Universidade Federal de Santa Catarina

Centro Tecnológico

Departamento de Informática e de Estatística

Curso de Pós-Graduação em Ciência da Computação

UM SIMULADOR PARA MULTICOMPUTADOR IMPLEMENTADO COMO UM NÚCLEO DE SISTEMA OPERACIONAL MULTIPROGRAMADO

por

Hamilcar Boing

Dissertação submetida à Universidade Federal de Santa Catarina para a obtenção do grau de Mestre em Ciência da Computação

Prof. Thadeu Botteri Corso

Orientador

Florianópolis, dezembro de 1996

UM SIMULADOR PARA MULTICOMPUTADOR IMPLEMENTADO COMO UM NÚCLEO DE SISTEMA OPERACIONAL MULTIPROGRAMADO

HAMILCAR BOING

ESTA DISSERTAÇÃO FOI JULGADA ADEQUADA PARA A OBTENÇÃO DO TÍTULO DE

MESTRE EM CIÊNCIA DA COMPUTAÇÃO

NA ÁREA DE CONCENTRAÇÃO DE SISTEMAS DE COMPUTAÇÃO, SUB-ÁREA DE SISTEMAS OPERACIONAIS E APROVADA EM SUA FORMA FINAL PELO PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO.

Prof. Thadeu Botteri Corso, M.Sc.
Orientador

Prof. Murilo Silva de Camargo, Dr.
Coordenador do Curso

BANCA EXAMINADORA:

Prof. Thadeu Botteri Corso, M.Sc. (Presidente)

Prof. Luiz Fernando Friedrich, Dr.

Prof. Luiz Fernando Jacinto Maia, Dr.

Agradecimentos

Em cada etapa vencida em nossa vida contamos com o apoio de muitas pessoas. Gostaria de agradecer a todas as pessoas que me apoiaram e contribuíram para a conclusão deste trabalho.

Um agradecimento especial ao prof. Hermann, que iniciou esta caminhada e ao prof. Thadeu, que aceitou assumir esta orientação e pela amizade e atenção.

Agradeço aos colegas Carlos Barros Montez e Carla Merkle pelo auxílio no desenvolvimento desta dissertação.

À minha família pelo convívio e carinho durante toda a minha vida.

À minha noiva, Adriana, com quem compartilhei grande parte do tempo enquanto desenvolvia este trabalho e que me apoiou e soube esperar por este momento para que, juntos, possamos compartilhar um futuro melhor.

"Lembra que o amanhã é fruto do teu hoje, para assim almejar a perfeição hoje, e não adiá-la para o nunca."

Elisa Maris Cadore

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

Atualmente existem diversos projetos de pesquisa envolvendo a construção de computadores com diversos processadores, desenvolvidos para resolver problemas de computação que envolvem processamento intenso, especialmente em virtude da queda de preços dos componentes de *hardware*. Seguindo essa tendência, os grupos de arquitetura de computadores e sistemas operacionais da Universidade Federal de Santa Catarina criaram o projeto Nó// visando o desenvolvimento de um multicomputador e seu sistema operacional. Como parte importante do projeto está previsto também o desenvolvimento de um simulador multiprogramado para servir como base de desenvolvimento de *software* para a máquina real.

Uma primeira versão do simulador foi desenvolvida por [MON95] e [CAM95] e se encontra funcional, porém com várias limitações, especialmente a nível de gerência de memória. O presente trabalho propõe a construção de um novo simulador, com o objetivo de eliminar problemas de gerência de memória da primeira versão citada e servir como base para o micronúcleo do sistema operacional do multicomputador Nó//.

Abstracts

Presently there are several research projects concerning to the construction of computers with many processors developed to solve computing problems related to hard processing, specially due to the low prices of hardware componentes. Folowing this tendency, the computers architeture and operating system groups of Universidade Federal de Santa Catarina created the Nó// project aiming to develop a multicomputer and its operating system, as well. As a important part of the project, it is also proposed the cuilding of a multiprograming simulator functioning as a software development base to the real machine.

The first version of the simulator was created by [MON95] and [CAM95] and is functioning in spite of some limitations, specially at the memory management level. This present work is developed with the purpose of building a new simulator with the expectation of solving some problems of the first version and to work as an operating system microkernel support to the Nó// multicomputer.

1 - Introdução

O projeto Nó// (leia-se Nó Paralelo) nasceu de uma proposta, apresentada em [COR93], para a construção de um ambiente (*hardware* e *software*) para processamento paralelo e está sendo desenvolvido na Universidade Federal de Santa Catarina. Nele é proposta a construção de um multicomputador, empregando soluções simples e de baixo custo em relação a outros multicomputadores, utilizando os processadores i486 da Intel. Seu sistema operacional será compatível, a nível de interface de programação, com UNIX. É previsto também a construção de um simulador, executável em computadores AT com sistema operacional DOS, com o objetivo de permitir o desenvolvimento de *software* em paralelo com a construção do *hardware.*

Uma motivação para este trabalho está em eliminar as limitações da versão atual do simulador a nível de gerência de memória, impostas pelo ambiente em que foi desenvolvido, utilizando-se recursos avançados da arquitetura dos processadores i486.

Outra motivação é o desenvolvimento do micronúcleo do sistema operacional do Nó//, composto de um gerenciador de memória e um escalonador de *threads*, com uma interface semelhante à do simulador, permitindo que o *software* nele desenvolvido seja portável para o Nó//. Como o simulador e o sistema operacional serão executados sobre a mesma arquitetura, o novo simulador será a base do micronúcleo do sistema operacional, com mínimas alterações, e estará praticamente pronto quando a máquina real estiver disponível.

A primeira versão do simulador desenvolvido por [CAM95] e [MON95] está funcional, apesar da limitação de memória principal imposta pelo ambiente DOS, que é de 640 *kbytes*. Dentro deste limite de memória devem estar o núcleo do DOS e o código do simulador. O restante da memória é alocada para representar a memória privativa dos nós de trabalho (figura 1.1).

Como cada nó de trabalho no Nó// terá, pelo menos, 1 *Mbyte* de memória principal, existe uma diferença muito grande entre a capacidade de memória

disponível na máquina real e a disponível no simulador, limitando a construção de *software* aos recursos do simulador.

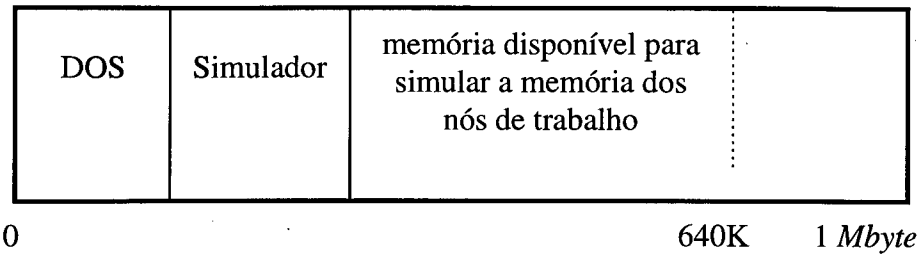

Figura 1.1: Distribuição de memória no modo real.

Este trabalho se enquadra dentro do projeto Nó// e tem como objetivos:

- desenvolver um novo simulador do multicomputador Nó//, utilizando os recursos avançados dos processadores i486, responsável por:
  - ◊ gerência de memória, permitindo utilizar toda a memória do computador, e, pelo menos, 1 *Mbyte* de memória por nó de trabalho;
  - ◊ gerência de processos;
  - ◊ supervisão de entrada e saída;
  - ◊ controle de interrupções.
- desenvolver uma interface semelhante à do simulador atual, que permita utilizar os *softwares* já desenvolvidos;
- propor um gerenciador de memória e um escalonador para o sistema operacional do Nó// que controle a execução de diversos *threads* por nó de trabalho.

O gerenciador de processos do simulador tem como função permitir simular os diversos nós do multicomputador Nó//.

A função do gerenciador de memória é, na etapa de inicialização, alocar memória em partições fixas para representar a memória física de cada nó de trabalho (figura 1.2), onde serão colocados códigos e dados dos processos. Essas partições, uma vez alocadas, não sofrem mais alterações de tamanho.

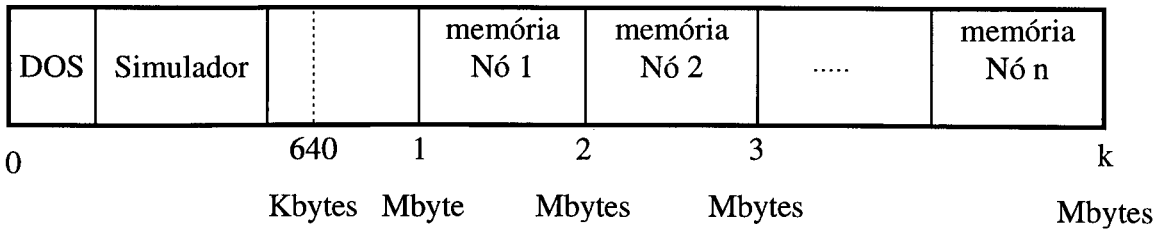

Figura 1.2: Organização da memória desejada.

A supervisão de entrada e saída e o controle de interrupções permitirá ao novo simulador gerenciar todos os recursos de *hardware* e dispositivos compartilhados do sistema.

Este trabalho está organizado em sete capítulos, além deste capítulo de introdução.

- O capítulo 2 descreve características de máquinas paralelas com memória distribuída, os multicomputadores.
- O capítulo 3 introduz o multicomputador Nó// e o simulador desenvolvido.
- O capítulo 4 apresenta os recursos para gerenciamento de memória dos processadores i486, que são a base da arquitetura do Nó//.
- O capítulo 5 descreve funções básicas de um micronúcleo e alguns exemplos de sistemas operacionais baseados neste enfoque.
- O capítulo 6 apresenta o novo simulador e as alterações necessárias para transformá-lo no micronúcleo do sistema operacional do Nó//.
- O capítulo 7 especifica as alterações necessárias para transformar o novo simulador no micronúcleo do sistema operacional do multicomputador Nó//.
- No capítulo 8 estão as conclusões sobre o trabalho, suas contribuições e perspectivas.

2 - Multicomputadores

Neste capítulo são descritas características de multicomputadores, em especial as topologias e suas redes de interconexão.

Multicomputadores são máquinas de processamento paralelo compostas de nós processadores autônomos e com memória distribuída (figura 2.1), desenvolvidos com o objetivo principal de aumentar a capacidade de processamento em relação a máquinas com apenas um processador.

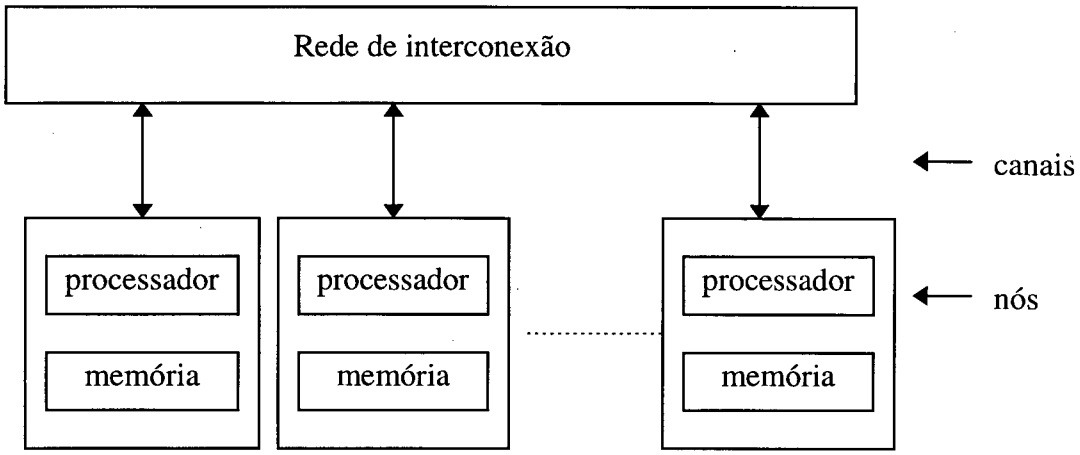

Figura 2.1: Multicomputador.

A programação de multicomputadores deve ser baseada na divisão em processos cooperantes, executados em nós diferentes para explorar paralelismo. Processos executados em nós diferentes se comunicam exclusivamente através da troca de mensagens, a partir de canais de comunicação bipontuais. O conjunto de canais de comunicação entre os nós é chamado de rede de interconexão.

O desempenho de um multicomputador está relacionada diretamente com a velocidade dos nós processadores e da rede de interconexão, por onde a troca de mensagens entre os processos é implementada. Quanto maior a troca de mensagens entre processos, maior a tendência da rede de interconexão tornar-se um "gargalo" do sistema.

Pode-se dividir os multicomputadores em dois grupos, em função da sua rede de interconexão: estática e dinâmica.

2.1 - Multicomputadores com redes de interconexão estáticas

Em multicomputadores com rede de interconexão estática os canais de comunicação entre dois nós são sempre fixos, ou seja, a topologia da rede de interconexão nunca é alterada.

O ideal para uma rede de interconexão seria ter todos os nós interligados, porém esta solução torna-se inviável em relação ao custo e a complexidade, à medida que se aumenta o número de nós do multicomputador. Por este motivo implementa-se topologias com limitada quantidade de ligações entre nós.

Problemas como a necessidade de identificar o melhor caminho para enviar uma mensagem e a necessidade de retransmissão de mensagens para esta chegar ao nó destino, quando os dois nós não estão ligados fisicamente por um canal de comunicação, são comuns a todas as redes de interconexão estáticas.

Algumas topologias de redes de interconexão estáticas são o anel, a grelha e o hipercubo.

2.1.1 - Rede de interconexão em anel

Na topologia em anel, mostrada na figura 2.2, os nós possuem dois canais de comunicação e formam um ciclo ocasionando distância máxima de comunicação de (**n**/2), onde **n** representa o número de nós do multicomputador.

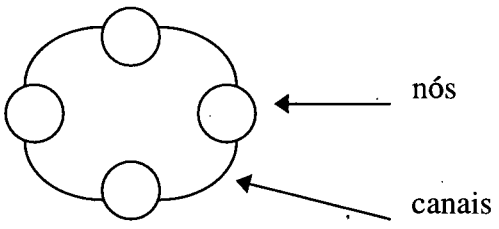

Figura 2.2: Rede de interconexão em anel.

Tipicamente, a topologia em anel é utilizada em equipamentos com poucos nós e que executam processos sem predominância de troca de mensagens.

2.1.2 - Rede de interconexão em grelha.

Na topologia em grelha, mostrada na figura 2.3, os nós e a rede de interconexão formam uma matriz quadrada de **n** linhas e colunas, totalizando $\mathbf{n}^2$ nós, cada qual conectado através de canais de comunicação com seus vizinhos imediatos.

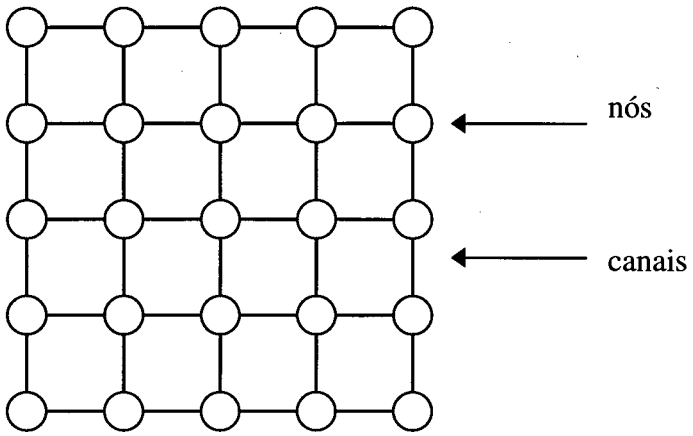

Figura 2.3: Rede de interconexão em grelha.

A distância de comunicação média entre os nós é de 2(**n**-1), onde **n** representa o número de nós do multicomputador. As redes de interconexão em grelha possuem a vantagem de apresentar vários caminhos alternativos de comunicação, possibilitando soluções para problemas de falhas.

2.1.3 - Rede de interconexão em hipercubo

Um multicomputador com rede de interconexão em hipercubo, mostrado na figura 2.4, utiliza $2^{\mathbf{n}}$ nós arranjados em **n** cubos dimensionais, onde cada nó tem $\log_2(\mathbf{n}^2)$ canais bidirecionais para nós adjacentes.

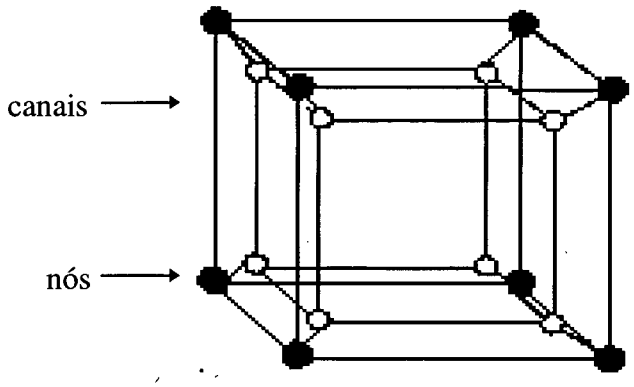

Figura 2.4: Multicomputador com rede de interconexão em hipercubo.

Os multicomputadores com rede de interconexão em hipercubo são os que apresentam, em média, a menor distância de comunicação entre nós. Alguns equipamentos como o *Cosmic Cube*, Ametek Séries 2010, Intel *Personal Supercomputer* e Ncube/10 são baseados em hipercubo.

2.2- Multicomputadores com rede de interconexão dinâmica

A rede de interconexão é composta por comutadores que podem ser reconfigurados por *software*. A distância entre dois nós sempre é de tamanho 1 (um). Toda troca de mensagem entre dois nós exige antes a configuração do canal de comunicação do comutador. Dois exemplos de rede de interconexão dinâmica são barramento e crossbar.

2.2.1- Rede de interconexão em barramento

Em um multicomputador com rede de interconexão dinâmica em barramento, mostrado na figura 2.5, os nós são conectados a um meio de comunicação único. A transferência de mensagens só podem ser feitas quando o barramento estiver livre. A capacidade do barramento é limitada a uma mensagem por vez.

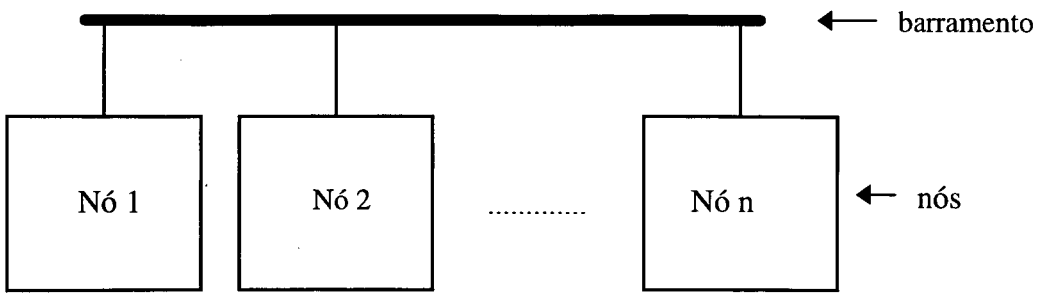

Figura 2.5: Multicomputador com rede de interconexão em barramento.

Barramentos são adequados para arquiteturas com poucas trocas de mensagens e de tamanho curto. Possuem simplicidade e alta capacidade de expansão, além de baixo custo.

2.2.2 - Rede de interconexão em *crossbar*

O crossbar é uma rede de interconexão dinâmica para conexão de **n** nós com alta velocidade de chaveamento, executadas através de um comutador de conexões eletrônico.

A figura 2.6 mostra um multicomputador com rede de interconexão dinâmica baseado em crossbar .

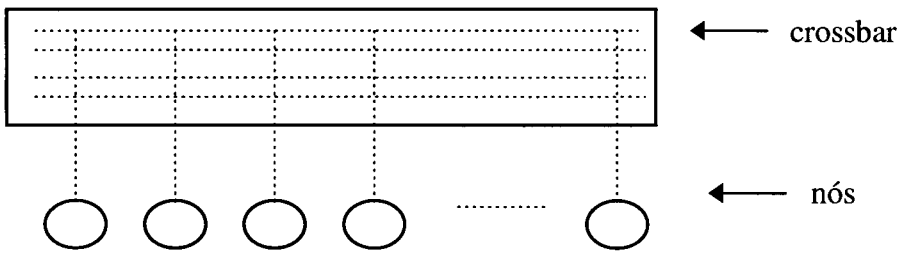

Figura 2.6: Multicomputador com rede de interconexão em crossbar

Diversas comunicações podem ser executadas simultaneamente entre diferentes nós, fornecendo excelente desempenho. As desvantagens de *crossbar* são o alto custo e a baixa capacidade de expansão, devido a quantidade de conexões necessárias ($\mathbf{n}^2$, onde **n** representa o número de nós do multicomputador).

3 - O multicomputador Nó //

Este capítulo apresenta as principais características do *hardware* do Nó//, seu sistema operacional e a primeira versão do simulador desenvolvido por [MON95] e [CAM95].

3.1 - O hardware

O multicomputador Nó// apresenta como principais características de *hardware* (figura 3.1):

- Sua arquitetura possui um nó de controle e vários nós de trabalho, interligados por dois dispositivos, um comutador de conexões e um barramento de serviços;
- Cada nó de trabalho tem processador, memória privativa e canais de comunicação;
- Os nós de trabalho se comunicam exclusivamente através de troca de mensagens, sendo a transmissão destas implementada através do comutador de conexões;
- O controle do comutador de conexões, do barramento de serviços e a alocação/liberação de nós de trabalho é feito pelo nó de controle;
- O barramento de serviços é utilizado para comunicação entre os nós de trabalho e o nó de controle, exclusivamente para troca de mensagens de controle.

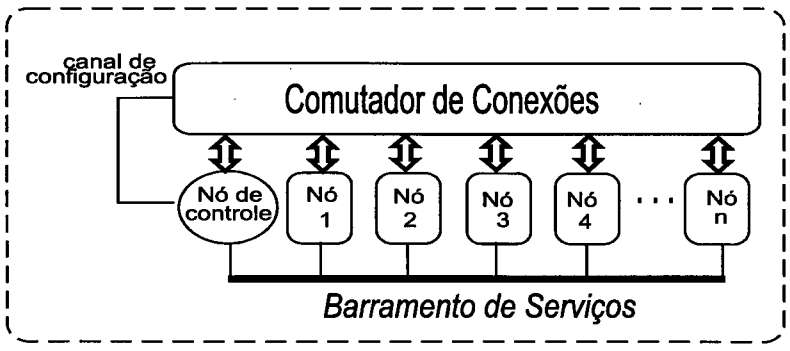

Figura 3.1: Arquitetura do Nó//.

⇒ Os nós de trabalho

Está previsto uma grande quantidade de nós de trabalho. Inicialmente, no primeiro protótipo, considera-se ideal de 16 a 32 nós. Cada nó de trabalho é composto de um processador i486, memória RAM de 1 ou 2 *Mbytes* e pequeno código especial em ROM responsável pela inicialização do sistema operacional e rotinas básicas de comunicação. Todos os nós de trabalho possuem ligação com o barramento de serviços e o comutador de conexões.

⇒ O nó de controle

O nó de controle possui a mesma configuração de um nó de trabalho, com exceção do código em ROM, que executa funções especiais de controle. Sua função é controlar as conexões através do comutador de conexões, as comunicações através do barramento de serviços e a alocação e liberação de nós de trabalho.

⇒ O comutador de conexões

O comutador de conexões é do tipo *crossbar*. Ele estabelece uma rede de interconexão dinâmica entre os nós processadores (nós de trabalho), permitindo comunicações bidirecionais para transmissão e recepção de mensagens, através de canais diretos entre os nós comunicantes, eliminando assim a necessidade de roteamento de mensagens. No primeiro protótipo está previsto apenas um canal de comunicação por nó, porém, na máquina real, está prevista a possibilidade de mais canais de comunicação por nó.

⇒ O barramento de serviços

O barramento de serviços é usado para transmissão de mensagens de controle entre os nós de trabalho e o nó de controle. O nó de controle consulta cada nó de trabalho pelo barramento de serviços, podendo receber requisições de conexão/desconexão de canais de comunicação via comutador de conexões, e alocação/liberação de nós de trabalho.

3.2 - O sistema operacional

O sistema operacional do Nó// será compatível com UNIX, com expansão que permitirá a troca de mensagens entre processos.

Especificado e desenvolvido em [MON95] e [CAM95], ele originalmente foi denominado CRUX, e oferece comandos dispostos em camadas, como mostrado na figura 3.2.

Camada das aplicações
Camada das chamadas de sistema
Camada das comunicações de alto nível
Camada das chamadas do núcleo
Camada das comunicações de baixo nível
Camada da máquina real (*Hardware*)

Figura 3.2: Camadas do sistema operacional do Nó//.

As chamadas da camada de sistema

As chamadas da camada de sistema correspondem às existentes em UNIX, estando prevista a compatibilidade com o padrão POSIX.1, proposto pela IEEE [LEW91].

As chamadas da camada de comunicações de alto nível

A camada das comunicações de alto nível provê suporte para troca de mensagens entre processos, através das chamadas:

- *Send*

  Um processo em um nó de trabalho qualquer solicita ao sistema operacional o envio de uma mensagem a um processo localizado em um determinado nó de trabalho.

- *Receive*

  Um processo em um nó de trabalho qualquer solicita ao sistema operacional a recepção de uma mensagem de um outro processo localizado em um determinado nó de trabalho.

- *ReceiveAny*

Um processo em um nó de trabalho qualquer solicita ao sistema operacional a recepção de uma mensagem de outro processo em um nó de trabalho qualquer.

As chamadas da camada do núcleo

As chamadas da camada do núcleo fornecem operações para pedido de alocação/liberação e conexão/desconexão de nós de trabalho. As operações de conexão/desconexão de nós de trabalho envolvem ações sobre o barramento de serviços e o comutador de conexões, e as operações de alocação/liberação de nós de trabalho envolvem ações sobre o barramento de serviços.

Para permitir a conexão e desconexão de nós de trabalho, existem as seguintes chamadas:

- *Connect*

  Um processo em um nó de trabalho qualquer solicita ao sistema operacional sua conexão através de um canal do comutador de conexões a um outro processo em um determinado nó de trabalho.

- *ConnectAny*

  Um processo em um nó de trabalho qualquer solicita ao sistema operacional sua conexão através de um canal do comutador de conexões a um outro nó de trabalho qualquer.

- *Disconnect*

  Um processo em um nó de trabalho qualquer solicita ao sistema operacional sua desconexão através do canal do comutador de conexões de um outro nó de trabalho.

Para permitir a criação e remoção de processos existem as seguintes chamadas na camada do núcleo:

- *Allocate*

  Um processo em um nó de trabalho qualquer solicita ao sistema operacional a alocação de um determinado nó de trabalho livre.

- *AllocateAny*

  Um processo em um nó de trabalho qualquer solicita ao sistema operacional a alocação de um nó de trabalho livre qualquer;

- *Deallocate*

  Um processo em um nó de trabalho qualquer solicita ao sistema operacional a liberação de um nó de trabalho.

As chamadas da camada de comunicações de baixo nível

O objetivo da camada de comunicações de baixo nível é de prover serviços de comunicação, via barramento de serviços, entre o nó de controle e os nós de trabalho.

As comunicações pelo barramento de serviços tem sempre o nó de controle como origem ou destino, e todas elas se desenvolvem sobre o comando deste nó. As comunicações pelo barramento de serviços são feitas através das seguintes chamadas:

- *W_SendReceive*

  Um processo em um nó de trabalho qualquer envia uma mensagem ao nó de controle e aguarda a recepção da resposta.

- *C_ReceiveAny*

  O processo executado no nó de controle aguarda a recepção de uma requisição de serviço enviado por um nó de trabalho.

- *C_Send*

  O processo executado no nó de controle envia uma mensagem a um determinado nó de trabalho.

3.3 - O simulador do Nó//

Como forma de acelerar o processo de desenvolvimento do ambiente completo do Nó//, foi desenvolvido um simulador em um computador AT 386/486, através de multiprogramação ([MON95] e [CAM95]). Este simulador tem como principal vantagem permitir o desenvolvimento de *software* em paralelo com a construção do multicomputador. Com ele será possível desenvolver, testar e depurar *software* antes de portá-lo para o Nó//.

Outra vantagem deste simulador foi fornecer dados para um trabalho de avaliação de desempenho, baseado na técnica de simulação (MER95).

O simulador foi desenvolvido em ambiente DOS e tomou como base parte do código do sistema operacional XINU [COM84]. Na primeira versão do simulador, foi considerado uma configuração mínima do Nó//, contendo até trinta e dois nós, com até 64 *kbytes* de memória e apenas um canal de comunicação por nó.

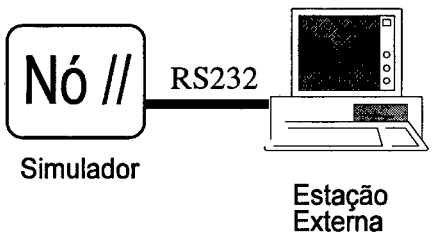

Figura 3.3: Simulador ligado à estação externa.

O simulador se comunica com uma estação de trabalho (figura 3.3), via interface serial RS232, através da qual inicialmente será recebido o código do sistema operacional, e após servirá como meio de comunicação com o usuário.

4 - O enfoque micronúcleo.

Este capítulo comenta a tendência atual no desenvolvimento de sistemas operacionais em camadas hierárquicas, concentrando-se na camada que gerência os recursos principais de um computador, o micronúcleo. O projeto do Nó// prevê a construção de um sistema operacional baseado neste enfoque.

4.1 - Conceitos básicos

A crescente quantidade de recursos que um sistema operacional deve gerenciar aumentaram sua complexidade e tamanho. Como uma forma de minimizar os problemas causados por esta tendência, é utilizada a técnica de desenvolvimento em camadas hierárquicas como forma de organizar o desenvolvimento de sistemas operacionais. Na técnica de desenvolvimento em camadas, cada função de uma camada é traduzida em chamadas à camada inferior, resultando em características importantes como legibilidade e facilidade de manutenção.

É tendência atual no desenvolvimento de sistemas operacionais agrupar-se a parte do código responsável por funções críticas, como gerenciamento de processos, em uma camada acima da camada física (*hardware*), normalmente chamada de micronúcleo.

O micronúcleo é uma pequena porção do sistema operacional que contém as rotinas mais utilizadas e de tarefas críticas, residindo em armazenamento primário. Suas funções geralmente incluem:

- controle de interrupções;
- manipulação de processos;
- escalonamento;
- comunicação entre processos;
- gerência de armazenamento primário;
- suporte a sistemas de arquivos.

Os principais objetivos da utilização de micronúcleos são basicamente o desempenho e a confiabilidade do sistema operacional.

Para garantir a confiabilidade do sistema todos os recursos de *hardware* que sejam compartilhados no computador, como DMA, interrupções, portas de entrada e saída, vídeo, memória e disco são gerenciados pelo micronúcleo e são acessados por processos através de chamadas ao sistema.

O processamento de interrupções é uma das suas mais importantes funções. Em sistemas multiusuários existe um constante fluxo de interrupções que devem ser atendidas pelo processador e que são gerenciadas pelo micronúcleo. Respostas rápidas a interrupções é essencial para que os recursos do sistema sejam bem utilizados.

O código do micronúcleo contém apenas as rotinas mais utilizadas pelo sistema, sendo um código pequeno e otimizado, o que garante a velocidade de processamento.

4.2 - Sistemas Operacionais baseados em micronúcleo

4.2.1 - O Amoeba

O Amoeba [Tan92] foi desenvolvido com duas pressuposições sobre o *hardware* do futuro:

1. computadores terão um grande número de CPU;
2. cada CPU terá dezenas de *Mbytes* de memória principal.

O micronúcleo do Amoeba tem como funções básicas:

- gerenciar processos e *threads*;
- suporte ao gerenciamento de memória em baixo nível;
- comunicação entre processos;
- gerenciamento de entrada e saída (I/O).

A primeira tarefa do micronúcleo é o gerenciamento de processos. Como muitos sistemas operacionais, o Amoeba suporta o conceito de processos. Além disto, suporta também múltiplos *threads*.

A segunda tarefa do micronúcleo é o gerenciamento de memória em baixo nível, fornecendo suporte para a criação e eliminação dinâmica de processos e *threads*.

A terceira tarefa é a comunicação entre processos. Duas formas de comunicação são possíveis: comunicação ponto a ponto e comunicação em grupo.

A última função é o gerenciamento de I/O de baixo nível. Para cada dispositivo existe um *driver* correspondente, que gerencia todos as requisições de I/O destinadas a ele.

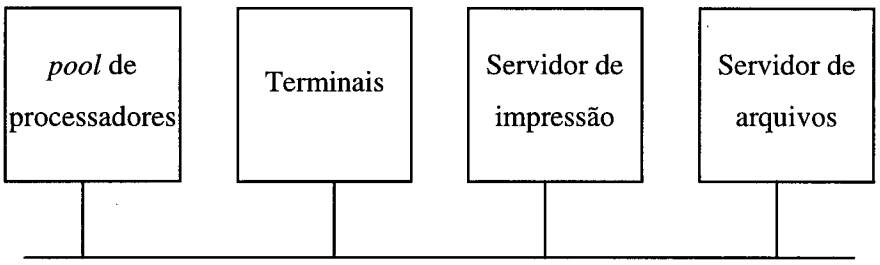

Figura 4.1: Arquitetura Amoeba.

A arquitetura do Amoeba é baseado no modelo mostrado na figura 4.1. Neste modelo, todo o poder de processamento é localizado em um ou mais *pool* de processadores. Um *pool* de processadores consiste de um substancial número de processadores, cada um com sua memória local e conexão de rede.

Os processadores do *pool* podem ser de diferentes arquiteturas. O Amoeba foi desenvolvido para trabalhar com sistemas heterogêneos e múltiplas arquiteturas, como Intel, Motorola e outras.

4.2.2 - O Mach

Outro exemplo de um sistema operacional moderno baseado em micronúcleo é o Mach [Tan92].

As principais metas quando do desenvolvimento do sistema operacional Mach foram:

1. prover uma base para construção de outros sistemas operacionais;
2. suportar um grande espaço de endereçamento;
3. possibilitar acesso transparente a recursos de rede;
4. explorar paralelismo a nível de sistema e aplicações;
5. tornar o Mach portável para uma coleção de máquinas.

A idéia geral do Mach era explorar multiprocessamento e sistemas distribuídos ao mesmo tempo, sendo capaz de emular diversos sistemas operacionais existentes, como UNIX, por exemplo.

O micronúcleo MACH .

O micronúcleo Mach foi desenvolvido baseado na capacidade de emular outros sistemas operacionais. Esta emulação é feita por uma camada de *software* que roda fora do micronúcleo, a nível de usuário, como mostrado na figura 4.2.

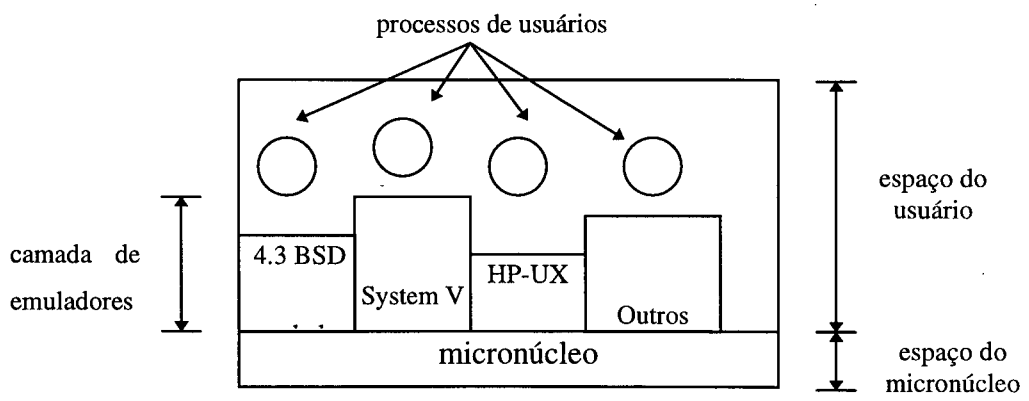

Figura 4.2: Arquitetura Mach.

O micronúcleo do Mach, como outros micronúcleos, é responsável pelo gerenciamento de processos, gerenciamento de memória, comunicação entre processos e serviços de entrada e saída.

Arquivos, diretórios e outras funções tradicionais de sistemas operacionais são controladas no espaço de usuário.

O micronúcleo do Mach gerencia:

- Processos;
- *Threads*;
- objetos de memória;
- portas;
- mensagens.

Um processo é a unidade básica para alocação de recursos, incluindo espaço de endereçamento e uma coleção de *threads*.

Uma *thread* é uma entidade executável. Ela tem um contador de programa e um conjunto de registradores associados a ela. Cada *thread* é parte de um único processo.

Objetos de memória são estruturas de dados que podem ser mapeados dentro do espaço de endereçamento de um processo, podendo ocupar uma ou mais páginas de memória, e formam a base do sistema de memória virtual do Mach.

Portas são os recursos que permitem a comunicação entre processos, através da troca de mensagens.

A troca de mensagens é implementada com caixas postais.

5 - Gerência de memória

Este capítulo descreve os métodos de gerência de memória mais utilizados e os recursos dos processadores i486 em modo protegido.

5.1 - Características gerais dos processadores i486

A organização de memória mais simples possível prevê apenas um processo de usuário na memória. Neste caso a memória é dividida entre o sistema operacional e o processo de usuário. Este tipo de gerenciamento de memória é utilizado apenas em pequenos microcomputadores. Nos demais, existe a possibilidade de ter-se mais de um processo na memória ao mesmo tempo. Nesses casos, os métodos normais de gerenciamento de memória são segmentação, paginação e a combinação de segmentação e paginação. Esses métodos envolvem o conceito de memória virtual, que pode ser definida como o mapeamento de endereços lógicos, utilizados em programas, diferentes dos endereços físicos disponíveis na memória principal.

Após a inicialização da máquina, os processadores i486 sempre começam processando no modo real, que é o modo compatível com os processadores 8086, e possuem poucos recursos.

As principais características dos processadores 8086 e do modo real dos processadores i486 são:

- capacidade de endereçamento de até 1 *Mbyte*;
- registradores de 16 bits;
- gerenciamento de memória através de segmentos sem proteção;
- segmentos de até 64 *kbytes*.

Os processadores i486 são processadores de 32 bits e possuem como principais características:

- operação nos modos real, protegido e virtual;
- registradores de 32 bits;
- endereçamento de memória física de até 4 *Gbytes* (2^{32} *bytes*);

- suporte à multiprogramação;
- gerenciamento de memória avançado nos modos protegido e virtual, incluindo segmentação, paginação e segmentação/paginação.

5.2 - Os recursos dos processadores i486

5.2.1 - Segmentação

A segmentação usa o gerenciamento de memória através de blocos de tamanhos variáveis, com a divisão de programas em entidades lógicas (código, dados e pilha). Cada segmento é composto de uma seqüência contígua de endereços físicos do tamanho necessário para acomodar as entidades lógicas dos programas.

A segmentação no modo protegido dos processadores i486 é baseada em tabelas de descritores de segmentos: a tabela de descritores globais (GDT) e as tabelas de descritores locais (LDT).

Cada descritor ocupa 8 *bytes* e contém como principais informações:

- endereço base do segmento;
- limite do segmento (tamanho);
- tipo de descritor (sistema ou aplicação);
- presença ou não na memória principal;
- nível de privilégio do descritor (DPL).

A estrutura dos descritores estão definidas em [INT90a], capítulo 5.

A tradução de um endereço lógico pelo sistema de segmentação é mostrada na figura 5.1, seguindo os seguintes passos:

1. um endereço lógico, utilizado em um processo, é formado por um seletor de segmento e um deslocamento;
2. o seletor de segmento aponta para uma entrada da GDT ou de alguma LDT, sendo que o seletor contém o endereço base e o limite (tamanho) do segmento, que são conferidos;
3. o endereço base é somado ao deslocamento do endereço lógico;

4. o resultado da soma gera o endereço linear, que é o mesmo que o endereço físico.

O mecanismo de segmentação fornece vários graus de proteção. Um descritor contém sempre um endereço base e o limite do segmento. Uma proteção é baseada no tamanho do segmento. Se o deslocamento de um endereço for maior que o limite (tamanho), ocorre um erro de violação de proteção.

Outra proteção é baseada em níveis de privilégio. Cada descritor de segmento contém um nível de privilégio do descritor (DPL), variando de 0 a 3, sendo 0 o maior nível de privilégio. Em um momento qualquer, o processador está operando em um nível de privilégio corrente (CPL), que é determinado pelo nível de privilégio do segmento de código em execução. Transferências entre diferentes níveis são providas através de um descritor especial chamado *call gate*.

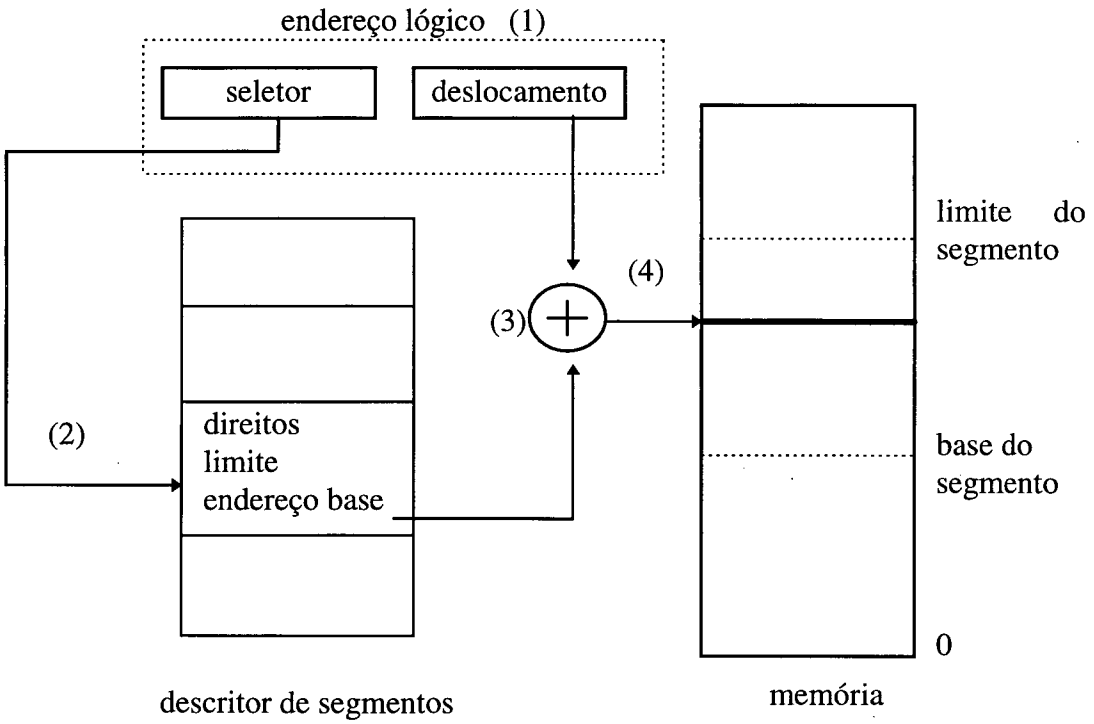

Figura 5.1: Tradução de endereços lógicos utilizando segmentação.

A figura 5.2 mostra a capacidade de proteção entre processos dos processadores i486 baseada em níveis de privilégios. Esta capacidade existe a partir da opção de definir uma LDT para cada processo, o que resulta em um intervalo isolado de endereçamento para cada um deles, com possibilidade de compartilhamento. O

compartilhamento é possível quando vários processos apontam para um mesmo seletor de segmento.

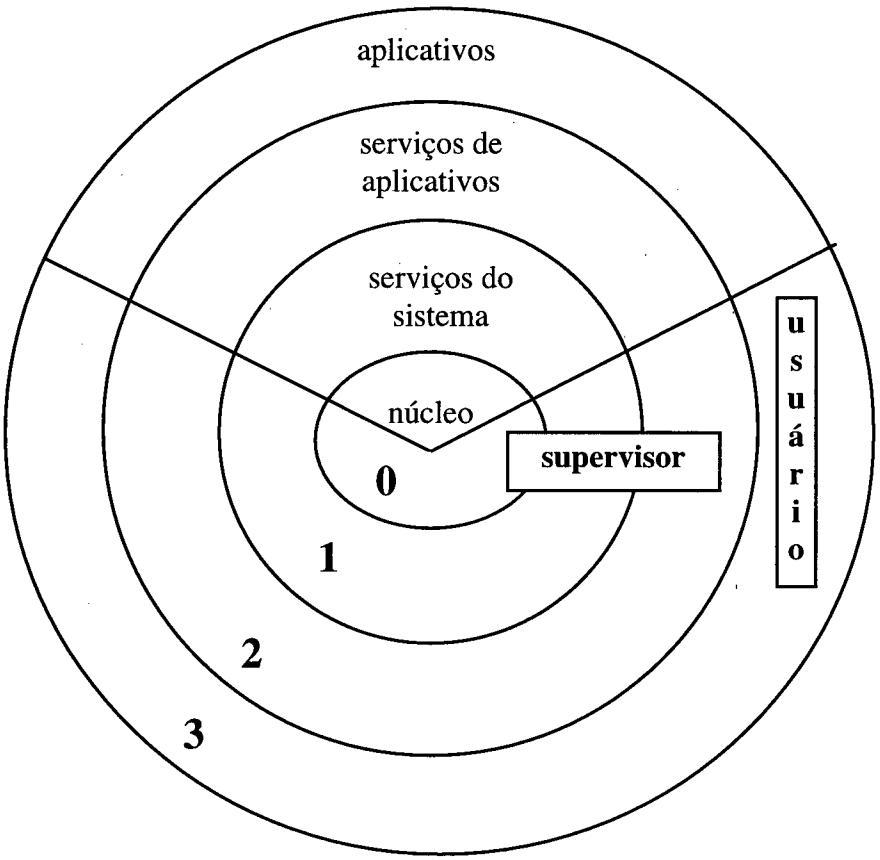

Figura 5.2: Proteção utilizando segmentação no modo protegido.

Como regra, um processo de um determinado nível pode acessar:

- segmentos de dados com o mesmo ou menor nível de privilégio (CPL do processo atual >= DPL do segmento acessado);
- segmentos de código com o mesmo ou maior nível de privilégio (CPL do processo atual <= DPL do processo a ser chamado).

5.2.2 - Paginação

A paginação usa o gerenciamento de memória através de páginas de tamanho fixo. Utilizando paginação, programas ocupam uma certa quantidade de páginas, sendo transparente a divisão de código, dados e pilha.

Os processadores i486 possuem um sistema de paginação de dois níveis, com páginas de tamanho de 4 *kbytes* (4096 *bytes*). A tradução do endereço lógico em endereço físico utiliza duas tabelas: a tabela de diretório de páginas (*page directory*) corresponde ao primeiro nível, e uma tabelas de páginas (*page table*), que correspondem ao segundo nível. A tabela de diretório de páginas pode endereçar até 1024 tabelas de páginas, que por sua vez podem endereçar até 1024 páginas físicas de 4 *kbytes* na memória.

Para aumentar a velocidade de tradução de endereços lógicos para físicos, existe uma *cache* interna nos processadores i486 de 32 entradas, chamada TLB (*translation loockaside buffer*).

Os bits de controle dos descritores de páginas, mostrados na figura 5.3, permitem a implementação de serviços especiais de um sistema operacional, sendo:

- o bit AVL para indicar se a página está disponível na memória principal ou não;
- o bit D para indicar se a página foi alterada;
- o bit A para indicar se a página foi acessada ou não;
- o bit U para definir dois níveis de proteção: usuário e supervisor;
- o bit W para definir direitos de leitura/escrita ou somente de leitura;
- o bit P para indicar se a página está presente na memória principal ou não.

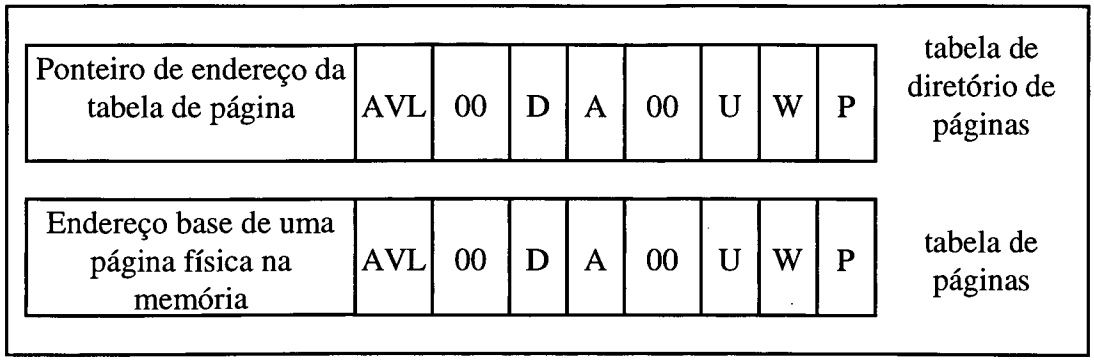

Figura 5.3: Descritores das tabelas utilizadas em paginação.

A tradução de um endereço lógico pelo sistema de paginação é mostrado na figura 5.4, seguindo os seguintes passos:

1. um endereço lógico (32 bits) é dividido em três campos;
2. os bits de 22 a 31 apontam para uma entrada na tabela de diretório de páginas, sendo que o endereço base desta tabela deve ser previamente carregado no registrador CR3 do processador;
3. nesta tabela cada entrada contém o endereço base de uma tabela de páginas;
4. os bits de 12 a 21 do endereço lógico apontam para um entrada na tabela de páginas;
5. esta entrada da tabela de páginas contém o endereço base de uma página física na memória;
6. os bits de 0 a 11 do endereço lógico informam o deslocamento dentro desta página física na memória, que é somado ao endereço base formando o endereço físico.

A proteção e o compartilhamento são menos naturais do que em segmentação. Como os processos não definem sua visão lógica da memória de forma independente, qualquer processo pode efetuar alterações em uma página, desde que referidas pelas tabelas do mecanismo de tradução de endereços, pois não existe proteção em relação a processos, apenas a nível de usuário ou supervisor e leitura ou escrita.

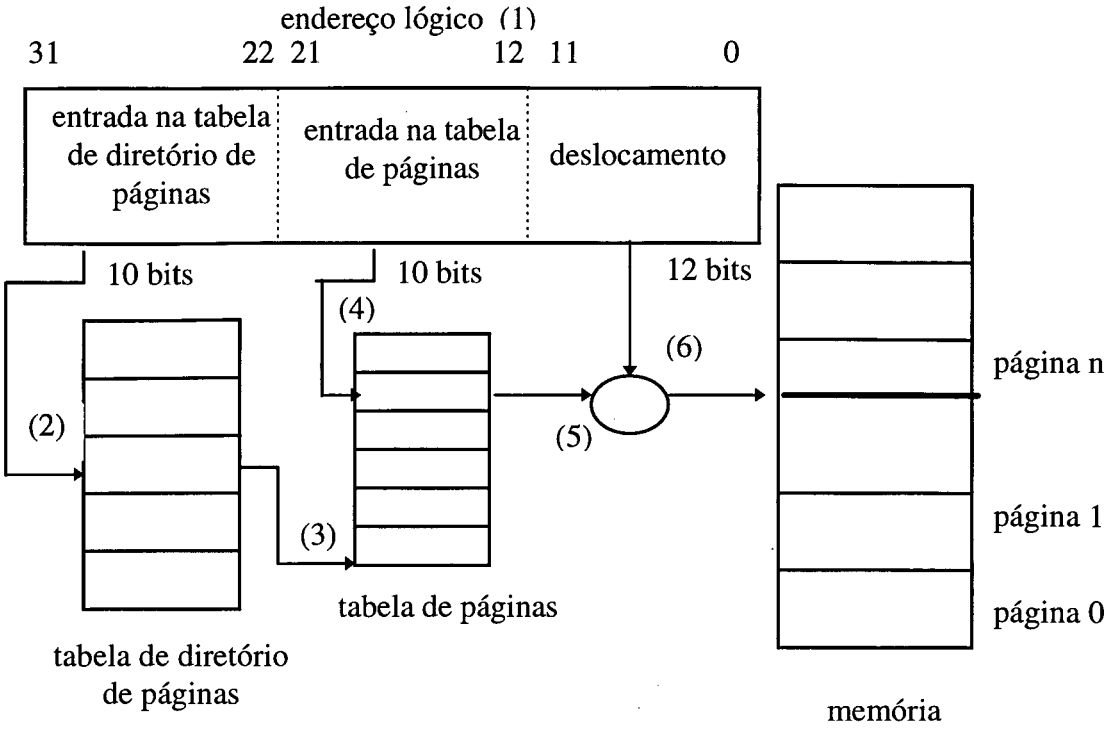

Figura 5.4: Tradução de endereços lógicos utilizando paginação.

Para efetuar compartilhamento, existe o problema de que nem todas as páginas podem ser compartilhadas, especialmente páginas que contém a pilha de um processo ou de dados de controle do supervisor, como os TSS dos processos (segmento utilizado para guardar o contexto de um processo no modo protegido).

Outro problema ocorre quando uma página compartilhada é transferida da memória para o disco. Todos os processos que a utilizam devem ser notificados (para cada entrada na tabela de diretório de páginas ou na tabela de páginas existe o bit P, que indica se a página está na memória principal ou em disco). Da mesma maneira, quando um processo é finalizado, as páginas compartilhadas não podem ter sua área liberada ao sistema operacional até que o último processo que a utilize termine sua execução.

A proteção é feita fisicamente, pois processos só podem acessar dados das páginas referenciadas na tabela de páginas, e de acordo com as restrições de acesso. O compartilhamento é suportado pela existência de diferentes processos com entradas nas tabelas de páginas que apontam para o mesmo conjunto de páginas físicas na memória.

5.2.3 - Segmentação/paginação

Muitos sistemas exploram a combinação de dois mecanismos de gerência de memória: segmentação e paginação. A desvantagem deste modo combinado de gerência de memória é a necessidade de manipular um grande número de tabelas (de segmentos e de páginas), envolvendo maior quantidade de cálculos para transformar endereço lógico em endereço físico. Apesar destes cálculos serem efetuados automaticamente pelo processador, o excesso de cálculos e tabelas resulta em menor desempenho.

A combinação de segmentação com paginação nos processadores i486 é feita criando-se as estruturas correspondentes aos dois mecanismos em separado. Com isto, todas as características dos dois métodos de gerenciamento de memória são herdados.

A tradução de um endereço lógico em um sistema de segmentação/paginação é mostrado na figura 5.5, seguindo os seguintes passos:

1. um endereço lógico possui um seletor de segmento e um deslocamento;
2. o seletor aponta para uma entrada da GDT ou de uma LDT;
3. a entrada na GDT ou LDT contém o endereço base do segmento; a esse endereço base é somado o deslocamento;
4. o resultado da soma forma um endereço linear, que será traduzido pelo sistema de paginação. Este endereço é dividido em três campos: diretório de páginas, tabela de páginas e deslocamento;
5. o primeiro campo faz referência à tabela de diretório de páginas;
6. a entrada na tabela de diretório de páginas aponta para a base de uma tabela de páginas;
7. o segundo campo do endereço linear aponta para uma entrada dentro da tabela de páginas;
8. o entrada na tabela de páginas contém o endereço base de uma página física na memória;
9. o endereço destino na memória física é calculado pelo terceiro campo do endereço físico, o deslocamento, que é somado ao endereço base da página física na memória.

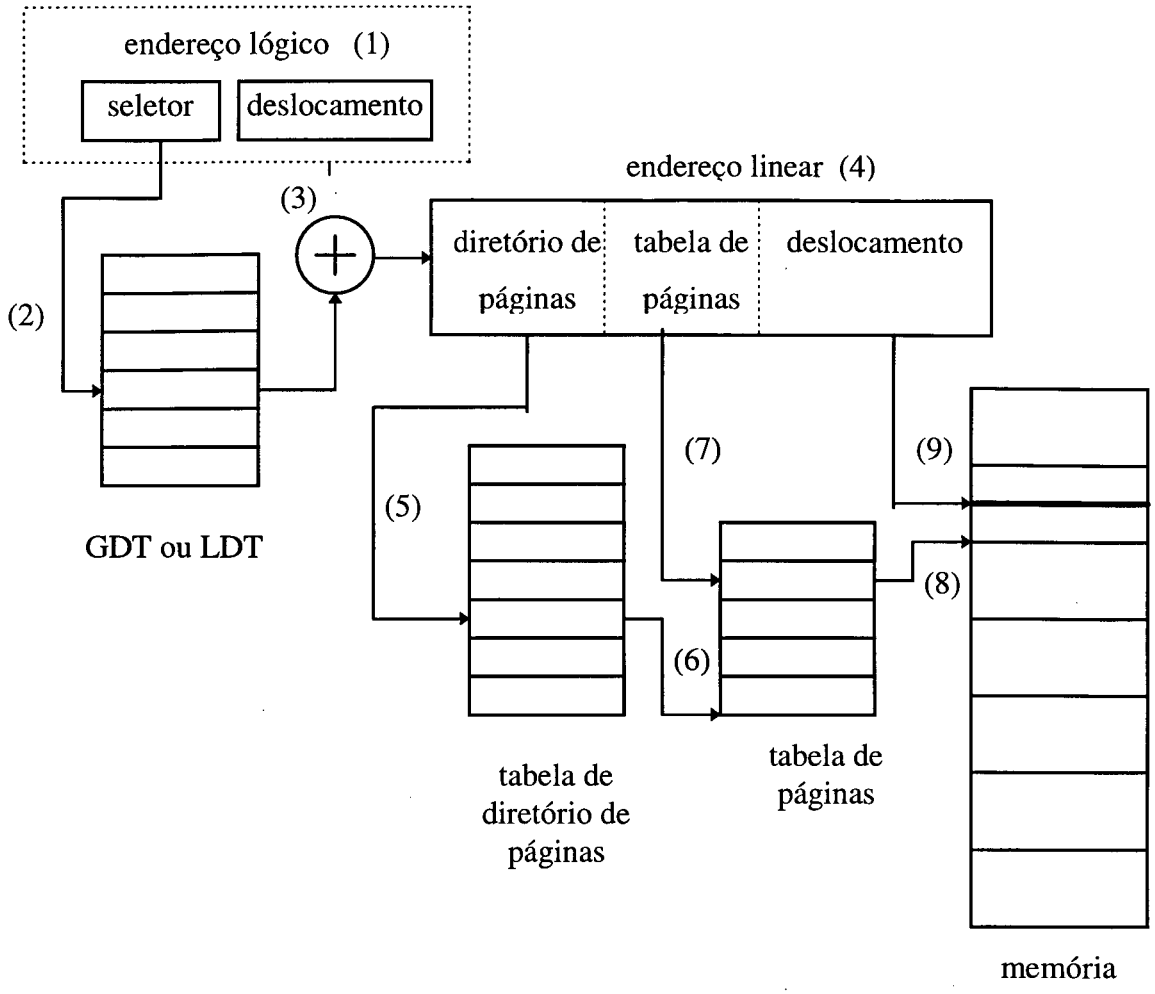

Figura 5.5: Tradução de endereços lógicos utilizando segmentação/paginação.

A proteção em segmentação/paginação é feita combinando os recursos de proteção dos dois mecanismos. Inicialmente é avaliada a proteção de segmento (limite e níveis de privilégio) e após a proteção de páginas (usuário/supervisor). O compartilhamento é feito de forma semelhante a segmentação: processos diferentes apontam para o mesmo seletor de segmento.

5.3 - A memória expandida

O DOS foi projetado como um sistema operacional para máquinas com processadores 8086/88, que endereçam até 1 *Mbyte* de memória. O conjunto dos primeiros 640 *kbytes* foi denominado memória convencional, sendo utilizada pelo sistema operacional e programas. O restante da memória acima dos 640 *kbytes* foi reservada para propósitos especiais, como a ROM BIOS e adaptadores de vídeo, entre outros. Esta área normalmente não é utilizada por completo.

A memória expandida é uma expansão de memória desenvolvida para computadores com processadores 8086/88 através de placas conectadas aos *slots* do computador e endereçadas através de entrada e saída (I/O). Nos computadores com processadores 80286, 80386, 80486 e *Pentium* esta memória pode ser simulada através de *software*, utilizando-se a memória estendida

A especificação de memória expandida desenvolvida pela Lotus, Intel e Microsoft (LIM-EMS), permite o acesso a até 8 *Mbytes* (32 *Mbytes* na versão 4.0) além do limite do DOS, através de um mecanismo de paginação de memória. A EMS utiliza 64 *kbytes* de memória contínua, chamada de moldura de páginas, na área acima de 640 *kbytes* de memória, que é utilizada como uma janela para o acesso a memória expandida.

Os pedidos de leitura/escrita endereçados à memória dentro da moldura de páginas são redirecionados para a área de memória expandida especificados pelo mapeamento atual.

O acesso à EMS é feito seguindo basicamente os seguintes passos:

- um programa aplicativo reserva páginas lógicas, de tamanho de 16 *kbytes*, de acordo com sua necessidade e dentro do limite disponível de memória expandida instalada;

- a moldura de páginas de 64 *kbytes* está dividida em páginas de 16 *kbytes*, na área de memória entre 640 *kbytes* e 1 *Mbyte*, que é endereçável pelo DOS. O programa pode especificar quais as páginas lógicas de memória expandida que serão mapeadas nas quatro páginas físicas da moldura de páginas. Desta forma, apenas 64 *Kbytes* de memória expandida estão disponíveis em cada instante;

- para poder acessar outras páginas de memória expandida, o programa deve solicitar outro mapeamento, fazendo a troca das páginas dentro da moldura de páginas.

O programa aplicativo deve utilizar as rotinas de um utilitário, chamado gerenciador de memória expandida (EMM), que fornece todas as rotinas necessárias para que um aplicativo utilize a memória expandida.

A memória expandida inicialmente foi desenvolvida como uma combinação de *software* e *hardware* para os computadores com processadores 8086. Nos computadores com processadores i486 a memória estendida é utilizada como memória expandida, através de gerenciadores que emulam memória expandida executando um núcleo em modo protegido.

O funcionamento da memória expandida EMS é mostrado na figura 5.6.

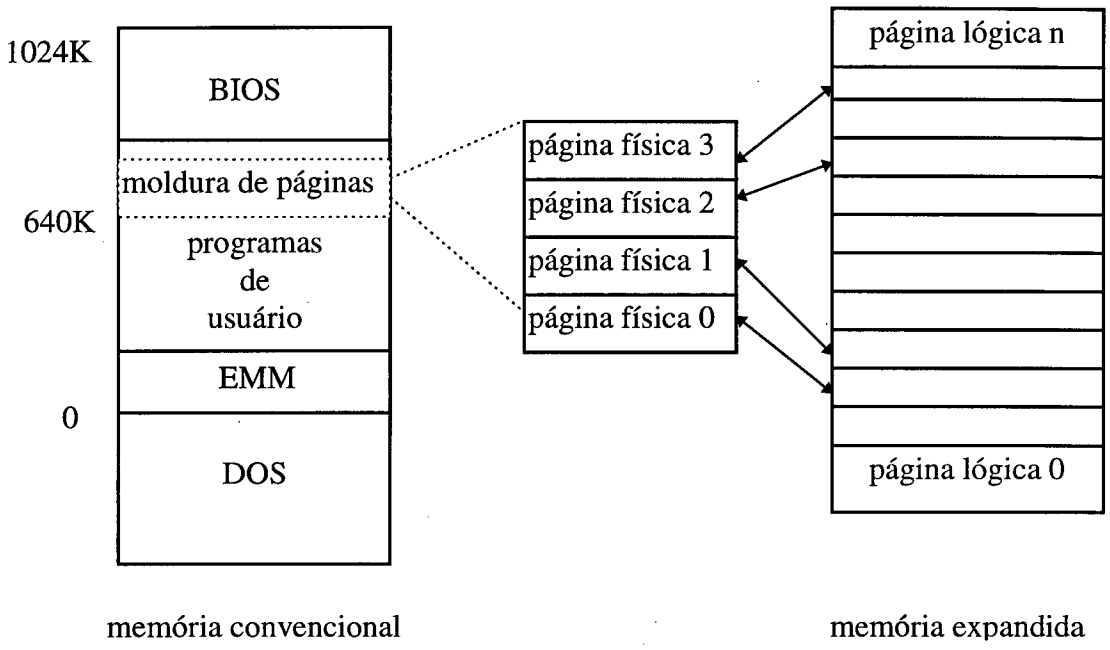

Figura 5.6: Funcionamento da memória expandida.

Na versão 4.0 da EMS diversas alterações foram implementadas, como por exemplo a possibilidade de definição do tamanho de página física da moldura de páginas, a definição do local da moldura de páginas em qualquer posição da memória convencional e a existência de várias molduras de páginas.

5.3 - A memória estendida

Memória estendida é toda memória acima do endereço de 1 *Mbyte* dos computadores PC-AT. A memória estendida existe a partir dos processadores 80286 em diante e só pode ser utilizada no modo protegido, com algumas exceções, diferenciando-a da memória expandida, que foi desenvolvida para possibilitar a expansão de memória para processos executados em processadores 8086 (modo real dos processadores i486).

Para acessar a memória estendida, o programa deve inicializar as estruturas do modo protegido. Basicamente, os passos para se ter acesso ao modo protegido são:

- criar a tabela descritora global (GDT);
- criar as tabelas descritoras locais (LDT), se existirem;
- atualizar os tipos de segmentos e os direitos de acesso nas tabelas descritoras;
- carregar todos os registradores de segmentos com descritores válidos que apontam para entradas da GDT ou de alguma LDT;
- carregar o registrador GDTR com o endereço base da tabela GDT;
- estabelecer o endereço das rotinas de atendimento de interrupções no modo protegido;
- alterar o bit PE do registrador de *flags* para entrar no modo protegido.

Após a entrada no modo protegido, deve existir rotinas para gerenciar todo o ambiente, com suporte à multiprogramação, operações de 32 bits, etc.

Na figura 5.7 é mostrada uma visão da distribuição da memória de um computador PC-AT, com memória principal superior a 1 *Mbyte*.

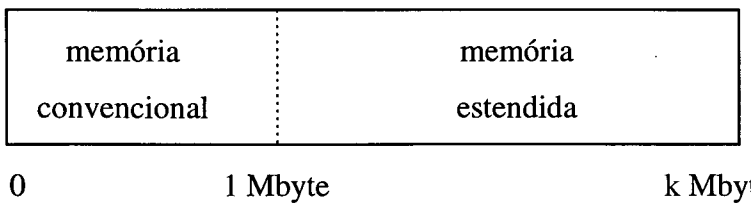

Figura 5.7: Memória nos computadores PC-AT.

A memória estendida existe em computadores que tenham uma quantidade de memória acima de 1 *Mbyte* e processadores que rodem em modo protegido. Nos computadores PC-XT não existe a memória estendida, pois os seus processadores só podem endereçar até 1 *Mbyte*. Nos computadores com processadores 80286, podem existir até 16 *Mbytes* de memória, enquanto que nos computadores baseados nos processadores 80386 e 80486 podem existir até 4 *Gbytes* de memória.

Para ter acesso à memória estendida é necessário utilizar os recursos do modo protegido. As exceções para acesso à memória estendida, via modo real, estão em duas rotinas da BIOS dos computadores PC-AT. A função 88h da INT 15h retorna a quantidade de memória estendida presente, e a função 87h da INT 15h move um bloco de memória entre qualquer posição de memória, convencional ou estendida.

A especificação de memória estendida (XMS) define uma interface padronizada, via *software*, para acesso à memória estendida. No DOS, o acesso a memória estendida XMS é possível através de um utilitário que o acompanha, chamado HIMEM.SYS.

Um resumo das funções XMS é mostrado na tabela 5.1.

NÚMERO	FUNÇÃO
00h	obtém versão da XMS
01h	aloca a área de memória superior
02h	libera a área de memória superior
03h	habilitação global da linha A20
04h	desabilitação global da linha A20
05h	habilitação local da linha A20
06h	desabilitação local da linha A20
07h	informa estado da linha A20
08h	requisita memória estendida livre
09h	aloca bloco da memória estendida
0Ah	libera bloco da memória estendida
0Bh	mover bloco da memória estendida
0Ch	bloqueia bloco da memória estendida
0Dh	desbloqueia bloco da memória estendida
0Eh	obtém informação do manipulador
0Fh	modifica o tamanho do bloco de memória estendida
10h	aloca bloco da memória superior
11h	libera bloco da memória superior

Tabela 5.1: Resumo das funções da memória estendida XMS.

O uso de processadores i486 permite o acesso de até 4 *Gbytes* de memória (convencional e estendida) utilizando registradores de 32 bits, desde que exista o gerenciador de memória estendida e as estruturas necessárias para utilizar os recursos do modo protegido.

5.4 - A DPMI (DOS Protected Mode Interface)

A DPMI é uma proposta de padronização de utilização da memória estendida XMS por extensores do DOS (programas que fornecem acesso a memória estendida a programas aplicativos). Ela define os mecanismos de acesso às estruturas necessárias do modo protegido dos processadores i486 . Através de suas funções, cada processo pode definir sua utilização de memória em uma LDT (a GDT é controlada exclusivamente pela DPMI), o que permite a utilização da memória estendida e um ambiente de multiprogramação sem conflito entre os processos. Ela também fornece suporte para a alternação entre os modos do processador (real, protegido e virtual) e acesso a memória convencional.

As principais rotinas da DPMI são mostradas nas tabelas 5.2 a 5.6.

NÚMERO	FUNÇÃO
0000h	alocar seletor da LDT
0001h	liberar seletor da LDT
0002h	mapear segmento do modo real no seletor
0003h	obter valor de incremento do próximo seletor
0004h	bloquear a memória do seletor
0005h	desbloquear memória do seletor
0006h	obter endereço de base do seletor
0007h	ajustar endereço de base do seletor
0008h	ajustar limite do seletor
0009h	ajustar direitos de acesso do seletor
000Ah	criar alias do seletor para segmento de código
000Bh	obter descritor
000Ch	ajustar descritor
000Dh	alocar seletor específico da LDT

Tabela 5.2: Funções DPMI para gerência da tabela de descritores locais (LDT) .

NÚMERO	FUNÇÃO
0100h	alocar bloco de memória do DOS
0101h	liberar bloco de memória do DOS
0102h	modificar tamanho do bloco de memória do DOS
0500h	informações sobre a memória livre
0501h	alocar bloco de memória
0502h	liberar bloco de memória
0503h	modificar tamanho de bloco de memória
0600h	bloquear região linear
0601h	desbloquear região linear
0602h	marcar região do modo real como paginável
0603h	relocar região do modo real

Tabela 5.3: Funções DPMI para gerenciamento de memória e paginação.

NÚMERO	FUNÇÃO
0200h	obter vetor de interrupção do modo real
0201h	ajustar vetor de interrupção do modo real
0202h	obter vetor do manipulador de exceção do processador
0203h	ajustar vetor do manipulador de exceção do processador
0204h	obter vetor de interrupção do modo protegido
0205h	ajustar vetor de interrupção do modo protegido
0900h	obter e desabilitar estado de interrupção virtual
0901h	obter e habilitar estado de interrupção virtual
0902h	obter estado de interrupção virtual

Tabela 5.4: Funções DPMI para o gerência de interrupções.

NÚMERO	FUNÇÃO
0300h	simular interrupção do modo real
0301h	chamar rotina do modo real com modelo de retorno FAR
0302h	chamar rotina do modo real com modelo de retorno por interrupção
0303h	alocar endereço da chamada-retorno ao modo real
0304h	liberar endereço da chamada-retorno ao modo real
0305h	obter estado dos endereços para salvamento
0306h	obter linha de endereços de comutação do modo da CPU

Tabela 5.5: Funções DPMI de tradução de modos da CPU.

NÚMERO	FUNÇÃO
0400h	obter versão da DPMI
0800h	mapeamento de endereços físicos

Tabela 5.6: Outras funções da DPMI.

É necessário o conhecimento dos recursos dos processadores i486 no modo protegido para utilizar com eficiência a DPMI. Os extensores do DOS, como a DPMI, fornecem suporte para que aplicativos utilizem os recursos de memória estendida, exclusivamente através das suas funções, para garantir uma utilização segura dos recursos de toda a memória do computador e dos recursos dos processadores como por exemplo, proteção e suporte a multiprogramação.

6 - O novo simulador do multicomputador Nó//

Neste capítulo estão descritas as características do novo simulador e as principais funções de seus módulos: gerenciador de processos, gerenciador de memória, comunicação entre processos, controle de entrada e saída e interrupções. É apresentado também as necessidades que motivaram o seu desenvolvimento.

6.1 - A necessidade de um novo simulador

O simulador descrito em [CAM95] e [MON95] utilizou o sistema operacional XINU [COM84] como base para seu o desenvolvimento. Esta escolha ocorreu pelo fato de o XINU se adequar à idéia de multiprogramação do simulador e pela facilidade de alteração de seu código.

O código fonte do XINU é escrito em linguagem C e *Assembly* e sofreu alterações para funcionar como a primeira versão do simulador. Basicamente, as alterações efetuadas foram a simulação dos nós de trabalho e dos dispositivos de comunicação do Nó//.

O simulador baseado no XINU possui limitações sérias que motivaram o desenvolvimento de um novo simulador. A principal limitação diz respeito a gerência de memória.

Sendo o XINU desenvolvido em ambiente DOS, que restringe o uso da memória principal a 640 *Kbytes*, o simulador tem ainda que compartilhar esta área de trabalho com o próprio DOS (núcleo do sistema operacional, vetor de interrupções e interpretador de comandos).

Como cada nó de trabalho do Nó// terá no mínimo 1 *Mbyte* de memória principal, existe uma grande diferença entre a memória disponível para cada nó de trabalho no simulador e na máquina real. Para simular uma configuração real de 8 nós, o simulador precisaria gerenciar pelo menos 8 *Mbyte*s de memória, em contraste com os menos de 640 *Kbytes* disponíveis.

Na prática, a situação é ainda pior. O XINU reserva apenas 64 *Kbytes* para alocar a memória dos nós de trabalho. Isto resulta em nós de trabalho no simulador com uma quantidade de memória disponível inferior a 16 *Kbytes*, levando em consideração uma configuração mínima do Nó// com 4 nós: o nó de controle, o nó de comunicação externa (NCE) e dois nó de trabalho.

A outra limitação é a de que o Nó// não será compatível com o DOS, portanto o código do simulador baseado no XINU não pode ser utilizado como base para o desenvolvimento do seu sistema operacional.

6.2 - O novo simulador.

A motivação inicial ao desenvolvimento de um novo simulador foi solucionar os problemas de gerência de memória da primeira versão.

O novo simulador foi desenvolvido em linguagem *Assembly*, utilizando os recursos avançados do modo protegido dos processadores i486 para gerência de memória, gerência de processos e proteção.

O desenvolvimento do simulador em linguagem *Assembly* permite utilizar o próprio simulador como base para o desenvolvimento do sistema operacional do Nó//, o CRUX. Como o simulador e o Nó// utilizam a mesma arquitetura, baseada nos processadores i486, apenas algumas alterações serão necessárias ao simulador para que ele funcione como o sistema operacional do Nó//.

A utilização de alguma interface disponível, como a DPMI, foi descartada porque o simulador desenvolvido com ela não seria portável para o Nó//, já que o mesmo possui ambiente próprio sem o sistema operacional DOS.

O simulador foi desenvolvido seguindo a tendência de construção de sistemas operacionais modernos, baseados em micronúcleos. As funções implementadas no novo simulador são:

- gerência de processos;
- gerência de memória;
- comunicação entre processos;
- controle de entrada e saída;
- controle de interrupções.

O novo simulador é executado no modo protegido dos processadores i486 e os processos são executados no modo virtual (figura 6.1).

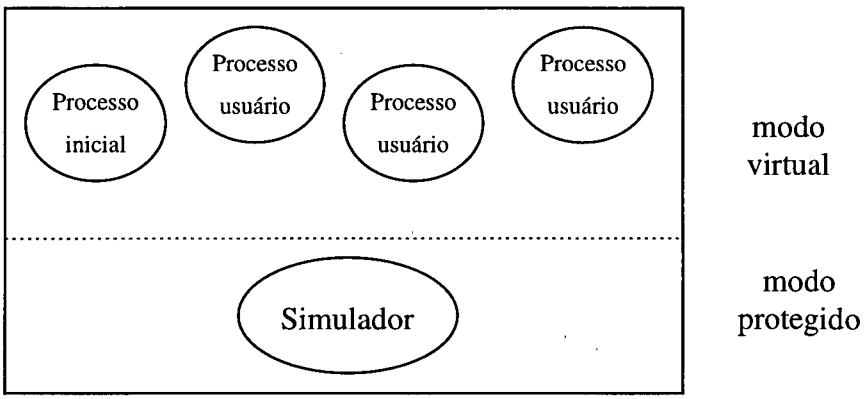

Figura 6.1: Modo de execução do simulador e seus processos.

A execução em modo protegido fornece recursos suficientes para as necessidades do novo simulador. Os recursos oferecidos pelo simulador são acessados através de *traps*, sendo fornecido recursos para :

- processos: criação , eliminação e alteração de estado;
- comunicação: simulação do barramento de serviços e do comutador de conexões, com primitivas para envio e recepção de mensagens;
- entrada e saída: leitura e escrita de portas;
- memória: alocação e liberação de memória dos nós de trabalho;
- controle e tratamento de interrupções.

6.2.1 - O gerenciador de processos.

O gerenciador de processos do simulador permite a execução de vários processos, um para cada nó simulado do multicomputador Nó//, possuindo processador virtual e memória privativa e a comunicação entre eles só é possível

através dos dispositivos de comunicação, que são simulados através de recursos avançados dos processadores i486. Não está implementado a execução de *threads* nesta nova versão do simulador.

O simulador utiliza intensamente todos os recursos avançados dos processadores i486 em modo protegido para executar suas funções. Processos são gerenciados através das seguintes estruturas:

- a tabela de descritores globais (GDT);
- o segmento de estado de cada processo (TSS);
- a tabela de seletores;
- a tabela de processos.

A GDT é criada na etapa de inicialização do simulador, quando ele ainda está executando no modo real dos processadores i486. Entradas livres na GDT são reservadas para a criação dinâmica de processos. Cada entrada, chamada descritor, descreve informações sobre um processo, como tipo de segmento (código, dados, pilha ou TSS), nível de proteção do segmento (DPL), memória reservada, modo de execução, entre outras.

As operações de alocação e liberação de nós de trabalho no multicomputador Nó// são simuladas através da utilização das entradas livres da GDT.

Os processos do simulador, executados em modo virtual dos processadores i486, necessitam apenas de um descritor na GDT, com informações sobre o segmento de estado do processo (TSS).

Para todo processo no simulador é criado um TSS, que é utilizado para armazenar o seu contexto. A cada escalonamento de processos, o contexto de *hardware* (registradores) do processo em execução é salvo automaticamente pela CPU no seu segmento de TSS, lendo um novo contexto do TSS do processo a ser executado.

A estrutura da tabela de processos é mostrada na tabela 6.1. Na tabela de processos é armazenado o estado de execução de cada processo.

Campo	Posição	Descrição	Tamanho	Comentários
0	0	TSS	2	posição na GDT
1	2	Pstate	1	estado do processo
2	3	Pprio	1	prioridade do processo
3	4	livre *	4	livre
4	8	Pname	18	nome do processo
5	26	BC_queue	1	fila de espera para comunicação via barramento de serviços
6	27	BC_NID_esperado	1	identificação do nó (NID) esperado para fazer comunicação pelo barramento de serviços
7	28	BC_msg	1	mensagem (*byte*) a ser transferido pelo barramento de serviços
8	29	NID_emissor	1	identificação do nó que tenta enviar uma mensagem ao processo
9	30	CB_NID_conectado	1	identificação do nó do processo ao qual o processo está conectado via comutador de conexões
10	31	CB_msg	1	mensagem (*byte*) transferido via comutador de conexões

Tabela 6.1: Tabela de processos do novo simulador.

O simulador executa um escalonamento preêmptivo, garantindo controle sobre processos e divisões de fatia de tempo de processador (*time-slice*) iguais. Todos os processos são considerados com a mesma prioridade pelo escalonador.

A comunicação entre processos é simulada através da tabela de processos. Cada processo possui entradas que controlam o seu estado na utilização do comutador de conexões e do barramento de serviços e os dados a serem transferidos.

A tabela de seletores é um vetor utilizado pelo gerenciador de processos para controlar a utilização das entradas livres da tabela de descritores globais (GDT). Todo seletor na GDT utilizado para criação dinâmica de processos possui uma entrada na tabela de seletores onde é armazenado o TSS do processo que a utiliza. A estrutura da tabela de seletores é mostrada na figura 6.2.

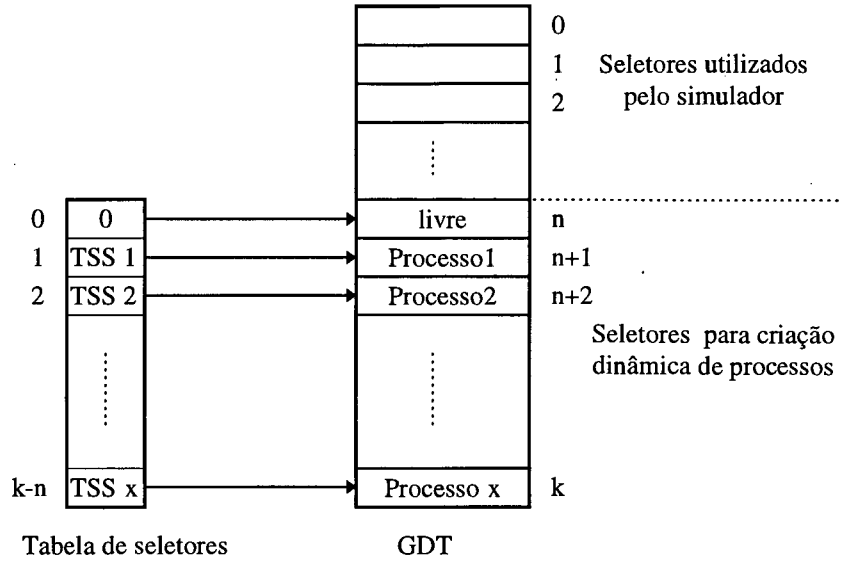

Figura 6.2: Estrutura da tabela de seletores.

Processos podem ser criados a partir de comandos de usuário, enviados pela estação de trabalho ao simulador, ou através de processos já em execução.

Processos para o simulador são programas completos, incluindo código, dados e pilha, mais o contexto de *hardware* (registradores do processador). Todos os processos tem área de memória exclusiva, dispondo de até 1 *Mbyte*.

Após a instalação do simulador e o início do processmento em modo protegido, o gerenciador de processos habilita um processo inicial, que faz parte do código do simulador, possuindo duas funções básicas:

1. iniciar o *boot* da máquina, semelhante ao *boot* do multicomputador Nó//;
2. executar um laço de espera quando nenhum processo estiver ativo.

Enquanto houver processos de usuário em execução, o processo inicial não é executado, não tomando fatias de tempo dos demais processos.

A interface de chamadas ao sistema é semelhante a da primeira versão do simulador baseado no XINU, sob o formato de uma biblioteca em linguagem C.

As primitivas do gerenciador de processos são:

Função CREATE: cria um novo processo.

parâmetros de entrada: nome do processo

parâmetros de saída: PID se criação com sucesso,

SYSERR.

A criação de um processo no novo simulador envolve as seguintes operações :

- alocação de um seletor disponível e cadastramento na tabela de seletores;
- cadastramento na tabela de descritores globais (GDT);
- cadastramento na tabela de processos;
- criação das tabelas de gerência de memória, e
- carga do processo.

Função KILL : elimina um processo.

parâmetros de entrada: PID

parâmetros de saída: OK

SYSERR

A eliminação de um processo no novo simulador envolve as seguintes operações:

- liberação na tabela de processos, e
- liberação na tabela de seletores.

Função ALTERA_STATUS: altera o estado de um processo.

parâmetros de entrada: PID e novo estado do processo

parâmetros de saída: OK

SYSERR

Função MY_PID : retorna a identificação (PID) de um processo.

parâmetros de entrada: nenhum.

parâmetros de saída: PID do processo que efetuou a chamada.

6.2.2 - O boot do novo simulador

O *boot* do multicomputador Nó//, descrito em [MON95], é composto de várias etapas. O código em ROM de cada nó é responsável apenas pelo recebimento do sistema operacional. Dois nós especiais possuem código diferenciado em ROM: o nó de controle e o nó de comunicação externa (NCE).

O código em ROM do nó de controle executa funções de inicializar o comutador de conexões e o barramento de serviços. Após a inicialização dos dois dispositivos, ele permanece a espera de requisições de serviços de alocação e liberação de nós de trabalho e conexões entre nós de trabalho via comutador de conexões.

O NCE é o responsável pelo estabelecimento de comunicação com a estação externa para o recebimento do sistema operacional.

Na primeira etapa do *boot*, o NCE aguarda da linha de comunicação um programa carregador e, após o seu recebimento, executa-o.

Na segunda etapa, o processo carregador recebe o descritor e o servidor CRUX, que é responsável pelo envio de requisições de processos ao servidor de arquivos na estação externa, carregando-os também no NCE.

Na terceira etapa o servidor CRUX requisita ao nó de controle um nó de trabalho livre, estabelecendo uma conexão entre o nó alocado e o NCE. O servidor CRUX passa a aguardar o programa carregador do processo INIT, o recebe e o reenvia ao nó alocado.

Na quarta etapa o servidor CRUX recebe o processo INIT e seu descritor e os envia para o processo carregador do INIT no nó alocado, via comutador de conexões.

Na última etapa o processo INIT carrega o processo SHELL (interpretador de comandos), que é sobreposto ao próprio processo INIT. O *boot* do sistema operacional do multicomputador Nó// é mostrado na figura 6.3.

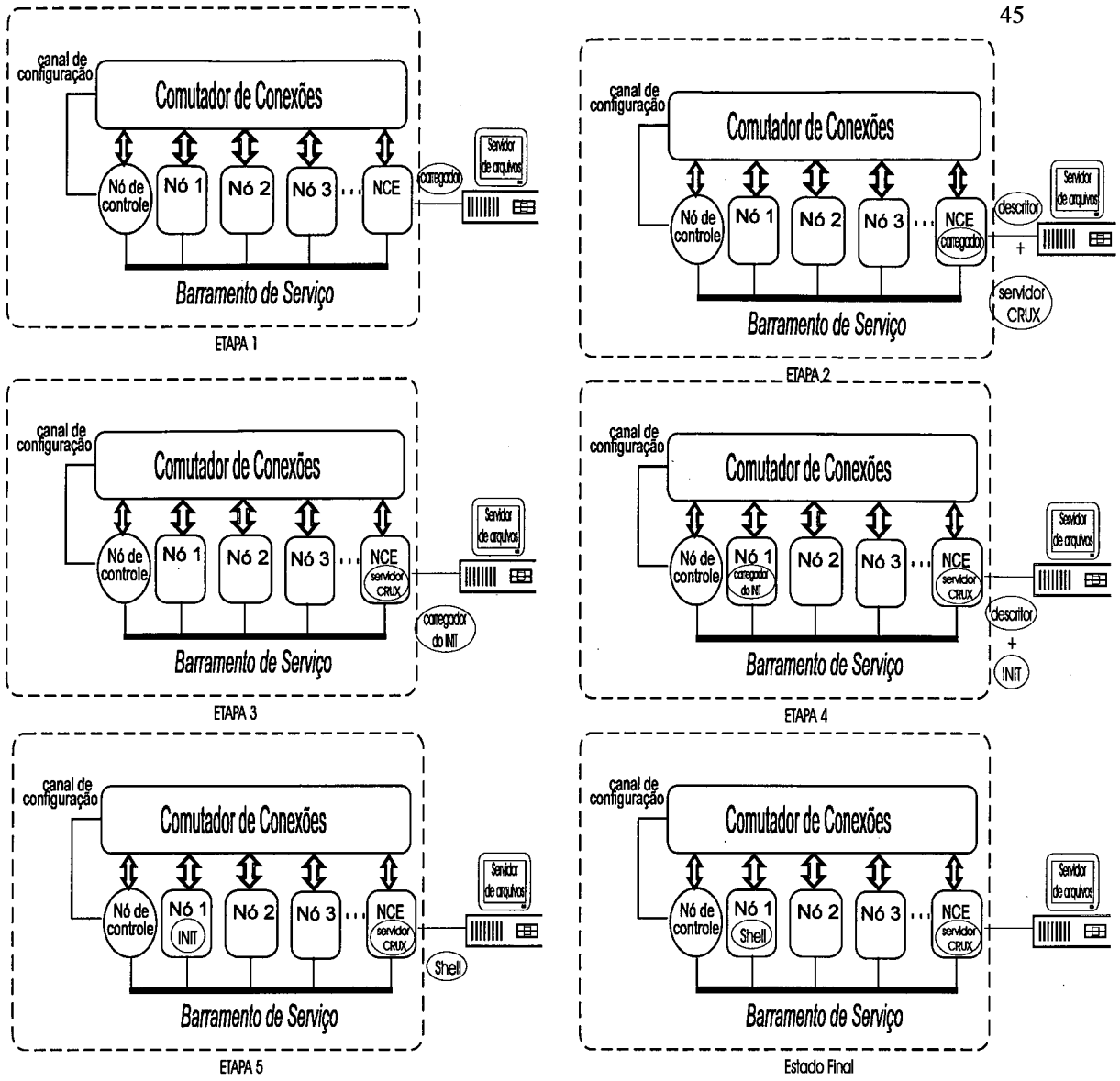

Figura 6.3: *Boot* do multicomputador Nó//.

No novo simulador o início do *boot* será executado pelo processo inicial, que é executado sempre após a criação das estruturas do simulador. O processo inicial é o responsável pela primeira etapa, que é o recebimento do carregador do servidor CRUX e a sua execução. Para isto, o processo inicial executa as seguintes tarefas:

1. requisita ao simulador a criação de um novo processo, o equivalente ao NCE;
2. o processo inicial recebe o carregador do servidor CRUX;
3. é feita uma requisição ao simulador para conexão com o novo processo para transferência de dados (equivalente a uma conexão no comutador de conexões);
4. envia o carregador ao novo processo;
5. executa uma chamada ao sistema, ativando o processo com o carregador.

O *boot* do simulador é mostrado na figura 6.4.

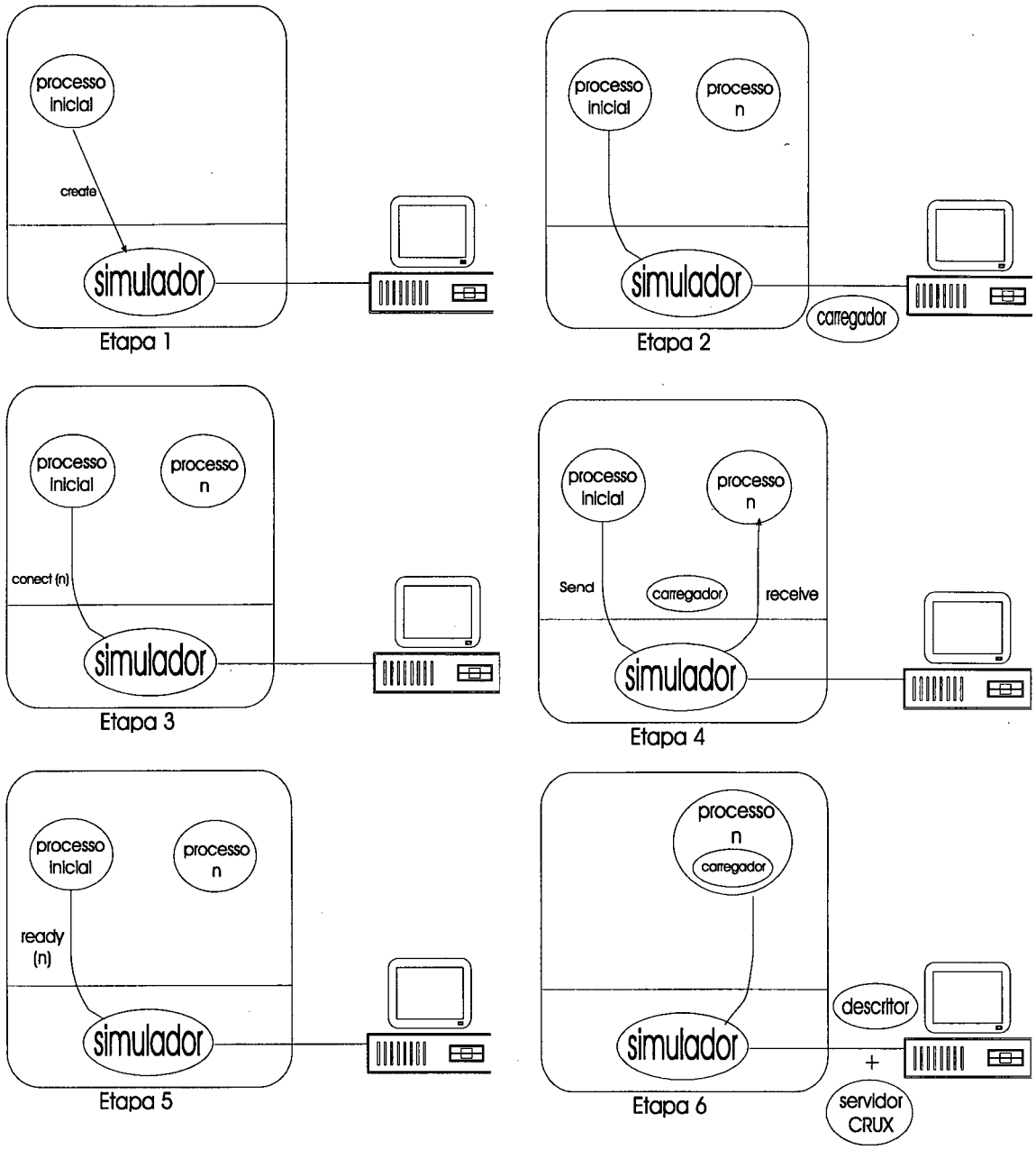

Figura 6.4: *Boot* no novo simulador.

Com a chamada de sistema ao simulador para ativar o novo processo, o escalonador de processos está apto a executá-lo. A partir deste ponto, o *boot* do simulador segue as mesmas etapas do Nó// e o processo inicial passa a ser inativo. As funções do nó de controle são implementadas como chamadas de sistema ao simulador.

6.2.3 - O gerenciador de memória.

No simulador do Nó//, a função do gerenciador de memória é, na etapa de inicialização, a de alocar memória em partições fixas para representar a memória física de cada nó de trabalho (figura 1.2), onde serão colocados código e dados dos processos. Essas partições, uma vez alocadas, não sofrem mais alterações de tamanho. No novo simulador cada nó possui uma porção fixa de 1 *Mbyte* de memória, iniciando no endereço 1 *Mbyte*.

Os primeiros 640 *Kbytes* de memória são reservado ao DOS (vetor de interrupções, núcleo do sistema e interpretador de comandos) e ao simulador e suas estruturas de dados para executar em modo protegido. A memória livre até o limite de 640 *kbytes* servirá para futuros aperfeiçoamentos no simulador, como implementação de comunicação entre processos através de caixas postais, servidor de arquivos, etc. A área entre 640 *Kbytes* e 1 *Mbyte* já era reservada pelo sistema operacional DOS para recursos como memória de vídeo em modo texto e *Shadow* RAM e não é utilizada pelo novo simulador.

Todos os processos terão disponíveis toda a área de memória de 1 *Mbyte* do nó de trabalho. O maior processo a ser executado no simulador poderá ter aproximadamente 870 *Kbytes*. O espaço de memória de 1 *Mbyte* de cada nó no simulador é dividido entre a área das estruturas de dados do simulador, a área de memória DOS (o simulador permite acessos a chamadas DOS através de interrupções de *software*) e a área disponível para código, dados e pilha do processo.

O novo simulador protege os processos entre si isolando a área de memória que eles utilizam, ou seja, não existe possibilidades de um processo alterar a área de memória de um outro processo. Processos devem se comunicar exclusivamente através das primitivas de troca de mensagens implementadas no novo simulador, executadas através de chamadas de sistema (interrupções de *software*).

Como parte do novo simulador, o gerente de memória é executado no modo protegido i486, enquanto que os processos de usuário são executados no modo virtual.

Foi escolhido para o novo simulador a gerência de memória através do método de segmentação e paginação combinados. A segmentação é obrigatória a todos os sistemas que sejam executados em modo protegido i486 e é implementada através da tabela de descritores globais (GDT).

Nos descritores da GDT cada processo é registrado com informações sobre sua área de memória, endereço base da tabela de paginação, modo de execução, nível de privilégio do descritor (DPL), endereço do seu segmento de estado de processo (TSS), etc. Quando um processo é criado, o gerente de processos registra os dados de cada processo em entradas livres da GDT e cria as tabelas de paginação (tabela de diretório de páginas e tabela de páginas) para aquele processo.

Todos os processos são executados em modo virtual, que é semelhante ao modo real dos processadores i486. No modo real, que é utilizado pelo sistema operacional DOS, são utilizados endereços de 20 bits, o que permite endereçar apenas o primeiro *Mbyte* de memória. Para que vários processos em modo virtual possam ser executados simultaneamente em endereços acima de 1 *Mbyte* de memória, foi necessário utilizar também a paginação.

Cada processo terá sua estrutura de paginação, que associa a visão lógica de endereços de 1 *Mbyte* utilizados pelos processos em endereços físicos no *Mbyte* correspondente a ele alocado, sendo esta conversão de endereços transparente ao processo (figura 6.5).

6.2.4 - A comunicação entre processos.

Nó multicomputador Nó//, a comunicação entre processos é feita exclusivamente através das chamadas de sistema.

Processos utilizam chamadas da camada de comunicações de alto nível para poder ter acesso aos recursos de comunicação. O simulador traduz estas chamadas básicas em chamadas da camada de comunicações de baixo nível, que são as chamadas que fornecem acesso aos dois dispositivos de comunicação: o barramento de serviços e o comutador de conexões.

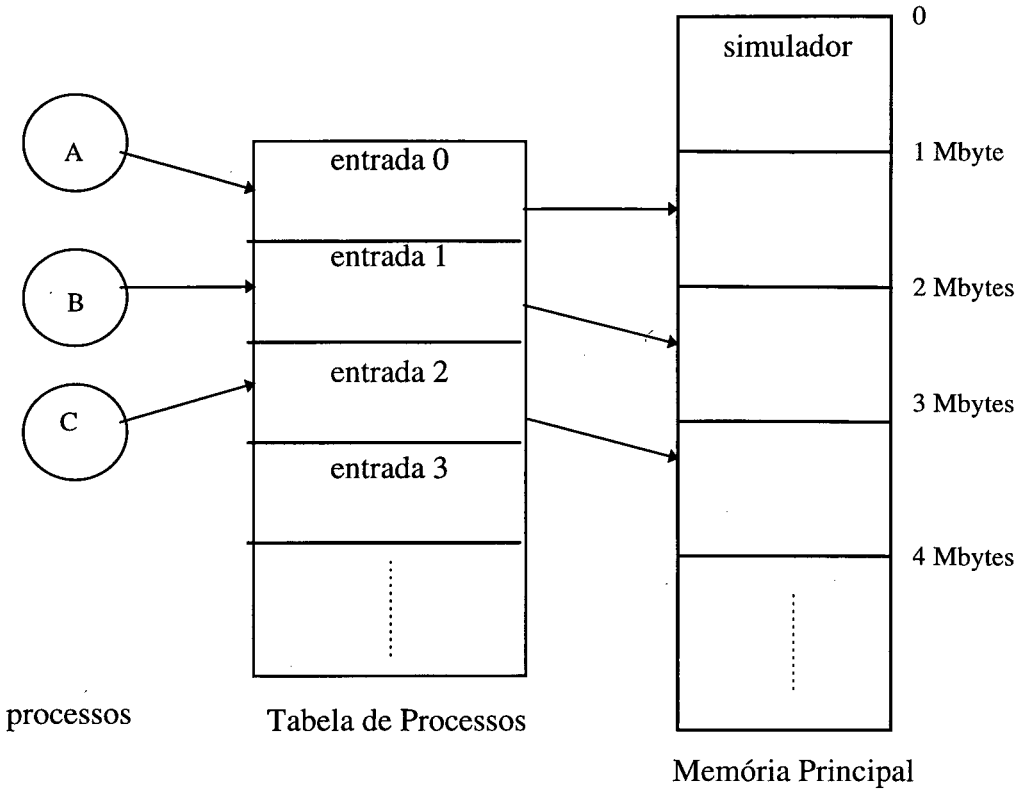

Figura 6.5: Distribuição da memória entre processos no novo simulador.

Todas as chamadas da camada de comunicações de alto nível implementadas na primeira versão do simulador permanecem inalteradas no novo simulador. Apenas as chamadas da camada de comunicações de baixo nível foram alteradas, para acessar os recursos deste, agora executado em modo protegido dos processadores i486. Estas alterações, apenas na camada de comunicações de baixo nível, permitem reutilizar todo *software* antes desenvolvido, sem alterações.

O simulador é quem executa as chamadas de camada de comunicação de baixo nível processadas pelo nó de controle no multicomputador Nó//. A simulação das conexões através dos dois dispositivos de comunicação é gerenciada através da tabela de processos. Nela é identificado o estado de conexão de um nó e a origem ou destino da comunicação que estiver sendo efetuada. A troca de mensagens é implementada com a transferência de dados *byte* a *byte* entre as áreas na memória principal correspondentes a cada nó.

Um processo deve, em primeiro lugar, enviar uma requisição de conexão com outro nó ao nó de controle. A função do nó de controle é gerenciar todas as conexões via comutador de conexões. Ele recebe uma mensagem com o pedido de conexão

entre dois nós de trabalho, envia uma mensagem ao nó destino sobre o pedido de conexão e a mensagem de resposta ao nó origem, todas via barramento de serviços. Caso a conexão seja possível, executa os comandos necessários, via canal de configuração do comutador de conexões, para habilitar o canal de comunicação entre nó origem e nó destino.

Após a conexão os dois nós podem efetuar a troca de mensagens pelo comutador de conexões, que é um canal bidirecional. Com o final da comunicação, os dois nós devem fazer o pedido de desconexão ao nó de controle.

Os procedimentos de comunicação entre dois nós é mostrado na figura 6.6.

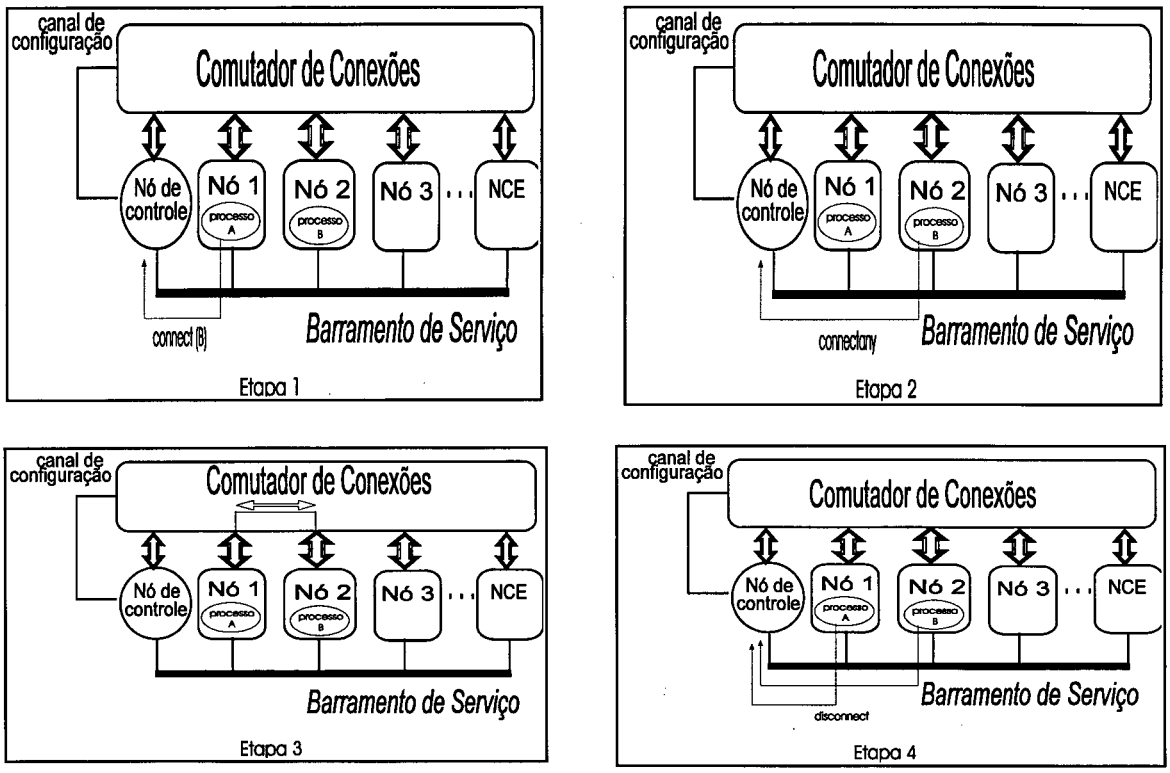

Figura 6.6: Comunicação entre dois nós.

As primitivas de comunicação da camada de comunicações de baixo nível disponíveis são:

Comutador de Conexões:

Função ***CB_Conecta***: conecta dois nós de trabalho via comutador de conexões.

parâmetros de entrada: identificação do nó 1 (NID)
identificação do nó 2 (NID)
parâmetros de saída: OK se conexão feita , ou
SYSERR se problemas na conexão.

Função ***CB_Desconcta***: desfaz uma conexão via comutador de conexões.

parâmetros de entrada: identificação do nó (NID)
parâmetros de saída: Ok.

Função ***CB_SendChar***: envia um caracter via comutador de conexões.

parâmetros de entrada: mensagem (byte).
parâmetros de saída: Ok.

Função ***CB_RecvChar***: recebe um caracter via comutador de conexões.

parâmetros de entrada: mensagem (byte).
parâmetros de saída: Ok.

Função ***CB_Send*** : envia uma mensagem via comutador de conexões.

parâmetros de entrada: mensagem e comprimento.
parâmetros de saída: Ok.

Função ***CB_Recv***: recebe uma mensagem via comutador de conexões.

parâmetros de entrada: mensagem e comprimento.
parâmetros de saída: tamanho da mensagem , se transferência correta,
ou 0 (zero) se transferência incorreta.

Barramento de Serviços:

Função ***BC_Send***: primitiva básica para enviar uma mensagem pelo barramento de serviços.

parâmetros de entrada: identificação do nó (NID) e mensagem (byte).
parâmetros de saída: Ok.

Função ***M_Send***: envia uma mensagem pelo barramento de serviços (nó de controle).

parâmetros de entrada: identificação do nó (NID) e mensagem (byte).
parâmetros de saída: Ok.

Função ***M_RecvAny***: recebe uma mensagem de qualquer nó de trabalho pelo barramento de serviços (nó de controle).

parâmetros de entrada: identificação do nó (NID) e mensagem (byte).
parâmetros de saída: Ok.

Função ***S_SendRec***: envia uma mensagem pelo barramento de serviços e aguarda o recebimento da resposta (nó de trabalho).

parâmetros de entrada: mensagem (byte).
parâmetros de saída: resposta (byte).

Função ***S_Send***: envia uma mensagem pelo barramento de serviços (nó de trabalho).

parâmetros de entrada: mensagem (byte)
parâmetros de saída: Ok.

Função ***S_Recv***: recebe uma mensagem pelo barramento de serviços (nó de trabalho).

parâmetros de entrada: nenhuma
parâmetros de saída: resposta (byte).

6.2.5 - O controle de entrada e saída

Todas as operações que envolvem entrada e saída nos processos deverão ser feitas exclusivamente através das chamadas de sistema (interrupções de *software*) ao simulador. Caso algum processo execute um operação de entrada e saída diretamente com as portas, ocorrerá um erro e o processo será encerrado.

O bloqueio do acesso às operações de entrada e saída é implementado através do segmento de estado do processo (TSS), via mapa de permissões de entrada e saída.

O mapa de permissões de entrada e saída é um *bitmap* que permite especificar individualmente quais portas de entrada e saída um processo executado em modo virtual dos processadores i486 pode acessar. No simulador a única possibilidade de um processo executar uma operação de entrada e saída é através de uma chamada de sistema.

Foram incorporadas duas novas chamadas de sistema à interface em linguagem C, para fornecer suporte a operações de entrada e saída de processos.

- função ***new_inp***: leitura de uma porta.
  parâmetros de entrada: endereço da porta de entrada e saída a ser lida;
  parâmetros de saída: valor lido da porta de entrada e saída.

- função ***new_outp*** : escrita em uma porta.
  parâmetros de entrada: endereço da porta de I/O;
  valor a ser enviado a porta de entrada e saída.
  parâmetros de saída: nenhum.

6.2.6 - O controle de interrupções

O novo simulador é responsável pelo atendimento de todas as interrupções de *hardware* e *software*, como forma de garantir segurança ao sistema.

As interrupções de *hardware* são ignoradas, com excessão da interrupção de relógio, que é utilizada pelo escalonador de processos. Foram mantidos os tempos da

interrupção de relógio do sistema operacional DOS, que é de aproximadamente 52 interrupções por segundo.

As interrupções de *software* são repassadas para os antigos tratadores do sistema operacional DOS, com exceção da interrupção 60h, que é a interrupção base das chamadas de sistema ao simulador. Interrupções de erros provocam o encerramento do processo. Embora seja possível acessar o DOS, não é aconselhável utilizá-lo, pois ele não estará disponível no Nó//, tornando o *software* incompatível.

O simulador gerencia interrupções, em modo protegido, através da montagem da tabela descritora de interrupções (IDT), utilizada pelos processadores i486. Através da IDT, o processador envia todas as interrupções ao simulador, que pode definir o tratamento adequado a elas.

Sempre que ocorrer uma interrupção, o fluxo de execução é desviado para o simulador, mas o contexto de *software* permanece o do processo. Este recurso permite ao simulador analisar o erro ocorrido, a pilha do processo, passagem de parâmetros através de registradores, etc.

A proteção nos atendimentos de interrupções é a mesma adotada a processos: o processador não permite a uma interrupção transferir a execução para um processo em segmento menos privilegiado. Caso isto ocorra, acontece um erro de proteção geral no sistema.

O nível de privilégio de um processo é dado pelo nível de privilégio do descritor (DPL) do processo na GDT. No simulador, processos de usuário tem nível 3 (menos privilegiado) e o simulador nível 0 (mais privilegiado). Níveis de privilégios intermediários não são utilizados no simulador (figura 6.7).

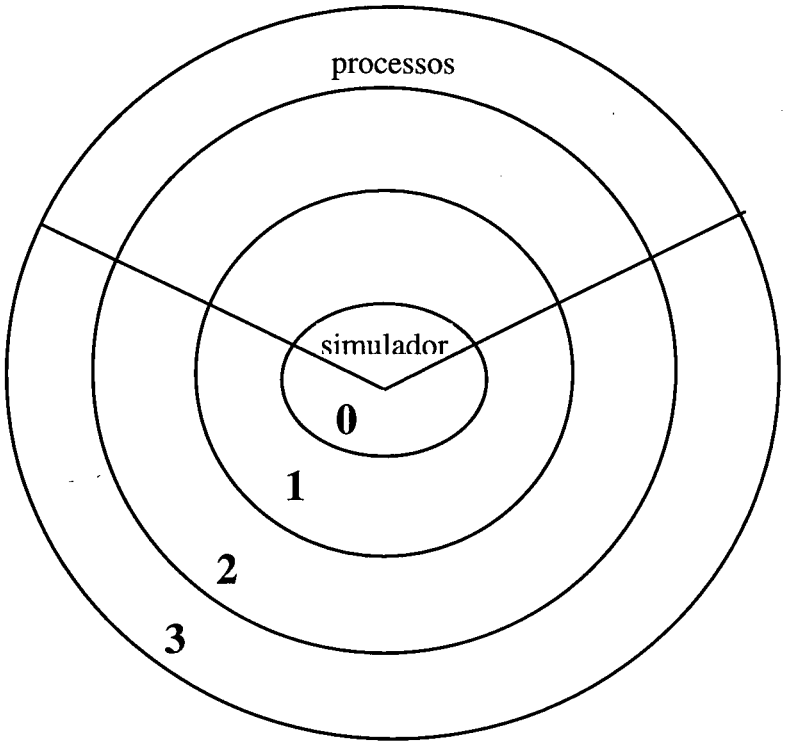

Figura 6.7: Níveis de privilégio no novo simulador.

No simulador interrupções só ocorrem a nível de processos. O código do simulador é executado sempre com interrupções desabilitadas, garantindo exclusividade para processamentos críticos, relacionados com processos e interrupções.

7 - O sistema operacional do multicomputador Nó//.

Neste capítulo estão descritas as alterações necessárias para transformar o novo simulador no sistema operacional do multicomputador Nó//.

7.1 - A inicialização do multicomputador.

As placas-mãe (*motherboards*) utilizadas no multicomputador Nó// possuem componentes de *hardware* que necessitam ser inicializados. O código em ROM que cada nó de trabalho possui é responsável apenas pelo recebimento do código do sistema operacional..

Os principais componentes que necessitam ser inicializados são:

- a controladora de interrupções, que gerencia interrupções mascaráveis ;
- o gerador de tempo (*timer*), utilizado para gerar a interrupção de relógio;
- o DMA.
- os dispositivos de comunicação (barramento de serviço e o comutador de conexões).

No simulador não é necessário inicializar estes componentes, pois o próprio sistema operacional DOS o faz.

7.2 - O boot.

O *boot* do simulador difere do sistema operacional do multicomputador Nó//, o CRUX, na sua instalação. O simulador é carregado como um programa DOS, através de um comando via linha de comandos e o sistema operacional do Nó// é carregado pelo código em ROM dos nós de trabalho.

As diferenças na instalação e as alterações necessárias são:

1. Sempre que o simulador é iniciado ele executa um teste sobre o tipo de processador. O simulador utiliza recursos avançados da arquitetura dos

processadores 80386 e só pode ser executado em computadores com estes processadores ou mais recentes, como os processadores 80486 e Pentium. A tentativa de executar o simulador em um computador fora destas especificações causará um erro e o simulador não será carregado, voltando o comando ao sistema operacional DOS.

No sistema operacional CRUX este teste pode ser retirado, pois o equipamento será construído dentro das especificações do projeto.

2. O simulador sempre altera o tratador de interrupções 15 hexadecimal (21 decimal) do sistema operacional DOS. Esta interrupção permite a processos executados em modo real (o modo virtual que os processos do simulador são executados é compatível como modo real) transferir dados entre memória convencional (abaixo de 1 *Mbyte*) e a memória estendida (acima de 1 *Mbyte*).

   Esta chamada caso utilizada, permitiria a um processo alterar áreas de dados em qualquer posição da memória, burlando o gerenciador de memória do simulador e quebrando a sua segurança.

   No sistema operacional CRUX esta alteração do tratador de interrupções pode ser eliminada, pois ela refere-se apenas ao sistema operacional DOS e não estará presente no CRUX. Esta interrupção poderá ser utilizada para implementar uma outra chamada qualquer ao sistema operacional.

3. As interrupções de *software* que acontecem em modo virtual, com exceção da interrupção 60 hexadecimal (96 decimal), são repassadas aos tratadores DOS pelo simulador. Apesar de não ser aconselhável utilizá-las pois não estarão disponíveis no CRUX, elas podem ser utilizadas, por exemplo, para depuração.

7.3 - O gerenciador de processos.

A estrutura atual do gerenciador de processos que permite a execução de vários processos poderá ser alterada para executar várias *threads*. Atualmente, cada processo possui uma área de memória privativa e deverá ser alterada para a execução das *threads*. As alterações devem alterar o mapeamento de memória, para que possa ser possível compartilhar código, dados, arquivos, etc.

As funções do processo inicial do simulador também deverão ser alteradas. No multicomputador Nó// a responsabilidade de executar o *boot* é do código em ROM presente em cada nó de trabalho. O processo inicial no multicomputador Nó// deverá executar apenas uma rotina de espera quando nenhum processo estiver sendo executado.

7.4 - O gerenciador de memória.

A gerência de memória do simulador, que é um ambiente multiprogramado, é baseada na proteção pelas tabelas de segmentação e paginação do processador i486, e nas características do modo virtual de endereçar apenas 1 *Mbyte* de memória, sendo possível existir apenas um processo por nó de trabalho.

No multicomputador Nó//, que é um ambiente multiprocessado, não é necessário a divisão de 1 *Mbyte* de memória para cada processo feita no simulador. O sistema operacional CRUX deverá gerenciar apenas um processo, dividindo a memória principal de cada nó de trabalho entre a área do seu próprio código e a do processo.

A área de memória do primeiro *Mbyte*, que no simulador é reservada, deverá ser liberada para a utilização do processo, ficando reservada apenas a área de memória do micronúcleo.

A proteção de memória poderá ser simplificada, utilizando-se apenas a proteção através de segmentação. A paginação, que é utilizada no simulador para possibilitar a execução de vários processos em modo virtual em endereços de memória acima de 1 *Mbyte*, poderá ser removida. A proteção por segmentação é necessária para separar a área de memória do sistema operacional da área de memória do processo e suas possíveis *threads*.

7.5 - Comunicação entre processos.

A comunicação entre processos no simulador é implementada como uma troca de dados entre as áreas de memória de cada processo, sendo o estado de comunicação de cada processo controlado através da tabela de processos. Os dois dispositivos de comunicação, o barramento de serviços e o comutador de conexões, são simulados através de interrupções de *software*.

No multicomputador Nó// estão presentes os dispositivos de comunicação. As rotinas presentes no simulador que tratam da comunicação entre processos devem ser substituídas por novas rotinas que implementem a comunicação entre processos em nós de trabalho diferentes através do comutador de conexões e as mensagens de controle enviadas ao nó de controle pelo barramento de serviços.

8 - Conclusões

O novo simulador representa um grande avanço dentro do projeto Nó//. Eliminando-se as restrições de gerência de memória da primeira versão do simulador, é possível desenvolver *software* com características próximas do sistema operacional CRUX. Será possível, por exemplo, terminar a implementação do interpretador SuperPascal desenvolvido em [MER96], que possui limitações pelas restrições da quantidade de memória do antigo simulador.

A técnica de desenvolvimento do novo simulador, com características de um micronúcleo, servirá como base para o desenvolvimento do sistema operacional do multicomputador Nó//. As diferenças entre simulador e sistema operacional, listadas no capítulo 6, não representam grandes dificuldades de implementação, o que permitirá um rápido desenvolvimento quando o primeiro protótipo do Nó// estiver pronto.

O simulador desenvolvido pode começar a ser utilizado para execução de *software*, mas ainda possui restrições que devem ser corrigidas com uma bateria de testes maior, pois existe apenas um *software* básico ([MER96]) desenvolvido para o Nó//. Até o momento, todos os esforços estavam concentrados no desenvolvimento do simulador e sistema operacional.

Algumas pendências ou melhorias possíveis no simulador são:

- alterar o gerenciador de processos para permitir a execução de *threads* ;
- implementar no gerenciador de memória a alocação de memória por demanda;
- implementar as chamadas de sistema especificadas no padrão POSIX.1 [LEW91];
- retirar a necessidade de compilação de programas para o Nó// com o XINU, isolando a interface desenvolvida de seu código, o que permitirá a diminuição do tamanho final do código dos programas já compilados;

O desenvolvimento deste trabalho foi extenso e muito dificultado pela falta de bibliografia e pela qualidade destas. As dificuldades em trabalhar com linguagem

Assembly e com os recursos avançados dos processadores i486 representaram a maior dificuldade na implementação do novo simulador.

8.1 - Contribuições

A partir da conclusão deste trabalho será possível acelerar o processo de desenvolvimento de *software* real para o multicomputador Nó//, antes mesmo da construção do primeiro protótipo dessa máquina.

Existem atualmente vários projetos e dissertações em desenvolvimento que aguardam por este trabalho. A primeira versão do simulador permitiu a especificação de vários projetos, mas que não podem ser desenvolvidos completamente pelas suas restrições. O novo simulador resolve os problemas da primeira versão e permitirá a conclusão dos projetos em desenvolvimento e o início de vários outros.

O código do sistema operacional do multicomputador Nó//, o CRUX, poderá ser desenvolvido utilizando-se o próprio simulador. As alterações a serem executadas referem-se as diferenças entre as arquiteturas do ambiente multiprocessado do multicomputador e o ambiente multiprogramado do simulador. São necessários apenas simplificações na etapa de *boot* e a implementação do código que executa a comunicação via barramento de serviços e comutador de conexões, que não estão presentes no simulador.

8.2 - Perspectivas

Diversos recursos e melhorias podem ser implementados no novo simulador ou no sistema operacional, dentre eles:

- implementar a gerência de memória por demanda.

  Atualmente, cada processo possui 1 *Mbyte* de memória reservada, utilizando-a ou não. Esta característica exige que exista memória para cada nó de trabalho simulado e limita o número de processos em função da memória principal do computador onde é executado o simulador. Com a alteração de alocação de memória por demanda será possível executar um número não definido de processos, sendo a quantidade de memória principal disponível o limitador.

- alterar o gerenciador de processos para permitir a execução de *threads*.

  A estrutura atual permite a criação de vários processos, cada qual com sua área de memória individual. Para implementar *threads* deve-se permitir o compartilhamento de áreas de memória entre elas. É possível compartilhar área de memória através da estrutura atual, utilizando os descritores da GDT e o segmento de estado de processo (TSS). Porém, para implementar *threads* este método não é o mais adequado. Pode-se criar uma estrutura para armazenar o contexto (registradores) de cada *thread* sem a necessidade de criar um novo processo. Criar um novo processo significa criar um novo TSS, novas tabelas de paginação e entradas na GDT.

  Threads necessitam apenas o compartilhamento de tempo do processador e criar todos estes recursos pode representar um consumo de memória e tempo de processamento do sistema (*overhead*) desnecessários. Pode-se criar uma tabela para cada processo com várias entradas, onde fique armazenado o contexto de cada *thread*, e a cada interrupção de relógio carregar o contexto de cada uma delas.

- implementar a comunicação entre processos através de caixas postais.

Existem áreas de memória no simulador que podem ser utilizadas para armazenar as mensagens enviadas por outros processos, funcionando como caixas postais. Estas área são a região de memória disponível abaixo de 1 *Mbyte* e a área de memória inicial de cada nó. Parte desta área é utilizada pelo gerenciador de memória para gerar as tabela de paginação de cada processo e o restante permanece disponível ao simulador. É necessário o desenvolvimento de primitivas de comunicação não bloqueantes para implementar caixas postais.

- implementar o servidor de arquivos localmente.

Atualmente são necessários dois computadores para executar o simulador. A estação externa executa as funções de enviar o sistema operacional na etapa de *boot*, de servidor de arquivos e comunicação com o usuário. O segundo computador executa o simulador e utiliza apenas processador e memória. Como o computador que executa o simulador é um computador completo, com disco, vídeo e teclado, pode-se alterar o simulador para rodar localmente as funções da estação externa, eliminado-se a necessidade de dois computadores para executar o simulador.

- executar uma avaliação de desempenho no código do simulador.

É ideal a execução de uma avaliação de desempenho no simulador, permitindo assim sua otimização ao utilizá-lo como base para o desenvolvimento do sistema operacional CRUX. Como uma das prerrogativas de um micronúcleo, a velocidade de execução é fundamental para o desempenho do sistema operacional.

Apêndice A - O simulador

Este apêndice descreve as funções do novo simulador, como utilizá-lo e possíveis alterações.

O novo simulador foi implementado totalmente em linguagem *Assembly*, sendo constituído pelos módulos descritos abaixo. O compilador utilizado foi o Microsoft MASM v6.1. Após a compilação é gerado o arquivo executável KERNEL.EXE.

Arquivo *KERNEL*.ASM

Arquivo base para compilação. Faz a inclusão dos demais arquivos.

Arquivo INSTALL.INC

Programa de início de execução, responsável pela criação e inicialização das estruturas de dados do simulador.

Seus funções principais são:

- testar se o tipo de processador é um 80386 ou posterior;
- alterar a interrupção INT 15h do DOS;
- ajustar os endereços na tabela de descritores globais (GDT) para o modo protegido;
- inicializar as tabelas de paginação do simulador e do processo inicial;

Arquivo GOPROT.INC

Programa que inicia o processamento em modo protegido, tendo como principais funções:

- carregar os registradores da CPU (LGDT, LIDT e CR3) para execução em modo protegido;
- habilitar a paginação.

Arquivo MONITOR.INC

Arquivo com código responsável pelo recebimento de todas as interrupções, de ocorridas durante o processamento do simulador. Recebe as chamadas ao sistema (interrupção 60 hexadecimal), desviando a execução para as rotinas de atendimento apropriadas.

Arquivo DATA.INC

Declaração e inicialização das variáveis do simulador. As principais variáveis são mostradas na tabela A.1.

VARIÁVEL	DESCRIÇÃO
TOTAL_MEMÓRIA	indica o total de memória disponível no computador.
NRO_MAX_PROCESSOS	indica o número máximo de processos que o simulador deverá gerenciar.
NRO_SEL_USUÁRIO	indica o número de seletores disponíveis na tabela de descritores globais (GDT) e, consequentemente, o tamanho da tabela de seletores.
TAM_KERNEL	tamanho do simulador, em páginas de 4 *kbytes*. Serve para reservar a área de memória do *kernel* nas tabelas de paginação dos processos.
GDT	tabela de descritores globais
IDT	tabela de rotinas de atendimento de interrupções. Seleciona a interrupção de relógio 0F8h direto para o escalonador e as demais para o simulador.
TAB_PROCESSOS	tabela de processos do *kernel*.
TAB_SELETORES	tabela de seletores utilizada para controlar o uso da GDT.
PAGINAÇÃO	segmento onde são criadas as tabelas de paginação do simulador e processo inicial.
TSS_PROCESSOS	segmento onde são criados os TSS dos processos criados dinamicamente pelo simulador.

Tabela A.1: Principais informações do arquivo Data.inc.

Arquivo TRATINT.INC

Tratadores de interrupção do simulador em modo protegido.

Arquivo RESCODE.INC

Tratamento de erros ocorridos durante a execução do simulador.

Arquivo TAREFA.INC

Código do processo inicial do simulador.

Arquivo ESCALONA.INC

Código do escalonador de processos do simulador.

Arquivo IO.INC

Código das rotinas de entrada e saída do simulador.

Arquivo CREATE.INC

Código da rotina de criação de processos do simulador.

Arquivo KILL.INC

Código da rotina de eliminação de processos do simulador.

Arquivo MYPID.INC

Código da rotina MY_PID, que retorna a identificação de um processo no simulador.

Arquivo BS.INC

Código das rotinas de comunicação pelo barramento de serviços.

Arquivo CB.INC

Código das rotinas de comunicação pelo comutador de conexões.

O programa KERNEL.EXE deve ser executado no sistema operacional DOS, não funcionando em sistemas operacionais que façam uso dos recursos do modo

protegido dos processadores i486, como o *Windows*. Os arquivos de *boot* do sistema operacional DOS (CONFIG.SYS e AUTOEXEC.BAT) devem ser alterados para carregar o menor número possível de programas residentes, liberando a memória para o novo simulador.

Uma configuração ideal para a execução do simulador é o arquivo CONFIG.SYS conter apenas os comandos FILES e BUFFERS e o arquivo AUTOEXEC.BAT não carregar nenhum programa residente.

As chamadas ao sistema são executadas através da interrupção de *software* 60 hexadecimal. As chamadas disponíveis para linguagem *Assembly* e C estão listadas na tabela A.2. A transferência de parâmetros entre processos e simulador é executada através dos registradores.

NÚMERO	CHAMADA	DESCRIÇÃO
00	S_M_Send	envia uma mensagem a um nó de trabalho específico via barramento de serviços.
01	S_M_RecAny	aguarda a recepção de uma mensagem via barramento de serviços de um nó de trabalho qualquer.
02	S_M_SendRec	envia uma mensagem e recebe a resposta via barramento de serviços.
03	S_CB_Conecta	pedido de conexão a um nó de trabalho via comutador de conexões.
04	S_CB_Desconecta	
05	S_CB_SendChar	envia um caracter (byte) via comutador de conexões.
06	S_CB_RecvChar	recebe um caracter (byte) via comutador de conexões.
07	S_CB_Send	envia uma mensagem via comutador de conexões.
08	S_CB_Recv	recebe uma mensagem via comutador de conexões.
40	Le_Porta	Lê uma porta de
41	Escreve_Porta	escreve um valor em uma porta de I/O
50	Create	Cria um novo processo
51	Kill	Elimina um processo
52	My_PID	Informa o PID de um processo
53	Altera_Status	Altera o estado de um processo

Tabela A.2: Chamadas de sistema do simulador.

Um exemplo de chamada ao sistema em linguagem *Assembly* é mostrado na figura A.1.

```
mov ah,41h          ;AH= número da função chamada
mov dx,61h          ;DX= endereço da porta de I/O
mov cx,03h          ;CX= valor a ser enviado pela porta de I/O
int 60h             ;gera uma interrupção de software
```

Figura A.1: Chamada ao simulador em linguagem Assembly.

A figura A.2 mostra exemplos de chamadas ao sistema através de funções em linguagem C.

```
CB_Conecta (nid1,nid2)      ; conecta dois nós via comutador de conexões
CB_Send (msg)               ; envia uma mensagem ao processo destino
CB_Desconecta (nid1)        ; desfaz a conexão via comutador de conexões.
```

Figura A.2: Chamada ao simulador em linguagem C.

Para incluir novas funções ao simulador deve-se seguir os passos abaixo:

- incluir no arquivo DATA.INC o número da função criada;
- incluir no arquivo MONITOR.INC a identificação da nova chamada;
- incluir no arquivo MONITOR.INC o arquivo com o código da nova chamada;
- criar o arquivo com o código da nova chamada;
- criar a chamada em linguagem C para ser utilizada como uma função da biblioteca de chamadas de sistema;
- recompilar o simulador.

Para aumentar a capacidade de processos em execução do simulador deve-se seguir os seguintes passos:

- alterar a variável TOTAL_MEMORIA do arquivo DATA.INC para a quantidade de memória disponível, em *Mbytes* (o valor deve ser múltiplo de 4);

- alterar a variável NRO_MAX_PROCESSOS do arquivo DATA.INC para a quantidade máxima de processos a serem executados, em função da quantidade de memória do computador a ser utilizado;
- alterar a variável NRO_SEL_USUÁRIO do arquivo DATA.INC, que deve ser maior ou igual ao número máximo de processos que o simulador irá gerenciar;
- recompilar o simulador.

Os estados dos processos no simulador são mostrados na tabela A.3. A função **Altera_Status** é utilizada para alterar estado de processos existente na tabela de processos. As chamadas de sistema que alteram o estado de execução de um processo, como suspensão e reativação, devem utilizar a chamada básica de alteração de estado de processos

NÚMERO	ESTADO	DESCRIÇÃO
01	Corrente	processo atual em execução.
02	Livre	entrada livre na tabela.
03	Pronto	pronto para execução.
04	Receive	aguardando recebimento de mensagem.
05	Sleep	processo suspenso por tempo determinado.
06	Suspenso	processo suspenso.
07	Wait	processo aguardando por um sinal (Signal).
08	CB_Receive	processo aguardando mensagem via comutador de conexões.
09	CB_Send	processo esperando para envio de mensagem pelo comutador de conexões.
10	BC_Receive	processo aguardando mensagem pelo barramento de serviços.
11	BC_Send	processo esperando para envio de mensagem pelo barramento de serviços.

Tabela A.3: Estados de processos no simulador.

O escalonador do simulador mostra na tela um indicador de progressão que indica quais processos estão ativos. Quando nenhum processo de usuário estiver ativo, apenas o processo inicial permanecerá ativo, mostrando um indicador de progressão próprio no vídeo.

A fatia de tempo (*time_slice*) reservada a cada processo foi mantida com o intervalo padrão de tempo da interrupção de relógio do DOS, ou seja, aproximadamente 13 mseg. Caso desejado, é possível reprogramar a interrupção de relógio, tornando a fatia de tempo para cada processo menor, aumentando a freqüência de troca dos processos em execução.

Glossário

Área de memória superior (HMA) área de memória entre 1024 e 1088 *Kbytes*.

Bloco de memória estendida (EMB) área de memória acima de 1088 *Kbytes*.

Bloco da memória superior (UMB) área de memória entre 640 e 1024 *Kbytes*.

CPL nível de privilégio corrente. Quando os processadores i486 operam no modo protegido o seu nível de privilégio atual (CPL), utilizado pelo mecanismo de proteção, é dado pelo nível de privilégio do segmento de código em execução. O CPL é especificado pelo valor do DPL na tabela de descritores globais (GDT) do segmento de código do processo emexecução.

Descritor elemento base das tabelas GDT e LDT utilizados pelos processadores i486 no modo protegido para gerenciamento de memória com segmentação. Um descritor contém dados dos segmentos, como endereço base, limite, o nível de privilégio (DPL), o tipo de segmento, etc.

DOS sistema operacional desenvolvido pela empresa Microsoft para computadores padrão IBM-PC-XT-AT e compatíveis.

DPL nível de privilégio do descritor. Indica o nível de privilégio de um descritor das tabelas GDT ou LDT quando o processador está operando no modo protegido dos processadores i486.

DPMI Interface do DOS com o modo protegido. Extensor do DOS que define as estruturas necessárias para os processadores i486 processarem no modo protegido e fornece acesso a memória estendida.

Endereço físico é o valor que aparece no barramento de endereços e corresponde a uma locação física na memória principal.

Endereço linear transformação de um endereço lógico (seletor / deslocamento) em um sistema com segmentação para um valor único, traduzido pelo sistema de armazenamento virtual.

Endereço lógico endereço utilizado por um programa em execução. O endereço lógico deverá ser mapeado em um endereço físico.

Exceções são transferências forçadas da execução de um processo para um outro processo conhecido como controlador (*handler*) através da execução de uma instrução.

GDT tabela descritora global, utilizada no modo protegido dos processadores i486 para armazenar os dados de segmentos de acesso global.

Interrupções são transferências forçadas da execução de um processo para um outro processo conhecido como controlador (*handler*) , ocorrendo em tempos aleatórios como resposta normalmente a um evento de *hardware*.

LDT tabela descritora local, utilizada no modo protegido dos processadores i486 para armazenar dados de segmentos de processos.

LIM-EMS padronização de acesso a memória expandida proposta pelas empresas Lotus, Intel e Microsoft.

Memória convencional área de memória física localizada a partir do endereço 0 até o limite de 640 *Kbytes*, sendo também chamada de memória do usuário.

Memória estendida memória endereçável pelos processadores dos computadores AT (computadores com processadores 80286 em diante) acima de 1 *Mbyte*.

Memória expandida memória desenvolvida para aumentar a capacidade de armazenamento da memória principal dos processadores 8086 (modo real

nos processadores i486), sendo chaveada por bancos e disponíveis em placas de expansão ou simuladas em memória estendida nos computadores AT.

Modo protegido modo dos processadores i486 que opera com tabelas de descritores de segmentos, fornecendo diversos recursos como gerenciamento de memória avançado e proteção .

Modo real modo dos processadores i486 compatível com os processadores 8086.

Modo virtual V86 modo dos processadores i486 que simula o funcionamento dos processadores 8086 sob a supervisão do modo protegido.

Página física um bloco de memória de tamanho fixo correspondente a endereços na memória física.

Página lógica um bloco de memória de tamanho fixo, utilizado por programas, que será traduzido pelo sistema de armazenamento virtual em uma página física.

POSIX.1 IEEE *Portable Operating System Interface for Computing Enviroments* - documento que padroniza a interface de programação do sistema operacional UNIX.

Processadores i486 neste trabalho este termo será utilizado para designar os processadores 80486, desenvolvidos pela empresa Intel. São praticamente compatíveis a nível de conjunto de instruções com processadores 80386 e Pentium.

Processadores 8086 processador de 16 bits lançado em 1978 pela Intel, sendo o primeiro da família de processadores x86.

RAM memória de acesso aleatório.

ROM memória somente de leitura.

Threads utilização de diversos fluxos de execução independentes dentro de um mesmo processo ([TAN92]).

Traps é uma exceção reportada imediatamente após a execução da instrução que a gerou.

TSS (Task State Segment) O TSS é um segmento definido por um seletor, existindo apenas na tabela de descritores GDT. Todo processo a ser executado no modo protegido deve ter seu TSS, onde é guardado o seu contexto.

São armazenadas informações sobre:

- registradores de uso geral: EAX, EBX, ECX, EDX, ESP, EBP, ESI e EDI;
- registradores de segmento: CS, DS, SS, ES, FS e GS;
- registrador de *flags* (EFLAGS);
- apontador de instruções (EIP);
- o seletor de TSS do processo anterior (se um retorno é esperado);
- o seletor da LDT do processo;
- o endereço lógico das pilhas dos níveis de privilégio 0, 1 e 2;
- informações sobre depuração, e
- o endereço base do mapa de permissões de I/O.

Referências Bibliográficas

[ARI90] Ben-Ari, M., "Principles of Concurrent and Distributed Programming", Prentice-Hall, EUA, 1990.

[BEN95] Benchimol, Isaac B., "Ferramentas de desenvolvimento para microcomputadores dedicados baseados em processadores 386/486", dissertação de mestrado, CPGCC-UFSC, Florianópolis, 1995.

[BRI91] Bricker, A. & et al., "Architetural Issues in Microkernel-base Operating Systems: the CHORUS experience", Computers Comunicatios, vol. 14, nº 6, pág. 347-357, agosto de 1991.

[BUR95] Burgess, Richard A., "Developing Your Own 32-Bit Operating System", Sams Publishing, EUA, 1995.

[CAM95] Campos, Rodrigo R. A., "Um sistema operacional fundamentado no modelo cliente-servidor e um simulador multiprogramado de multicomputador", dissertação de mestrado, CPGCC-UFSC, Florianópolis, 1995.

[COM84] Comer, D., "*Operating System Design: The XINU Approach*", Prentice-HAll, EUA, 1984.

[COR93] Corso, Thadeu B., "Ambiente para programação paralela em multicomputador", Relatório Técnico UFSC-CTC-CEC Nº 1, Florianópolis, 1995.

[DPM90] "*DOS Protected Mode Interface* (DPMI) - *Protected Mode API for DOS" Extend Aplications*, versão 0.9, 1990.

[DUN91] Duncam, Ray, "Ampliando o DOS", Editora Ciência Moderna, Rio de Janeiro, 1991.

[HWA93] Hwang, Kai, "Advanced computer architeture: paralelism, scalability, programmability", McGraw-Hill, EUA, 1993.

[INT87] Intel, "80386 System Software Writer's Guide", Intel, EUA, 1987.

[INT90a] Intel, *"i486 Microprocessor Programmer's Manual"*, Intel, EUA, 1990.

[INT90b] Intel, *"i486 Microprocessor Hardware Reference Manual"*, Intel, EUA, 1990.

[INT91] Intel, *"Intel 486SX Microprocessor, Intel 487SX Math Coprocessor"*, *Data Book*, Intel, EUA, 1990.

[LEW91] Lewis, D., *"POSIX Programmer's Guide"*, O'Reilly & Associates, 1991.

[MER95] Merkle, Carla & Boing, Hamilcar, "Simulação do Nó//, Departamento de Informática e Estatística", UFSC, Relatório técnico a ser publicado, CPGCC- UFSC, Florianópolis, 1995.

[MER96] Merkle, Carla, "Ambiente para a execução de programas paralelos escritos na linguagem superpascal em um multicomputador com rede de interconexão dinâmica", dissertação de mestrado, CPGCC-UFSC, Florianópolis, 1996.

[MON95] Montez, Carlos B., "Um sistema operacional com micronúcleo distribuído e um simulador multiprogramado de multicomputador", dissertação de mestrado, CPGCC-UFSC, Florianópolis, 1995.

[POU94] Pountian D., "The Chorus Microkernel", Byte, pág. 131-136, janeiro de 1994.

[QUA92] Qualitas Inc, *"Qualitas C/C++ Libraries for DPMI"*, documentação do *software*, Eua, 1992.

[SEG92] Segal, Bernardo; Nakajune, Cesar Kaiyo & Celestino, Sílvio Alves, "Conhecendo a família 80486: hardware e software". Livros Érica Editorial Ltda. São Paulo, 1992.

[STE92] Stein, B. O., "Projeto do Núcleo de um Sistema Operacional Distribuído", dissertação de mestrado, CPGCC-UFRS, Porto Alegre, 1992.

[TAN87] Tanenbaum, A.S., *"Operating Systems: design and implementation"*, Prentice Hall, EUA, 1987.

[TAN92] Tanenbaum, A. S., *"Modern Operating System"*, Prentice Hall, EUA, 1992.

[YOU91] Young, Michael J., "MS-DOS Programação Avançada", Editora Ciência Moderna, Rio de Janeiro, 1991.