IMPLEMENTAÇÃO DE UM TERMINAL X PARA O SISTEMA OPERACIONAL PLURIX

Luiz Fernado Huet de Bacellar

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. Newton Faller, Ph.D.
(presidente)

Prof. Edil S. Tavares Fernandes, Ph.D.

Prof. Júlio Salek Aude, Ph.D.

RIO DE JANEIRO, RJ - BRASIL

MARÇO DE 1990

BACELLAR, LUIZ FERNANDO HUET DE

Implementação de um Terminal X para o Sistema Operacional Plurix [Rio de Janeiro] 1990

IX, 104 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 1990)

Tese - Universidade Federal do Rio de Janeiro, COPPE

1. Sistemas Computacionais Gráficos

I. COPPE/UFRJ II. Título (série).

À minha esposa Adriana

AGRADECIMENTOS

Ao Doutor Newton Faller pelo apoio, incentivo e orientação fornecidos no transcorrer deste trabalho.

Aos Doutores Edil Severiano Tavares Fernandes e Júlio Salek Aude pela honra de tê-los participando da banca.

Aos meus Pais por terem tornado possível a chegada a este ponto.

Aos Amigos do NCE que de alguma forma contribuiram para a realização deste trabalho e ao NCE por ter tornado viável o desenvolvimento e a implementação do trabalho.

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

IMPLEMENTAÇÃO DE UM TERMINAL X PARA O SISTEMA OPERACIONAL PLURIX

Luiz Fernando Huet de Bacellar

Março, 1990

Orientador: Prof. Newton Faller, Ph.D.
Programa: Engenharia de Sistemas e Computação

Este trabalho consiste na definição e implementação de um sistema integrado de hardware e software, denominado Terminal X. O objetivo principal é dotar o Plurix, um sistema operacional com filosofia Unix, de uma interface gráfica com seus usuários.

Apresentamos primeiramente o sistema de janelas X Window, que foi utilizado para a definição da arquitetura do Terminal X implementado. A seguir demonstramos como foi realizada a implementação do Terminal X e ao final fazemos uma avaliação do trabalho executado, sugerindo possíveis melhorias e ampliações no sistema.

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

AN X TERMINAL IMPLEMENTATION FOR THE PLURIX OPERATING SYSTEM

Luiz Fernando Huet de Bacellar

March, 1990

Thesis Supervisor: Prof. Newton Faller, Ph.D.
Department: Systems Engineering and Computer Science

This work consists of the definition and implementation of a hardware and software integrated system named X Terminal. Its main purpose is to provide Plurix, an Unix-like operating system, with a graphic user interface.

First we present the X Window system, which was used to define the architecture of the X Terminal. Later we explain how the X Terminal was actually implemented and finally we make an evaluation of the work done, suggesting possible improvements and extensions to the system.

ÍNDICE

CAPÍTULO I

INTRODUÇÃO

I.1. Apresentação do Trabalho

Este trabalho consiste na definição e implementação de um sistema integrado de hardware e software, denominado Terminal X, para dotar o sistema operacional Plurix de uma interface gráfica com seus usuários. O sistema operacional Plurix, um sistema com filosofia Unix, foi desenvolvido no Núcleo de Computação Eletrônica da Universidade Federal do Rio de Janeiro e é executado no computador Pegasus-32X, também desenvolvido no mesmo local [1 - 2 - 3].

A definição da arquitetura Terminal X baseou-se no sistema de janelas X Window, desenvolvido no Massachusetts Institute of Technology, EUA. É um sistema que fornece ao seu usuário uma biblioteca de rotinas gráficas para serem utilizadas na criação de aplicações que necessitem de recursos gráficos. O sistema X Window tem seu código colocado em domínio público.

O hardware do Terminal X foi definido de acordo com as características da arquitetura a ser utilizada e de forma a suportar todas as exigências quanto aos recursos a serem oferecidos pelo sistema X Window. O software do

Terminal X foi implementado através da adaptação das funções do sistema X Window à arquitetura e ao hardware do Terminal X. Para a adaptação do X Window ao Terminal X foi necessário modificar seu código original e implementar novos procedimentos de software para a interação com o hardware definido.

O Terminal X apesar de ter sido implementado para o sistema operacional Plurix, tem o objetivo de, em sua versão final, poder ser ligado, através de redes de computadores, a qualquer sistema computacional que utilize o sistema X Window como interface gráfica com seus usuários.

O Terminal X foi desenvolvido e implementado no Núcleo de Computação Eletrônica da Universidade Federal do Rio de Janeiro.

I.2 Motivação

O advento das estações de trabalho com suas telas de exibição de alta resolução, trouxeram o potencial das interfaces gráficas para uma variedade de aplicações. As interfaces gráficas facilitam o processo de aprendizado, uso e entendimento das aplicações. Diversas interfaces gráficas proprietárias surgiram, provocando dificuldades no transporte de aplicações desenvolvidas em um sistema computacional que utilize uma determinada interface, para outro com interface distinta.

Diante deste fato, surgiu a necessidade da busca de uma interface gráfica que fosse padrão em sistemas computacionais de diferentes fabricantes. Baseado nesta idéia, foi desenvolvido nos EUA o sistema de janelas X Window. Este sistema tem sua arquitetura dividida em diversas camadas ou níveis. Essas camadas se sobrepõem conforme aumenta a complexidade das bibliotecas gráficas oferecidas ao usuário. A finalidade é permitir ao usuário a utilização das diversas bibliotecas de acordo com as necessidades de sua aplicação.

O sistema X Window foi colocado em domínio público e sua arquitetura passou a ser adotada como um padrão em sistemas gráficos. No entanto, somente a camada do nível básico do sistema X Window passou a ser utilizada como um padrão de fato. Foi iniciada uma intensa discussão sobre como seriam definidas as camadas de mais alto nível que utilizariam as rotinas do nível básico para fornecer ao usuário um ambiente gráfico sofisticado.

Concorrentemente a estes acontecimentos, foi desenvolvido no NCE/UFRJ o sistema operacional Plurix. Surgiu então a necessidade de dotar o Plurix de uma interface que permitisse aos seus usuários o desenvolvimento de aplicações gráficas. Porém, o Pegasus-32X, computador no qual o Plurix é executado, não possui o suporte de um hardware gráfico. Como o Pegasus foi projetado como computador protótipo em 1984 e ainda hoje funciona desta forma, procurou-se uma solução que se

tornasse independente deste computador.

A solução encontrada veio com o Terminal X. Estes produtos surgiram nos EUA no final de 1988 e funcionam basicamente como sistemas computacionais dedicados ao serviço de exibição gráfica. O fato dos Terminais X utilizarem o sistema X Window para serem implementados foi fundamental na escolha da solução. Somente as camadas do sistema X Window que vão até o nível básico são necessárias para a implementação dos Terminais X. Desta forma, adotou-se uma solução gráfica para o Plurix que possibilitará a implantação das demais camadas que serão executadas sobre a camada básica, assim que for definido um padrão para estas.

I.3. Proposições Adotadas no Trabalho

Duas proposições básicas foram adotadas como orientações no trabalho de desenvolvimento do Terminal X para o sistema operacional Plurix.

A primeira proposição é manter total compatibilidade entre o software implementado para o Terminal X e o sistema de janelas X Window. Esta linha foi seguida a partir do fato do nível básico do sistema X Window ter se tornado um padrão em ambientes que oferecem recursos gráficos. Sem tal compatibilidade, o Terminal X se torna inviável de ser aproveitado para execução de aplicações genéricas já existentes para o sistema X Window, inclusive

aquelas comerciais.

A compatibilidade permite não só a execução de aplicações que tenham sido implementadas em outros sistemas computacionais que utilizem o sistema X Window, como também o aproveitamento de outros níveis de interface desenvolvidos sobre o nível básico do X Window. Além disto, aplicações dedicadas que venham a ser desenvolvidas para o Terminal X, poderão ser transportadas para outros sistemas que possuam a biblioteca básica de rotinas gráficas do sistema X Window implementada neste terminal.

Como segunda proposição adotada neste trabalho, foi considerado que o desenvolvimento da base de hardware do Terminal X deveria seguir duas orientações. Ao mesmo tempo deveria ser o estado da arte na área eletrônica de exibição gráfica e também permitir que o trabalho de pesquisa desenvolvido gerasse, ao final, um produto que fosse viável comercialmente no mercado de Informática brasileiro. Estas orientações foram seguidas, pois reconhecemos que a interação entre as áreas acadêmica e industrial é fundamental para o desenvolvimento da Informática no Brasil.

I.4. Organização do Trabalho

Este trabalho está organizado em mais seis capítulos além deste de introdução. O capítulo II apresenta o sistema de janelas X Window. São definidos os componentes

de um sistema de janelas e a seguir, são mostradas as principais características da arquitetura do sistema X Window. Procura-se mostrar os principais conceitos envolvidos neste sistema e suas dependências quanto ao ambiente computacional no qual está sendo executado.

No capítulo III é feita a definição de um Terminal X e dos motivos que proporcionaram o seu desenvolvimento. São explicadas as arquiteturas existentes para sua implementação e é mostrada a arquitetura e o caminho adotados na implementação do Terminal X realizada neste trabalho.

Nos capítulos IV e V é explicado como foi feita a implementação do Terminal X para o sistema operacional Plurix. O capítulo IV descreve a implementação dos recursos necessários para o desenvolvimento de aplicações gráficas no Terminal X. O capítulo V trata da implementação da parte do sistema X Window que irá interagir com o hardware definido para o Terminal X.

O capítulo VI relata como foram feitos os testes no ambiente Plurix/Terminal X implementado. Procura-se mostrar as principais metodologias adotadas e os problemas encontrados durante este trabalho. É feita também uma análise do desempenho do sistema desenvolvido e são sugeridas áreas de atuação para possíveis melhorias.

Por fim, no capítulo VII são apresentadas as

conclusões tiradas em função do trabalho desenvolvido e das proposições iniciais deste trabalho. Procura-se mostrar o caminho a ser seguido para a evolução do Terminal X como um sistema computacional gráfico.

CAPÍTULO II

O SISTEMA X WINDOW

O sistema X Window, ou simplesmente X, nasceu da necessidade de se poder utilizar estações de trabalho de diferentes fabricantes de uma forma transparente sob o ponto de vista da aplicação a ser criada. O começo de seu desenvolvimento data do ano de 1984 no Massachusetts Institute of Technology (MIT) nos EUA [4]. A idéia inicial era desenvolver o núcleo de um sistema de janelas com rotinas gráficas que pudesse ser facilmente transportado para as estações de trabalho "bit-mapped" de qualquer fabricante. Esse núcleo forneceria rotinas básicas de baixo nível que permitiriam o desenvolvimento de aplicações de mais alto nível que pudessem ser executadas nas diversas estações de trabalho, praticamente sem ser necessário a realização de adaptações.

Junto à idéia da facilidade de transporte, foi incorporada a possibilidade de, através do sistema X, o usuário poder interagir com um ambiente distribuído. Em uma rede que interligue estações de trabalho e computadores de grande porte, a aplicação gráfica de um usuário que está utilizando qualquer um dos elementos da rede, poderá exibir os resultados em qualquer outro elemento que possua uma tela gráfica.

A partir destas idéias foram desenvolvidas diversas versões do sistema X Window. Este sistema aparece também como seqüência aos sistemas de janelas VGTS e W, ambos desenvolvidos para o sistema V, um software para sistemas distribuídos construído na Universidade de Stanford, EUA [4]. De ambos, o sistema X herdou diversas características como, por exemplo, o conceito de hierarquia com relação aos recursos oferecidos pelo sistema.

Com a versão 10.4 o sistema X Window foi colocado pelo MIT como um sistema de domínio público. Isto gerou uma grande popularidade do mesmo, fazendo com que diversas aplicações e variações fossem desenvolvidas em cima deste sistema, dentre as quais podemos citar os Terminais X que tratamos neste trabalho. A versão do sistema X utilizada para o desenvolvimento do Terminal X foi a 11.2.

II.1. As Partes de um Sistema de Janelas

Para que um sistema de janelas seja definido é importante primeiramente se fazer a definição formal de janela e de alguns termos associados [5].

Uma memória para armazenamento de uma imagem ("frame memory" ou "frame buffer") é um buffer utilizado para o refrescamento da informação visual exibida em um monitor através da varredura de um feixe de elétrons sobre uma tela de fósforo. Uma memória de imagem "bit-mapped" convencional tem pelo menos a quantidade necessária para

armazenar um valor correspondente a cada pixel na tela de exibição. Em várias aplicações a tela de exibição é dividida por software ou firmware em janelas, através das quais diferentes tarefas são executadas. Janelas são regiões retangulares dentro das quais imagens são desenhadas, armazenadas e manipuladas. Cada janela representa um pedaço da memória de imagem. Janelas podem se sobrepor na tela, com uma janela totalmente visível e outras parcialmente, ou totalmente, ocultas.

Um sistema de janelas tem como funções básicas controlar onde uma janela aparece na tela, qual seu tamanho e alterar o seu conteúdo. Para desempenhar estas funções um sistema de janelas é dividido em três partes segundo SCHEIFLER e GETTYS [4]:

- o gerenciador de janelas;
- o gerenciador de entradas;
- o sistema de janelas básico.

Um sistema de janelas, sob o ponto de vista da aplicação, ou seja de um procedimento que irá utilizá-lo para exibir seus resultados, possui diversas interfaces de acordo com SCHEIFLER e GETTYS [4]. Essas interfaces são implementadas pelas três partes, acima mencionadas, que compõem o sistema. Para se definir cada uma destas partes é importante primeiro analisar as interfaces de um sistema de janelas.

A interface de programação é uma biblioteca de

rotinas e tipos fornecida em uma determinada linguagem de programação que permite a interação da aplicação com o sistema de janelas. As interfaces de programação podem ser de baixo nível ou de alto nível, sendo ambas normalmente desenvolvidas junto com o sistema de janelas básico.

A interface de programação é dita de alto nível quando são desenvolvidas rotinas que utilizam a biblioteca de baixo nível para a elaboração de objetos gráficos mais complexos do que as primitivas gráficas fornecidas por esta biblioteca. Normalmente o conjunto das rotinas que compõem a interface de programação de alto nível é denominado "toolkit".

A interface com a aplicação é a interação mecânica do usuário com a aparência visual específica de uma determinada aplicação. Ela define como a informação é apresentada e manipulada dentro da aplicação.

A interface de gerenciamento é a interação mecânica do usuário tratando com o controle geral do dispositivo de hardware em si e dos dispositivos de entrada sobre os quais o sistema de janelas e as aplicações estão sendo executadas. A interface de gerenciamento define como as aplicações são arrumadas e rearrumadas na tela, e como o usuário troca de aplicação.

A interface com o usuário é a junção da interface com a aplicação e a interface de gerenciamento. A interface

com o usuário responde por toda a interação entre o usuário e o sistema computacional por ele utilizado para executar a aplicação desejada.

Uma vez feita a análise das funções das interfaces de um sistema de janelas, pode-se definir como as três partes nas quais um sistema de janelas é dividido, irão implementar as funções de cada interface.

O **gerenciador de janelas** implementa parte da interface de gerenciamento. É apenas um programa que, tal como uma aplicação, utiliza a biblioteca de rotinas da interface de programação para controlar o tamanho e o posicionamento das janelas das aplicações na tela de exibição. Ao gerenciador de janelas é dada uma autorização especial para realizar esta função, autorização que as demais aplicações não possuem.

O gerenciador de janelas tipicamente permite que uma aplicação mova ou redimensione as suas janelas e controle a pilha de janelas que se sobrepõem de acordo com regras pré-definidas. O conjunto de regras que especificam os tamanhos permitidos e a posição das janelas é a política para o layout das janelas de um sistema.

O **gerenciador de entradas** implementa o restante da interface de gerenciamento. Ele controla qual aplicação enxerga uma sinalização proveniente de um dos dispositivos de entrada do sistema, normalmente um teclado ou um

"mouse". A finalidade do gerenciador de entradas é evitar que aplicações que não estejam ativas ou associadas a um dispositivo de entrada não sejam interrompidas por sinais enviados por este dispositivo.

A principal parte de um sistema de janelas é o **sistema de janelas básico.** Ele é o substrato no qual as aplicações e os gerenciadores de janelas e de entradas são construídos. No sistema de janelas básico estão as rotinas que realizam a interface do sistema com o dispositivo de hardware em si e com os dispositivos de entrada.

Para a implementação do Terminal X definido neste trabalho, foi utilizada a parte do sistema de janelas básico do sistema X Window, junto com as rotinas da biblioteca que implementam a interface de programação de baixo nível. As demais partes como o gerenciador de janelas e o gerenciador de entradas, por serem transportadas automaticamente para o sistema de janelas básico, e a interface de programação de alto nível, que é implantada sobre a interface de programação de baixo nível, não fazem parte do escopo deste trabalho.

II.2. A Arquitetura do Sistema X Window

II.2.1. Características Básicas

O desenvolvimento do sistema X Window foi baseado em duas importantes características quanto a sua arquitetura

[4]. A primeira é que as aplicações devem ser independentes do sistema computacional no qual estão sendo executadas. A segunda, o sistema deve utilizar redes computacionais de uma forma transparente sob o ponto de vista da aplicação executada.

Sobre a independência do sistema X quanto ao computador no qual o sistema está sendo executado, existem diversos fatores a se ressaltar. Ao se utilizar um sistema com este tipo de independência, pode-se transportar uma aplicação de um sistema computacional para outro distinto praticamente sem modificações. Caso os dois sistemas executem um sistema operacional compatível e possuam um mesmo processador central, o transporte pode ser feito ao nível do programa objeto da aplicação.

A independência quanto ao sistema computacional é obtida no sistema X através da padronização da interface de programação. Através desta padronização, uma aplicação encontra as mesmas funções gráficas disponíveis em qualquer sistema X independentemente das características do dispositivo de exibição gráfica suportado pelo sistema.

Quanto a transparência de acesso à redes computacionais, o sistema X permite que aplicações que sejam executadas em um determinado computador da rede, possam exibir seus resultados em algum outro computador desta rede, possivelmente uma estação de trabalho. Normalmente estes dois sistemas são distintos, com

arquitetura e sistema operacional diferentes.

Estas características levaram a arquitetura do sistema X a se basear no modelo cliente-servidor. Neste modelo, é possível implementar naturalmente um sistema de janelas com uma filosofia que coincide com as duas características citadas.

II.2.2. O Modelo Cliente-Servidor

O conceito do modelo cliente-servidor é definido em COMER [6], a partir de um ambiente onde exista uma rede interligando diversos sistemas computacionais. O modelo cliente-servidor é uma extensão natural da comunicação entre processos em um único sistema computacional.

O termo **servidor** se aplica a qualquer programa que ofereça um serviço que possa ser requisitado através da rede. O servidor aceita as requisições, realiza o seu serviço e retorna o resultado ao sistema que gerou a requisição.

Um programa em execução é denominado **cliente** quando ele envia uma requisição a um servidor através da rede e espera por uma resposta.

Normalmente os servidores são programas aplicativos, ou seja, processos, sendo executados em um determinado sistema. Desta forma, o servidor de um determinado serviço

pode ser executado em um sistema compartilhando tempo com outros processos, ou pode ser executado em algum sistema computacional específico para ele, podendo ser inclusive em computadores pessoais.

Múltiplos servidores podem oferecer o mesmo serviço e podem ser executados na mesma máquina ou em várias máquinas. Isto acarreta que servidores de um mesmo serviço que estejam em máquinas distintas, tenham endereços distintos, um para cada máquina. No entanto, dentro de cada uma destas máquinas, estes servidores possuem um mesmo subendereço ou porta. O número das portas de servidores bastante difundidos e conhecidos pelos usuários da rede é fixo, de forma que ao se desejar um destes serviços, conhecendo-se o endereço da máquina onde está o servidor, basta enviar requisições para a porta correspondente a este servidor.

II.2.3. O Modelo Cliente-Servidor no Sistema X Window

O modelo cliente-servidor no sistema X funciona de forma a manter a independência da aplicação em relação ao hardware que a exibe [4]. Assim sendo, para cada dispositivo físico de exibição irá existir um servidor para controlá-lo, de forma a manter a aplicação/cliente independente do dispositivo.

Um dispositivo físico de exibição, ou "display", é constituído de um ou mais monitores que utilizem a

tecnologia de exibição por varredura, de um teclado, de um "mouse" e do hardware para controle destes dispositivos. Um sistema computacional pode possuir mais de um "display" e nesse caso tem-se um servidor sendo executado para cada "display". É importante notar que um "display" pode ter mais de um monitor a ele ligado, desde que todos os monitores sejam controlados por um hardware único.

Uma aplicação/cliente se comunica com um servidor através de qualquer canal de comunicação bidirecional, confiável e que se possa enviar mensagens de tamanho igual a um byte. Existe um protocolo definido pelo sistema X, o protocolo X, que faz a comunicação do cliente com o servidor através do canal de comunicação. Se o cliente e o servidor estão na mesma máquina, o canal pode ser baseado em qualquer mecanismo de comunicação entre processos. Caso contrário, se o cliente e o servidor estão em máquinas distintas interligadas através de uma rede, é necessário estabelecer uma conexão entre os dois através da rede.

O fato do sistema X Window não necessitar de um canal de comunicação complexo para a comunicação entre o cliente e o servidor, faz com que o sistema possa ser utilizado em diversos ambientes. Como exemplo, pode-se citar o uso do protocolo X sobre os protocolos de rede TCP/IP ("Transmission Control Protocol / Internet Protocol") e DECNET ("Digital Equip. Corp. Network Protocol") [4].

Pode-se ter no sistema X vários clientes com conexões

abertas simultaneamente a um único servidor e um cliente pode abrir conexões a vários servidores ao mesmo tempo. A tarefa básica do servidor é multiplexar os pedidos dos diversos clientes em relação ao "display" que ele controla, e separar as sinalizações de entrada provenientes do teclado e do "mouse", enviando-as para o cliente apropriado. A visualização de um exemplo de utilização do sistema X Window é mostrada na figura (II.1).

De acordo com as definições do item II.1, o servidor do sistema X, ou **servidor X,** engloba o sistema de janelas básico do sistema X Window. Ele fornece os recursos e os mecanismos fundamentais para se implementar diversas interfaces com o usuário. Todas as dependências com o hardware são colocadas no servidor fazendo com que qualquer cliente/aplicação seja independente deste hardware. Além disso, o protocolo de comunicação entre cliente e servidor é independente do meio físico de comunicação, fazendo com que o cliente/aplicação seja realmente independente da máquina na qual o servidor está sendo executado.

O sistema X fornece uma interface de programação, a **biblioteca X** [7], na qual estão todas as rotinas necessárias para que o cliente/aplicação faça a conexão com o servidor e solicite as tarefas desejadas. Ao se ligar a aplicação às rotinas fornecidas pela biblioteca X, a forma de comunicação com o servidor se torna totalmente

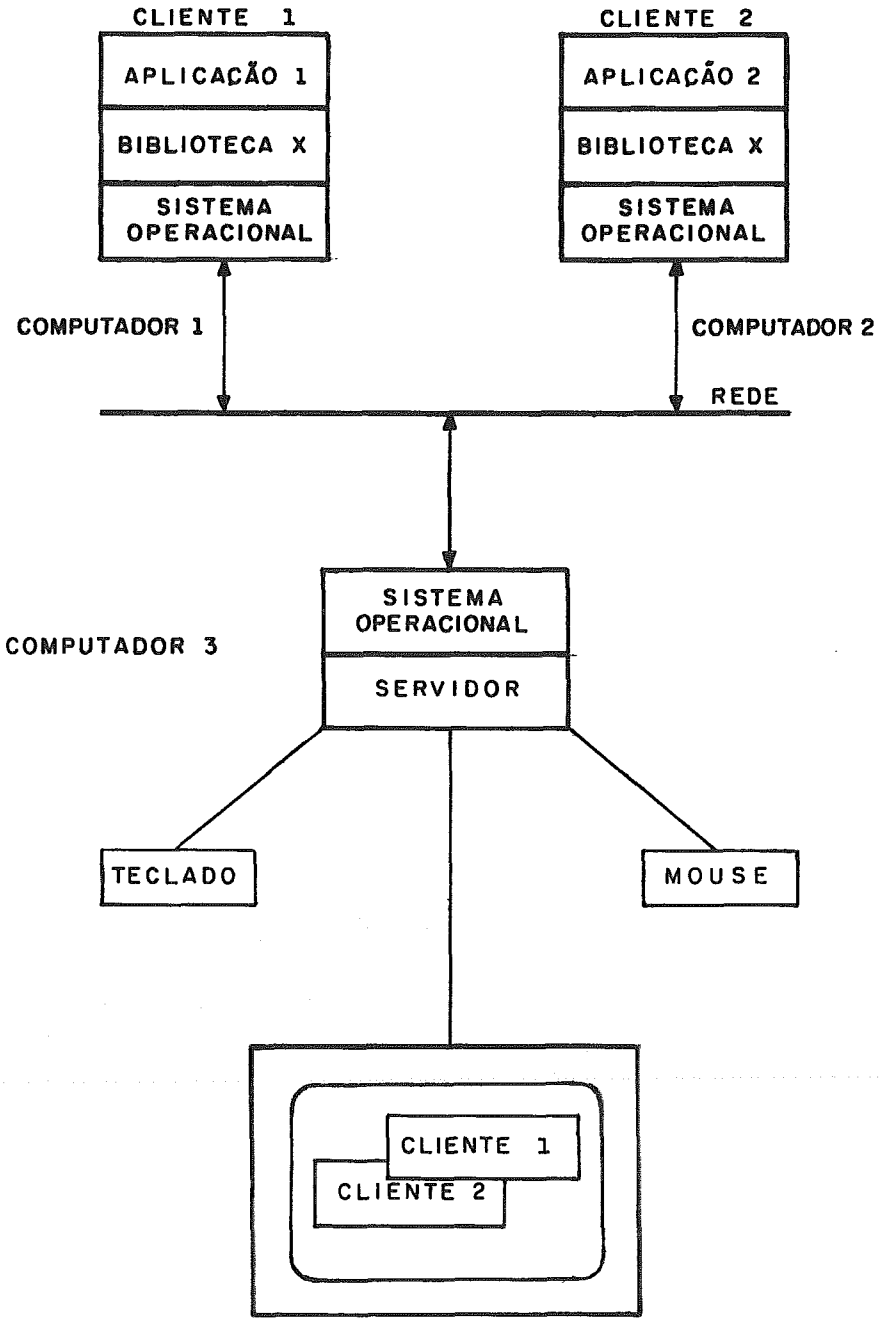
CLIENTE 1
APLICAÇÃO 1
BIBLIOTECA X
SISTEMA OPERACIONAL
CLIENTE 2
APLICAÇÃO 2
BIBLIOTECA X
SISTEMA OPERACIONAL
COMPUTADOR 1
COMPUTADOR 2
REDE
COMPUTADOR 3
SISTEMA OPERACIONAL
SERVIDOR
TECLADO
MOUSE
CLIENTE 1
CLIENTE 2

FIGURA II.1

transparente para o cliente. Nas rotinas da biblioteca estão todas as dependências quanto ao mecanismo que será utilizado para enviar as mensagens do protocolo X. A aplicação utiliza estas rotinas para especificar a qual servidor ou servidores ela fará conexão para exibir seus resultados.

II.2.4. O Protocolo X para a Comunicação Cliente-Servidor

O **protocolo X** [7] foi projetado de forma que, se o servidor está esperando por uma resposta de um certo cliente, ele deve poder continuar a servir outros clientes. Isto evita que, no caso de um cliente com erro, o servidor não fique parado esperando algo deste cliente.

Outra característica importante a se ressaltar a nível de comunicação entre cliente e servidor, é que o servidor X deve ser sempre projetado para aceitar conexões com clientes que transmitam os bytes na ordem "big-endian" ou "little-endian" [4]. Isto ocorre, pois no sistema X existem informações de tamanho 16 bits ou 32 bits sendo transmitidas no canal de comunicação entre o cliente e o servidor, canal este que é definido com largura de 8 bits.

Para resolver este problema, o sistema X foi concebido de tal forma que o cliente sempre receba e transmita informações em sua ordem de byte natural. Caso as ordens de byte nas máquinas em que estão o cliente e o servidor sejam diferentes, o servidor é o responsável pela

troca da ordem dos bytes vindos do cliente. Caso as ordens sejam iguais nada é feito, evitando assim o "overhead" que existiria se a ordem dos bytes na comunicação fosse fixa e tanto o cliente quanto o servidor estivessem em máquinas com ordem oposta a ordem fixada.

Além do problema quanto a ordem de byte na comunicação, existe o problema quanto ao alinhamento das informações que são transmitidas através do protocolo de comunicação entre cliente e servidor. Isto é resolvido no protocolo X fazendo com que as informações sejam transmitidas em blocos com tamanhos múltiplos de 32 bits. Além disso, quantidades de 16 bits e 32 bits dentro de um bloco são sempre alinhadas em espaços de 16 bits e 32 bits respectivamente. Isto significa que, no protocolo X, ao se transmitir uma informação de tamanho de um byte e depois outra de 16 bits, entre as duas irá sempre uma informação de tamanho igual a um byte sem nenhum significado. Este byte é denominado "pad".

O protocolo X é dividido em quatro tipos de formato para a transmissão de informações [7]:

- formato para requisições;
- formato para respostas;
- formato para erro;
- formato para evento.

O formato para as requisições do cliente ao servidor é sempre de um "header" de quatro bytes, seguido de zero

ou mais bytes de dados adicionais. O "header" é composto de quatro bytes, sendo um com o "opcode" da requisição, dois que indicam o tamanho da informação adicional em unidades de quatro bytes e um byte genérico.

O formato para resposta do servidor ao cliente contém sempre 32 bytes seguidos de zero ou mais bytes de dados adicionais. Nos 32 bytes fixos existe um campo de 32 bits que diz o número de bytes adicionais em unidades de quatro bytes.

O formato para comunicar um erro ao cliente em relação a alguma requisição, tem sempre tamanho de 32 bytes. Um dos bytes dá o código do erro e os demais informações como o "opcode" da requisição que falhou e o número de seqüência desta requisição. No sistema X, todas as requisições entre um cliente e um servidor são numeradas seqüencialmente.

O formato para um servidor notificar um cliente que algum evento ocorreu, tem também tamanho fixo de 32 bytes. Todos os eventos possuem um byte com o código do tipo de evento que está sendo comunicado.

II.2.5. Os Recursos Fornecidos pelo Servidor X

Os recursos básicos fornecidos pelo servidor X são janelas, fontes de caracteres, cursores controlados por "mouse" e imagens não visíveis na tela ("pixmaps") [4]. O

cliente requisita a criação de um recurso e o servidor aloca o recurso ao cliente, retornando um identificador do recurso. Qualquer cliente que conheça o identificador de um recurso pode usar e manipular o recurso livremente, mesmo que este recurso tenha sido criado por outro cliente. Isto é permitido com a finalidade de se poder criar gerenciadores de janelas independentemente das aplicações que irão utilizar as janelas. Além disto, pode-se ter no sistema X, aplicações desenvolvidas através de múltiplos processos que compartilham os mesmos recursos.

Todas as operações em um recurso devem ter como parâmetro o identificador do recurso. Desta forma, uma aplicação pode multiplexar o uso de várias janelas em uma conexão única com um servidor, através da rede de computadores. Similarmente, cada evento gerado pelo servidor irá conter o identificador da janela na qual o evento ocorreu.

As janelas e as imagens não visíveis na tela são, no sistema X Window original, regiões de memória acessíveis através de descritores de arquivos fornecidos pelo sistema operacional. A diferença entre as janelas e as imagens não visíveis é que as primeiras estão situadas no "frame buffer" do sistema gráfico do computador, enquanto as outras estão na memória de trabalho do mesmo sistema computacional. Isto significa que as imagens não visíveis só aparecerão na tela de exibição caso sejam copiadas em

alguma das janelas que estejam visíveis nesta tela.

É importante ressaltar também o conceito de hierarquia de janelas no sistema X [4]. No topo da hierarquia, está a janela **raiz** que cobre toda a tela de exibição e é criada pelo próprio servidor. Nas aplicações, as janelas de mais alto nível na hierarquia, são criadas como subjanelas, ou **filhas,** da janela **raiz.** Dada uma janela de uma aplicação, suas subjanelas podem ser empilhadas em qualquer ordem e se sobreporem arbitrariamente. O modelo da hierarquia de janelas do sistema X procura, desta forma, representar uma pilha de papéis sobre uma mesa de trabalho.

Quando uma janela cobre parcialmente ou totalmente outra janela, diz-se que a primeira está ocultando a segunda. Uma janela **filha** pode ocultar totalmente a janela **mãe,** podendo inclusive possuir dimensões maiores que a da **mãe.** Neste caso, a janela **filha** terá sua parte visível sempre restrita a área ocupada por sua **mãe** e sua área excedente ficará invisível. A janela **raiz** limita o tamanho máximo de uma janela qualquer. Nenhuma janela terá suas dimensões maiores que a da janela **raiz,** que cobre toda a tela de exibição do monitor ligado ao servidor do sistema X Window.

II.2.6. Suporte do Sistema Operacional para o Sistema X

O sistema X Window foi desenvolvido, desde sua versão

inicial, em computadores que utilizavam o sistema operacional Unix. O sistema Unix utilizado inicialmente para o desenvolvimento foi o Unix 4.2 BSD da Universidade de Berkeley, EUA. A partir da versão 11.2 do sistema X, foi utilizada a versão 4.3 do Unix de Berkeley [8] para o desenvolvimento. A facilidade de transporte de softwares entre sistemas Unix-like e o grande número de usuários deste tipo de sistemas, fez com que o sistema X fosse transportado e adaptado para outros computadores que utilizassem sistema operacional Unix que não fosse o desenvolvido em Berkeley.

O sistema X pode ser também implantado sobre outros sistemas operacionais, o que, por exemplo, já foi realizado para o sistema VAX/VMS segundo POUNTAIN [9] e IBM-PC/DOS segundo GETTYS [10]. Entretanto, o transporte do sistema X para computadores com sistemas operacionais que não possuam facilidades para a comunicação entre processos, suporte para a comunicação através de rede de computadores e capacidade para executar múltiplas tarefas, é bem mais custosa do que para aqueles que possuam tais características, conforme ANGEBRANNDT [11].

Isto ocorre, porque o sistema X utiliza para implementar o modelo cliente-servidor, rotinas oferecidas pelo sistema operacional Unix 4.3 BSD. Estas rotinas do sistema implementam protocolos para a comunicação entre processos e entre sistemas computacionais interligados através de rede. O sistema X Window original está

preparado para utilizar rotinas do sistema que implementam três famílias de protocolos para comunicação [11]: o "Unix domain" para a comunicação local, ou seja, quando o cliente e o servidor estão num mesmo computador; o TCP/IP e o DECNET para a comunicação externa, quando o cliente e o servidor estão em máquinas distintas.

Utilizando as rotinas de comunicação oferecidas pelo sistema operacional, o sistema X se torna independente do protocolo que está sendo utilizado para a comunicação do cliente com o servidor. Nos sistemas que não oferecem rotinas para a comunicação entre processos e entre computadores, será necessário criá-las de alguma forma para que se possa fazer a implementação do sistema X.

Além das rotinas para comunicação oferecidas pelo sistema Unix 4.3 BSD, o sistema X utiliza um grande número de outras rotinas oferecidas por este sistema operacional como suporte durante a execução de um processo. Estas rotinas muitas vezes não são padrão entre outros sistemas Unix-like. Desta forma, ao se transportar e adaptar o sistema X para outros sistemas operacionais Unix-like, é necessário criar novas rotinas no sistema que irá suportar o X Window e/ou modificar o código do sistema X nos trechos em que são chamadas algumas rotinas oferecidas pelo sistema operacional.

CAPÍTULO III

ARQUITETURAS PARA UM TERMINAL X

A idéia de um Terminal X tem como finalidade básica dotar um sistema computacional ou uma rede de computadores de um serviço de exibição gráfico. O nome Terminal X origina-se do inglês "X Terminal" que por sua vez vem da expressão "X Window Terminal". Essa expressão confunde-se com outras como "X Window Display Station" ou "Network Display Station" [12] na intenção de representar a finalidade básica descrita de um Terminal X.

Os Terminais X surgiram nos EUA no final de 1988 [12] e situam-se entre os terminais de texto alfanuméricos, semigráficos ou gráficos, e as estações de trabalho com ou sem disco. Seu surgimento só foi possível graças a grande difusão, através dos usuários de sistemas computacionais, do sistema X Window. Os Terminais X se baseiam na arquitetura deste sistema para serem implementados.

Uma das maiores vantagens dos Terminais X é proporcionar ao usuário um ambiente similar ao de uma estação de trabalho gráfica possuindo no entanto complexidade de hardware e custo inferiores [13]. Utilizando um Terminal X, um usuário pode entrar na rede a qual o terminal está ligado e ter acesso às diversas

aplicações em diferentes "hosts", exibindo-as em múltiplas janelas na tela do monitor gráfico do terminal.

III.1. Motivação para o Desenvolvimento dos Terminais X

A grande utilização do sistema X Window como interface com o usuário de computadores, principalmente aqueles que utilizam sistema operacional com filosofia Unix, fez com que fosse colocado em prática o conceito da utilização de um ambiente gráfico distribuído. O usuário de um determinado sistema, que esteja ligado a uma rede de computadores e que interage com seu computador através do sistema X, pode executar aplicações gráficas em outros sistemas que estão ligados à rede e exibir os resultados no seu computador. Isto é possível pois a utilização do modelo cliente/servidor pelo sistema X Window permite que a aplicação possa ser executada em sistemas distintos ao sistema que executa o servidor X e que irá exibir os resultados da aplicação.

Este tipo de ambiente distribuído, é desejável quando se está ligado a uma rede através de uma estação de trabalho com recursos gráficos, de baixa capacidade de processamento, e a aplicação gráfica exige, para ser executada de forma viável, um sistema computacional com desempenho maior do que o da estação. Neste caso, utiliza-se os sistemas de maior capacidade de processamento que estejam ligados à rede para a execução da aplicação. Através do protocolo X, a aplicação se

comunica com o servidor X que é executado na estação de trabalho e exibe os resultados nesta.

Esta forma de se utilizar as estações de trabalho com capacidade gráfica, fez com que as estações passassem a ser utilizadas com duas finalidades práticas. A primeira, na qual a aplicação gráfica não é muito complexa, a estação pode executar ao mesmo tempo, tanto o cliente/aplicação como o servidor gráfico. A segunda finalidade, na qual o cliente/aplicação é bastante complexo e exige um sistema de maior capacidade de processamento para executá-lo, a estação de trabalho funciona praticamente como um terminal/servidor ligado à rede. Este servidor recebe através da rede, dados enviados por outro sistema para serem exibidos na sua tela gráfica. A utilização do sistema X Window no segundo caso é fundamental pois a padronização do protocolo X permite que qualquer sistema que possua a biblioteca X implantada, possa ser utilizado para a execução da aplicação. Além disto, qualquer estação de trabalho que esteja executando o servidor X, pode ser utilizada para exibir os resultados de qualquer aplicação.

A utilização das estações de trabalho, em certas aplicações, como terminais/servidores gráficos de uma rede de computadores, motivou a idéia da criação de um sistema mais simples que a própria estação. Este sistema teria como finalidade ser um servidor gráfico dentro do sistema X Window e receberia requisições de serviços através do

protocolo X transmitido pela rede a qual o servidor estivesse ligado [14]. A sua simplicidade residiria no fato de não ser necessário um sistema operacional para controlá-lo. O programa servidor seria o único a ser executado pelo hardware deste sistema gráfico. Além disto, sob o ponto de vista de hardware, como não seria necessário executar diversos processos, não haveria o hardware complexo necessário para a implementação de sistemas computacionais de uso geral. Não existiria inclusive a necessidade de memória secundária, pois o programa servidor gráfico poderia ser carregado a partir de uma ROM ou através da rede [15 - 16].

A esses terminais/servidores do sistema X Window foi dado o nome de Terminais X ("X Window Terminals" ou "X Terminals"). Sua simplicidade permite que tenham um custo menor em relação às estações de trabalho, facilitando sua difusão. Com isto, pode-se ter uma quantidade maior de servidores gráficos dentro de um ambiente distribuído através de uma rede de computadores, aliviando inclusive o trabalho de exibição das estações ligadas a esta rede.

III.2. Definição de Arquiteturas para Terminais X

III.2.1. Arquitetura Tradicional Usando Rede

A arquitetura tradicional de um Terminal X origina-se naturalmente da arquitetura de uma estação de trabalho simplificada, de acordo com o que foi descrito no item

anterior. Um Terminal X, nesta arquitetura, consiste basicamente de um processador e memória local executando as funções necessárias para dar suporte ao sistema X Window e para suportar a comunicação através de uma rede de computadores [15]. Além disto, possui um monitor de média ou alta resolução, monocromático ou colorido, um teclado, um dispositivo de apontamento e posicionamento ("mouse"), e uma interface que permita a conexão a redes de computadores, normalmente através do padrão Ethernet. Pode também possuir um processador auxiliar para realizar as tarefas de exibição gráficas.

Tal como as estações de trabalho sem disco, os Terminais X também não o possuem e, além disto, não são dotados de um sistema operacional. Assim sendo, também de forma similar às estações sem disco, a carga do programa necessário para gerenciar e executar suas funções é feita através da rede. Para isto, o Terminal X se comunica com um computador, através de um programa de carga do sistema, que deve estar preparado para fornecer o programa servidor necessário ao terminal. O nome (endereço) deste computador é solicitado durante a inicialização dos Terminais X. Uma forma alternativa para a carga deste mesmo programa servidor é através da utilização de ROM, quando é feita uma cópia do programa armazenado na ROM para a memória de trabalho do Terminal X [16].

Basicamente um Terminal X executa o servidor do sistema X e é capaz de se comunicar com vários sistemas

computacionais interligados em rede, através de um protocolo padrão, normalmente o TCP/IP ou DECNET, implementado em um meio Ethernet. É em cima do protocolo padrão utilizado na rede que são trocadas as mensagens do protocolo X entre o cliente/aplicação, que está sendo executado em determinado computador, e o servidor, no Terminal X. O fato da comunicação com outros computadores ser feita normalmente através de Ethernet, onde a taxa de transmissão de dados é da ordem de 10 Mbps, é fundamental para o desempenho do sistema X quando executado de forma distribuída através dos Terminais X. A velocidade com que as mensagens são trocadas entre o cliente/aplicação e o servidor e, conseqüentemente, a velocidade com que a aplicação exibe os resultados, depende diretamente da taxa de transmissão do meio de comunicação entre os Terminais X e os sistemas computacionais que executam a aplicação. Quanto maior for esta taxa, menor será o tempo entre a execução da aplicação e a visualização dos seus resultados em um Terminal X.

III.2.2. Arquitetura Alternativa Usando Interface Serial

Uma arquitetura alternativa para um Terminal X é similar à arquitetura tradicional, utilizando-se um processador e memória local, além de um monitor, de um teclado e de um "mouse". Porém, nesta arquitetura, ao invés de se utilizar a interface para a interligação do Terminal X a outros sistemas computacionais baseada na conexão através de rede, utiliza-se uma interface serial

do tipo RS232-C para interligá-lo diretamente a um único sistema computacional [15]. Este sistema é que está ligado através de uma rede a outros sistemas. As aplicações distribuídas através da rede, se comunicam com o Terminal X através deste sistema computacional.

O servidor X é separado em duas partes. Uma sendo executada no Terminal X e a outra executada no computador ao qual o terminal está ligado. A divisão é feita de tal forma que através da linha de comunicação serial que liga o Terminal X ao computador, só passam informações necessárias para a exibição gráfica na tela do monitor do Terminal X e sinalizações de entradas provenientes deste [15].

Nesta arquitetura, para que não fique muito lenta a apresentação do resultado de uma aplicação na tela do Terminal X, é necessário que o terminal possua capacidade de receber através da linha serial comandos gráficos de alto nível. Isto ocorre, pois caso o Terminal X receba somente comandos gráficos para o traçado de pixel por pixel da tela a ser exibida, a passagem pela linha serial de uma imagem de, por exemplo, centenas de milhares de pixels levaria dezenas de segundos. Desta forma, o processador do Terminal X deve executar um programa capaz de receber comandos gráficos do tipo trace reta, preencha área, desenhe padrão e outros, para diminuir o tempo de traçado de imagens. Normalmente, coloca-se junto ao processador de uso geral do Terminal X, um processador

gráfico dedicado ao traçado de primitivas gráficas. Com a utilização do processador gráfico, as primitivas gráficas são transmitidas, através da linha serial, no formato próprio para o processador.

Esta arquitetura para a implementação de Terminais X possui uma vantagem quanto a complexidade e o custo em relação a arquitetura tradicional. Por não possuir interface para a conexão a redes de computadores nem as rotinas para implementar esta conexão através de algum protocolo padrão, os Terminais X implementados através da arquitetura alternativa têm um menor custo e um hardware menos complexo do que aqueles implementados pela arquitetura tradicional. Sua aplicação típica é em sistemas computacionais de médio porte que não possuam recursos gráficos e que estejam ligados à redes de computadores. A conexão de um Terminal X a interfaces seriais destes sistemas, permitiria ao usuário do sistema o acesso a um ambiente gráfico distribuído através do sistema X Window podendo executar múltiplas tarefas e exibí-las em janelas na tela do monitor gráfico do Terminal X.

III.3. Arquitetura Utilizada no Terminal X Implementado

O objetivo final deste trabalho é a implementação de um Terminal X, baseado na arquitetura tradicional descrita no item III.2.1, para o sistema operacional Plurix, um sistema com filosofia Unix desenvolvido no NCE/UFRJ. Para

isto, dois fatores básicos e fundamentais levaram a divisão do trabalho de implementação em duas etapas. Primeiro a implementação do Terminal X com a arquitetura alternativa, utilizando interface serial, e a seguir a evolução para o objetivo final, o terminal baseado na arquitetura tradicional, utilizando rede de computadores.

O primeiro fator responsável por esta divisão é que o sistema operacional Plurix não possui, até a presente data, chamadas ao sistema que permitam a conexão do computador onde o Plurix está sendo executado a um outro sistema, utilizando um protocolo padrão, por exemplo o TCP/IP. As rotinas para permitir esta conexão usando o protocolo citado, começaram a ser definidas enquanto este trabalho vinha sendo desenvolvido.

Além deste, o outro fator que gerou a divisão do trabalho foi que o supermicrocomputador desenvolvido no NCE/UFRJ para executar o sistema operacional Plurix, o Pegasus-32X, não possui nenhuma interface de comunicação com alta taxa de transmissão. O Pegasus-32X, um projeto desenvolvido em 1984, só possui como forma básica para comunicação com o meio exterior, interfaces seriais padrão RS232-C. Da mesma forma que o protocolo de comunicação do Plurix, durante este trabalho começou a ser definida uma estação de trabalho de alta capacidade de processamento, que irá utilizar como sistema operacional o Plurix e que possuirá uma interface de comunicação padrão Ethernet, permitindo a transmissão de dados em taxas de até 10 Mbps.

Somente após a implementação desta estação de trabalho é que será possível interligar o Terminal X a um sistema computacional, que execute o Plurix e que possua uma interface de comunicação padrão Ethernet.

Os dois motivos anteriores, levaram então a que este trabalho passasse por uma etapa intermediária na qual o Terminal X se baseia na arquitetura alternativa, utilizando interface serial, descrita no item III.2.2. A arquitetura utilizada na implementação tem ainda uma variação em relação ao que foi definido neste item uma vez que, por não ser possível ligar o Pegasus-32X a redes de computadores, não é possível distribuir os clientes/aplicações por outros sistemas computacionais. Desta forma, esses clientes devem ser executados também no Pegasus.

A definição do hardware do Terminal X incluindo um processador gráfico, capaz de executar diversas primitivas gráficas, tornou viável a implementação do terminal utilizando a arquitetura alternativa. O servidor do sistema X Window foi dividido em duas partes, de acordo com a definição desta arquitetura, sendo uma delas executada no Pegasus-32X e a outra no hardware do Terminal X, conforme é mostrado na figura (III.1). A comunicação entre as duas partes se restringe somente a comandos gráficos e a sinalizações de entrada provenientes do hardware do Terminal X. Os comandos gráficos utilizados são transmitidos a nível das primitivas gráficas que o

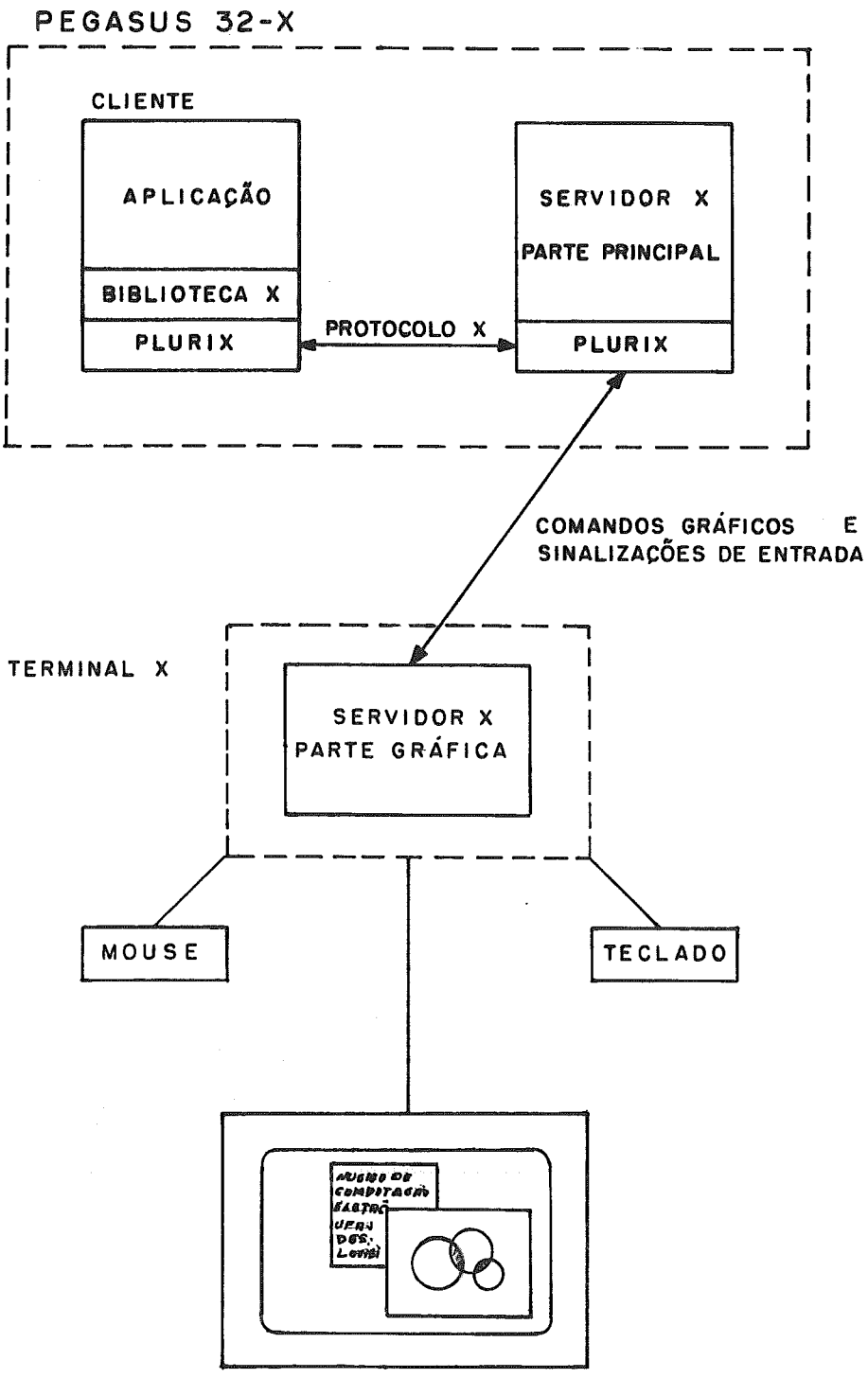
PEGASUS 32-X
CLIENTE
APLICAÇÃO
BIBLIOTECA X
PLURIX
PROTOCOLO X
SERVIDOR X
PARTE PRINCIPAL
PLURIX
COMANDOS GRÁFICOS E
SINALIZAÇÕES DE ENTRADA
TERMINAL X
SERVIDOR X
PARTE GRÁFICA
MOUSE
TECLADO

FIGURA III.1

processador gráfico executa. Não é necessário desta maneira transmitir ponto a ponto as imagens a serem exibidas na tela do monitor do terminal, e com isto, diminui-se o fluxo de informações passadas entre o Pegasus e o Terminal X. Desta forma, é possível interligar as duas partes do servidor X através de uma linha de comunicação serial com velocidade de 9600 bps. Essa taxa de transmissão é a taxa máxima das interfaces seriais existentes no Pegasus-32X.

A segunda etapa só se tornará possível, ao ser implementada tanto a estação de trabalho com interface de comunicação padrão Ethernet, como o protocolo padrão TCP/IP para conexão do Plurix a redes de computadores. A implementação da primeira etapa foi feita de forma mais transparente possível quanto a arquitetura utilizada, para que a passagem para a segunda etapa possa ser feita sem ser necessário redefinir todo o trabalho desenvolvido na primeira etapa.

CAPÍTULO IV

IMPLEMENTAÇÃO DO CLIENTE

O trabalho de implementação do cliente do sistema X Window consiste fundamentalmente em implantar a biblioteca X no sistema computacional onde o cliente/aplicação irá ser executado. A biblioteca X, que implementa a interface de programação de baixo nível dentro do sistema X Window, fornece as rotinas necessárias para que o cliente/aplicação faça a conexão com o servidor X desejado e solicite todas as tarefas necessárias para a exibição de seus objetivos.

A implantação da biblioteca X dentro de um sistema computacional, permite a implementação de diversos clientes dentro deste sistema. Cada programa cliente deverá, depois de compilado, ser ligado às rotinas da biblioteca X necessárias para o sua execução. O procedimento é similar ao das bibliotecas existentes para as diversas linguagens de programação, utilizadas pelos programas escritos nas respectivas linguagens.

A biblioteca X é fornecida junto às demais partes do sistema X Window original. Todas as rotinas oferecidas pela biblioteca X estão escritas na linguagem de programação C e utilizam diversas rotinas, inclusive as de

chamada ao sistema, existentes na biblioteca C do sistema Unix 4.3 BSD. O trabalho de implantação da biblioteca X consiste na adaptação de suas rotinas à biblioteca C existente no sistema operacional sobre o qual ela será implantada, seguido da compilação dos diversos módulos que formam a biblioteca X.

Neste trabalho, conforme foi explicado no capítulo anterior, o sistema computacional utilizado para executar os clientes do sistema X foi o Pegasus-32X rodando o sistema operacional Plurix. A implantação da biblioteca X no sistema Pegasus/Plurix, foi realizada considerando-se como premissa fundamental modificar as rotinas da biblioteca original o mínimo possível. Essa premissa foi considerada, pois criando-se no Plurix um ambiente capaz de suportar a versão atual utilizada da biblioteca X, o custo de tempo para a implantação das futuras versões será mínimo, desde que estas não possuam modificações estruturais. Além disto, clientes/aplicações desenvolvidos em outros sistemas utilizando a biblioteca X implantada naqueles sistemas, devem ser transportados para o sistema Pegasus/Plurix sem ser necessário realizar alterações no respectivo código. Isto só é garantido se as bibliotecas de todos estes sistemas forem essencialmente idênticas. Possíveis modificações só serão necessárias nas chamadas ao sistema feitas diretamente pelas aplicações, caso haja diferenças entre os sistemas operacionais.

IV.1. Implementação da Comunicação com o Servidor X

A primeira etapa da implantação da biblioteca X no sistema Pegasus/Plurix consiste em adaptar para este sistema, o mecanismo original de comunicação do cliente com o servidor X. A biblioteca X original possui dois módulos onde estão todas as rotinas responsáveis pela comunicação entre o cliente e o servidor:

- XConnDis.c;
- XlibInt.c.

Destes dois módulos, somente o módulo XConnDis.c possui chamadas a rotinas do sistema operacional que tratam com o protocolo de comunicação a ser utilizado para o diálogo entre o cliente e o servidor. O módulo XConnDis.c é o responsável pela implementação da conexão do cliente com o servidor. As demais trocas de mensagens entre o cliente e o servidor, para as quais utiliza-se as rotinas do módulo XlibInt.c, são feitas de forma transparente quanto ao tipo de protocolo utilizado. Isto acontece, pois o sistema operacional fornece um descritor através do qual é possível o acesso ao canal de comunicação criado pela conexão do cliente com servidor. O cliente passa a receber e enviar mensagens para o servidor através deste descritor. Cabe ao sistema operacional formatar as mensagens que passam no canal de acordo com o protocolo utilizado para a criação do mesmo.

IV.1.1. Implantação do Módulo para Conexão

O módulo XConnDis.c original está preparado para utilizar rotinas do sistema Unix 4.3 BSD que implementam a

conexão do cliente com o servidor através dos protocolos "Unix domain", para a comunicação local, TCP/IP ou DECNET, para a comunicação remota. A opção quanto ao tipo de protocolo a ser utilizado ocorre na compilação deste módulo, definindo-se para o pré-processador qual dos "#ifdef" existentes para cada protocolo será utilizado. A escolha de um determinado protocolo implica na utilização de chamadas a rotinas do sistema que utilizam este protocolo para realizar a conexão do cliente com o servidor. Estas rotinas recebem normalmente como parâmetro, estruturas pré-definidas para cada protocolo utilizado. Estas estruturas são definidas, pelo sistema Unix 4.3 BSD, através de arquivos públicos de inclusão ("*.h") e seus membros representam dados inerentes a cada protocolo.

Para a realização do trabalho de implantação do módulo XConnDis.c no sistema Pegasus/Plurix, baseado na premissa já citada de não modificar a biblioteca X original, adotou-se a seguinte metodologia, que será detalhada posteriormente. Primeiramente, foi necessário fazer a escolha de uma das três opções oferecidas por este módulo quanto ao tipo de protocolo. Feita a escolha, foi necessário fazer um estudo de como se realiza a conexão entre o cliente e o servidor utilizando o protocolo escolhido. Isto foi feito através da análise do funcionamento das rotinas do sistema Unix 4.3 BSD chamadas pelo módulo XConnDis.c para implementar a conexão cliente-servidor no protocolo escolhido. Depois de feita a

análise, criaram-se as rotinas que dentro do ambiente Pegasus/Plurix emulam as chamadas ao Unix, permitindo a conexão do cliente com o servidor X.

Como o Plurix não implementa nenhum dos três protocolos mencionados, a escolha quanto ao tipo de protocolo pode ser feita de forma arbitrária. Dentre as opções existentes, foi escolhida a de trabalhar como se fosse possível utilizar o protocolo TCP/IP. Esta escolha se deve unicamente ao fato simbólico de ser para este protocolo que pretende-se desenvolver as rotinas de comunicação do Plurix.

Para o protocolo TCP/IP, o módulo XConnDis.c utiliza as chamadas ao sistema Unix 4.3 BSD **socket, connect, gethostname, gethostbyname,** durante o processo de conexão com o servidor X. Estas chamadas ao sistema possuem, dentro do contexto do protocolo TCP/IP e do sistema X Window, as seguintes funções [6 - 8]:

. **socket** - cria um dos extremos do canal de comunicação a ser utilizado para o diálogo com o servidor X; o outro extremo do canal fica aberto até que seja feita a conexão com o servidor; devolve o descritor que identificará este canal;

. **connect** - recebe como parâmetros o descritor do canal criado pela chamada socket e o endereço do computador onde está sendo executado o servidor X; tenta associar o outro extremo do

canal indicado pelo descritor ao computador cujo endereço foi recebido como parâmetro; se bem sucedida completa a conexão entre o cliente e o servidor;

. **gethostbyname** - recebe como parâmetro o nome de um "host" e devolve o endereço Internet desse "host"; utilizada para a identificação do endereço do computador, a partir de seu nome, no qual o servidor X está sendo executado;

. **gethostname** - devolve o nome do "host" no qual o processo que a chamou está sendo executado; é chamada pelo cliente quando este não recebeu o nome do "host" para fazer a conexão com o servidor X e então assume o "host" que o está executando.

Feita análise das rotinas do Unix 4.3 BSD utilizadas para implementar a conexão cliente-servidor no sistema X original, pôde-se partir para a implementação das rotinas para emular as funções das chamadas ao Unix, dentro do ambiente Pegasus/Plurix. Como foi mencionado anteriormente, o sistema Pegasus/Plurix não oferece mecanismos que possibilitem a sua conexão a redes de computadores. O sistema operacional Plurix não fornece chamadas ao sistema que possibilitem a comunicação com outros sistemas através de algum protocolo padrão e o computador Pegasus-32X não possui uma interface, tipo Ethernet ou similar, que torne viável sua ligação a redes

computacionais. Assim sendo, devido a impossibilidade de ligar o sistema Pegasus/Plurix a redes e a conseqüente inviabilidade de distribuir os clientes por outros sistemas computacionais, partiu-se para a implementação do cliente no mesmo sistema que o servidor, no caso o Pegasus/Plurix.

O fato do cliente e do servidor estarem no sistema Pegasus/Plurix, significa que serão dois processos sendo executados simultaneamente por este sistema. Desta forma, é necessário a existência de um mecanismo que permita a intercomunicação entre dois processos simultâneos. Através deste mecanismo é que será feita a troca de mensagens entre o cliente e o servidor.

O sistema operacional Plurix não oferece mecanismos diretos que permitam a criação de canais bidirecionais para a comunicação entre processos. Foi necessário então criar, dentro dos recursos disponíveis, um meio que permitisse a comunicação entre o cliente e o servidor X dentro do ambiente Pegasus/Plurix.

Uma das duas formas possíveis encontradas, para implementar a comunicação entre os dois processos, foi através de arquivos tipo "FIFO" ("First In - First Out")[17]. Como "FIFO" é um mecanismo de comunicação unidirecional, é necessário a criação de dois "FIFO's", um para a comunicação na direção cliente-servidor e outro para direção oposta. Esta opção foi descartada, pois seria

necessário que o cliente trabalhasse com dois descritores, um para cada "FIFO". Como a biblioteca X possui suas rotinas originais para comunicação estruturadas em função de apenas um descritor para o canal de comunicação cliente-servidor, seria necessário uma alteração substancial no código destas rotinas. A própria rotina responsável pela realização da conexão do cliente com servidor devolve como resposta, o valor do descritor do canal bidirecional criado através da conexão. A alteração desta e de outras rotinas para suportar dois descritos iria contra a premissa básica utilizada nesta implementação de não modificar a biblioteca X original.

A outra forma encontrada foi a utilização de duas interfaces de comunicação seriais do Pegasus interligadas. Através deste mecanismo, o cliente envia e recebe mensagens por intermédio de uma linha serial. O servidor X realiza o mesmo utilizando outra linha serial. O fio de transmissão de uma linha está ligado no fio de recepção da outra e vice-versa. No Plurix, é necessário somente um descritor para realizar operações de leitura e escrita em linhas de comunicação seriais. Isto ocorre, pois linhas seriais são canais de comunicação bidirecionais com acesso através de um único dispositivo físico. Esta forma é compatível com as rotinas da bibiloteca X original, e poderia ser utilizada.

Este tipo de implementação permite ainda a simulação de um ambiente gráfico distribuído. A partir do momento

que se tenha um outro sistema computacional executando o Plurix, pode-se implantar naquele sistema a biblioteca X e, através de uma de suas interfaces seriais, ligá-lo ao Pegasus. Desta forma, pode-se criar um cliente/aplicação em outro sistema enquanto o servidor é executado no Pegasus. Esta configuração poderia melhorar o desempenho da aplicação executada, uma vez que o cliente e o servidor estariam sendo executados em sistemas computacionais distintos e de forma paralela.

O único problema da implementação utilizando interfaces seriais, é que a taxa de transmissão máxima dessas interfaces no Pegasus é de 9600 bps. Isto prejudicaria a comunicação entre o cliente e o servidor. Porém, consideramos esta solução transitória. Como já foi mencionado, durante este trabalho foi iniciado o desenvolvimento de um ambiente que irá possibilitar o uso do Plurix para se ter acesso a redes de computadores com alta taxa de transmissão. Por isto, e por ser a única solução compatível e disponível para o momento, a implementação no Plurix da comunicação entre os processos cliente e servidor foi realizada utilizando-se interfaces seriais.

Concebido o mecanismo para intercomunicação entre os processos cliente e servidor, a implementação da rotina que emula a chamada **socket** dentro do ambiente Pegasus/Plurix consistiu na abertura (chamada ao sistema **open**) de um canal serial "tty" em modo "raw" tanto para

transmissão como para recepção. O modo "raw" é necessário pois o sistema não deve tratar os bytes transmitidos e recebidos no "tty" utilizado. A chamada implementada devolve o descritor do canal "tty" aberto. O canal "tty" aberto não deve possuir **login,** uma vez que deve ser controlado exclusivamente pelo processo cliente.

A chamada **connect** perdeu, dentro desta implementação, sua função pois a conexão com o outro extremo do canal de comunicação é feita fisicamente. Uma vez enviada a mensagem pelo cliente, o servidor automaticamente a receberá pois deverá estar "escutando" o outro extremo do canal. A rotina **connect** implementada retorna o valor zero indicando que não houve erro na sua execução, para manter compatibilidade com a rotina original do Unix 4.3 BSD.

As chamadas **gethostname** e **gethostbyname** perdem também seu sentido, pois nesta implementação o cliente não precisa saber o endereço nem o nome do sistema onde o servidor X está sendo executado. Estes dados são necessários na implementação original, pois a rotina **connect** utiliza essas informações para completar a conexão com o servidor. As rotinas implementadas devolvem respectivamente o nome "Pegasus" e um ponteiro para uma estrutura, simplesmente para manter a compatibilidade a nível do código original do módulo XConnDis.c

IV.1.2. Implantação do Módulo para Troca de Mensagens

O módulo da biblioteca X responsável por fornecer as rotinas para a troca de mensagens entre o cliente e o servidor é, como foi mencionado anteriormente, o XlibInt.c. Estas rotinas utilizam basicamente duas chamadas ao sistema Unix 4.3 BSD, a **readv** e a **writev,** para implementar a comunicação do cliente com o servidor.

A função destas duas chamadas ao sistema é, dentro do contexto da comunicação do cliente com o servidor, receber e transmitir mensagens. Para executar esta tarefa elas recebem como parâmetro o descritor obtido durante a conexão do cliente com o servidor. Este descritor permite o acesso, através do sistema operacional, ao canal criado para a comunicação. É importante lembrar que as mensagens recebidas e enviadas através das chamadas **readv** e **writev,** são totalmente transparentes quanto ao protocolo implementado no canal de comunicação. Cabe ao sistema, através do descritor recebido, identificar o protocolo utilizado e tratar os bytes que fluem na comunicação cliente-servidor de acordo com este protocolo.

O sistema operacional Plurix não possui as chamadas **readv** e **writev** sendo necessário implementá-las e colocá-las na biblioteca C do Plurix. Estas rotinas realizam a mesma ação das chamadas **read** e **write** existentes no Plurix. A diferença é que a **read** e a **write** recebem como parâmetros, além do descritor mencionado, um ponteiro para um buffer e o número de bytes que devem ser lidos ou escritos neste buffer, enquanto a **readv** e a **writev** recebem

um vetor de estruturas cujos membros são buffers para ler ou escrever [8]. A implementação da **readv** e da **writev** foi baseada na utilização das duas chamadas similares fornecidas pelo Plurix.

IV.2. Modificações Realizadas no Ambiente Plurix

IV.2.1. Modificações nos Compiladores do Plurix

Os módulos da biblioteca X original possuem uma grande quantidade de identificadores de rotinas e variáveis que só diferem entre si a partir do décimo quarto caractere. O pré-processador, o compilador C, o montador e o editor-ligador construídos para o Plurix, não eram capazes de diferenciar esses identificadores. Só reconheciam como identificadores distintos aqueles com diferenças nos treze primeiros caracteres.

Para que não fosse necessário alterar os identificadores de grande parte das rotinas e das variáveis da biblioteca X original, foi necessário fazer modificações no pré-processador, no compilador C e no montador para que passassem a reconhecer como identificadores distintos aqueles com diferenças nos trinta e um primeiros caracteres. Esta modificação foi suficiente para que esses programas diferenciassem todos os identificadores existentes nos módulos da biblioteca X.

Porém, o problema continuou existindo em relação ao

editor-ligador de módulos objeto do Plurix. Algumas rotinas internas definidas e utilizadas em módulos distintos da biblioteca X original, têm os seus identificadores diferenciados a partir do décimo quarto caractere. Na criação dos programas-objeto destes módulos, são gerados pelo montador identificadores idênticos para essas rotinas. Isto ocorre, pois o editor-ligador de programas-objeto do Plurix resolve referências a símbolos globais com identificadores de, no máximo, treze caracteres. Certamente o editor-ligador do Unix 4.3 BSD trabalha com identificadores de, pelo menos, trinta e um caracteres.

O padrão ANSI da linguagem C recomenda que o número mínimo de caracteres considerados em identificadores de símbolos globais (aqueles que o editor-ligador recebe) seja de seis. Como o editor-ligador do Plurix supera esta recomendação e o número de símbolos globais que não tiveram seus identificadores reconhecidos como distintos foi pequeno, o problema foi resolvido sem se alterar o editor-ligador. Foram feitas definições para o pré-processador substituindo os identificadores das rotinas que se enquadravam neste caso por identificadores similares, porém com diferenças nos treze primeiros caracteres. Essas definições foram inseridas em um arquivo de inclusão utilizado por todos os módulos que compõem a biblioteca X.

Foi necessário resolver também uma incompatibilidade

provisória entre o código da linguagem C apresentado nos módulos da biblioteca X original e o código analisado pelo compilador do Plurix. No início deste trabalho, não havia sido ainda implementado no compilador a geração de código para atribuições feitas entre estruturas. Necessitou-se mudar todas as atribuições de estruturas existentes no código original, por chamadas à rotina do sistema para a cópia física das regiões de memória onde estavam situadas as estruturas em questão. Somente algum tempo depois de implantada a biblioteca X no Plurix é que o compilador passou a aceitar este tipo de atribuição.

IV.2.2. Inclusão de Rotinas na Biblioteca C

Além das chamadas ao sistema, mencionadas no item IV.1, necessárias para implementar a comunicação do cliente com o servidor, mais uma rotina foi implementada e incluída na biblioteca C do Plurix.

Esta rotina, chamada **ffs,** é utilizada pelo módulo XOpenDis.c da biblioteca X. Sua função é devolver o número de ordem do primeiro bit menos significativo, de uma máscara de bits, que está em 1 (um).

Outras duas rotinas, a **bcopy** e a **bzero,** também são utilizadas por módulos da biblioteca X e não existem na biblioteca C do Plurix. Porém, elas podem ser substituídas pelas rotinas do Plurix **memcpy** e **memset** respectivamente [17], que realizam funções idênticas. A substituição foi

realizada fazendo-se definições para o pré-processador de troca dos identificadores dessas rotinas. Essas definições foram inseridas em um arquivo de inclusão utilizado por todos os módulos da biblioteca X.

IV.2.3. Implantação dos Arquivos de Inclusão

Diversos arquivos de inclusão ("*.h") necessários para a compilação dos módulos da biblioteca X não pertenciam ao ambiente Plurix. Alguns destes arquivos foram fornecidos junto a biblioteca X e outros pertenciam ao ambiente do sistema Unix 4.3 BSD.

Os arquivos fornecidos pela biblioteca X foram implantados nos respectivos diretórios em que são procurados pelos módulos da biblioteca que os incluem. Os arquivos pertencentes ao ambiente do sistema Unix tiveram que ser implementados a partir de suas definições obtidas em manuais deste sistema.

Além dos arquivos de inclusão, foi necessário implantar um arquivo fornecido junto a biblioteca X que serve como base de dados para a informação de eventuais erros ocorridos durante a comunicação do cliente com o servidor. Este arquivo é aberto pela rotina da biblioteca responsável pelo tratamento destes erros, para enviar a mensagem correspondente ao erro ocorrido. O caminho percorrido, para a abertura do arquivo, na árvore do sistema de arquivos é definido a nível de código. Por

isto, o arquivo deve ser colocado no diretório determinado pelo código da rotina.

O trabalho de implantação de todos os arquivos requisitados pelos módulos da biblioteca X nos seus respectivos diretórios, é de fundamental importância para atender a premissa de modificar o mínimo possível a biblioteca X implantada. Futuras versões desta biblioteca não necessitam de alterações já que encontram um ambiente no Plurix preparado para a biblioteca X.

IV.2.4. Incompatibilidades com os Utilitários do Plurix

Junto a biblioteca X original é fornecido um arquivo **Makefile** para a implantação da biblioteca X em sistemas compatíveis com o Unix 4.3 BSD. Este arquivo é interpretado por um utilitário deste sistema, denominado **make,** para a execução das tarefas necessárias na implantação da biblioteca.

O formato do arquivo **Makefile** fornecido possui incompatibilidades com o formato dos arquivos tipo **Makefile** reconhecidos pelo utilitário **make** do Plurix [17]. Com isto, foi necessário analisar os procedimentos descritos no arquivo **Makefile** original e implementar um novo arquivo **Makefile** para o Plurix.

Dentre os procedimentos descritos no arquivo **Makefile** original, existe um que utiliza o utilitário **awk** do Unix.

Este utilitário não foi implementado no sistema Plurix. Neste procedimento, o utilitário **awk** recebe como entrada uma rotina escrita em uma linguagem própria para ser interpretada pelo utilitário. O **awk** utiliza esta rotina para modificar um determinado arquivo de inclusão existente na biblioteca e gerar um novo arquivo de inclusão com um formato pré-definido.

Para implementação do procedimento descrito foi utilizado um outro sistema computacional com Unix, que possuia o utilitário **awk.** Foi necessário fazer pequenas mudanças na rotina recebida como entrada pelo **awk** utilizado, pois também existiam incompatibilidades quanto a linguagem interpretada por este **awk** e a linguagem interpretada pelo **awk** do sistema Unix 4.3 BSD.

IV.3. Alterações na Biblioteca X Original

Mesmo sendo o trabalho de implantação da biblioteca X feito de forma a não alterar o código original dos módulos desta biblioteca, algumas mudanças foram inevitáveis. Essas alterações foram realizadas tipicamente quando a biblioteca X utiliza recursos existentes no sistema Unix 4.3 BSD que possuem similares no sistema Plurix, mas que funcionam de forma diferente.

A grande maioria das mudanças concentrou-se nas chamadas ao sistema Unix que têm chamadas similares no sistema Plurix, porém são incompatíveis sob algum aspecto.

A incompatibilidade existiu em três casos básicos: divergência quanto ao nome da chamada, quanto ao nome dos parâmetros passados para a chamada e quanto a forma da chamada retornar o seu resultado.

Os dois primeiros casos foram resolvidos colocando-se em arquivos de inclusão utilizados por todos os módulos da biblioteca, definições para o pré-processador de substituição dos identificadores utilizados no Unix para aqueles utilizados no Plurix.

No terceiro caso, foi inevitável fazer alterações no código das rotinas que utilizam a chamada **ioctl** que, por ser independente de implementação, é incompatível quanto a forma de retornar o seu resultado. No Unix 4.3 BSD implementado num VAX, esta chamada recebe como parâmetro o endereço da posição de memória onde deve retornar seu resultado [8]. No Plurix, esta chamada retorna diretamente o valor de seu resultado [17], devendo haver uma atribuição do valor retornado à variável localizada no endereço de memória passado como parâmetro na chamada **ioctl** do Unix de Berkeley.

Foi necessário também fazer alterações em arquivos de inclusão tipicamente existentes na biblioteca X para concentrar todas as dependências quanto ao sistema utilizado dentro do universo dos Unix compatíveis. Essas alterações ocorreram por causa das variações dos nomes de alguns arquivos públicos de inclusão no universo dos

sistemas Unix compatíveis.

CAPÍTULO V

IMPLEMENTAÇÃO DO TERMINAL X

O desenvolvimento de um Terminal X divide-se em duas etapas: a implementação de uma base de hardware para fornecer os recursos necessários para o funcionamento do sistema X Window e a implementação do servidor do sistema X sobre o hardware criado, gerenciando os recursos existentes e comunicando-se com os clientes.

O hardware do Terminal X foi definido de forma a suportar a arquitetura utilizada neste trabalho, para a implementação do terminal. A implementação, conforme foi definido no item III.3, prevê a passagem do Terminal X por uma etapa intermediária, na qual o terminal se baseia na arquitetura alternativa, utilizando interface serial. Na etapa final, o terminal passa a ser implementado através da arquitetura tradicional, ligado diretamente a redes de computadores. A definição do hardware do Terminal X foi feita de maneira a permitir a evolução natural da arquitetura alternativa para a arquitetura tradicional, sem a necessidade de mudanças no trabalho implementado durante a etapa intermediária.

A implementação do servidor X foi feita de acordo com as características da arquitetura utilizada para a

implementação do Terminal X. O servidor foi dividido em duas partes, de forma a permitir sua implementação sobre a arquitetura alternativa. Uma parte do servidor foi desenvolvida sobre o sistema operacional Plurix executado no computador Pegasus-32X. Esta parte, apesar de estar no Pegasus, possui dependências quanto as características do hardware do Terminal X. A outra parte, executada sobre o hardware do terminal, é responsável pela interação direta com o hardware gráfico e com os dispositivos de entrada. Sua função básica é ligar os recursos de hardware oferecidos pelo Terminal X à parte do servidor executada no Pegasus.

O trabalho de implementação do servidor X foi realizado, analogamente ao desenvolvimento do hardware, de forma a permitir a passagem para a arquitetura tradicional sem a necessidade de redefinir o trabalho realizado. O protocolo utilizado para a comunicação das duas partes em que se dividiu o servidor, foi implementado de forma que para se juntar essas duas partes, o trabalho necessário consiste basicamente na eliminação deste protocolo. As rotinas que necessitam de serviços gráficos passam a ser ligadas diretamente aos procedimentos responsáveis pela execução dos serviços.

V.1. Definição do Hardware do Terminal X

A definição do hardware do Terminal X se baseou no hardware das estações de trabalho com recursos gráficos.

Foi a partir destas estações que surgiu a idéia dos Terminais X. Utilizou-se também, as características de Terminais X comerciais nos EUA [13] para a escolha de alguns componentes a serem utilizados no Terminal X implementado. No processo de escolha dos componentes de hardware, levou-se ainda em consideração o equilíbrio entre a eficiência e o custo.

Utilizou-se um processador de uso geral, o MC68000 da Motorola [18], ligado a uma memória local de 1 Mbyte. A utilização do MC68000, um processador com barramento interno de 32 bits e externo de 16 bits, levou em conta sua velocidade de processamento comparada ao custo associado. O MC68000, utilizado na freqüência de 10 MHz sem estados de espera no acesso à memória, é um processador de 0.83 Mips com um custo bastante reduzido. A quantidade de memória foi definida de forma a suportar a implementação do Terminal X baseado na arquitetura tradicional onde o servidor X é todo executado no próprio terminal. A estimativa de 1 Mbyte foi feita a partir de dados obtidos dos terminais comerciais americanos. Foi previsto a possível expansão da memória.

Foram implementadas quatro interfaces seriais. A primeira, para a comunicação com o Pegasus-32X. A segunda, para a ligação com um teclado alfa-numérico. O dispositivo de apontamento, o "mouse", ficaria ligado na terceira interface e a quarta seria reservada para futuras expansões. Todas as interfaces seriais podem atingir a

velocidade de 38.4 Kbps.

Com respeito a futura implementação do Terminal X na arquitetura tradicional, algumas previsões de hardware foram realizadas. Foi utilizado um circuito "timer" para implementar as funções que necessitam de contagem de tempo no servidor X. Essas funções são supridas na arquitetura alternativa, pelas rotinas de relógio fornecidas pelo sistema operacional Plurix. A parte do servidor X que ficará no Terminal X nesta arquitetura não utilizará o "timer" implementado.

Foi feita uma previsão no hardware desenvolvido para permitir a implementação de uma interface Ethernet. Esta interface não foi implementada, pois conforme mencionado no item III.3 não seria utilizada na primeira fase da implementação a comunicação através de rede. A implementação, por motivo de compatibilidade de circuitos, foi deixada para ser realizada quando existir um sistema computacional que utilize o Plurix e permita a conexão ao Terminal X através de interface Ethernet.

Na parte gráfica do Terminal X, foram colocados os conhecimentos mais recentes a respeito de circuitos eletrônicos existentes para a exibição de imagens em monitores através de varredura. A resolução escolhida foi de 1280 por 1024 pixels a serem exibidos na tela do monitor. Definiu-se ainda, que seria utilizada uma tabela de conversão de cores ("look-up table") com 256 posições.

Isto significa que a memória de imagem deve ter no mínimo oito planos de 1280x1024 bits. Na implementação, a memória de imagem teve oito planos de 2048x1024 bits. O número de planos da memória de imagem é definido como sendo sua profundidade e representa, através da utilização da tabela de cores, o número máximo de variações de cores que um determinado pixel da tela pode ter.

Para controlar a exibição no monitor das informações armazenadas na memória de imagem, foi utilizado um processador gráfico comercial, o ACRTC HD63484 da Hitachi [19]. Este processador não só realiza a tarefa de exibição como é capaz de executar diversas primitivas gráficas a partir de comandos recebidos. Pode-se traçar pontos, retas, círculos, elipses, áreas, polígonos e desenhar padrões através da passagem dos respectivos comandos junto com os parâmetros desejados [19]. Estas funções são diretamente usadas pelo sistema X Window e devem estar presentes em todos os servidores X implementados. A utilização de um processador para executar estes procedimentos não só diminui o tempo de execução, como também irá diminuir o fluxo de informações passadas entre o Pegasus e o Terminal X.

Conforme foi mencionado, a tabela para conversão de cores deve possuir 256 entradas. Na sua saída, tem-se a informação digital do código da cor selecionada. Esta informação é dividida em três grupos R, G e B ("red", "green" e "blue"). Foi determinado que se teria oito bits

de informação para cada um destes grupos. Isto nos dá a possibilidade de escolher 256 cores em um universo de 16 Mega (1024x1024) cores.

A informação que sai da tabela de cores deve ser convertida na informação analógica da intensidade com que devem ser "acesos" os fósforos vermelho, verde e azul na tela do monitor. Isto nos leva a três conversores digitais-analógicos com oito bits de entrada. Para a resolução desejada e de acordo com as características de monitores comerciais, a freqüência de conversão deve ser da ordem de 108 MHz.

Para implementar a tabela de cores e os conversores foi utilizado o circuito integrado Bt458 da Brooktree [20]. Este circuito possui internamente uma tabela de cores e três conversores que satisfazem as condições desejadas. O Bt458 deve ser alimentado com um relógio ("clock") com a freqüência de conversão acima mencionada. Foram utilizados circuitos integrados com tecnologia ECL para implementar esta função.

V.2. Implementação do Servidor X

O código do X Window, colocado em domínio público pelo MIT, fornece junto às demais partes do sistema, uma implementação de um servidor X. Esse servidor, denominado "sample server" ou servidor modelo , é dependente da arquitetura do sistema computacional com recursos gráficos

para a qual foi concebido. A finalidade desta implementação é ser usada como base para o desenvolvimento de novos servidores em sistemas com arquiteturas distintas [11 - 21].

A arquitetura do Terminal X difere da arquitetura para o qual o "sample server" foi desenvolvido em diversos pontos. Por exemplo, o Terminal X possui um processador dedicado capaz de traçar primitivas gráficas enquanto o servidor modelo realiza o cálculo destas primitivas por algoritmos de software e faz acesso diretamente a memória de imagem para traçá-las.

Além disto, existem as diferenças com relação ao sistema operacional. De forma similar à biblioteca X, o servidor modelo também foi desenvolvido para o sistema Unix 4.3 BSD. Os mesmos problemas mencionados no capítulo IV quanto à comunicação cliente-servidor e às incompatibilidades com o Plurix, aparecem no servidor modelo.

Apesar das diferenças, pela complexidade que envolve as diversas funções que são executadas pelo servidor, a implementação do servidor para o Terminal X partiu do modelo de servidor fornecido. A estrutura de como o servidor X é subdividido foi mantida. As partes do código do servidor modelo foram analisadas quanto a viabilidade de serem utilizadas no servidor a ser implementado. Aquelas nas quais foi detectada a compatibilidade, foram

aproveitadas. Outras foram parcialmente alteradas e as que interagem com a parte de exibição gráfica foram em grande parte reescritas.

V.2.1. Arquitetura do Servidor X

O servidor X modelo fornecido junto ao sistema X Window é dividido em quatro partes [11]:

- DIX ("Device Independent X server");
- OS ("Operating System");
- DDX ("Device Dependent X server");
- Interface para extensões.

Na camada DIX está o código do servidor modelo que pode ser utilizado integralmente em qualquer servidor que vier a ser implementado a partir do servidor modelo. Esta camada, executa as funções do servidor que são independentes do dispositivo que o executa. Isto significa que as rotinas implementadas na camada DIX utilizam as rotinas de outras camadas para fazer a comunicação com o sistema operacional e para interagir com o hardware gráfico.

A camada OS é a que interage com o sistema operacional do computador no qual o servidor está sendo executado. No caso do servidor modelo, o sistema é o Unix 4.3 BSD. Nesta camada, ficam concentradas a maioria das chamadas ao sistema operacional necessárias para que o servidor execute as funções de comunicação com os clientes

e acesso aos arquivos necessários.

Na camada DDX estão concentradas todas as dependências do servidor modelo quanto as características do hardware gráfico sobre o qual ele deve ser executado. Esta camada é responsável pela interação do servidor com o sistema operacional na intenção de realizar operações sobre o hardware gráfico. É importante lembrar, que o servidor modelo está preparado para interagir diretamente com a memória de imagem e isto é feito através do sistema operacional.

A interface para extensões define uma série de procedimentos padrões necessários para a implementação de novas funções no servidor modelo fornecido. Na realidade, o sistema X Window aceita a possibilidade de se criar extensões às suas funções básicas. Essas extensões devem ser feitas de forma padronizada nas diversas implementações de servidores X. Sem isto, um servidor alterado poderá não ficar compatível com uma aplicação desenvolvida para outro servidor, sem utilizar as novas funções. A interface para extensões no servidor modelo, fornecido junto ao sistema X Window, é especificada por FISHER [22].

No servidor implementado para o Terminal X, a camada DIX foi implantada no Plurix praticamente sem alterações. A camada OS, foi modificada de forma a atender a arquitetura do Terminal X e para ficar compatível com o

Plurix. Na camada DDX foram feitas diversas modificações em relação ao código fornecido pelo servidor modelo. Neste trabalho não foi necessário realizar a implementação de extensões para as funções básicas oferecidas pelo servidor modelo.

V.2.2. Implantação da Camada DIX no Terminal X

A camada DIX do servidor modelo foi implantada no Terminal X para ser executada no Pegasus/Plurix. O código das rotinas que compõem esta camada é praticamente independente do sistema operacional e do hardware gráfico. Somente as rotinas para gerenciamento de memória, **malloc, realloc** e **free,** e as rotinas **ffs, bcopy** e **bzero,** já mencionadas no item IV.2.2, são solicitadas à biblioteca C do sistema. O trabalho de implantação da camada DIX se resumiu a problemas idênticos aos já mencionados no item IV.2.

Problemas com relação aos compiladores do Plurix, já haviam sido resolvidos durante a implantação da biblioteca X. O problema do editor-ligador foi resolvido de forma similar ao que foi explicado no item IV.2.1. O mesmo ocorreu com relação à atribuição de estruturas, não implementada, no início deste trabalho, pelo compilador C do Plurix. Este problema apareceu igualmente nas rotinas das camadas OS e DDX, sendo resolvido da mesma forma.

Com relação à incompatibilidade do utilitário **make** do

Plurix e do Unix 4.3 BSD, um novo arquivo **Makefile** para compilar a camada DIX, teve de ser criado. O mesmo ocorreu para as camadas OS e DDX. A falta do utilitário **awk** foi novamente sentida durante a implantação da camada DIX. Dois arquivos pertencentes a esta camada, o Xatom.h e o initatoms.c, são gerados a partir da passagem através do **awk,** do arquivo BuiltInAtoms presente na mesma camada. Estes dois arquivos estão relacionados com a definição de **atoms** e não devem ser editados livremente.

Um **atom** para o sistema X Window é um identificador numérico único que corresponde a uma "string" de caracteres utilizada para identificação de, por exemplo, **tipos** de variáveis pertencentes ao sistema X. Mudanças nos **atoms** pré-definidos ou mesmo na ordem na qual eles aparecem nos arquivos gerados pelo **awk,** é equivalente a forçar uma nova versão "minor" para o servidor do sistema X. Por isso, existe a necessidade dos referidos arquivos serem gerados por um procedimento fixo escrito em linguagem interpretada pelo utilitário **awk.** Novamente recorreu-se a outro sistema computacional que possui o utilitário **awk** para gerar os arquivos Xatom.h e initatoms.c.

Na camada DIX estão as principais rotinas do servidor X. Pode-se destacar a rotina para iniciar todo o ambiente controlado pelo servidor, o "dispatcher" que implementa a política "round-robin" para distribuir as tarefas existentes, as rotinas que iniciam a execução das

requisições feitas ao servidor e as rotinas responsáveis pelo gerenciamento de todos os recursos oferecidos pelo servidor.

As rotinas da camada DIX estão agrupadas em módulos extensos, chegando vários deles a ter em média 3.000 linhas de código escrito na linguagem C. A compilação desses módulos foi realizada de forma a só serem compiladas as rotinas da camada DIX, que estavam sendo necessárias em cada instante da implementação deste trabalho. Os detalhes da metodologia utilizada para a compilação destes módulos serão descritos no próximo capítulo.

V.2.3. Implementação da Camada OS no Terminal X

A camada OS é a responsável pela interação do servidor com o sistema operacional. Essa interação se dá com o intuito do servidor realizar dois procedimentos básicos: a comunicação com os diversos clientes do sistema X Window e o acesso a arquivos existentes no sistema com a finalidade de obter informações contidas nestes arquivos. O trabalho de implementação da camada OS fornecida pelo servidor modelo, consiste na modificação e adaptação das rotinas que interagem com o Unix 4.3 BSD para a interação com o Plurix e com a arquitetura definida para o Terminal X.

. A Comunicação com os Clientes do Sistema X

Nos módulos connection.c, io.c e WaitFor.c da camada OS do servidor modelo, estão concentradas todas as rotinas utilizadas para fazer a conexão com novos clientes e, uma vez feita a conexão, realizar a comunicação com esses clientes.

O servidor modelo está preparado para realizar a conexão com novos clientes através do protocolo TCP/IP. Os demais protocolos, mencionados no capítulo anterior, não estão implementados no servidor fornecido. Assim sendo, o servidor modelo chama rotinas do sistema Unix 4.3 BSD que utilizam o protocolo citado, para realizar a conexão.

Como na primeira etapa da implementação do Terminal X não foi utilizado o protocolo TCP/IP, foi necessário alterar todas as rotinas que utilizam essas chamadas ao sistema para adaptá-las a arquitetura utilizada. Essa arquitetura, conforme foi definido no capítulo anterior, determina que o servidor irá se comunicar com o cliente, através de duas interfaces seriais do Pegasus interligadas.

A adaptação do servidor modelo à arquitetura do Terminal X foi feita de forma a preservar as características e a funcionalidade da implementação utilizando o protocolo TCP/IP. Com isto, a passagem para a segunda etapa prevista neste trabalho, na qual será utilizado o protocolo TCP/IP, será realizada mais rapidamente. Para atingir este objetivo, foi necessário

entender como é feita a implementação utilizando o TCP/IP, para depois fazer a adaptação à arquitetura utilizando linhas seriais.

Na inicialização do servidor é chamada a rotina **CreateWellKnownSockets,** pertencente ao módulo connection.c, que cria uma porta dentro do protocolo TCP denominada **well-known port.** Esta porta terá um número fixo para todas as implementações de servidores X Window em redes TCP/IP. Quando um cliente deseja iniciar a conexão com um servidor, deverá enviar o pedido de conexão para o endereço IP do sistema na rede que executa o servidor e para a porta fixa dos servidores X dentro do protocolo TCP. Os servidores X ficam sempre "escutando" esta porta na espera de novas conexões. A partir do momento que é recebido o pedido de conexão, uma nova porta no protocolo TCP é criada. Esta porta é que irá indicar, para o servidor, o canal de comunicação estabelecido com o novo cliente. A partir deste instante, toda a comunicação com o novo cliente é feita através desta nova porta e a **well-known port** permanece aberta para a "escuta" de novos pedidos de conexão. Para realizar toda a operação descrita, o servidor modelo utiliza as chamadas **socket, bind** e **accept** do sistema Unix 4.3 BSD.

Na implementação no Terminal X, a chamada **socket** foi substituída pela chamada **open** do Plurix. A chamada **open** é utilizada para abrir canais seriais "tty" em modo "raw" tanto para transmissão como para recepção, de forma

similar ao que foi descrito para o cliente no item IV.1.1. Uma vez abertos os canais "tty", o servidor possui os descritores para os canais de comunicação com os quais será feita a comunicação com os clientes. Desta forma, a **well-known port** é trocada pelos descritores dos canais por onde é feita a comunicação com os clientes.

A chamada **bind,** que atribui o número fixo a porta TCP do servidor X, perdeu sua função nesta implementação. A chamada **accept,** que cria uma nova porta para a comunicação com o cliente que pediu conexão, foi trocada pela chamada **fcntl** do Plurix. Esta troca foi realizada somente para preservar as características do código do servidor modelo. Como a chamada **accept** cria um novo descritor para o novo cliente, a chamada **fcntl** foi utilizada para duplicar o descritor do canal serial já aberto, criando assim um novo descritor para o mesmo canal. A partir do momento em que esta operação é realizada, a comunicação com o novo cliente será feita através do novo descritor.

O módulo io.c contém as duas rotinas responsáveis pela transmissão de informações aos clientes e pela recepção de requisições destes clientes. Estas rotinas, no servidor modelo, recebem uma estrutura que possui o descritor do canal de comunicação com o cliente. Esta funcionalidade foi preservada na implementação do servidor no Terminal X, através dos descritores dos canais seriais criados para a comunicação cliente-servidor. São utilizadas, pelo servidor modelo, as chamadas ao sistema

read e **writev,** para implementar as funções de comunicação. A chamada **read** é idêntica no Plurix e a chamada **writev** já havia sido implementada no Plurix durante a implantação da biblioteca X.

A rotina de transmissão utiliza também a chamada **select,** do Unix 4.3 BSD, quando tenta escrever e o canal de comunicação está bloqueado pelo sistema. Esta chamada coloca o processo em estado de espera pela liberação do canal desejado, durante um tempo máximo determinado através de parâmetro passado para a chamada. Com isto, o processo não ocupa o processador do sistema em um "loop" de espera pela liberação do canal. Esta chamada não possui similar no Plurix nem existem ainda mecanismos que permitam sua simulação. A solução adotada foi colocar o processo servidor em um "loop" finito de espera pela liberação do canal desejado, tornando esta implementação menos eficiente sob o ponto de vista de ocupação do processador do sistema.

O módulo WaitFor.c possui uma única rotina, a **WaitForSomething.** Esta rotina é chamada pelo "dispatcher" da camada DIX para realizar um "loop" esperando que exista algum cliente pedindo conexão, ou que algum cliente já existente envie uma requisição, ou que algum dispositivo de entrada seja ativado. Novamente é utilizada, pelo servidor modelo, a chamada ao sistema **select** para realizar a espera por um determinado tempo, sem utilizar o processador do sistema. Neste caso, a chamada **select** foi

substituída pela chamada do Plurix **ioctl** passando um parâmetro para saber se existem informações nos canais de comunicação com os clientes ou no canal que recebe sinalizações de entrada provenientes do Terminal X. Esta implementação, de forma semelhante ao caso anterior, é menos eficiente, pois o processo ficará em "loop" de espera sendo executado pelo processador do sistema Pegasus.

No módulo WaitFor.c existe ainda a utilização de um "timer" do sistema para contar o tempo desde a última ativação de um dispositivo de entrada. Isto é realizado com a intenção de preservar o fósforo da tela do monitor conectado ao servidor X. Quando o servidor não for utilizado durante um período de tempo determinado, a tela é apagada. Ela só é acesa novamente caso seja feita alguma sinalização de entrada. A chamada ao "timer" e a estrutura que contém a informação de tempo são diferentes nos sistemas Unix 4.3 BSD e Plurix. Modificações foram realizadas de forma a compatibilizar a função descrita com os recursos oferecidos pelo Plurix.

. Acesso aos Arquivos Utilizados pelo Servidor X

O servidor X Window modelo interage com o sistema operacional para ter acesso a três tipos de informações: a uma base de dados sobre cores, a uma lista dos sistemas computacionais que podem fazer conexão com o servidor e aos arquivos que contêm as fontes de caracteres utilizadas

pelo servidor.

Os módulos osinit.c e oscolor.c são responsáveis pela interação com o sistema operacional para a utilização de uma base de dados que fornece informações sobre o código digital dos níveis de diversas cores, a partir de nomes de cor pré-definidos. Esses módulos utilizam chamadas a rotinas de uma biblioteca do Unix 4.3 BSD para a inicialização e o acesso a esta base de dados.

Esta biblioteca, denominada **libdbm** fornece todas as rotinas necessárias para a criação e utilização de uma base de dados construída através de algoritmo usando técnica de "hash". A biblioteca **libdbm** não está definida no Plurix. Porém, junto ao servidor modelo são fornecidas as rotinas que compõem esta biblioteca desenvolvidas para um outro sistema operacional com filosofia Unix. Para adaptá-las ao Plurix foi necessário analisar todas essas rotinas, pois elas utilizam a passagem de estruturas como parâmetro quando são chamadas ou quando retornam um valor.

Essas estruturas representam, no caso do sistema X, o código digital dos níveis das cores obtido a partir de uma "chave" que é o nome da cor. As rotinas fornecidas tiveram que ser analisadas, pois, como já foi mencionado anteriormente, o compilador C do Plurix ainda não aceitava as operações mencionadas feitas com estruturas. Com isto, foi necessário alterar as rotinas para que fossem passados e recebidos ponteiros para as estruturas, ao invés das

próprias estruturas.

A partir do momento em que foi implantada a biblioteca **libdbm** no Plurix, pôde-se gerar a base de dados, no diretório que o servidor X a procura. Esta base é gerada a partir de um arquivo de dados que contém o nome de várias cores e seus respectivos níveis RGB ("red", "green" e "blue"). Este arquivo é fornecido juntamente com o sistema X Window. Depois de gerada a base de dados, foram alteradas as chamadas às rotinas da biblioteca **libdbm,** feitas pelos módulos osinit.c e oscolor.c, para a passagem de ponteiros como parâmetros no lugar de estruturas.

O módulo access.c é o responsável pela interação do servidor modelo com o sistema operacional, para ter acesso a lista dos nomes dos sistemas computacionais que podem fazer conexão com o servidor. Esta lista é utilizada pelo servidor com as finalidades de segurança e proteção. Caso um cliente, que esteja sendo executado em um determinado sistema computacional, faça o pedido de conexão ao servidor, esta lista é consultada para saber se este sistema está ou não habilitado a se comunicar com o servidor.

Como foi definido que os clientes estarão, na primeira fase da implementação do Terminal X, sendo executados também no Pegasus, a lista dos nomes dos sistemas habilitados à conexão não foi construída. Assim

sendo, o procedimento de consulta não foi inicialmente considerado e as rotinas do módulo access.c não foram utilizadas na implementação do servidor X realizada durante a primeira etapa deste trabalho.

Os módulos fileio.c e filenames.c são utilizados para o acesso, através do sistema operacional, aos arquivos que contêm as fontes de caracteres utilizadas pelo servidor X. Estes módulos utilizam as chamadas ao sistema **open, read** e **close** que são idênticas no Unix 4.3 BSD e no Plurix. Poucas mudanças foram necessárias para a implantação destes módulos.

O último módulo pertencente a camada OS é o utils.c. Este módulo possui rotinas utilitárias responsáveis pela colocação de mensagens de erro, pelo processamento da linha de comando de chamada ao servidor, pela impressão do "help" do servidor e pelos procedimentos de interrupção e reinício do servidor. Esse módulo também foi implantado sem a necessidade de se realizar mudanças significativas.

V.2.4. Implementação da Camada DDX no Terminal X

Na camada DDX estão todas as rotinas responsáveis pela interação do servidor X Window com o hardware para exibição gráfica e com os dispositivos de entrada, o teclado e o "mouse". O hardware para exibição gráfica consiste normalmente de dois dispositivos básicos: a memória de imagem, onde são desenhadas as imagens que

aparecerão na tela do monitor, e a tabela de conversão de cores, onde é feita a transformação da informação armazenada na memória de imagem para o código digital da cor correspondente.

O servidor modelo fornecido junto ao sistema X Window interage, utilizando as rotinas da camada DDX, com os dispositivos gráficos e de entrada, através do sistema operacional. São utilizadas as chamadas ao sistema **open, close, read** e **write** para fazer o acesso a cada um deste dispositivos independentemente. Na implementação do servidor do Terminal X isto ocorreu de forma distinta. As rotinas da camada DDX, utilizando as mesmas chamadas ao sistema, fazem o acesso a todos esses dispositivos através de um único canal de comunicação serial "tty" que liga o Pegasus ao Terminal X. Além disto, essas rotinas não se comunicam diretamente com a memória de imagem, pois o Terminal X possui um processador gráfico dedicado ao controle desta memória.

Através da linha serial que liga o Pegasus ao Terminal X, as rotinas enviam requisições, para o processador gráfico e para programação da tabela de cores, e lêem as sinalizações de entrada, provenientes do teclado e do "mouse". No Terminal X, o processador MC68000 executa o programa responsável pela comunicação com o Pegasus. Esta programa recebe as requisições, faz o processamento necessário e se comunica com o processador gráfico HD63484 ou com o Bt458 para realizar a função requisitada. Em

relação ao teclado e ao "mouse", o mesmo programa verifica o estado destes dispositivos, enviando para o Pegasus mensagens sinalizando a ocorrência de ações nos respectivos dispositivos.

O mecanismo de comunicação com o hardware e a utilização do processador gráfico são as principais diferenças entre a arquitetura para a qual foi implementado o servidor modelo e a arquitetura do Terminal X. Mesmo assim, devido à quantidade e à complexidade das tarefas realizadas pela camada DDX, o trabalho de implementação desta camada no Terminal X foi baseado na camada DDX do servidor modelo e dividido em duas etapas. Na primeira etapa, as rotinas que compõem esta camada no servidor modelo, foram analisadas e, posteriormente, adaptadas ou reescritas para o Terminal X, conforme as diferenças existentes entre as duas arquiteturas. Nesta etapa, procurou-se otimizar o tempo necessário para que se tivesse uma implementação do Terminal X funcionando. Na segunda etapa, será feita uma otimização no desempenho do Terminal X, aproveitando-se melhor todas as funcionalidades oferecidos pelo processador gráfico HD63484. Esta divisão é proposta ainda por ANGEBRANNDT [11] como estratégia para a implementação do servidor modelo sobre outros sistemas gráficos, o que é o caso deste trabalho, devido a utilização do processador gráfico.

A camada DDX do servidor modelo é subdividida em

cinco outras subcamadas:

- MI ("Machine Independent");
- CFB ("Color Frame Buffer");
- MFB ("Monochrome Frame Buffer");
- SNF ("Server Natural Format");
- Interface com hardware.

Para descrever a função de cada uma dessas subcamadas é necessário definir alguns conceitos e termos associados ao sistema X Window e em particular, ao servidor modelo fornecido. A partir disto, será possível explicar como essas subcamadas foram implementadas no Terminal X.

. Termos e Conceitos Associados ao Sistema X

O sistema X Window trabalha com uma entidade denominada "drawable". Um "drawable" é qualquer região de memória na qual pode-se desenhar imagens. Conforme já foi mencionado no item II.2.5, o sistema X Window oferece dois recursos que são do tipo "drawable". Esses recursos são as **janelas,** áreas na memória de imagem e conseqüentemente, visíveis na tela gráfica, e **"pixmaps",** áreas na memória de trabalho do sistema e portanto, não são visíveis na tela gráfica.

Os "pixmaps" são utilizados pelo sistema X Window para a construção de padrões gráficos. Esses padrões são usados normalmente para a construção de imagens formadas a partir da repetição destes padrões por toda a área da imagem. O sistema X trabalha com dois tipos de padrões: um

denominado "tile" ou ladrilho e outro denominado "stipple" ou pontilhado.

Um "tile" é um padrão onde os pontos podem ter várias cores. O número de cores que cada ponto de um "tile" pode ter é determinado pela profundidade dos "drawables" do sistema. No Terminal X, cada ponto de um "tile" corresponde a um byte, pois a memória de imagem, onde estão as janelas, possui profundidade igual a oito.

Um "stipple" é um padrão onde os pontos são binários, ou seja, um "stipple" é um "tile" com profundidade igual a um. O ponto que possuir o bit em "1" significa que será utilizada uma determinada cor, normalmente denominada "foreground color" (cor de desenhar). O ponto de um "stipple" que possuir o bit em "0" identifica que será utilizada a "background color" (cor de fundo) caso o "stipple" seja opaco ou haverá ausência de cor no caso contrário.

. Funcionamento da Camada DDX no Servidor Modelo

Umas das funções da camada DDX é fornecer uma interface de software para um dispositivo de hardware virtual. Este dispositivo deve fornecer um conjunto de 16 primitivas de desenho que permitem a execução de todas as operações gráficas, definidas no sistema X, sobre as janelas e os "pixmaps". Essas primitivas incluem cópia de imagens, preenchimento de polígonos e arcos, e desenho de

pontos, linhas, retângulos, arcos e textos. Pode-se definir também, para as primitivas onde é pertinente, a largura das linhas e o estilo ou padrão para preenchimento. Além destas primitivas definidas pelo sistema X, outras primitivas gráficas devem existir na camada DDX para uso interno do servidor, como por exemplo, o preenchimento de áreas utilizado na criação das janelas.

A função da subcamada MI ("Machine Independent"), pertencente a camada DDX do servidor modelo, é fornecer uma simulação por software do dispositivo de hardware virtual, construído a partir de primitivas mais simples, denominadas **Pixblit.** A idéia desta subcamada é ser independente do hardware gráfico sobre o qual serão aplicadas as primitivas gráficas oferecidas pelo servidor. Desta forma, esta subcamada pode ser transportada para diferentes sistemas, deixando o trabalho de adaptação as características do hardware para as subcamadas de mais baixo nível que implementam as primitivas Pixblit.

As subcamadas CFB ("Color Frame Buffer") e MFB ("Monochrome Frame Buffer") fornecem as rotinas que implementam respectivamente as primitivas Pixblit sobre "drawables" com diversas cores (profundidade maior que um) e com duas cores (profundidade igual a um). As rotinas destas subcamadas devem ser adaptadas as características do hardware dos diferentes sistemas para os quais sejam transportadas. Além das primitivas Pixblit, as subcamadas CFB e MFB devem fornecer as demais primitivas gráficas de

uso interno do servidor e as rotinas para iniciar e alterar a tabela de cores, caso ela exista no sistema implementado.

É importante notar que as rotinas da subcamada MFB são necessárias mesmo que o sistema, sobre o qual está sendo implementado o servidor X, tenha a memória de imagem com profundidade maior que um. Isto se explica, pois o sistema X trabalha com "pixmaps" de profundidade igual a um para a criação de padrões do tipo "stipple". Como "pixmaps" são "drawables", qualquer primitiva pode ser aplicada a eles.

A subcamada SNF ("Server Natural Format") contém as rotinas para tratamento com arquivos de fontes de caracteres no formato SNF. Os arquivos de fontes de caracteres são obtidos neste formato através da compilação das fontes de caracteres fornecidas pelo sistema X no formato BDF (uma pequena variação do Adobe 2.1 Bitmap Distribution Format da Adobe Systems, Inc.). O compilador de fontes de caracteres é fornecido juntamente com o servidor modelo. Uma vez compilados, os arquivos das fontes são abertos pelas rotinas já mencionadas da camada OS do servidor e as fontes necessitadas são obtidas através das rotinas da subcamada SNF.

A última subcamada da camada DDX é a responsável pela interface com o hardware gráfico, com a tabela de cores, com o teclado e com o "mouse". Esta subcamada no servidor

modelo é que chama as rotinas do sistema operacional para abrir, ler, escrever e fechar estes dispositivos. O servidor modelo fornece diversos exemplos dessa interface para sistemas estrangeiros como SUN, IBM, DEC e outros.

. Metodologia Utilizada na Implementação da Camada DDX

Conforme descrito anteriormente a metodologia para implementação da camada DDX no Terminal X foi definida em duas etapas. Na primeira etapa o compromisso maior foi com simplicidade e, conseqüentemente, com o tempo para se colocar o Terminal X funcionando. Numa segunda etapa, são feitas as diversas otimizações e neste caso o compromisso maior é com o desempenho do sistema.

A metodologia de como se obter o servidor modelo funcionando no menor espaço de tempo, sobre um hardware genérico é indicada por ANGEBRANNDT [11]. O método consiste basicamente no aproveitamento das rotinas existentes na subcamada MI e na alteração das rotinas das subcamadas MFB e CFB de acordo com as características do hardware.

Assim sendo, na primeira etapa da implementação do servidor X no Terminal X, foram utilizadas as rotinas da subcamada MI. As rotinas da subcamada MFB também foram aproveitadas, pois só são utilizadas para a manipulação com "pixmaps" utilizados na criação de padrões "stipple".

As rotinas da subcamada CFB foram reescritas para a interação com "drawables" de profundidade igual a oito, a profundidade da memória de imagem do Terminal X. As primitivas **Pixblit** foram escritas utilizando as primitivas fornecidas pelo processador gráfico do Terminal X. Foram criadas rotinas que replicam um "tile" por uma determinada área, rotinas para traçado de retas e pontos e para preenchimento de áreas. Essas rotinas enviam os comandos para execução das respectivas informações através da linha serial que comunica o Pegasus com o Terminal X. Algumas rotinas desta subcamada que são fornecidas pelo servidor modelo, tiveram que ser preservadas para a interação com "drawables" do tipo "pixmaps".

As rotinas para a interação com os arquivos das fontes de caracteres, contidas na subcamada SNF, não são dependentes das características do sistema que executa o servidor X, e desta forma, puderam ser integralmente aproveitadas. Essas rotinas não serão alteradas inclusive na segunda etapa da implementação da camada DDX no Terminal X, pois o desenho de textos é realizado utilizando as primitivas das subcamadas MI, MFB e CFB.

Na subcamada que faz a interface com o hardware do Terminal X, foram escritas as rotinas que contêm as informações quanto a resolução da memória de imagem do Terminal X e as rotinas que enviam comandos para escrita na tabela de cores do Terminal X. Foram desenvolvidas também as rotinas para introdução das condições iniciais e

controle dos dispositivos de hardware responsáveis pela exibição da memória de imagem na tela do monitor do Terminal X.

Na segunda etapa da implementação da camada DDX, as rotinas da subcamada MI, responsáveis pelas primitivas gráficas definidas pelo sistema X Window, serão reescritas para o aproveitamento integral das primitivas oferecidas pelo processador gráfico HD63484. Desta forma, não será mais necessário, na maior parte dos casos, a utilização das primitivas Pixblit, oferecidas pela subcamada CFB, para a interação com processador gráfico do Terminal X.

CAPÍTULO VI

TESTES E AVALIAÇÃO DA IMPLEMENTAÇÃO

VI.1. Testes Realizados

Os testes foram iniciados pelo hardware do Terminal X. Primeiramente, foi testada a parte do hardware de uso geral, ou seja, aquela que não é dedicada ao uso gráfico. O processador MC68000 foi colocado em funcionamento para executar um programa monitor, gravado em ROM, para teste e depuração dos demais circuitos. A partir deste monitor foi realizado o teste da memória de 1 Mbytes e das interfaces seriais.

No monitor foi implementada uma rotina para carregar programas para serem executados na memória do sistema. Essa rotina foi desenvolvida de forma a carregar programas no formato executável, recebidos através de uma das interfaces seriais. Esses programas eram gerados no Pegasus-32X e enviados através de uma de suas interfaces seriais para o Terminal X.

Por esse mecanismo, foi possível o desenvolvimento de um segundo programa monitor mais complexo. Esse programa foi criado com a finalidade de testar o hardware gráfico. Através deste, pôde-se programar e testar as diversas

primitivas gráficas existentes no processador gráfico HD63484. Foi testado o controle do processador gráfico sobre a memória de imagem e a utilização da informação proveniente desta memória pelos conversores do Bt458. Foram feitos também os teste para o acesso a tabela de conversão de cores situada no Bt458. Finalmente, o monitor colorido foi ligado ao hardware do Terminal X e pôde-se visualizar as operações do processador gráfico sobre a memória de imagem.

Uma vez testado todo o hardware do Terminal X, passou-se para a implementação de um programa cliente do sistema X Window no Pegasus-32X. Para isso, foram utilizados quatro programas clientes cujos fontes escritos em linguagem C, são fornecidos juntamente com o sistema X original. Esses programas foram compilados no ambiente criado com a implantação da biblioteca X no Pegasus/Plurix. Resolvidos os problemas iniciais de incompatibilidade das chamadas ao sistema operacional, pois os aplicativos também são desenvolvidos para o Unix 4.3 BSD, obteve-se quatro programas clientes executáveis e que utilizam diversas chamadas às rotinas da biblioteca X.

Esses aplicativos foram executados, ainda sem o servidor X funcionando, para observação do protocolo X para pedido de conexão, transmitido através de uma linha de comunicação serial. Nesse pedido, o cliente envia sua versão no sistema X (no caso versão 11.2) e a ordem natural de como é feito o acesso, no computador em que é

executado, aos dois bytes de uma palavra de 16 bits (acesso na ordem "little-endien" ou "big-endian").

Duas linhas de comunicação serial que não possuiam **LOGIN** foram interligadas no Pegasus e foi implementado um programa simples para receber o protocolo X para pedido de conexão, na linha serial reservada para o servidor. Esse programa lia os bytes recebidos através da linha e imprimia o resultado na tela de um terminal do Pegasus. Esse teste serviu para detectar que as chamadas ao sistema Plurix para a abertura das linhas seriais não estavam inicializando corretamente essas linhas em modo "raw". Os bytes recebidos estavam sendo alterados pois eram tratados pelo sistema operacional tanto na transmissão pelo cliente, quanto na recepção pelo programa de teste. Esse problema foi resolvido rapidamente passando-se os parâmetros corretos para a abertura das linhas seriais em modo "raw".

A partir destes testes, passou-se para a implementação do servidor X no Pegasus. Essa implementação foi feita por partes, sendo analisados e compilados somente os módulos do servidor X que estavam sendo testados. Dentro de cada módulo, foram separadas as rotinas que não estavam sendo testadas, colocando-se elas entre comentários. Nas rotinas que estavam em teste, foram inseridos "printf" com o nome da rotina para indicar a passagem por aquele ponto. Essa metodologia foi fundamental para o desenvolvimento do servidor X dentro do

ambiente Pegasus/Plurix e Terminal X, pois conforme já foi mencionado, diversos módulos do servidor X são extensos e cada um possui internamente um grande número de rotinas. Sem a utilização desta metodologia, diante da complexidade e da quantidade das funções executadas pelo servidor X, seria difícil localizar, em caso de erro, em qual rotina estava o problema.

As primeiras rotinas compiladas foram as que iniciam o servidor X. Essas rotinas são responsáveis por atribuir às diversas estruturas utilizadas pelo servidor X os parâmetros com as características do hardware que o servidor irá controlar. Além disto, essas rotinas iniciam o hardware gráfico, preenchendo a tabela de cores com um determinado conteúdo, criando a janela raiz que ocupa toda a tela de exibição do monitor do Terminal X, colocando o cursor inicial na tela, monitorando o teclado e o "mouse" e criando as cores iniciais para desenho ("foreground") e fundo ("background"). Para implementação das rotinas que iniciam o hardware, foi necessário o teste da comunicação do Pegasus com o Terminal X e o teste do protocolo definido para requisição das operações a serem realizadas pelo Terminal X.

Depois foram testadas as rotinas do servidor para a leitura e escrita no canal de comunicação serial estabelecido com o cliente. Foi iniciada a comunicação do cliente com o servidor, através da análise por parte do servidor do protocolo X para conexão enviado pelo cliente

e a resposta do servidor enviando para o cliente as características do hardware gráfico e os identificadores dos recursos já criados. Esses recursos incluem, por exemplo, a janela raiz, a tabela de cores inicial e as cores de "foreground" e de "background" iniciais.

Uma vez testada e estabelecida a conexão entre o cliente e servidor, começaram a ser testadas individualmente cada uma das requisições do protocolo X utilizadas pelos quatro clientes mencionados. Pode-se desta forma, testar a implementação de todas as funções desempenhadas pelo servidor X, juntamente com as operações realizadas pelo hardware do Terminal X. Foi igualmente possível testar o protocolo criado para a comunicação do Pegasus com o Terminal X, enviando requisições para operações gráficas e lendo sinalizações provenientes do "mouse" e do teclado. Para a implementação das operações realizadas pelo Terminal X, foram utilizados como base os programas monitores construídos para teste do hardware do Terminal X.

A utilização dos programas clientes fornecidos junto ao sistema X Window facilitou bastante a realização dos testes. A operação desses programas, utilizando as diversas rotinas da biblioteca X, foi analisada antes deles serem compilados e executados. Através da visualização dos resultados obtidos na tela do Terminal X, durante a execução desses programas clientes, pôde-se observar tanto os erros existentes como constatar a

correta operação das rotinas do servidor X e do Terminal X.

VI.2. Avaliação da Implementação

Uma primeira avaliação da implementação realizada do Terminal X nos permite dizer que o objetivo principal foi alcançado. Pôde-se, em um período de tempo relativamente curto, dotar o sistema operacional Plurix de uma interface gráfica padrão, o sistema X Window.

Numa análise mais profunda, algumas modificações podem ser sugeridas. Conforme já foi mencionado no capítulo V, a primeira otimização deve ser feita na implementação da camada DDX do servidor. A segunda etapa da implementação desta camada, descrita naquele capítulo, pode ser considerada fundamental para a melhoria do desempenho do servidor dentro do Terminal X implementado.

Esta afirmação foi comprovada através dos testes realizados com os quatro clientes mencionados no item anterior. A execução das tarefas gráficas foi bastante prejudicada pelo fato de não se estar utilizando todo o potencial oferecido pelo processador gráfico. A utilização do processador somente para o traçado de primitivas gráficas de mais baixo nível, tornou o servidor menos eficiente pois as primitivas de mais alto nível estavam sendo implementadas por software.

Além disto, outras funções, como por exemplo as rotinas para "clipping" de áreas, tiveram que ser realizadas por software, em função da emulação das primitivas de mais alto nível também por software. A partir do instante em que o processador gráfico seja utilizado para o traçado destas primitivas pode-se realizar também o "clipping" por intermédio do processador, pois esta é uma das funções que ele oferece.

A pouca utilização do processador gráfico aumentou também o fluxo de informações na linha de comunicação serial que interliga o Pegasus ao Terminal X. Como a velocidade desta linha não é alta isto prejudicou ainda mais o desempenho do sistema implementado. Por esse e pelos demais motivos explicados, pode-se afirmar que haverá uma melhora considerável no desempenho do Terminal X, quando este passar pela segunda etapa da implementação da camada DDX do servidor X utilizado.

Um outro fator a ser analisado é a ligação do cliente com o servidor, implementada através de duas linhas seriais do Pegasus. A respeito deste mecanismo, foi feito um teste bastante conclusivo. No lugar de se executar o cliente juntamente com o servidor no Pegasus, o cliente foi implementado em outro computador que utiliza o sistema operacional Plurix. Esse computador, um EBC 32010, foi adquirido pelo NCE/UFRJ durante o desenvolvimento deste trabalho. O EBC 32010 é um supermicrocomputador baseado no microprocessador MC68010 e sua capacidade de processamento

é inferior a do Pegasus-32X.

A execução do cliente no EBC 32010 permitiu a constatação que, na implementação realizada do Terminal X, a velocidade da comunicação entre o cliente e o servidor não é relevante em relação ao desempenho do sistema. Pôde-se observar que a limitação do sistema está realmente na implementação do servidor. Para se chegar a esta conclusão, foi utilizada uma das linhas de comunicação serial do EBC 32010 para ligá-lo ao Pegasus. Os programas clientes, já mencionados no item anterior, foram implementados no EBC, depois de implantada a biblioteca X neste sistema, e o servidor X foi mantido no Pegasus.

Como os dois processos passaram a ser executados em sistemas computacionais distintos, esperava-se um aumento na velocidade de visualização das requisições enviadas pelo cliente ao Terminal X. Porém, o que foi observado foi que o servidor executado no Pegasus não conseguia retirar, em tempo hábil, as requisições recebidas e o "buffer" do Plurix, executado no Pegasus, não era suficiente para acumular estas mensagens. Desta forma, depois de algum tempo de execução, o servidor perdia parte das mensagens enviadas pelo cliente interrompendo a execução deste.

Só foi possível a execução dos aplicativos/clientes do sistema X no EBC 32010 e do servidor X no Pegasus, através da utilização da rotina **XSync** da biblioteca X. Esta rotina é utilizada para sincronizar o cliente com o

servidor, pois o servidor só irá atendê-la depois que executa todas as requisições já existentes. Ao atendê-la o servidor responde ao cliente e só então este continua sua execução.

Este teste serviu não só para confirmar a necessidade de otimizações no servidor X, como também para comprovar uma das características do sistema X. Pôde-se criar um ambiente distribuído, com a execução do cliente em um sistema computacional e do servidor em outro. Esta arquitetura, que só era prevista na segunda fase da implementação do Terminal X, quando então ele poderá ser ligado por meio de redes de alta velocidade a outros sistemas que utilizem o Plurix, pode ser simulada através da utilização das linhas de comunicação serial. Além disto, teve-se a oportunidade de se testar a implantação da biblioteca X realizada no ambiente Pegasus/Plurix, em um outro computador distinto que também utiliza o Plurix.

A avaliação final que se pode fazer da implementação realizada do Terminal X é que ela foi totalmente válida e a metodologia utilizada foi igualmente correta. As deficiências apareceram nos pontos onde foram previstas, mas foram totalmente compensadas pela possibilidade que se teve de rapidamente poder executar aplicações/clientes do sistema X Window no sistema operacional Plurix e visualizar seus resultados no Terminal X. Somente este fato já incentiva a continuação do trabalho e ajuda inclusive, a indicar outras áreas do servidor que estejam

deficientes, onde novos procedimentos poderão ser utilizados.

CAPÍTULO VII

CONCLUSÕES

O trabalho desenvolvido permitiu a rápida implementação de uma interface gráfica no sistema operacional Plurix. Esta interface, conforme já foi explicado no capítulo II, é uma interface de baixo nível pois oferece somente rotinas gráficas básicas para implementação de aplicações.

Outras bibliotecas podem ser implementadas sobre a biblioteca básica já existente. Estas bibliotecas teriam seu transporte facilitado, pois são desenvolvidas utilizando as rotinas básicas padronizadas pelo sistema X Window que foram implementadas no Plurix. Dentre as bibliotecas de mais alto nível existentes, pode-se destacar três: a X Toolkit, fornecida junto ao próprio sistema X Window, a Motif, definida e desenvolvida pela Open Software Foudation, e a Looking Glass, oferecida pela Visix Software.

Estas bibliotecas são utilizadas por diversos fabricantes de sistemas computacionais gráficos e existe atualmente uma disputa sobre qual delas se tornará padrão para o desenvolvimento dos aplicativos gráficos a serem executados nesses sistemas computacionais.

Estudos estão sendo feitos sobre cada uma das três bibliotecas mencionadas, enquanto aguarda-se a definição de qual delas será utilizada pela maioria dos sistemas gráficos. Pretende-se implantar a que for definida como padrão, sobre a biblioteca básica do sistema X Window. Com isto, os aplicativos já desenvolvidos para utilizarem esta interface gráfica de alto nível, podem ser transportados para o ambiente Plurix/Terminal X. Da mesma forma, pode-se desenvolver softwares aplicativos no Terminal X e transportá-los para outros sistemas distintos que possuam a mesma interface gráfica.

Além da implantação de bibliotecas de mais alto nível sobre a interface gráfica implementada, outras áreas de atuação podem ser adotadas para a continuação do trabalho desenvolvido. Algumas já foram definidas dentro deste próprio trabalho, como a otimização do servidor e a implementação da interface Ethernet e das rotinas para a comunicação utilizando o protocolo TCP/IP. Outras áreas podem ser citadas de forma a dotar o Terminal X de outros recursos gráficos mais sofisticados e também para gerenciar os recursos já oferecidos pelo sistema X Window.

Uma área interessante é o estudo da extensão PEX ("Phigs+ Extension to X") que está sendo definida para o protocolo X Window. A interface Phigs+ é um conjunto de rotinas definidas pela ISO ("International Standards Organization") para o desenvolvimento de aplicações gráficas que fazem a exibição de imagens com a perspectiva

em três dimensões (3D). Esse conjunto de rotinas requer uma grande quantidade de processamento para a exibição destas imagens e é comum a definição de um hardware acelerador gráfico para a execução, de forma mais eficiente, das diversas tarefas associadas as rotinas para a exibição em 3D. A extensão PEX vem sendo definida pelo MIT para possibilitar a implantação, sobre o protocolo X, das rotinas Phigs+. Isto permitirá que o servidor X receba requisições para execução de funções 3D, podendo-se definir e implementar um hardware acelerador gráfico para ser controlado pelo servidor X.

Além desta área, existe também a possibilidade do estudo de algoritmos para serem utilizados nos gerenciadores de janelas para o sistema X Window e para implementar algumas funções mais complexas definidas neste sistema. Estas funções podem incluir, por exemplo, o armazenamento de imagens que são ocultas quando se exibe outras imagens sobre as imagens originais. Este procedimento é definido para o sistema X Window, porém a responsabilidade de atualização da memória de imagem quando alguma área oculta se torna visível, é deixada a cargo do cliente que é dono da janela na qual a área exibida está contida. Deixar esta tarefa a cargo do servidor X exige o estudo de um algoritmo para implementar esta função, pois a quantidade de memória, além da memória de imagem, necessária para o armazenamento das áreas ocultas é, teoricamente, infinita pois o sistema X Window não limita o número máximo de janelas que podem se

sobrepor.

Os algoritmos para os gerenciadores de janelas também devem ser estudados de forma a se otimizar a alocação das janelas na tela do monitor do Terminal X. O estudo pode partir do gerenciador de janelas que é fornecido junto ao sistema X original, e que não realiza a função de gerenciamento de forma otimizada, podendo apenas ser utilizado como exemplo de gerenciador.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] FALLER, N. et alii, "O Projeto PEGASUS-32X/PLURIX", Anais do XVII Congresso Nacional de Informática, Rio de Janeiro, Nov. 1984.

[2] FALLER, N., ANIDO, M. L. e SALENBAUCH, P., "Técnicas de Projeto Utilizadas na Construção do Supermicrocomputador PEGASUS-32X e do Sistema Operacional PLURIX", Anales de La Convención Informática Latina, CIL 85, Barcelona, Abr. 1985.

[3] FALLER, N. e SALENBAUCH, P., "PLURIX: A multiprocessing UNIX-like Operating System" Proceedings of the Second IEEE Workshop on Workstation Operating System, pp. 29-36, Washington, Set. 1989.

[4] SCHEIFLER, R.W. e GETTYS, J., "The X Window System", ACM Transactions on Graphics, vol. 5, no. 2, pp. 233-248, 1986.

[5] WESTMORE, R.J., "A Window-Based Graphics Frame Store Architecture", ACM Transactions on Graphics, vol. 7, no. 4, pp. 233-248, 1988.

[6] COMER, D., "Internetworking with TCP/IP - Principles, Protocols and Architecture", New

Jersey, Prentice Hall, 1st Edition, 1986.

[7] SCHEIFLER, R.W., GETTYS, J. e NEWMAN, R., "X Window System - C Library and Protocol Reference", Bedford, Digital Press, 1st Edition, 1988.

[8] 4.3 BERKELEY SOFTWARE DISTRIBUTION, "UNIX Programmer's Manual", U.S.A, Berkeley University, 7th Edition, 1986.

[9] POUNTAIN, D., "The X Window System", BYTE, vol. 14, no. 1, pp. 353-360, 1989.

[10] GETTYS, J., "Network Windowing Using the X Window System", Dr. Dobb's Journal, vol. 14, no. 3, pp. 42-53, 1989.

[11] ANGEBRANNDT, S. et alii, "Strategies for Porting the X v11 Sample Server", X Window v11.2 System Manuals, Massachusetts Institute of Technology, 1988.

[12] MANUEL, T., "Now, Low-Cost Terminals Can Run Like Work Stations", Electronics, vol. 61, no. 18, pp. 45-46, 1988.

[13] BALDWIN, H., "Why All The Shouting Over X Terminals?", Unix World, vol. 6, no. 10, supplement Unix Networking, pp. 75-80, 1989.

[14] BRUNET, J., "Workstations Feel The Squeeze", Unix World, vol. 6, no. 3, pp. 56-63, 1989.

[15] BRUNET, J., "New Challenge To Character Terminals", Unix World, vol. 6, no. 5, pp. 79-83, 1989.

[16] SCHNATMEIER, V., "First X Terminal Offers Network Display", Unix World, vol. 6, no. 2, pp. 151, 1989.

[17] FALLER N. et alii, "Manual de Referência PLURIX 2.1", Rio de Janeiro, NCE/UFRJ, 2a. Edição, Jun. 1989.

[18] MOTOROLA Inc., MC68000 16- / 32-Bit Microprocessor, Austin, Texas, March 1985.

[19] HITACHI America, Ltd., HD63484 Advanced CRT Controller User's Manual, San Jose, California, April 1985.

[20] BROOKTREE Corp., Bt458 256x24 Color Palette RAMDAC, San Diego, California, 1986.

[21] ANGEBRANNDT, S. et alii, "Definition of the Porting Layer for the X v11 Sample Server", X Window v11.2 System Manuals, Massachusetts Institute of Technology, 1988.

[22] FISHER, B., "X11 Server Extensions Engineering

Specification", X Window v11.2 System Manuals, Massachusetts Institute of Technology, 1988.