struct buf_tx buf_tx;

struct buf_tx *msg_rx_addr;

struct msg_tx {
 unsigned int ctl;
 unsigned int tp;
 unsigned int dado1;
 unsigned int dado2;
} msg_tx;

struct tab_tx_temp {
 unsigned long t_g;
 unsigned long t_tx;
 unsigned long t_f_tx;
};

struct tab_rx_temp {
 unsigned long t_rx;
 unsigned long t_f_rx;
 unsigned long t_r;
};
```

```
struct tab_tx_temp tx_temp[N_TOTAL_EV];
struct tab_rx_temp rx_temp[N_TOTAL_EV];
```

VAREXT.H

```
/*
************************************************************************
*                                                                      *
*                              VAREXT.H                                *
*                                                                      *
*                                                                      *
*          Contem as variaveis globais do SI   (externas)              *
*                                                                      *
*                                                                      *
************************************************************************
*/

/*
 *    Descritor da RAT (Request Allocation Table) da fila gerenciada pelo GF
 */

 extern struct RAT rat;

 extern struct FILA *fim_fila;

/*  extern struct MSG_REQ pedido_raw [MAXMSG]; */
 extern struct REQUEST pedido;

/* variaveis extraidas do buffer pedido_raw */

 extern unsigned char del_prior_status;
 extern unsigned char exec_key;
 extern unsigned long tam_arq;
 extern unsigned long hora_limite;
 extern unsigned int  num_bytes_buf;
 extern unsigned long n_ordem;
 extern unsigned char privilegio;

 extern struct DESCR_USUARIO usuario;

 extern char nome_grupo [9];
 extern char formulario;
 extern unsigned char data_corrente [6];

 extern unsigned char n_ped_ord;
 extern unsigned char posicao;

 extern unsigned int  num_grupos;

 extern unsigned int contador_fila;
 extern unsigned long req_id;
 extern unsigned long hora_print_after;


 /*   Variaveis de manipulacao do arquivo-fila gerenciado pelo GF          */

 extern char *arquivo_fila;
 extern int  handle_gf;
 extern char gf_addr [4];
```

```
/*   Tabela dos SIs consumidores na rede                                    */

extern struct CNSMR *first_free_si;
extern struct CNSMR *first_si;
extern struct CNSMR *last_si;

extern unsigned int num_si;
extern unsigned int num_si_livres;

extern unsigned int num_pool_si;

/*   Tabela dos grupos de SI consumidores na rede                           */

extern struct GRUPOS_SI *pool;

extern struct TRABALHO pool_trab;

/*    Variaveis do GET_DOS e FREE_DOS                                       */

  extern int psp_antigo;
  extern int break_antigo;

/*    Variaveis do SALVA E RECUPERA CONTEXTO                                */

   extern int my_psp;

extern int RAX;
extern int RBX;
extern int RCX;
extern char *RDX;
extern char *RSI;
extern char *RDI;
extern char *RES;

/* +++  variaveis para troca de "signals" entre os processos  +++ */

extern int exchgr;                              /* entre o PP e a rede */
extern int sigcnt;                         /* Numero de signals pendentes */

/* +++  area de controle de mensagens transmitidas e recebidas  +++ */

extern struct ECB_DESC *first_ecb;
extern char *first_msg;            /* buffers */
extern struct ECB_DESC *first_msg_ecb;
extern struct ECB_DESC *last_msg_ecb;

/* Variaveis utilizadas na autenticacao da area de rascunho */

extern struct ASSOCIACAO  old_assoc, assoc_dup;

/* +++ area de controle de mensagens de resposta dos SIs +++       */

extern unsigned int esp_exc;         /* exchange para recepcao de msg do SA  */
extern struct ECB_DESC ecbrx;        /* ecb de comunicacao com o SA e SI     */
extern struct ECB_DESC ecbtx;        /* ecb de comunicacao com o SA e SI     */
extern int    signal_resp_si;        /* signal entre GF e SI                 */
extern char   resp_si [3];           /* buffer de rx vindo do SI = 3 bytes   */
```

```
/* +++ variaveis para controle da instalacao do processo em background +++ */

extern unsigned _asizds;                    /* tamanho do programa em bytes */
extern unsigned numpar;                  /* tamanho do programa em paragrafos */
extern unsigned _atopsp;

/* Indica se ja foi gerada a ocorrencia de erro nos arquivos de config. */
extern unsigned char enviou_erro_config;

extern int backg;            /* Flag para instalacao em foreground (0) ou
                                                        background (1)    */
extern int sa_instalado;     /* flag que indica se o SA esta' (1) ou
                                                        nao (0) instalado */

extern int polling;          /* Flag para realizacao de polling (0) nao faz
                                                              (1) faz   */
extern int num_ecb;

extern int max_cnsmr;

extern unsigned long hora;
extern unsigned long hora_inicial;

/* tabela usada pala set date */
extern unsigned tab_dia[12];


#define         NIL             0xffff

#define         SINC_ADDR       0xf
#define         SIMUL_ADDR      0xf

#define         HEAD_TAM16
#define         PREAM_LEN       8

#define         BUS_RATE10          /* Mbps */

#define         MEU_SOCK0x223
#define         MEU_NO          1

#define         NO_SERV         0

#define         PK_USU          0
#define         PK_TRANSP       1
#define         PK_REDE         2
#define         RESP_USU3
#define         RESP_TRANSP     4
#define         RESP_REDE       5
#define         INICIO_TX       6
#define         FIM_TX          7
#define         FIM_WAIT8
#define         TRANSM_IDLE     9
#define         BUS_LIVRE       10
#define         RESP_REQ_TRANS  11
#define         COLISAO_DET     12
#define         INICIO_RECEP    13
#define         FIM_RECEP       14
#define         ERRO_RECEP      15
```

```
#define         IDLE            0
#define         PROC_TX         1
#define         PROC_RX         2
#define         TRANSMITINDO    3
#define         WAIT_EOC 4
#define         HOLD            5
#define         WAIT_RANDOM     6
#define         RECEBENDO       7

#define         N_MAX_EVENT     10
#define         N_TOTAL_EV      12

#define         MSG_SINC 0
#define         MSG_INFO 1

#define         t_gera          1
#define         t_resp          2
#define         t_proc_transp   300
#define         t_proc_rede     120
#define         t_proc_enlace   150

#define         holding_time    10

#define         wait_time       55

extern struct   exc_info {
 unsigned char b1;
 unsigned char b2;
} exc_info;

extern int recoloc;

extern int ecb_end_lib,ecb_end;

extern int bus_state;

extern int collision_count;

extern unsigned long cur_time;

extern unsigned long cur_time_sis;

extern int event;

extern int var_len;

extern int state,state_tx,state_rx;

extern int station_state,tx_station_state,rx_station_state;

extern int previous_state,previous_tx_state,previous_rx_state;


struct event_entry {
  int               event;
  unsigned long     time;
  struct event_entry *next;
};
```

```
extern struct event_entry *event_list;

extern struct event_entry *apont_free;

extern struct event_entry estr_evento[N_MAX_EVENT];

extern struct event_entry *event_list_ext;

extern struct event_entry *apont_free_ext;

extern struct event_entry estr_evento_ext[N_MAX_EVENT];

extern int fila_tx [16];

extern int put_tx;
extern int get_tx;

struct buf_tx {
  int msg_ctl;
  int msg_tp;
  int msg_info;
  int station;
  int tp;
  int dst;
  int src;
  int len;
  char dado[1000];
};

extern struct buf_tx buf_tx;

extern struct buf_tx *msg_rx_addr;

extern struct msg_tx {
 unsigned int ctl;
 unsigned int tp;
 unsigned int dado1;
 unsigned int dado2;
} msg_tx;

struct tab_tx_temp {
 unsigned long t_g;
 unsigned long t_tx;
 unsigned long t_f_tx;
};

struct tab_rx_temp {
 unsigned long t_rx;
 unsigned long t_f_rx;
 unsigned long t_r;
};

extern struct tab_tx_temp tx_temp[N_TOTAL_EV];
extern struct tab_rx_temp rx_temp[N_TOTAL_EV];
```

SAFUNC.H

```
/*
************************************************************************
*                                                                      *
*                              SAFUNC.H                                *
*                                                                      *
*       Arquivo a ser incluido quando da manipulacao de funcoes para   *
*       acesso a um Servidor de Arquivos, sem transparencias.          *
*                                                                      *
************************************************************************

*/

/* Numero de sockete arbitrario usado na comunicacao. */

#define         SOCK_GF         0x220
#define         SOCK_ARB 0x221

#define         TEMP_ESP 100     /* tempo de espera p/resposta do S.A. */

#define         MAX_FRAGS  4             /* fragmentos do ECB */

#define         FLG_TMP  0xFFFF          /* flag p/ teste de timeout */

/* Descritor de um Event Control Block ECB */

struct ECB_DESC {

        char            link[4];/* uso do sistema */
        unsigned esr_off,/* ponteiro p/ rotina de servico */
                        esr_seg;
        char            in_use,          /* indicador de ECB em uso */
                        retcod;          /* codigo de retorno */
        int             len;             /* numero de bytes recebidos/transm */
        char            reserv[12];
        int             src_socket;      /* processo que fez requisicao ao SO */
        char            lsn;             /* numero de sessao */
        unsigned dst_net,/* subrede destino */
                        dst_node,        /* no destino */
                        dst_socket;      /* processo destino */
        char            frag_cnt;        /* numero de fragmentos */
        unsigned frags[MAX_FRAGS * 3];
        };


/* descritor de endereco de estacao da rede */

struct ENDEST {
        unsigned int dst_net;
        unsigned int dst_node;
        unsigned int dst_socket;
        };
```

```
struct TXWORD {

        unsigned int ax;
        unsigned int bx;
        unsigned int cx;
        unsigned int dx;
        unsigned int di;
        unsigned int si;
        unsigned char handle;
        unsigned int autl;
        unsigned int auth;
      };


struct TXBYTES {

        unsigned char al,ah;
        unsigned char bl,bh;
        unsigned char cl,ch;
        unsigned char dl,dh;
      };


union TXDAT {

        struct TXWORD x;
        struct TXBYTES h;
      };


struct RXWORD {

        unsigned int ax;
        unsigned int bx;
        unsigned int cx;
        unsigned int dx;
        unsigned int di;
        unsigned int si;
        unsigned int cflag;
      };


struct RXBYTE {

        unsigned char al,ah;
        unsigned char bl,bh;
        unsigned char cl,ch;
        unsigned char dl,dh;
      };


union RXDAT {

        struct RXWORD x;
        struct RXBYTE h;
      };
```

```
struct RXAUX {

        unsigned int ax;
        unsigned int bx;
        unsigned int cx;
        unsigned int dx;
        unsigned int di;
        unsigned int si;
        unsigned int cflag;
        unsigned char hd;
        unsigned int autl;
        unsigned int auth;
      };
```

QUEUE.H

```
/*
************************************************************************
*                                                                      *
*                              QUEUE.H                                 *
*                                                                      *
*              Estrutura de dados do Gerenciador de Fila               *
*                                                                      *
************************************************************************
*/

#define TRUE     1
#define FALSE    0

#define SEM_HORA_LIMITE 0xffffffff

/* Limites */

#define TAM_BUF_REQ       1024          /* tamanho do buffer de requisicoes(600) */
#define MAX_BUF_REQ       10            /* numero de buffers de requisicoes
                                           disponiveis */
#define MAX_FRAGS          4            /* preparado para transmissao de no
                                           maximo 4 fragmentos por ECB */

/* apontador NULO */

#define NULO            0


/* status de um ECB */

#define RX_ECB          2               /* em recepcao */
#define TX_ECB          1               /* em transmissao */
#define LV_ECB          0               /* livre */

/* codigos para controle de protocolo na troca de mensagens */

#define FIM_MSG      'Z'                /* fim de mensagem */
#define FIM_BLC       'X'         /* fim de bloco */

/* Indicam se e uma requisicao de usuario ou administrador de grupo ou geral */

#define USUARIO      '\x00'
#define ADM_GRUPO    '\x40'
#define ADM_GERAL    '\x80'
#define MASC_ID      '\xC0'
#define MASC_PRIV    '\x3F'

/* Indica a ocorrencia que esta sendo cadastrada para o administrador */

#define ERRO_FILA    0x51
#define ERRO_CONFIG  0x52

/*codigos de retorno a uma requisicao */
```

```
#define X_OK            '\x00'          /* comando atendido */
#define X_SERV_AUS      '\xDA'          /* timeout no SA */
#define X_OVER_SOL      '\xC1'          /* excedido numero de solicitacoes */
#define X_TIME_OUT     '\x33'         /* Time-out */
#define X_INV_PAS       '\x56'          /* acesso na permitido */
#define X_ERRO_PAR      '\x57'          /* ausencia de parametros obrigatorios*/
#define X_NCONF_IMP     '\xC3'          /* impressora nao configurada */
#define X_INV_CMD       '\xC4'          /* comando invalido ou nao implementado */
#define X_INV_OP '\xC5'          /* solicitacao de status invalida */
#define X_IMP_EXC       '\xC6'          /* impressora exclusiva */
#define X_FIM_SOL       '\xC7'          /* fim das solicitacoes na fila */
#define X_IMP_OCP       '\xC8'          /* impressora exclusiva ja alocada */
#define X_INV_ACE       '\x89'          /* acesso a solicitacao nao permitido*/
#define X_SOL_AUS       '\xCA'          /* solicitacao ausente */
#define X_IMP_ATV       '\xCB'          /* impressora em atividade */
#define X_SEM_IMP       '\xCC'          /* nao ha impressao em andamento */
#define X_SEM_INT       '\xCD'          /* nao ha impressao interrompida */
#define X_PAG_INV       '\xCE'          /* pagina invalida */
#define X_IMP_CPT       '\xCF'          /* impressora e compartilhavel */
#define X_ARQ_AUS       '\xD0'          /* arquivo nao encontrado */
#define X_INV_FORM      '\xD1'          /* formulario invalido */
#define X_ERRO_IO       '\xD2'          /* erro de io no arquivo de coleta */
#define X_DUP_ARQ       '\xD3'          /* arquivo de coleta ja existe */
#define X_BLQ_DST       '\xD4'          /* imp. bloqueada ou desativada */
#define X_N_INST_IMP '\xD5'             /* associacao na presente */
#define X_NOT_ASSOC     '\xD5'          /* associacao na presente */
#define X_NOT_OPEN     '\xD6'           /* arquivo nao aberto */
#define X_OVER_REDIR    '\xD7'          /* nao ha mais buffer de solicitacao
                                           para continuacao do redireciona/o */
#define X_FALT_NOM      '\xD8'          /* Nome.amp nao encontrado */
#define X_FALT_SA       '\xD9'          /* Nome de SA nao encontrado */


#define EVER         ;;

#define INT_MASK     0x00FF
#define LONG_MASK    0x00FFFFFF

#define MAXSOL       125    /* numero maximo de pedidos          */
#define MAXBUF       256    /* tamanho dos buffers de solicitacao */
#define HEADER_SIZE  512    /* tamanho do header (encadeamento de pedidos */
#define MAXUSER        8
#define MAXGRUPO       8
#define NUMAXGRUPO    31    /* numero maximo de grupos que podem ser catalog.*/
#define BUF_ID_SIZE   37    /* tamanho em bytes de req_id + identificacao do */
                            /* usuario + tamanho do pedido quando armazenado */
                            /* no buffer de requisicao                       */

#define INT_POLLING  270    /* Intervalo para realizacao do polling de ativi-*/
                            /* dade dos SIs                                  */


/*    Comandos do GF                                          */

#define R_INCLUI      0x01
#define R_STATUS      0x02
#define R_EXCLUI      0x03
#define R_REORDENA    0x04
#define R_BLQSOL      0x05
```

```
#define R_LIBSOL      0x06
#define R_ALTEXECKEY  0x07
#define R_ALTPARMS    0x08
#define R_BLQFILA     0x09
#define R_LIBFILA     0x0A
#define R_DSTVFILA    0x0B
#define R_ATVFILA     0x0C
#define R_ASSOCNSMR   0x0D
#define R_STATUSCNSMR 0x0E
#define R_POLLING     0x0F
#define R_ARQCOL      0x10
#define R_IMPEXCL     0x11    /* impressora exclusiva             */
#define R_GRUPOS      0x12    /* mostra os grupos cadastrados     */
#define R_FREEZE      0x13    /* solicita interrupcao de impressao */
#define R_LIBERA      0x14    /* libera impressao "congelada"     */
#define R_DESATIVA    0x15    /* desativa SI apos impress.corrente */
#define R_REATIVA     0x16    /* reativa SI apos impress. corrente */
#define R_DESATCNSMR  0x17    /* status de desativacao do SI      */
#define R_CTRLALTDEL  0xA0    /* broadcast de desassociacao       */

/*    Encadeamento da fila                                        */

#define FIM_FILA     0x0000

/*    Status de fila                                              */

#define OK           0x00
#define HOLD_QUEUE   0x01
/* #define reservado    0x02 */
#define EXCECAO      0x03

/*    Status de pedidos                                           */

#define OK           0x00
/* #define reservado    0x01 */
#define HOLD_REQ     0x02
/* #define reservado    0x03 */
#define GONE         0x04
#define DELETAR      0x05
#define DELETANDO    0x06
#define T_EXPIRADO   0x07
#define BLOQ_ADM     0x08
#define BLOQ_USR     0x09
#define ERRO_CKSUM   0x0A
#define PARALISADO   0x0B
#define PRINT_UNTIL  0x0C
#define PRINT_AFTER  0x0D
/* #define reservado    0x0E */
#define POS_LIVRE    0x0F

/*    Mensagens para controle de impressora                       */

#define MSG_FREEZE   0x03
#define MSG_LIBERA   0x04
#define MSG_DESATIVA 0x05
#define MSG_REATIVA  0x06

/*    Status dos SIs consumidores                                 */
```

```
#define LIVRE        0x00
#define INOPERANTE   0x10
#define OCUPADO      0x20
#define DESATIVADO   0x30
#define FREEZE       0x40
/* #define reservado    0x50 */
/* #define reservado    0x60 */
/* #define reservado    0x70 */
#define EXCL_LIV     0x80
#define EXCL_INOP    0x90
#define EXCL_OCUP    0xA0
#define EXCL_DESAT   0xB0
/* #define reservado    0xC0 */
/* #define reservado    0xD0 */
/* #define reservado    0xE0 */
/* #define reservado    0xF0 */

/*    Chaves de execucao                                          */

#define ANYGROUP     0x00   /* consumivel por qualquer grupo      */
#define ANYONE       0x00   /* consumivel por qualquer consumidor */

#define MAIOR_PRDD   0x00   /* mais prioritario                   */
#define PRDD_DFLT    0x40   /* prioridade default                 */
#define MENOR_PRDD   0x70   /* menos prioritario                  */

#define DAI          0x80   /* Delecao Apos Impressao             */


/*
 *    Estruturas auxiliares a construcao da fila gerenciada pelo GF
 */

 /* descritor da area de recepcao de requisicoes */

 struct REQ_BUF {
        char b[TAM_BUF_REQ];
              };


 struct FILA {
                     struct   FILA  *link;
               unsigned char  del_prior_status;
               unsigned char  chave_execucao;
            };

 struct REQUEST {
               char buffer [MAXBUF];
               };

 struct RAT  {
               unsigned char status_fila;
               unsigned char reservado;
                      struct FILA *inicio_fila;
                        char req_id_bck [3];
                      struct FILA *inicio_fila_vazia;
                      struct FILA *fila_print_after;
                      struct FILA fila [MAXSOL];
               unsigned char check_sum_header;
```

```
            };

struct ASSOCIACAO {
               struct ENDEST end; /* endereco do SA da area de rascunho */
               unsigned char hd;  /* handle da autenticacao             */
               unsigned int autl; /* parte baixa da autenticacao        */
               unsigned int auth; /* parte alta  da autenticacao        */
            };

struct CNSMR {
               struct CNSMR *link;
               struct ENDEST end;          /* endereco do SI */
               unsigned char grupo_cnsmr;
               unsigned char exec_key_req;
               unsigned char estado_si_req;
               struct FILA   *paux_req;
               unsigned char cont_polling;
               unsigned int  percent;
               unsigned char privilegio;
            };

 struct GRUPOS_SI {
               char tipo;
               char nome_pool [9];
               char chave_grupo;
               char end_serv_rasc [6];
               char banner_default [13];
               char path_rasc [9];
            };

 struct TRABALHO {
               char tipo;
               char nome_pool [9];
               char chave_grupo;
               char end_serv_rasc [6];
               char banner_default [13];
               char path_rasc [65];
            };

struct PARAMETROS {
               unsigned char id;
               unsigned int  versao;
               unsigned char num_ecb;
               unsigned char num_si;
               unsigned char max_cnsmr;
            };

struct DESCR_USUARIO { char id;
                       char grupo [MAXGRUPO + 1];
                       char nome [MAXUSER + 1];
                     };
```

AMPBASIC.ASM

```
;
;       Rotinas de impressao do SI
;

        TITLE   servrot

_TEXT   SEGMENT BYTE PUBLIC 'CODE'
_TEXT   ENDS
_DATA   SEGMENT WORD PUBLIC 'DATA'
_DATA   ENDS
_CONST  SEGMENT WORD PUBLIC 'CONST'
_CONST  ENDS
_BSS    SEGMENT WORD PUBLIC 'BSS'
_BSS    ENDS

DGROUP  GROUP   _CONST, _BSS, _DATA
        ASSUME  CS:_TEXT, DS:DGROUP, SS:DGROUP, ES:DGROUP

_DATA   SEGMENT

INF_MINHA_SUBR          EQU             732BH

EXTRN   _ecbrx          :WORD
EXTRN   _sa_instalado   :WORD      ; flag que indica se o SA esta' instalado

_DATA   ENDS

_TEXT   SEGMENT

        PUBLIC  _drive_local
        PUBLIC  _open_sok
        PUBLIC  _close_sok
        PUBLIC  _txmsg
        PUBLIC  _rxmsg
        PUBLIC  _name
        PUBLIC  _await
        PUBLIC  _asignal
        PUBLIC  _abort_rx
        PUBLIC  _com_ocorr
        PUBLIC  _inst_par
        PUBLIC  _vrf_sub

_drive_local    PROC    NEAR

                PUSH    SI
                PUSH    DI
                PUSH    ES

                MOV     AX, 4409H
                MOV     BL, 0
                INT     21H
                MOV     AX, 3
                AND     DX, 1000H
                JZ      LOCAL
```

```
                MOV     AX, 0

LOCAL:          POP     ES
                POP     DI
                POP     SI
                RET

_drive_local    ENDP

_open_sok       PROC    NEAR

                PUSH    BP
                MOV     BP,SP

                PUSH    DI

                MOV     AX,0B800H
                INT     2FH
                CMP     AL, 0
                JZ      SHORT SEM_AMP
                TEST    BX, 8
                JZ      SHORT SEM_AMP

                TEST    BX, 40H                 ; testa se o SA esta' instalado
                JNZ     CONT_OPEN_SOK
                MOV     _sa_instalado, 0

CONT_OPEN_SOK:  MOV     DI, [BP + 4]
                MOV     AX, 730FH
                INT     2AH
                MOV     AX, 0
                JMP     SHORT FIM1
SEM_AMP: MOV    AX, 0FFH
FIM1:
                POP     DI
                POP     BP
                RET

_open_sok       ENDP

_close_sok      PROC    NEAR

                PUSH    BP
                MOV     BP,SP

                PUSH    DI
                PUSH    DS

                MOV     DI, [BP + 4]
                MOV     AX, 7310H
                INT     2AH

                POP     DS
                POP     DI
                POP     BP
                RET

_close_sok      ENDP
```

```
_txmsg  PROC    NEAR

        push    bp
        mov     bp,sp

        push    si
        push    di

        mov     ax,7306h
        mov     di,[bp + 4]
        int     2ah

        pop     di
        pop     si

        pop     bp

        ret

_txmsg  ENDP


_rxmsg  PROC    NEAR

        push    bp
        mov     bp,sp

        push    si
        push    di

        mov     ax,7307h
        mov     di,[bp + 4]
        int     2ah

        pop     di
        pop     si

        pop     bp

        ret

_rxmsg  ENDP

_name   PROC    NEAR

;
;       Converte nome para endereco
;
;       Saida: 1 Nome nao encontrado
;              0 Sucesso
;
        push    bp
        mov     bp,sp

        push    si
        push    di

        mov     ax,7308h
        mov     si,[bp + 4]     ; nome     => DS:SI
```

```
        mov     di,[bp + 6]     ; endereco => ES:DI
        push    ds
        pop     es
        int     2ah

        mov     ah,0

        pop     di
        pop     si

        pop     bp

        ret

_name   ENDP

_await PROC     NEAR

        push    bp
        mov     bp,sp

        push    si
        push    di

        mov     ax,7301h
        mov     bx,[bp + 4]     ; exchange => ES:BX
        push    ds
        pop     es
        mov     cx,[bp + 6]     ; temporizacao para time-out => CX
        int     2ah

        pop     di
        pop     si

        pop     bp

        ret

_await  ENDP

_asignal PROC   NEAR

        push    bp
        mov     bp,sp

        push    si
        push    di

        mov     ax,7302h
        mov     bx,[bp + 4]     ; exchange => ES:BX
        push    ds
        pop     es
        int     2ah

        pop     di
        pop     si

        pop     bp
```

```
        ret

_asignal  ENDP


_abort_rx PROC   NEAR

        push    si
        push    di

        mov     ax,7312h
        mov     di,OFFSET DGROUP:_ecbrx    ; ECB => DS:DI
        int     2ah

        pop     di
        pop     si

        ret

_abort_rx  ENDP


;COMUNICA OCORRENCIA

_com_ocorr      PROC    NEAR

        PUSH    BP
        MOV     BP, SP

        PUSH    AX
        PUSH    BX
        PUSH    CX
        PUSH    DI
        PUSH    SI
        MOV     BX,00H
        MOV     DX,[BP + 4]       ; DL <- NUMERO DA OCORRENCIA
        MOV     DH, 80H
        MOV     CX,02H
        MOV     AX,7336H
        INT     2AH
        POP     SI
        POP     DI
        POP     CX
        POP     BX
        POP     AX
        POP     BP
        RET
_com_ocorr      ENDP


_inst_par  PROC  NEAR

        push    bp
        mov     bp,sp
        push    si

        mov     ax,7324h
        mov     si,[bp + 4]     ; parametros de instalacao em DS:SI
        int     2ah
```

```
        pop     si
        pop     bp

        ret

_inst_par  ENDP

_vrf_sub PROC    NEAR    ; Coloca o # da subrede na otica da estacao requisitante
                         ; PAR1 = Subrede da estacao que vai receber a informacao
                         ; PAR2 = Subrede do elemento mencionado
                         ; Retorna Subrede do elemento mencionado
        PUSH    BP
        MOV     BP,SP
        MOV     CX,[BP + 4]
        MOV     AX,[BP + 6]
        OR      CX,CX           ; Ja' Possui a mesma visao que eu - AX ja' Setado
        JZ      VRFS1
        CMP     AX,CX
        JE      VRFS2
        OR      AX,AX
        JNZ     VRFS1
        MOV     AX,INF_MINHA_SUBR
        XOR     CX,CX
        INT     2AH             ; Retorna em AX a minha Subrede
VRFS1:  POP     BP
        RET
VRFS2:  XOR     AX,AX
        POP     BP
        RET
_vrf_sub ENDP


_TEXT   ENDS
END
```

INSTALL.ASM

```
;
;          Rotina de instalacao de processos desenvolvidos em  "C"
;               Funciona para o compilador Microsoft vs 4.00
;
;           - inclui o processo no sistema e termina a execucao,
;             permanecendo a memoria ocupada residente no DOS.
;
;           - os valores necessarios para o preenchimento do des-
;             critor do processo, sao obtidos em parte  do modulo
;             de inicializacao "CRT0.OBJ", atraves das variaveis pu-
;             blicas _atopsp ( topo da stack do C ), _psp ( offset e
;             segmento do PSP do programa ), _asizds ( tamanho to-
;             tal do programa dado em bytes).
;
;           - para  outras versoes de C , o  segmento do PSP pode
;             ser obtido atraves da funcao 62h do MSDOS, ou  pela
;             funcao  'getpsp()'  definida na biblioteca BIBBAS.
;
;           - a determinacao do tamanho total do programa  depende
;             da forma de alocacao de cada compilador, sendo calcu
;             lado por :
;                          tam = fm_seg - PSP_seg + fm_off/16 + 1
;             sendo :  tam     - tamanho da memoria em paragrafos
;                      PSP_seg - segmento do PSP
;                      fm_seg  - segmento do fim da memoria
;                      fm_off  - offset do fim da memoria ;
;
;           - o final da memoria utilizada pelo programa deve con-
;             siderar inclusive  areas reservadas pelo  compilador
;             para alocacao "dinamica" de memoria.
;
;           - para compatibilizar a ativacao do processo pelo Esca
;             lador com uma chamada normal de procedure em  C , e'
;             necessario inicializar o registro BP com o offset do
;             inicio da stack.
;
;

        TITLE   INSTALA


_TEXT   SEGMENT BYTE PUBLIC 'CODE'
_TEXT   ENDS
_DATA   SEGMENT WORD PUBLIC 'DATA'
_DATA   ENDS
_CONST  SEGMENT WORD PUBLIC 'CONST'
_CONST  ENDS
_BSS    SEGMENT WORD PUBLIC 'BSS'
_BSS    ENDS
STACK   SEGMENT STACK   'STACK'
stack_top       label   word
STACK   ENDS

DGROUP  GROUP   _CONST, _BSS, _DATA
```

```
        ASSUME  CS:_TEXT, DS:DGROUP, SS:DGROUP, ES:DGROUP

_DATA   SEGMENT

EXTRN   _numpar:WORD            ; numero de paragrafos calculado antes da
                                ; chamada da funcao
EXTRN   __atopsp:WORD           ; topo da pilha
EXTRN   __psp:WORD              ; endereco do psp

EXTRN   _psp_antigo:WORD ; variavel para salvar psp
EXTRN   _break_antigo:WORD      ; varialvel para salvar estado do break

EXTRN   _my_psp:WORD           ; psp do GF


                db      128(?)
PILHA           dw      0

IDENT           dw      0
PRIOR           db      0
PSP             dw      0
STACK_OFF       dw      0
STACK_SEG       dw      0
ENTRY_OFF       dw      0
ENTRY_SEG       dw      0
_DS             dw      0
_ES             dw      0
_BP             dw      0
;I24_SEG        dw      0       variavel para salvar end. da int 24
;I24_OFF        dw      0       variavel para salvar end. da int 24
DESCRITOR       equ     IDENT
topo            dw      OFFSET DGROUP : (stack_top)

GET_REC_DOS     equ     7304h
LIB_REC_DOS     equ     7305h
GET_BREAK       equ     3300h
SET_BREAK       equ     3301h
SET_PSP         equ     5000h
GET_PSP         equ     5100h

_DATA   ENDS

_TEXT   SEGMENT

        PUBLIC  _install
        PUBLIC  _residente
        PUBLIC  _aloca_area

        PUBLIC  _get_dos
        PUBLIC  _free_dos

        PUBLIC  _get_my_psp

        EXTRN   _int_21h:NEAR           ; rotina para acesso ao DOS

_install PROC   NEAR

        push    bp
        mov     bp,sp                   ; salva  stack do C
```

```
        mov     ax,cs
        mov     ENTRY_SEG,ax            ;
        mov     bx,[bp+4]               ; offset da rotina de entrada do processo
        mov     ENTRY_OFF,bx            ; inicializa entry-point do processo
        mov     bx,[bp+6]               ; identidade do processo
        mov     IDENT,bx
        mov     al,[bp+8]               ; prioridade do processo
        mov     PRIOR,al

        mov     bx,[__psp]              ; segmento do PSP
        mov     PSP,bx

        mov     ax,ss
        mov     STACK_SEG,ax            ; salva segmento da stack
        mov     bx,[__atopsp]           ; inicia pilha do processo como o
        mov     STACK_OFF,bx            ; topo da pilha do "C"
        mov     _BP,bx                  ; no inicio, BP deve ser igual a SP
        mov     bx,ds
        mov     _DS,bx                  ; completa o DESCRITOR
        mov     bx,es
        mov     _ES,bx
;
        mov     ax,7300h        ; INCLUDE para o processo descrito
        mov     bx,ds                   ; na estrutura apontada por ES:BX
        mov     es,bx
        mov     bx,OFFSET DGROUP:DESCRITOR
        int     2Ah
        test    ah,al
        jnz     ERRO
        pop     bp
        ret
;
ERRO:   cli
        mov     bx,[STACK_SEG]          ; restaura pilha do "C"
        mov     ss,bx
        mov     sp,bp                   ; BP nao foi destruido
        pop     bp
        sti                             ; retorna codigo de erro em AL
        ret

_install ENDP



_residente PROC NEAR
        mov     ax,topo
        mov     cl,4
        shr     ax,cl
        add     ax,15h
        mov     dx,DGROUP
        sub     dx,_TEXT
        add     dx,ax
        mov     ax,3100H
        call    _int_21h        ; Termina processo e mantem residente.
_residente ENDP

_aloca_area     PROC    NEAR

        push    bp
```

```
        mov     bp, sp
        mov     cx, [bp + 4]
        mov     ax, topo
        add     cx, ax
        mov     topo, cx
        pop     bp
        ret
_aloca_area     ENDP


_get_dos PROC NEAR
;
        push    si
        push    di
        push    bp
;
;       aloca recurso MS-DOS
;
        xor     bx,bx
        xor     cx,cx
        mov     ax,GET_REC_DOS
        int     2ah
;
;       pega PSP do foreground e salva
;       coloca a minha PSP (background)
;
        mov     ax,GET_PSP
        call    _int_21h
        mov     DS:[_psp_antigo],bx
        mov     bx,_my_psp
        mov     ax,SET_PSP
        call    _int_21h
;
;       pega, salva e desabilita CTRL-BREAK
;
        mov     ax,GET_BREAK
        call    _int_21h
        mov     DS:[_break_antigo],dx
        mov     dl,0                    ; estado do break: OFF
        mov     ax,SET_BREAK
        call    _int_21h
;
;       salvo endereco da int 24
;
;       push    ds
;       push    si
;
;       mov     bx,0
;       mov     ds,bx
;       mov     si,90h
;       mov     bx,word ptr ds:[si]
;       mov     I24_OFF,bx
;       mov     bx, word ptr ds:[si + 2]
;       mov     I24_SEG,bx
;       mov     word ptr ds:[si], OFFSET  ROT_I24
;       mov     word ptr ds:[si + 2], SEG ROT_I24
;
;       pop     si
;       pop     ds
```

```
;
        pop     bp
        pop     di
        pop     si
        ret
;
_get_dos ENDP

_free_dos PROC NEAR
;
        push    si
        push    di
        push    bp
;
;       salvo endereco da int 24
;
;       push    ds
;       push    si
;
;       mov     bx,0
;       mov     ds,bx
;       mov     si,90h
;       mov     bx, I24_OFF
;       mov     word ptr ds:[si], bx
;       mov     bx, I24_SEG
;       mov     word ptr ds:[si + 2], bx
;
;       pop     si
;       pop     ds
;
;       restaura CTRL-BREAK
;
        mov     dx,DS:[_break_antigo]
        mov     ax,SET_BREAK
        call    _int_21h
;
;       restaura PSP
;
        mov     bx,DS:[_psp_antigo]
        mov     ax,SET_PSP
        call    _int_21h
;
;       libera recurso MS-DOS
;
        xor     bx,bx
        xor     cx,cx
        mov     ax,LIB_REC_DOS
        int     2ah
;
        pop     bp
        pop     di
        pop     si
        ret
;
_free_dos ENDP

;ROT_I24      PROC FAR
;
;       mov     al,3
```

```
;       iret
;
;ROT_I24 ENDP

_get_my_psp PROC NEAR
;
        push    si
        push    di
        push    bp
;
;       obtem a PSP do GF
;
        mov     ah,62h
        call    _int_21h
        mov     DS:[_my_psp], bx
;
;
;
        pop     bp
        pop     di
        pop     si
        ret
;
_get_my_psp ENDP

_TEXT   ENDS

_SEGMEM SEGMENT MEMORY  'MEMORY'
_SEGMEM ENDS


END
```

SSR.ASM

```
;
;       Rotina de servico para o SCI
;
;       Envia Signal para introduzir o processo novamente na fila do escalador
;
;          rx_ssr     => signal quando do recebimento de requisicoes da rede
;          tx_ssr     => coloca a ECB novamente em recepcao
;          esp_ssr    => signal quando do recebimento de respostas do S.A.
;

        TITLE   servrot

_TEXT   SEGMENT BYTE PUBLIC 'CODE'
_TEXT   ENDS
_DATA   SEGMENT WORD PUBLIC 'DATA'
_DATA   ENDS
_CONST  SEGMENT WORD PUBLIC 'CONST'
_CONST  ENDS
_BSS    SEGMENT WORD PUBLIC 'BSS'
_BSS    ENDS

CGROUP  GROUP   _TEXT
DGROUP  GROUP   _CONST, _BSS, _DATA
        ASSUME  CS:_TEXT, DS:DGROUP, SS:DGROUP, ES:DGROUP

_DATA   SEGMENT

EXTRN   _first_ecb     :WORD
EXTRN   _first_msg     :WORD
EXTRN   _exchgr:        WORD
EXTRN   _esp_exc:       WORD
EXTRN   _sigcnt:        WORD
EXTRN   _signal_resp_si:WORD
EXTRN   _gf_addr_aux:   WORD
EXTRN   _first_msg_ecb: WORD
EXTRN   _last_msg_ecb:  WORD
EXTRN   _ecb_end_lib:    WORD
EXTRN   _ecb_end:       WORD
EXTRN   _recoloc:       WORD

EXTRN   _RAX:           WORD
EXTRN   _RBX:           WORD
EXTRN   _RCX:           WORD
EXTRN   _RDX:           WORD
EXTRN   _RSI:           WORD
EXTRN   _RDI:           WORD
EXTRN   _RES:           WORD

_DATA   ENDS

_TEXT   SEGMENT

        PUBLIC  _rx_ssr
        PUBLIC  _tx_ssr
```

```
        PUBLIC  _esp_ssr
        PUBLIC  _get_ds
        PUBLIC  _get_cs
        PUBLIC  _rot_resp_si
        PUBLIC  _get_ad_loc
        PUBLIC  _int2a
        PUBLIC  _libera_msg
        PUBLIC  _sai

EXTRN   _serv_no: near

_rx_ssr PROC    FAR

        ; Coloca ECB com mensagem no final da fila de msg a tratar, verificando
        ; inicialmente se a fila nao esta' vazia

;       mov     ax, DS:_first_msg_ecb
;       cmp     ax, 0
;       jnz     ENCADEIA_MSG
;       mov     ax, di
;       mov     DS:_first_msg_ecb, ax
;
;ENCADEIA_MSG:
;
;       mov     WORD PTR [di], 0
;       mov     ax, DS:_last_msg_ecb
;       cmp     ax, 0
;       jz      short SEGUE
;       mov     bx, DS:_last_msg_ecb
;       mov     WORD PTR [bx], di
;
;SEGUE: mov     ax, di
;       mov     DS:_last_msg_ecb, ax
;
;       inc     _sigcnt                         ; signal
;       mov     ax,7302h
;       mov     bx,DGROUP
;       mov     es,bx
;       mov     bx,OFFSET DGROUP:_exchgr ; ES:BX -> apontador para a
;       int     2Ah                             ;          exchange
;       ret

        mov     _ecb_end,di
        mov     bx,[di + 36]                     ; offset dos dados
        push    di
        push    bx
        call    _serv_no                ; chama rotina de tratamento
        pop     bx
        pop     di
        cmp     _recoloc,0
        je      n_poe_fila

n_trata:                                ; recoloca ECB em recepcao
        mov     ax,OFFSET CGROUP:[_rx_ssr]
        mov     ds:[di + 4],ax
        mov     ds:[di + 6],CS
        mov     word ptr ds:[di + 10],1024         ; TAM_BUF_REQ
        mov     WORD PTR ds:[di + 24],223H
        mov     bx,0
```

```
        mov     ds:[di + 27],bx
        mov     ds:[di + 29],bx
        mov     ds:[di + 31],bx
        mov     bl,01h
        mov     ds:[di + 33],bl
        mov     word ptr ds:[di + 34],1024        ; TAM_BUF_REQ

        mov     bx,3aH                  ; sizeof (struct ECB_DESC)
        mov     ax,di
        sub     ax,_first_ecb
        cwd
        div     bx
        mov     cx,1024                 ; TAM_BUF_REQ
        mul     cx

        mov     bx,_first_msg
        add     bx,ax
        mov     ds:[di + 36],bx         ; offset
        mov     ds:[di + 38],DGROUP

        mov     ax,7307H                ; coloca em rx
        int     2ah

n_poe_fila:
        ret

_rx_ssr ENDP


_tx_ssr PROC    FAR

        ; Quando acabar a tx da ECB, recoloco-a em rx

        mov     ax,OFFSET CGROUP:[_rx_ssr]
        mov     ds:[di + 4],ax
        mov     ds:[di + 6],CS
        mov     ds:[di + 10],600        ; TAM_BUF_REQ
        mov     WORD PTR ds:[di + 24],220H
        mov     bx,0
        mov     ds:[di + 27],bx
        mov     ds:[di + 29],bx
        mov     ds:[di + 31],bx
        mov     bl,01h
        mov     ds:[di + 33],bl
        mov     ds:[di + 34],600        ; TAM_BUF_REQ

        mov     bx,3aH                  ; sizeof (struct ECB_DESC)
        mov     ax,di
        sub     ax,_first_ecb
        cwd
        div     bx
        mov     cx,600                  ; TAM_BUF_REQ
        mul     cx

        mov     bx,_first_msg
        add     bx,ax
        mov     ds:[di + 36],bx         ; offset
        mov     ds:[di + 38],DGROUP
```

```
        mov     ax,7307H                ; coloca em rx
        int     2ah

        ret

_tx_ssr ENDP


_esp_ssr PROC   FAR

        mov     ax,7302h          ; signal para a exchange
        mov     bx,ds
        mov     es,bx
        mov     bx,OFFSET DGROUP:_esp_exc ; apontada por ES:BX
        int     2Ah
        ret

_esp_ssr ENDP

_get_ds PROC    NEAR
        mov     ax, ds
        ret
_get_ds ENDP

_get_cs PROC    NEAR
        mov     ax, cs
        ret
_get_cs ENDP

_rot_resp_si PROC FAR
       mov  bx,offset _signal_resp_si
       mov  ax,seg _signal_resp_si
       mov  es,ax
       mov  ax, 7302H
       int  2AH
       ret
_rot_resp_si ENDP


_get_ad_loc     PROC    NEAR

                PUSH    BX
                PUSH    CX
                PUSH    DX
                PUSH    SI
                PUSH    DI
                PUSH    ES
                PUSH    DS

                MOV     DI, WORD PTR [_gf_addr_aux]
                PUSH    DI
                PUSH    DS

                MOV     AX, 730DH    ; GET ADDR
                INT     2AH

                POP     ES
                POP     DI
```

```
                MOV     CX,4
                CLD
REP             MOVSB

                POP     DS
                POP     ES
                POP     DI
                POP     SI
                POP     DX
                POP     CX
                POP     BX
                RET

_get_ad_loc     ENDP


_int2a  PROC NEAR

        push    bx
        push    cx
        push    dx
        push    si
        push    di
        push    es

        mov     ax, ds:[_RAX]
        mov     bx, ds:[_RBX]
        mov     cx, ds:[_RCX]
        mov     dx, ds:[_RDX]
        mov     si, ds:[_RSI]
        mov     di, ds:[_RDI]
        mov     es, ds:[_RES]

        int     2ah

        pop     es
        pop     di
        pop     si
        pop     dx
        pop     cx
        pop     bx

        ret

_int2a  ENDP


_libera_msg     PROC    NEAR

                                        ; recoloca ECB em recepcao
        mov     di,_ecb_end_lib
        mov     ax,OFFSET CGROUP:[_rx_ssr]
        mov     ds:[di + 4],ax
        mov     ds:[di + 6],CS
        mov     word ptr ds:[di + 10],1024          ; TAM_BUF_REQ
        mov     WORD PTR ds:[di + 24],223H
        mov     bx,0
        mov     ds:[di + 27],bx
```

```
        mov     ds:[di + 29],bx
        mov     ds:[di + 31],bx
        mov     bl,01h
        mov     ds:[di + 33],bl
        mov     word ptr ds:[di + 34],1024          ; TAM_BUF_REQ

        mov     bx,3aH                  ; sizeof (struct ECB_DESC)
        mov     ax,di
        sub     ax,_first_ecb
        cwd
        div     bx
        mov     cx,1024                 ; TAM_BUF_REQ
        mul     cx

        mov     bx,_first_msg
        add     bx,ax
        mov     ds:[di + 36],bx         ; offset
        mov     ds:[di + 38],DGROUP

        mov     ax,7307H                ; coloca em rx
        int     2ah

        ret

_libera_msg     ENDP


_sai    PROC    NEAR

        mov     ax,4c00h
        int     21h

_sai    ENDP


_TEXT   ENDS
END
```

LIBCGF.ASM

```
cgroup group _text
assume cs:cgroup

dgroup group _data
assume ds:dgroup

_data segment

extrn _hora          :word
extrn _arquivo_fila  :word
extrn _data_corrente :byte
extrn _time_i        :byte
extrn _time_f        :byte

_data ends

_text segment public 'code'

public _int_21h
public _get_int_dos
public _creat
public _open
public _lseek
public _read
public _write
public _dup_handle
public _close
public _strlen
public _strchr
public _strcpy
public _get_dos_date
public _memset
public _memcpy
public _puts
public _atoi
public _memcmp
;public _exit
public _get_hora_i
public _get_hora_f


_int_21h proc near

        db     0cdh
_sys_vet db    21h
        db     0c3h

_int_21h endp

_get_int_dos proc near

        push    es
        push    di
```

```
        mov     ax,7319H
        mov     cx,103h
        int     2ah
        mov     _sys_vet,dl

        pop     di
        pop     es

        ret

_get_int_dos endp

_creat proc near

        mov     cx,00             ; atributo do arquivo

        mov     dx,_arquivo_fila  ; arquivo a ser aberto

        mov     ax,3c00h

        call    _int_21h

        jnc     FIM_CREAT
        mov     ax,0ffffh

FIM_CREAT:
        ret

_creat endp

_open proc near

        push    bp
        mov     bp,sp

        mov     ax,[bp + 6]     ; al: modo de abertura: 0 -> leitura
                                ;                       1 -> escrita
                                ;                       2 -> leitura e escrita
        mov     ah,3dh

        mov     dx,[bp + 4]     ; arquivo a ser aberto
        call    _int_21h

        jnc     FIM_OPEN
        mov     ax,0ffffh

FIM_OPEN:
        pop     bp

        ret

_open endp

_lseek proc near

        push    bp
        mov     bp,sp

        ; Parametros passados na pilha
```

```
        mov     bx,[bp + 4]             ; file handle
        mov     cx,0                    ; cx:dx -> long que indica o offset
        mov     dx,[bp + 6]             ; a ser dado da poicao indicada
        mov     ax,[bp + 8]             ; posicao inicial, que pode ser:
                                        ;       0 -> beginning of file
                                        ;       1 -> current position
                                        ;       2 -> end-of-file

        mov     ah,42h

        call    _int_21h

        jnc     short FIM_LSEEK
                                ; 'Acess denied' ou 'Invalid handle'
        mov     ax,0            ; FALSE = erro no le_conf

FIM_LSEEK:
        pop bp

        ret

_lseek endp


_read proc near

        push    bp
        mov     bp,sp

        ; Parametros passados na pilha
        mov     bx,[bp + 4]             ; file handle
                                        ; ds:dx -> endereco do buffer para
        mov     dx,[bp + 6]             ; onde serao lidos os dados
        mov     cx,[bp + 8]             ; numero de bytes a serem lidos

        mov     ah,3fh

        call    _int_21h

        jc      short ERRO_READ         ; 'Acess denied' ou 'Invalid handle'
        cmp     ax,cx
        jz     short FIM_READ

ERRO_READ:
        mov     ax,0            ; FALSE = erro na leitura do aruivo

FIM_READ:
        pop bp

        ret

_read endp


_write proc near

        push    bp
        mov     bp,sp
```

```
        ; Parametros passados na pilha
        mov     bx,[bp + 4]             ; file handle
                                        ; ds:dx -> endereco do buffer para
        mov     dx,[bp + 6]             ; onde serao lidos os dados
        mov     cx,[bp + 8]             ; numero de bytes a serem lidos

        mov     ah,40h

        call    _int_21h

        jc      short ERRO_WRITE        ; 'Acess denied' ou 'Invalid handle'
        cmp     ax,cx                   ; verifica se escreveu correto
        jz      short FIM_WRITE

ERRO_WRITE:
        mov     ax,0            ; FALSE = erro na leitura do aruivo

FIM_WRITE:
        pop bp

        ret

_write endp


_dup_handle proc near

        ; Duplica handle do arquivo para depois fecha-lo em _close

        push    bp
        mov     bp,sp

        ; Parametro passado na pilha
        mov     bx,[bp + 4]             ; file handle

        mov     ah,45h
        call    _int_21h

        jnc     FIM_DUP
        mov     ax, 00h
FIM_DUP:
        pop     bp

        ret

_dup_handle endp


_close proc near

        ; Fecha arquivo de configuracao = close

        push    bp
        mov     bp,sp

        ; Parametro passado na pilha
        mov     bx,[bp + 4]             ; file handle

        mov     ah,3eh
```

```
        call    _int_21h

        pop     bp

        ret

_close endp


_strlen proc near

        push    bp
        mov     bp,sp

        push    di

        mov     di,[bp + 4]     ; string => ES:DI
        push    ds
        pop     es
        mov     al,0            ; fim de string
        mov     cx,0ffffh       ; ate' quando procurar o fim da string
        cld
repnz   scasb
        mov     ax,cx
        sub     ax,0ffffh
        neg     ax              ; tamanho da string
        dec     ax              ; exclui-se o 0x00

        pop     di

        pop     bp

        ret

_strlen endp

_strchr proc near

        push    bp
        mov     bp,sp

        push    di

        ; calcula tamanho da string

        mov     di,[bp + 4]     ; string => ES:DI
        push    ds
        pop     es
        mov     al,0            ; fim de string
        mov     cx,0ffffh       ; ate' quando procurar o fim da string
        cld
repnz   scasb
        sub     cx,0ffffh
        neg     cx              ; tamanho da string

        ; procura caracter

        mov     di,[bp + 4]     ; string => ES:DI
        mov     ax,[bp + 6]     ; caracter procurado
```

```
repnz   scasb

        dec     di
        cmp     cx,0
        jnz     short ACHOU
        cmp     al,0
        jz      short ACHOU
        mov     ax,0
        jmp     short FIM_STRCHR
ACHOU:  mov     ax,di

FIM_STRCHR:

        pop     di

        pop     bp

        ret

_strchr endp

_strcpy proc near

        push    bp
        mov     bp,sp

        push    si
        push    di

        ; calcula tamanho da string origem

        mov     di,[bp + 6]     ; string origem => ES:DI
        push    ds
        pop     es
        mov     al,0            ; fim de string
        mov     cx,0ffffh       ; ate' quando procurar o fim da string
        cld
repnz   scasb
        sub     cx,0ffffh
        neg     cx              ; tamanho da string

        mov     di,[bp + 4]     ; string destino => ES:DI
        mov     si,[bp + 6]     ; string origem => DS:SI
rep     movsb

        pop     di
        pop     si

        pop     bp

        ret

_strcpy endp

_get_dos_date proc near

        mov     ah,2ah                  ; get date
        call    _int_21h
        sub     cx, 1990
```

```
        mov     [_data_corrente + 2],cl     ; ano do pedido (referencia -> 1990)
        mov     [_data_corrente + 1],dh     ; mes do pedido (1 - 12)
        mov     _data_corrente,dl           ; dia do pedido (1 - 31)

        mov     ah,2ch                      ; get_time
        call    _int_21h
        mov     [_data_corrente + 3],ch     ; hora do pedido (0 - 23)
        mov     [_data_corrente + 4],cl     ; min. do pedido (0 - 59)
        mov     [_data_corrente + 5],dh     ; seg. do pedido (0 - 59)

        ret

_get_dos_date endp

_memset proc near

        push    bp
        mov     bp,sp

        push    di

        mov     di,[bp + 4]     ; endereco a ser preenchido  => ES:DI
        push    ds
        pop     es
        mov     ax,[bp + 6]     ; byte a preencher o buffer
        mov     cx,[bp + 8]     ; numero de bytes
        cld
rep     stosb

        pop     di

        pop     bp

        ret

_memset endp


_memcpy proc near

        push    bp
        mov     bp,sp

        push    si
        push    di

        mov     di,[bp + 4]     ; buffer destino => ES:DI
        push    ds
        pop     es
        mov     si,[bp + 6]     ; buffer origem => DS:SI
        mov     cx,[bp + 8]     ; numero de bytes a serem copiados
        cld
rep     movsb

        pop     di
        pop     si

        pop     bp
```

```
        ret

_memcpy endp

_puts proc near

        push    bp
        mov     bp,sp

        mov     ah,9
        mov     dx,[bp + 4]
        call    _int_21h

        pop     bp

        ret

_puts endp

_atoi proc near

        push    bp
        mov     bp,sp

        push    si

        mov     cx,10           ; dividiremos e multiplicaremos por 10 = CX
        mov     si,[bp + 4]     ; string => DS:SI
        cld
        xor     bx,bx
CONT_ATOI:
        xor     ax,ax
        lodsb
        cmp     al,0
        jz      short FIM_ATOI
        sub     al,30h
        mul     cx
        add     bx,ax

        mov     ax,bx           ;
        mul     cx              ; = "mul bx,10"
        mov     bx,ax           ;

        jmp     short CONT_ATOI

FIM_ATOI:
        mov     ax,bx
        div     cx
        div     cx

        pop     si

        pop     bp

        ret

_atoi endp
```

```
_memcmp proc near

        push    bp
        mov     bp,sp

        push    si
        push    di

        mov     di,[bp + 4]     ; 1o. buffer => DS:DI
        mov     si,[bp + 6]     ; 2o. buffer => DS:SI
        mov     cx,[bp + 8]     ; numero de bytes a serem comparados
CONTINUA:
        mov     al,BYTE PTR [di]
        cmp     al,BYTE PTR [si]
        jnz     DIFERENTE
        inc     si
        inc     di
        loop    CONTINUA
        mov     ax,0
        jmp     short FIM

DIFERENTE:
        mov     ax,0ffffh

FIM:    pop     di
        pop     si

        pop     bp

        ret

_memcmp endp

;_exit proc near

;        push    bp
;        mov     bp,sp

;        mov     ax,[bp + 4]
;        mov     ah,4ch
;        int     21h ==> *** CUIDADO: Se for usar, substituir por call _int_21h

;        pop     bp

;        ret

;_exit endp

_get_hora_i proc near

        mov     ah,2ch                  ; get_time
        call    _int_21h
        mov     _time_i,cl              ; min. do pedido  (0 - 59)
        mov     [_time_i + 1],dh        ; seg. do pedido  (0 - 59)
        mov     [_time_i + 2],dl        ; cent. do pedido (0 - 99)

        ret

_get_hora_i endp
```

```
_get_hora_f proc near

        mov     ah,2ch                  ; get_time
        call    _int_21h
        mov     _time_f,cl              ; min. do pedido  (0 - 59)
        mov     [_time_f + 1],dh        ; seg. do pedido  (0 - 59)
        mov     [_time_f + 2],dl        ; cent. do pedido (0 - 99)

        ret

_get_hora_f endp


_text ends

end
```

REFERÊNCIAS BIBLIOGRÁFICAS

[1] CHLAMTAC, IMRICH; JAIN, RAJ, "A methodology for building a simulation model for efficent design and performance analysis of local area networks", Simulation, Fevereiro 1984

[2] CHLAMTAC, IMRICH; FRANTA, WILLIAM R., "A generalized simulation for computer networks", Simulation, Outubro 1982

[3] CHET, DAVID LANCTOT, "Performance evaluation of protocols for high speed bus networks via simulation", University of California, Los Angeles, 1984

[4] BAKER, DENNIS J.; HAUSER, JAMES P.; THOET, WILLIAM A., "A Distributed Simulation and Prototyping Testbed for Radio Communication Networks", IEEE Communications, Janeiro 1988

[5] FAWAZ, AYMAN; GIRALT, DIDIER; LUDWIG, LESTER, "The Protocol Workroom: an Experimental Protocol and Distributed System Research facility for U. C. Berkeley", University of California, Berkeley, Novembro 1985

[6] UNGER, BRAIN; BIRTWISTLE, GRAHAM; CLEARY, JOHN; HILL, DAVID; LOMOW, GREG; NEAL, RADFORD; PETERSON, MURRAY; WITTEN, IAN; WYVILL, BRIAN, "Jade: A simulation and software prototyping environment", University of Calgary, Fevereiro 1984

[7] UNGER, BRAIN; BIRTWISTLE, GRAHAM; CLEARY, JOHN; HILL, DAVID; LOMOW, GREG; NEAL, RADFORD; PETERSON, MURRAY; WITTEN, IAN; WYVILL, BRIAN, "Jade: A distributed software prototyping environment", ACM Operating Systems Review, Julho 1983

[8] RIGHTER, RHONDA; WALRAND, JEAN, C., "Distributed Simulation of Discrete Event Systems", Proceedings of the IEEE, Janeiro 1989

[9] HEIDELBERGER, P., "Statistical analysis of parallel simulations", Proc. Winter Sim. Conf., 1986

[10] RAZOUK, RAMI R., "Modeling and Simulation of data Communication Networks using SARA", Computer Networks and Simulation III, 1986

[11] AMPLUS INFORMÁTICA, "Rede Local Amplinet - Manual de Referência Técnica", Junho 1988

[12] SHRIVASTAVA, S. K.; PANZIERI, F., "The Design of a Reliable Remote Procedure Call Mechanism", IEEE Transactions on Computers, Julho 1982

[13] MAGEE, JEFFREY N., "Provision of Flexibility in Distributed Systems", Imperial College, London, Abril 1984

[14] NASSEHI, M. MEHDI, "CRMA: an Access scheme for High-Speed Lans and Wans", IEEE Proc. Supercomm., Abril 1990

[15] WONG, J. W., "Throughput of DQDB Networks under Heavy Load", EFOC/LAN-89, Junho 1989

[16] IEEE 802.3, "Carrier Sense Multiple Access with Collision Detection (CSMA-CD) Access Method and Physical Layer Specifications", Junho 1983

[17] IEEE 802.4, "Token-Passing Bus Access Method and Physical Layer Specifications", Junho 1984

[18] IEEE 802.5, "Token Ring Access Method and Physical Layer Specifications", Dezembro 1984

[19] RODRIGUES, PAULO H. DE AGUIAR, "Access Protocols for High-Speed Fiber Optical Local Networks", University of California, Los Angeles, 1984

[20] CARMO, ROSA MARIA L. R., "Especificação de uma Ferramenta para Avaliação de Algoritmos Distribuídos", PUC-RJ, Abril 1990

[21] TANENBAUM, ANDREW S., "Computer Networks", Prentice-Hall, 1981

[22] ZIMMERMANN, H., "OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection", IEEE Trans. Commun., Abril 1980

[23] PEACOCK, J. KENT; WONG J. W.; MANNING, ERIC G., "Distributed Simulation Using a Network of Processors", Computer Networks 3, 1979

[24] IEEE 802.2, "Logical Link Control", Julho 1984

[25] MICROSOFT Co., "Microsoft MS-DOS versão 4.01 Guia do Usuário e Referência do Usuário", 1989

[26] AMPLUS INFORMÁTICA, "AMPLIWARE versão 2.05 Manual do Administrador e Manual do Usuário", 1990

[27] MICROSOFT Co., "Microsoft C Compiler Language Reference and Run-Time Library Reference", 1986

[28] MICROSOFT Co., "Microsoft CodeView Window-Oriented Debugger", 1986

[29] MICROSOFT Co., "Microsoft Macro Assembler User Guide and

Reference", 1985

[30] PERISCOPE COMPANY, Inc., "Periscope Manual version 2.02", 1986

[31] KERNIGHAN, BRIAN W.; RITCHE, DENNIS M., "C a Linguagem de Programação", Editora Campos, 1986

[32] CHISTOPHER, T.; EVENS, M.; GARGEYA, R. R.; LEONHARDT, T., "Structure of a Distributed Simulation System", IEEE COMPSAC, 1982

[33] CONCEPCION, A. I., "Mapping Distributed Simutatiors onto the Hierarchical Multibus Multiprocessor Architecture", Proc. SCS Dist. Sim. Conf., 1985

[34] MISRA, J., "Distributed Discrete-Event Simulation", Cumputing Surveys, vol 18, Março 1986

[35] LANDWEHR, C. E., "An Abstract type for Statistics Collection in SIMULA", ACM Trans. Prog. Languages Sys., vol. 2, Out. 1980

[36] CHANDY, K. M.; MISRA, J., "Assynchronous Distributed Simulation via Sequence of Parallel Computations", Comm. of the ACM, vol. 24, no. 4, Abril 1981

[37] JEFFERSON, D. R.; SOWIZRAL, H., "Fast Concurrent Simulation using the Time Warp Mechanism", Proc. SCS Distr. Sim. Conf., 1985

[38] JEFFERSON, D. R., "Virtual Time", ACM Trans. Prog. Lang. and Sys., vol. 7, no. 3, 1985