UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E DE ESTATÍSTICA
CURSO DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UM SISTEMA DE ARQUIVOS PARA O AMBIENTE MULTICOMPUTADOR NÓ //

por

Jorge Diego Callaú

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

Prof. Luis Fernando Friedrich, Dr.
Orientador

Florianópolis, fevereiro de 1997

UM SISTEMA DE ARQUIVOS PARA O AMBIENTE MULTICOMPUTADOR NÓ //

JORGE DIEGO CALLAU

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 SISTEMAS OPERACIONAIS E APROVADA EM SUA FORMA FINAL PELO PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO.

Prof. Luiz Fernando Friedrich, Dr.
Orientador

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

BANCA EXAMINADORA :

Prof. Luiz Fernando Friedrich, Dr. (Presidente)

Prof. Thadeu Botteri Corso, M.Sc .

Prof. Paulo José de Freitas Filho, Dr.

Oferecimento:

À minha esposa Christiane,
aos meus filhos Pablo e Andres,
aos meus pais Jorge e Rosário.

AGRADECIMENTOS

Este é o resultado de um trabalho que contou com a colaboração de várias pessoas. Agradeço a todas elas e em especial ao meu orientador Luis Fernando Friedrich pela paciência, sabedoria dedicadas na orientação.

À Universidade Federal de Santa Catarina e ao CNPq. Pela oportunidade e o apoio financieiro.

A todos os professores, colegas e funcionários do CTC que , de uma forma ou outra, contribuíram na minha formação, com seus ensinamentos, conselhos e amizade.

A todos os amigos que aqui fiz, levarei sempre no coração.

Aos meus irmãos Andres e Daniel (*in memorium*), pelo incentivo.

Ao Alberto e Claudete, Fernando e Claudineia ,Francisco e Lúcia e toda a familia, pela amizade carinho e apoio.

Aos meus sogros Gildes e Sonia, pelo amor e apoio.

À minha esposa Christiane e meus filhos Pablo e Andres, pela paciência e resignação.

A DEUS, obrigado.

SUMÁRIO

1. INTRODUÇÃO **10**

2. MÁQUINAS PARALELAS **13**

2.1 INTRODUÇÃO 13
2.2 MULTICOMPUTADORES 13
2.3 REDES DE INTERCONEXÃO 15
2.3.1 Redes de Interconexão Estáticas *19*
2.3.2 Redes de Interconexão Dinâmicas *21*

3. COMUNICAÇÃO EM SISTEMAS DISTRIBUÍDOS **24**

3.1 INTRODUÇÃO 24
3.2 COMUNICAÇÃO ENTRE PROCESSOS 24
3.2.1 Disciplinas de Comunicação *26*
3.3 AS CHAMADAS DE PROCEDIMENTO REMOTO 27

4. ARQUITETURA DO MULTICOMPUTADOR NÓ// **30**

4.1 INTRODUÇÃO 30
4.2 OS NÓS 30
4.3 O COMUTADOR DE CONEXÕES 31
4.4 O CONTROLADOR DE NÓS E CONEXÕES (CNC) 32
4.5 O BARRAMENTO DE SERVIÇOS 32
4.6 O NÓ DE COMUNICAÇÃO EXTERNA (NCE) 34

5. SIMULADOR DO MULTICOMPUTADOR NÓ // **35**

5.1 INTRODUÇÃO 35
5.2 DESCRIÇÃO DO SIMULADOR 35
5.3 IMPLEMENTAÇÃO DO SIMULADOR 37
5.4 ALTERAÇÕES NO XINU 39

6. O SISTEMA CRUX **42**

6.1 INTRODUÇÃO 42
6.2 PROCESSO CRUX 42
6.3 AS CAMADAS DO SISTEMA 44
6.3.1 Camada das chamadas do sistema *45*
6.3.2 Camada das comunicações de alto nível *45*

6.3.3 Camada das chamadas do núcleo *46*
6.3.4 Camada das comunicações de baixo nível *48*
6.4 O SERVIDOR CRUX 49

7. SISTEMA DE ARQUIVOS PYXIS 51

7.1 INTRODUÇÃO 51
7.2 IDENTIFICAÇÃO DAS ENTIDADES DO SISTEMA 52
7.3 IDENTIFICAÇÃO PERANTE O USUÁRIO 52
7.3.1 Nomes de nodos *53*
7.3.2 Nomes de arquivos *53*
7.4 IDENTIFICAÇÃO PERANTE O SISTEMA DE ARQUIVO 54
7.4.1 Identificadores de Usuários *54*
7.4.2 Identificadores de Caixas Postais *54*
7.4.3 Identificadores de Arquivos *55*
7.5 COMUNICAÇÃO E LOCALIZAÇÃO DOS SERVIDORES 55

8. UM SISTEMA DE ARQUIVOS PARA O NÓ // 59

8.1 INTRODUÇÃO 59
8.2 O SUPORTE BÁSICO DE COMUNICAÇÃO DO PYXIS 59
8.2.1 O Micronúcleo *60*
8.2.2 O Mecanismo de Caixa Postal Proposto *64*
8.2.3 O Servidor de Caixa Postal *65*
8.3 MIGRAÇÃO DO PYXIS PARA O NÓ// 69

9. CONCLUSÃO 74

10. REFERÊNCIAS BIBLIOGRÁFICAS 76

INDICE DE FIGURAS

FIGURA 2-1 : MULTICOMPUTADOR. 14
FIGURA 2-2 :SISTEMAS DE PROCESSAMENTO PARALELO 15
FIGURA 2-3 : GRELHA QUADRADA 3 X 3. 19
FIGURA 2-4 : HIPERCUBO DE DIMENSÃO 4. 20
FIGURA 2-5 : MULTICOMPUTADOR BASEADO EM BARRAMENTO. 22
FIGURA 2-6 : REDE ÔMEGA 2 X 2. 22
FIGURA 2-7 : CROSSBAR 4 X 4. 23
FIGURA 3-1: DISCIPLINA DE COMUNICAÇÃO PRODUTOR-CONSUMIDOR 26
FIGURA 3-2: MODELO CLIENTE-SERVIDOR 27
FIGURA 3-3: CHAMADA DE PROCEDIMENTO REMOTO 28
FIGURA 4-1: A ARQUITETURA DO NÓ //. 30
FIGURA 4-2: A ESTRUTURA INTERNA DE UM NÓ. 31
FIGURA 5-1: NÓ // CONECTADO A UMA ESTAÇÃO DE TRABALHO. 36
FIGURA 5-2: TRECHO DE CÓDIGO QUE CRIA PROCESSOS XINU. 37
FIGURA 5-3: TRECHO DE CÓDIGO QUE SIMULA UM NÓ 38
FIGURA 5-4: O XINU UTILIZADO COMO BASE PARA O SIMULADOR DO NÓ //. 39
FIGURA 6-1: UM PROCESSO COLOCADO NA MEMÓRIA DE UM NÓ. 42
FIGURA 6-2: AS CAMADAS DO SISTEMA. 44
FIGURA 6-3: ARQUITETURA DO CRUX. 45
FIGURA 6-4: UTILIZAÇÃO DE UM SERVIDOR DE ARQUIVOS EXTERNO AO NÓ //. 49
FIGURA 7-1: O PYXIS EM UM SISTEMA COMPUTACIONAL 52
FIGURA 7-2: DIAGRAMA DE SINTAXE DE UM NOME DE NODO 53
FIGURA 7-3: DIAGRAMA DE SINTAXE DE UM NOME DE ARQUIVO 53
FIGURA 7-4: IDENTIFICADOR DE USUÁRIO.. 54
FIGURA 7-5: IDENTIFICADOR DE CAIXA POSTAL 55
FIGURA 7-6: IDENTIFICADOR DE ARQUIVO 55
FIGURA 7-7: EXEMPLO DA TABELA DE LOCALIZAÇÃO DOS SERVIDORES 57
FIGURA 8-1: COMUNICAÇÃO COM CAIXA POSTAL 60
FIGURA 8-2 MICRONÚCLEO DO CRUX 61
FIGURA 8-3 : CONFIGURAÇÃO DE UM MICRONÚCLEO ANTES (A) E DEPOIS (B) DA INTRODUÇÃO DO SCP, E CAMADA DE CAIXAS POSTAIS. 65
FIGURA 8-4 : TABELA DE CAIXAS POSTAIS MANTIDAS PELO SCP 66
FIGURA 8-5: O PYXIS EM UM MULTICOMPUTADOR 70

RESUMO

A procura por sistemas de computação formados por vários processadores capazes de atingir maior poder de processamento tem levado cientistas e pesquisadores da área de Arquitetura de Computação e Sistemas operacionais, a propor e desenvolver diferentes modelos de arquitetura de alto desempenho. O Projeto Nó// (lê-se nó paralelo) do qual participam grupos de pesquisa da Universidade Federal de Santa Catarina e do Rio Grande do Sul, visa o desenvolvimento de um ambiente completo para programação paralela, incluindo a construção de um multicomputador com rede de interconexões dinâmica . A apresentação deste trabalho vem colaborar com a concepção desse multicomputador e a consolidação da idéia do ambiente multicomputador Nó // a partir da definição da organização e das necessidades de comunicação de um sistema de arquivo para ser executado no Nó // .

ABSTRACT

The search for computer systems composed by many processors that can reach to a powerful making up of a process has guide the scientists and reserarches in Computer Architeture and Operational System, to propose and develop different architeture models whit high performance. The Project Nó // (read parallel node) in which participate researches for Federal Universities of Santa Catarina and Rio Grande do Sul, ains to develop a complet enviromment for parallel programming, including the construction of a multicomputer whit dynamic interconnection network. The presentation of this work will collaborate in the conception of this multicomputer and in the consolidation of the idea about Nó // multicomputer environment , from the definition of the organization and comunication needs of the file system to be executed in Nó //.

CAPITULO 1

1. Introdução

Devido a constante queda nos preços dos componentes dos computadores, desde sua invenção até hoje, cientistas e pesquisadores têm procurado construir máquinas com capacidades de processamento cada vez maiores. A velocidade com a qual a tecnologia tem evoluído é tamanha que aquilo que hoje é considerado um padrão, amanhã tornar-se-á obsoleto.

Por outro lado, a construção de modelos utilizando arquiteturas de computadores que apresentam alta performance de processamento, como os computadores vetoriais, processadores matriciais e os multicomputadores, formados por vários processadores existentes no mercado, podem viabilizar um produto com capacidade de processamento muito maior do que a de um computador monoprocessador tradicional.

Entretanto a utilização dessas máquinas paralelas implicam na confecção de uma nova classe de sistemas operacionais (denominados Sistemas Operacionais Distribuídos), que se preocupam em tornar transparente, aos programas de aplicação, a arquitetura empregada.

Dentro do contexto da Universidade Federal de Santa Catarina (UFSC), há em andamento um projeto para construção de um ambiente de programação paralela, destinado ao desenvolvimento de aplicações em alto desempenho. Este projeto chamado Nó// (lê-se nó paralelo) conta ainda com a colaboração da Universidade Federal do Rio Grande do Sul (UFRGS), envolvendo pesquisadores das áreas de Arquitetura de Computadores, Sistemas Operacionais e Simulação.

Os objetivos principais do projeto Nó// são a construção de um Multicomputador com rede de interconexão dinâmica, e o desenvolvimento de um Sistema Operacional Distribuído.

Numa fase inicial do projeto foi concretizada a construção de um simulador para a máquina Nó//, servindo como plataforma para o projeto e implementações de experiências de software na arquitetura proposta. Algumas destas experiências incluem: Sistema Operacional, CRUX [CAM95] e [MON95] e a implementação de uma linguagem de programação, Super Pascal [MER 95].

Em paralelo o hardware tem sido desenvolvido. Espera-se, até meados de 1997, dispor de um protótipo com até 8 processadores.

Enquanto não se tem a máquina física o simulador tem servido como plataforma de desenvolvimento de software de forma que a migração para a máquina real seja feita sem grandes modificações.

O projeto apresentado neste trabalho vem colaborar com a concepção desse multicomputador através da determinação da viabilidade de um Sistema de Arquivos, baseado no modelo ***cliente/servidor*** , adaptado ao ambiente proposto. O trabalho está dividido em duas partes principais:

1) Definição da plataforma básica para a implementação da migração do Sistema de Arquivos PYXIS para o ambiente multicomputador Nó//.

2) O estudo e projeto de mecanismos de comunicação mais apropriados para o tipo de arquitetura considerado. Os conceitos abordados foram os de threads, multiplos fluxos de execução de tarefas, derivados do conceito de processos leves (light weight process) descrito em [TAN92] e o conceito de caixas postais , o qual foi escolhido para implementar o conceito básico de comunicação no multicomputador Nó// de modo a manter uma comunicação uniforme entre os nós processadores.

O trabalho está dividido em oito capítulos. O primeiro fornece uma visão global do contexto em que se situa a proposta aqui definida. Os capítulos 2 a 6 descrevem as partes envolvidas na definição do trabalho de dissertação , desde uma introdução a máquinas paralelas, e o problema da comunicação em arquiteturas distribuídas, até a descrição e estudo dos componentes do ambiente proposto:

Arquitetura do Nó //, o Simulador e o Sistema operacional.

No capítulo 7 é apresentado o Sistema de Arquivos PYXIS usado como base para o sistema proposto.

Por fim o capítulo 8 apresenta o sistema de arquivos para o ambiente Multicomputador Nó //.

CAPITULO 2

2. Máquinas Paralelas

2.1 Introdução

A construção de máquinas paralelas exige a interconexão de processadores, memórias e canais de comunicação. As máquinas paralelas de uso geral podem ser divididas em três grandes grupos em função do grau de integração dos seus componentes [AUS91]: multiprocessadores, multicomputadores e redes locais.

Esses grupos se distinguem pelo grau e velocidade das interações possíveis entre seus componentes. Os **multiprocessadores** (ou máquinas paralelas com memória compartilhada) são computadores individuais com processadores que acessão memórias compartilhadas através de redes de interconexão dinâmicas. Os **multicomputadores** (ou máquinas paralelas com memória distribuída) são compostos de nós autônomos, constituídos de processadores, memória privativa e canais de comunicação, ligados através de redes de interconexão formadas por canais de comunicação bipontuais exclusivos. As **redes locais** são compostas de computadores independentes completos interconectados através de canais de comunicação compartilhados.

2.2 Multicomputadores

A conjunção das seguintes características permite distinguir os multicomputadores das outras máquinas paralelas [REE87]:

- ***Grande número de elementos processadores homogêneos.*** O baixo custo dos componentes e a facilidade de montagem permitem a construção de máquinas com grande quantidade de nós processadores.
- ***Interação baseada em trocas de mensagens***. Os processos de um programa paralelo que executam em nós diferentes podem interagir somente através de trocas de mensagens.

- ***Alta velocidade das comunicações.*** Os canais dos multicomputadores constituem meios confiáveis de comunicação, apresentando velocidades de uma ordem de grandeza superior a das redes locais.
- ***Granularidade média de processamento***. Os multicomputadores encorajam a exploração do paralelismo real pela decomposição de programas em vários processos cooperantes. Entretanto, uma granularidade muito fina pode se tornar inadequada pelo predomínio das comunicações.
- ***Programação*** - O desenvolvimento de programas paralelos compostos de vários processos cooperantes deve contar com linguagens, compiladores e sistemas operacionais adequados para explorar automaticamente o paralelismo fornecido pelo multicomputador.
- ***Grande número de canais de comunicação bipontuais***. As redes de interconexão formam topologias regulares com muitos canais conectando pares de nós dessas máquinas.

As redes de interconexões são fator decisivo na determinação do desempenho geral dos multicomputadores pois, mesmo com altas velocidades de processamento dos nós, para a execução dos processos cooperantes, são necessárias trocas de mensagens pela rede de interconexão (Figura 2-1). Dessa forma, o desempenho do multicomputador depende de uma combinação adequada da velocidade de processamento dos nós e da velocidade com que é possível trocar mensagens nas redes de interconexões.

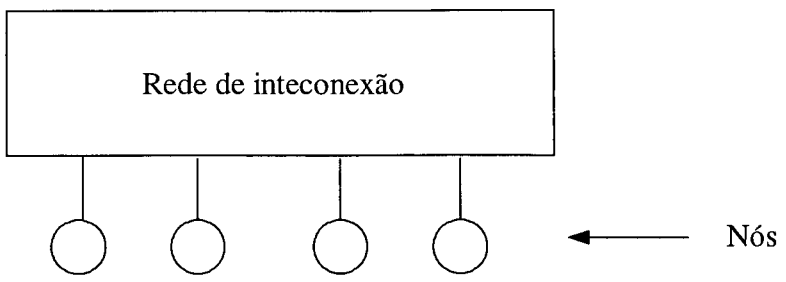

Figura 2-1 : **Multicomputador.**

2.3 Redes de Interconexão

As redes de interconexão são constituídas por entidades de ***hardware*** (canais de comunicação) e ***software*** (controle do estabelecimento dos canais) que são projetadas para facilitar a comunicação entre processos e processadores.

Um ambiente de computação paralela pode ser representado como na Figura 2-2 [FEN81] :

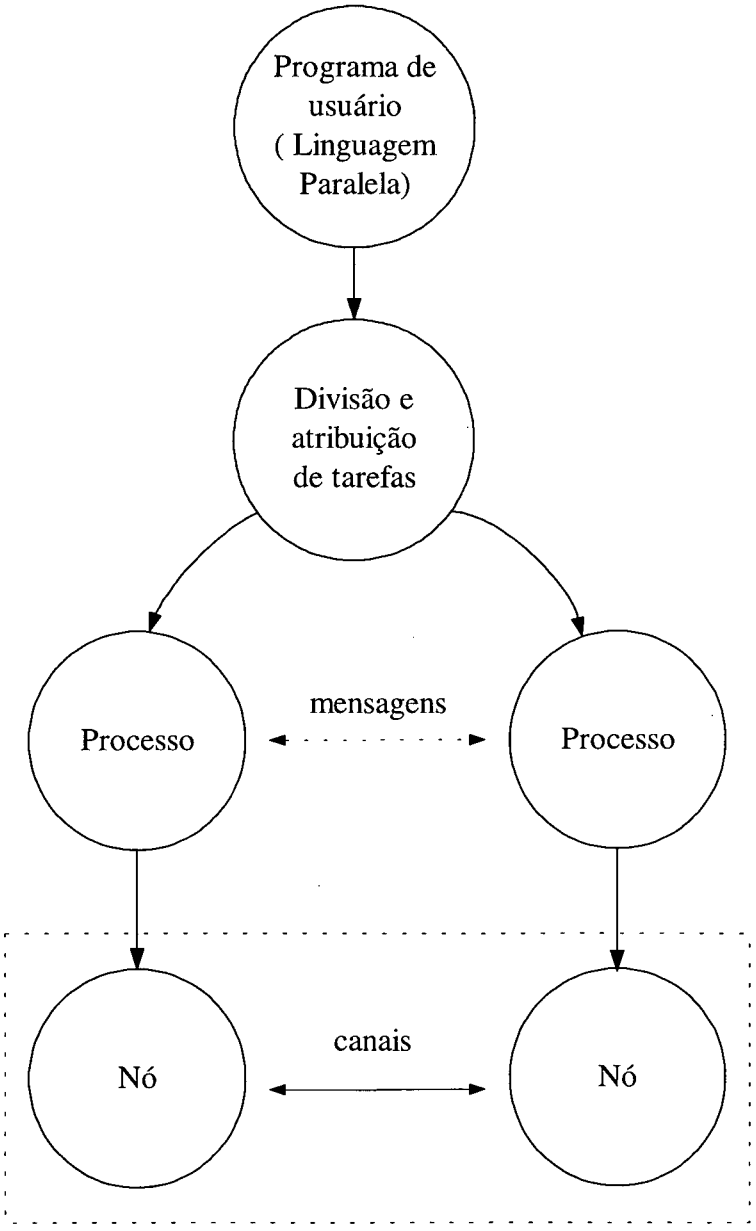

Figura 2-2 **:Sistemas de Processamento Paralelo**

A rede de interconexão é o elemento principal de uma arquitetura paralela. No passado, essas redes serviam somente à comunicação de dados, resultado direto do conceito de trocas de mensagens no processamento distribuído. A tendência atual é de projetar redes de interconexão de tal forma que também as funções a nível de sistema, tais como alocação de recursos e sincronização possam ser facilmente implementadas usando facilidades da rede.

As redes de interconexão são consideradas como **recursos,** da mesma forma que processadores. Elas influenciam fortemente o projeto de algoritmos, linguagens, compiladores e escalonadores. Conseqüentemente, o problema do projeto de redes de interconexão efetivas torna-se cada vez mais crítico encorajando a busca de arquiteturas paralelas inovadoras.

Na seleção de uma arquitetura de rede de interconexão, algumas importantes decisões de projeto podem ser identificadas : a topologia da rede, as técnicas de reconfiguração, a estratégia de controle, o método de comutação e combinação entre algoritmo e arquitetura.

- **Topologia da Rede de Interconexão**

A topologia de uma rede de inteconexão pode ser representada por um grafo onde os vértices representam os nós processadores e as arestas representam canais de comunicação. A topologia das arquiteturas paralelas é um parâmetro de extrema importância por ter influência sobre o paralelismo a nível de **hardware.** As redes de interconexão bipontuais se dispõem de forma regular e podem ser agrupadas em duas categorias : estáticas e dinâmicas [FEN81].

Um grande número de propostas de redes de interconexão têm aparecido na literatura e uma enorme quantidade de pesquisa é centrada no projeto e análise dessas redes.

- **Técnicas de Reconfiguração das Redes Dinâmicas**

Técnicas de reconfiguração são usadas para alocar recursos de **hardware**, tais como nós processadores e canais de comunicação, para uma tarefa específica. Os recursos de **hardware** alocados são normalmente interconectados para formar uma topologia de rede adequada para a tarefa. A reconfiguração é executada quando uma tarefa solicita recursos através de um controlador do sistema. Uma técnica de

reconfiguração eficiente deve encaixar várias topologias de rede com um pequeno **overhead**, enquanto a utilização de recursos permanece alta.

A técnica de reconfiguração é particularmente importante no escalonamento de sistemas de grande escala para computação de alto desempenho já que existe uma necessidade de combinar arquitetura e algoritmo e de alocar recursos de comunicação e processadores. A reconfiguração depende da habilidade da rede para adicionar novas conexões arbitrárias para uma rede com conexões existentes quaisquer. A rede pode permitir que qualquer conexão arbitrária seja estabelecida a qualquer instante ou o estabelecimento de alguma conexão pode ser bloqueado por uma conexão existente que usa um ponto de cruzamento que é necessário para a nova conexão. Dentre os principais termos que descrevem a habilidade das redes para estabelecer conexões pode-se ressaltar : não bloqueante e bloqueante.

Uma rede é **não bloqueante** se qualquer conexão desejada entre nós não utilizados pode ser estabelecida imediatamente sem interferência de quaisquer conexões existentes. Uma rede é **bloqueante** se existem grupos de conexões que impedem que conexões adicionais sejam estabelecidas entre nós não utilizados.

As classes de redes relevantes ao bloqueio são um resultado direto de várias topologias de rede onde a relação custo/desempenho principal é entre bloqueio e quantidade de pontos de cruzamento. Em geral, quanto mais pontos de cruzamento, mais rotas alternativas possíveis e menor a chance de bloqueio. Muito do esforço no estudo e desenvolvimento dessas topologias de rede referem-se à obtenção de alguma capacidade de conexão necessária com um número mínimo de pontos de cruzamento.

- **Estratégia de Controle e Método de Comutação das Redes Estáticas**

A estratégia de controle refere-se ao transporte de mensagens, roteamento e gerência de armazenamento temporário.

Roteamento pode ser implementado por dois métodos de comutação diferentes: comutação de circuitos e comutação de pacotes. Na comutação de circuitos um caminho físico é estabelecido entre uma fonte e um destino. Na comutação de pacotes, dados são colocados em um pacote e roteados através da rede de interconexão sem o estabelecimento de uma conexão física.

Na comutação de circuitos, uma vez estabelecida a conexão, dados podem ser transmitidos diretamente da entrada para a saída a altas velocidades dependentes somente das características elétricas do canal. Na comutação de pacotes há a necessidade de roteamento e de gerência de armazenamento temporário de pacotes enquanto eles estão a caminho do destino.

- **Combinação entre Algoritmo e Arquitetura**

O problema de mapeamento surge quando o padrão de comunicação do algoritmo paralelo difere da topologia da rede de interconexão do multicomputador [HWA87]. Mapeamento refere-se a forma utilizada para atribuir processos a recursos tais como os nós com o objetivo de minimizar o número total de passos de roteamento.

A exploração do paralelismo real provido pelos multicomputadores impõe a construção de programas paralelos sob a forma de redes de processos que se comunicam exclusivamente através de trocas de mensagens. Essas redes vêm sendo consideradas como um bom modelo de programação paralela e têm dado origem a várias linguagens como, por exemplo, Occam [INM84] e Super Pascal [HAN94a].

Com o objetivo de fornecer suporte às linguagens acima citadas, existem duas abordagens de combinação entre algoritmo e arquitetura paralela nos multicomputadores :

- se todas as tarefas paralelas no processamento são conhecidas *a priori* elas podem ser estaticamente mapeadas nos nós da rede antes do processamento começar;
- se novas tarefas iniciam e tarefas existentes terminam conforme o desenrolar do processamento, elas devem ser atribuídas dinamicamente aos nós da rede.

2.3.1 Redes de Interconexão Estáticas

As topologias estáticas apresentam canais ligando direta e estaticamente nós sem a possibilidade de reconfiguração. A comunicação entre nós não diretamente ligados ocorre por intermédio de outros nós. Dois exemplos significativos de redes estáticas são a grelha quadrada e o hipercubo

A **grelha quadrada** consiste de uma coleção de nós conectados como na Figura 2-3 *: **Grelha quadrada 3 x 3.***.Ela possui n^2 nós, onde n representa o número de nós em cada linha ou coluna. A Figura 2-3apresenta uma grelha quadrada com $n = 3$, também denominada grelha 3 x 3.

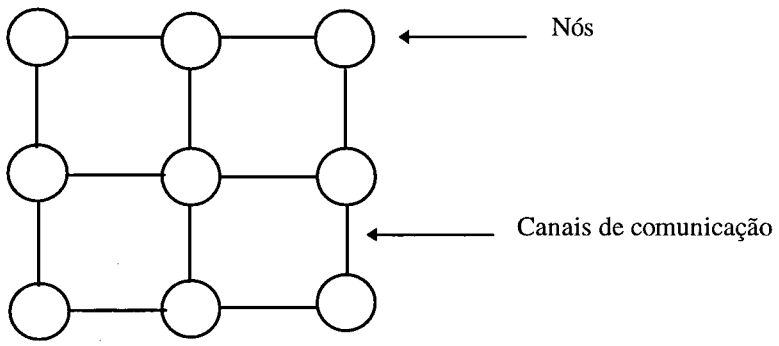

Figura *2-3* : *Grelha quadrada 3 x 3.*

O **hipercubo** ou n-cubo binário [REE87] é uma máquina cuja topologia é caracterizada pelo parâmetro n, denominado sua dimensão, que determina o número de ligações de cada nó (Figura 2-4 : Hipercubo de dimensão 4.). Um hipercubo de dimensão n possui 2^n nós. Um hipercubo de dimensão n+1 é obtido como a composição de dois hipercubos de dimensão n. Dessa forma um hipercubo de dimensão 0 possui um único nó, um de dimensão 1 é composto de dois nós, um de dimensão 2 é formado de quatro nós e assim por diante.

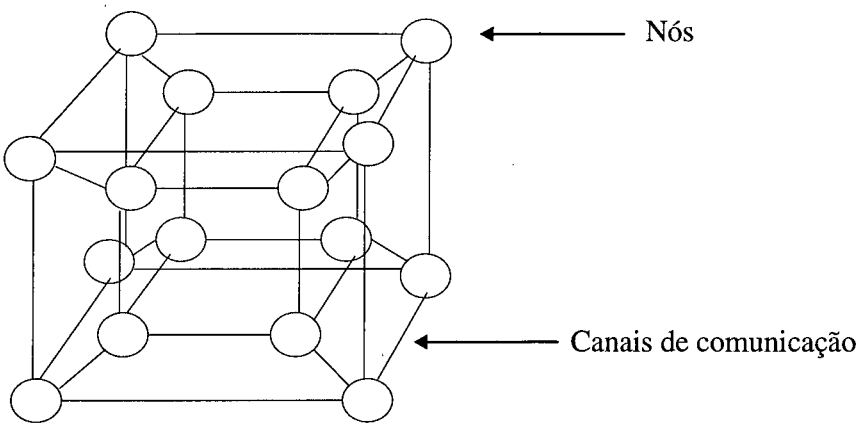

Figura 2-4 : *Hipercubo de dimensão 4.*

Alguns aspectos importantes a considerar nas topologias estáticas são : o tipo de aplicação para as quais elas são mais eficientes, a possibilidade de crescimento do número de nós e o comprimento máximo do caminho que uma mensagem percorre em cada topologia.

Aplicações específicas com seus respectivos padrões de comunicação sugerem a utilização de uma topologia determinada. Como exemplo de aplicação para a topologia grelha quadrada tem-se a multiplicação de matrizes.

Escalabilidade é a qualidade de uma máquina ser capaz de crescer em número de nós para aumentar sua potência de processamento. Essa é uma característica importante na maioria dos ambientes de computação onde a necessidade por capacidade de processamento tende a crescer com o passar do tempo de forma a exceder o que está disponível. A solução usual é abandonar o computador atual e comprar um novo. Um sistema escalável é capaz de crescer conforme crescem as necessidades dos usuários.

A topologia de uma máquina pode ter influência muito importante no fator escalabilidade. O número de canais por nó não se altera com o acréscimo de linhas e de colunas da grelha quadrada. Já o número de conexões de cada nó cresce com a dimensão do hipercubo. No projecto do *hardware* de qualquer topologia uma dimensão máxima sempre deve ser fixada.

Por exemplo, um nó possuindo 10 canais pode ser usado para construir qualquer hipercubo com até 2^{10} (1024) nós.

Tanto na grelha quadrada como no hipercubo, uma mensagem pode eventualmente percorrer vários nós para atingir seu destino. No entanto, a rota percorrida entre os nós mais distantes em ambas as topologias para o mesmo número de nós, definida como diâmetro da rede, é mais longa na grelha. Na grelha quadrada com n nós em cada linha ou coluna, o diâmetro é igual a $2n$-2 [HWA93] enquanto no hipercubo é igual a sua dimensão.

Um dos aspectos que determina a flexibilidade das topologias é a possibilidade de executar uma aplicação feita para uma topologia diferente. Assim, pode-se executar eficientemente uma aplicação feita para a grelha quadrada em um hipercubo se o hipercubo contiver a grelha quadrada como sub-rede.

2.3.2 Redes de Interconexão Dinâmicas

Na topologia dinâmica, os canais dos nós não estão ligados diretamente uns aos outros mas utilizam um comutador de conexões para essa tarefa. As ligações podem ser reconfiguradas pelo ajuste dos elementos de comutação ativos da rede de interconexão.

Topologias da categoria dinâmica são classificadas em barramentos, redes multiestágio e *crossbar*.

- **Barramentos** - são uma coleção de fios e conectores interconectando os vários nós de processamento de forma compartilhada (Figura 2-5). Têm baixo custo mas possuem limitação na sua capacidade de transferência acarretando degradação de desempenho do sistema quando sobrecarregados.

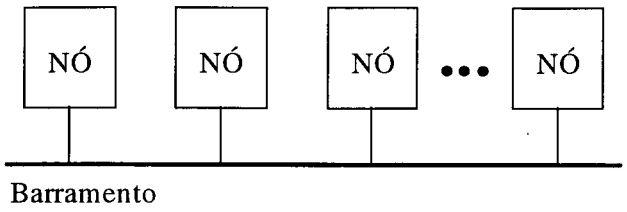

Figura 2-5 : **Multicomputador baseado em barramento.**

• **Redes multiestágio** - são constituídos de vários estágios de pontos de cruzamento. Apenas o primeiro e o último estágios são conectados a nós de processamento. Pontos de cruzamento nos estágios intermediários são conectados a outros pontos de cruzamento. Vários estágios geralmente diminuem o número de pontos de cruzamento necessários para grandes redes. Vários estágios também aumentam o tempo para conectar dois nós que torna-se significativo em redes de muitos estágios. Um exemplo de rede multiestágio é a rede ômega na qual o tempo para conectar qualquer par de nós tem sempre a mesma duração (Figura 2-6).

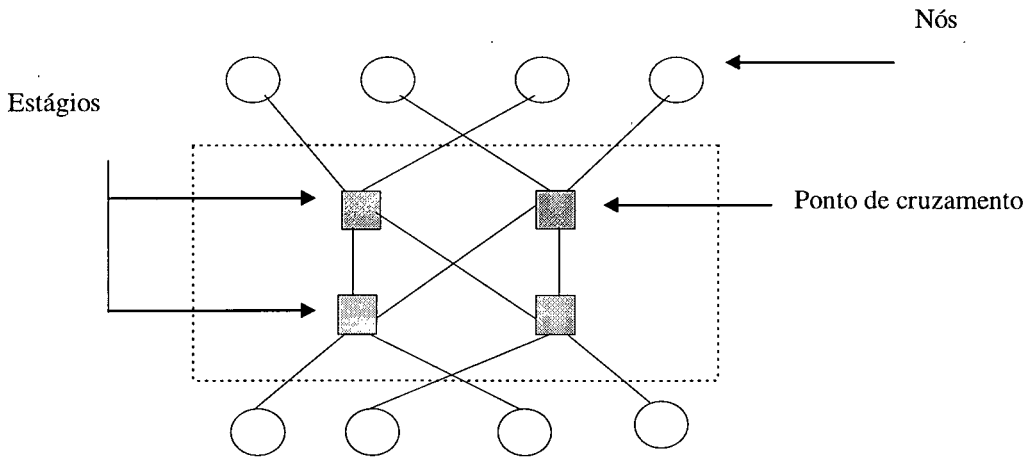

Figura 2-6 : **Rede ômega 2 x 2.**

•**Crossbar** - Um *crossbar* é uma rede de interconexão que possui um ponto de cruzamento para cada par de nós. Nessa rede, apenas um ponto de contato precisa ser fechado para estabelecer a conexão de qualquer par de nós. Ela é a única rede de

comutação de circuitos verdadeiramente de estágio único [BRO83]. O *crossbar* com n nós de entrada e n nós de saída possui n^2 pontos de cruzamento (Figura 2-7).

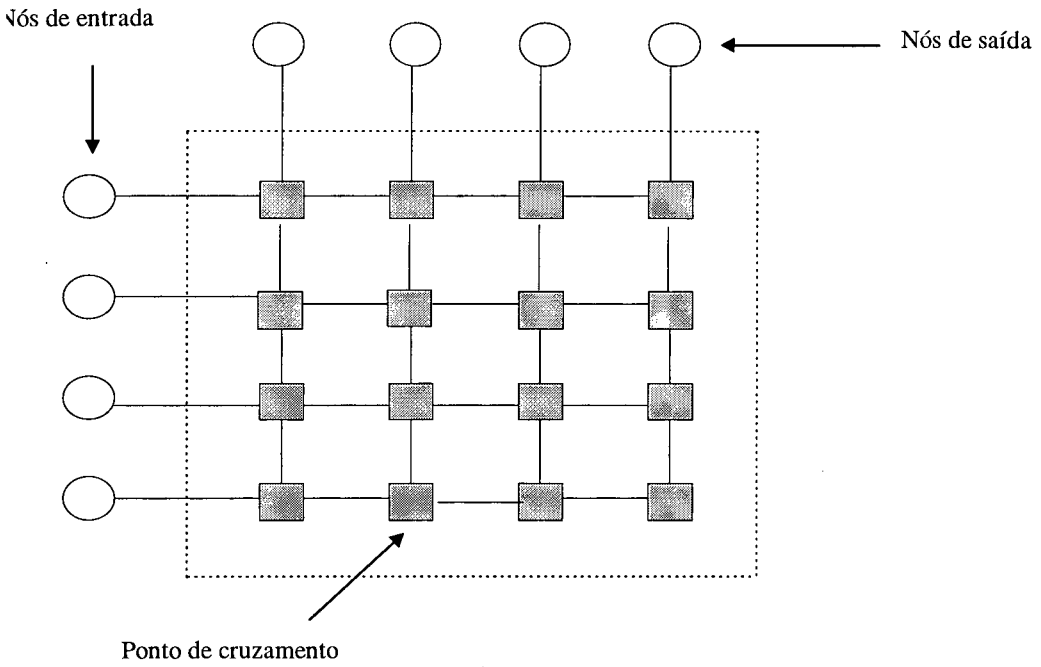

Figura 2-7 : **Crossbar 4 x 4.**

O *crossbar* é a rede de interconexão dinâmica mais poderosa [HWA93]. Ela é não bloqueante possuindo reduzido tempo para conexão de dois nós. Entretanto ela é de alto custo pelo grande número de pontos de cruzamento necessários.

"O custo" em redes de comutação é normalmente referido como "*complexidade*" ou número de elementos de comutação necessários. O custo de um *crossbar* pode ser medido pelo número de entradas multiplicado pelo número de saídas. Assim, o custo não está necessariamente relacionado ao preço, sendo antes uma medida para comparar a complexidade relativa das redes.

Quanto à escalabilidade é importante ressaltar que os avanços da tecnologia não podem ser ignorados. Apesar do estudo de redes de interconexão do tipo *crossbar* com tecnologia ótica já estar em desenvolvimento, o tempo necessário para comutar linhas de sinais óticos ainda é lento quando comparado com a tecnologia eletrônica atual. Entretanto, avanços nesse sentido podem viabilizar no futuro redes de grandes dimensões [VAR87].

CAPITULO 3

3. Comunicação em Sistemas Distribuídos

3.1 Introdução

Um sistema de computação distribuído é uma coleção de computadores independentes que se apresentam para o usuário como um único computador [TAN92]. Como os sistemas operacionais distribuídos são implementados em processadores que possuem memória privativa, se faz necessário a implementação do mecanismo de troca de mensagens entre processos.

3.2 Comunicação entre Processos

Em ambientes distribuídos, o sistema operacional deve prover a comunicação uniforme entre processos, independentemente do nó em que está, do nó que comanda as conexões, e da rede de conexão. Cada nó da máquina deve conter um micronúcleo residente que implementa serviços que suportem mais que um processo.

Um processo se comunica com outro pelo envio e recebimento de mensagens. Basicamente os serviços de comunicação entre processos são representados por primitivas do tipo *send (mensagem)*, para o envio de uma mensagem a um processo destinatário, e *receive (mensagem)* para o recebimento de uma mensagem de um processo origem.

O projeto das primitivas de comunicação depende de decisões referentes à disciplina de comunicação, endereçamento, sincronismo, armazenamento temporário e fluxo.

As disciplinas de comunicação são definidas como os padrões de troca de mensagens entre dois processos.

O endereçamento consiste da forma na qual um processo se refere a outro, podendo ser feito o endereçamento direto, através de seus identificadores, ou endereçamento indireto envolvendo um elemento intermediário chamado caixa postal, para o qual os processos emissores enviam uma mensagem e do qual os processos receptores obtém uma mensagem.

Quanto ao sincronismo a comunicação pode ser síncrona ou assíncrona. Na forma síncrona, as primitivas de comunicação implementam o bloqueio do processo emissor até que o processo receptor esteja apto a receber a mensagem. Na forma assíncrona, o processo emissor é desbloqueado após a mensagem ter sido copiada para o núcleo de transmissão e apenas o processo receptor permanece bloqueado até a chegada de uma mensagem. O armazenamento temporário de mensagens é utilizado no modelo assíncrono, onde um processo receptor mais lento pode gerar um conjunto de mensagens pendentes. Este problema pode ser resolvido pelo núcleo gerindo filas de mensagens, uma por processo ou caixa postal, com um tamanho limitado. No caso de ocorrer esgotamento do espaço de armazenamento temporário, o processo emissor fica bloqueado até o processo receptor executar uma primitiva de recepção, liberando o espaço.

O fluxo de mensagens em uma comunicação pode ser unidirecional ou bidirecional. O fluxo unidirecional é normalmente utilizado em comunicações assíncronas, onde um processo emissor não permanece esperando por um resposta do processo receptor, não sendo necessário a comunicação no outro sentido. O fluxo bidirecional é utilizado em comunicações síncronas, normalmente aplicações cliente-servidor (descrito adiante), onde o processo servidor após receber e processar requisições de clientes, responde com o resultado da operação.

3.2.1 Disciplinas de Comunicação

Existem vários tipos de disciplinas de comunicação, que são definidas como padrões de troca de mensagens entre dois processos [COR93]. Essas disciplinas são utilizadas para facilitar implementações de comunicação adequadas para as aplicações. Embora existam vários tipos de disciplinas, são tratadas a seguir apenas dois modelos.

- **Modelo Produtor-Consumidor**

No modelo produtor-consumidor, um processo que possui um canal unidirecional de emissão através do qual executa uma seqüência de envio de mensagens é chamado ***produtor.*** Um processo que possui um canal unidirecional de recepção através do qual executa uma seqüência de recepção de mensagens é chamado ***consumidor***. Quando um canal de emissão de um produtor é conectado ao canal de recepção de um consumidor, desenvolve-se entre eles uma disciplina de comunicação do tipo produtor consumidor (Figura 3-1).com o uso de esta disciplina é possível construir um *pipeline de processos.*

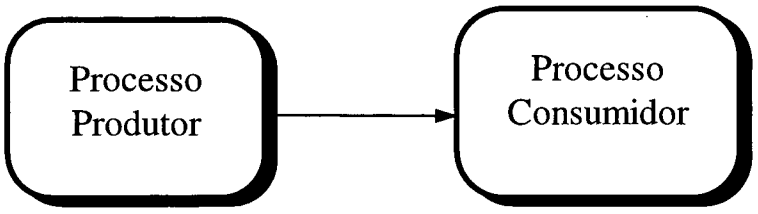

***Figura 3-1*: Disciplina de Comunicação Produtor-Consumidor**

- **Modelo Cliente-Servidor**

No modelo cliente-servidor o sistema operacional pode ser dividido em módulos, cada um dos quais se encarrega de um serviço do sistema . Esses módulos podem ser distribuídos por vários processadores na máquina. Neste modelo, um

processo servidor permanece aguardando pedidos de serviços, vindos de processos clientes, na forma de mensagens. Estes processos, cliente e servidor, possuem um canal bidirecional para envio e recebimento de mensagens. Após o recebimento de uma requisição de serviço, o processo servidor analisa a mensagem, processa o pedido, retorna ao cliente o resultado da operação e volta a aguardar a chegada de outras requisições (Figura 3-2).

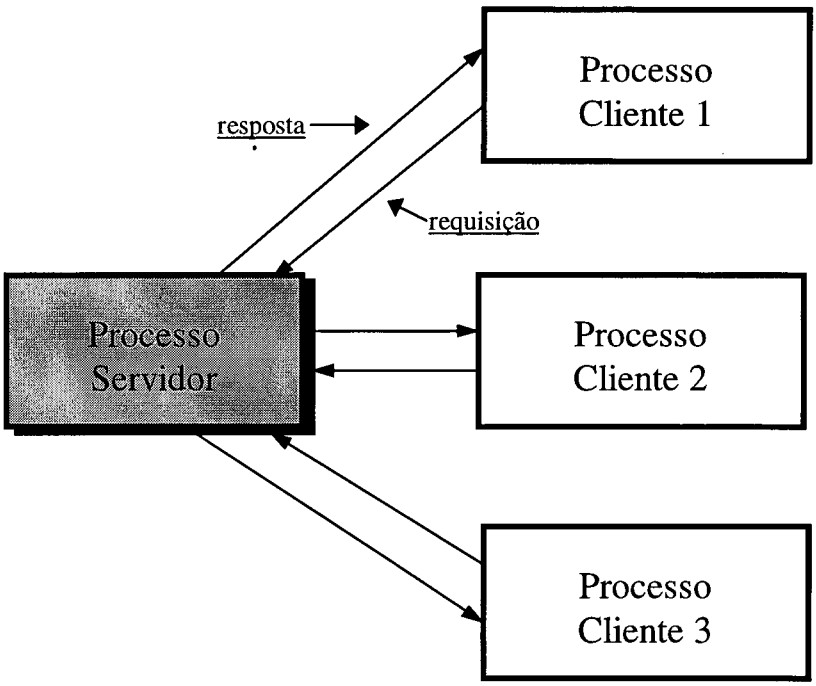

Figura 3-2: **Modelo Cliente-Servidor**

3.3 As Chamadas de Procedimento Remoto

A chamada de procedimento remoto é implementada sobre o modelo cliente-servidor, no qual processos usuários requisitam serviços de processos servidores. Como nos sistemas operacionais distribuídos os processos são colocados em processadores com memórias privativas e as requisições necessitam de comunicação entre processos. Chamadas de procedimento remoto (RPC), podem ocultar essas

comunicações tornando a programação em ambientes distribuídos similar a de ambientes centralizados.

As aplicações, a fim de acessarem os serviços oferecidos pelo sistema, utilizam *stubs* clientes que criam mensagens, com informações sobre o tipo de serviço e seus parâmetros, para serem enviadas ao servidor (Figura 3-3) [COU88]. Quando o servidor recebe a requisição através de um *stub* servidor, atende a requisição e coloca o resultado da operação em um *stub* servidor. O *stub* servidor, por sua vez , coloca o resultado da operação em uma mensagem enviada ao *stub* cliente. Assim, o processo cliente tem a impressão de ter executado um procedimento local, sem perceber as funções de envio, recepção, montagem e desmontagem de mensagens.

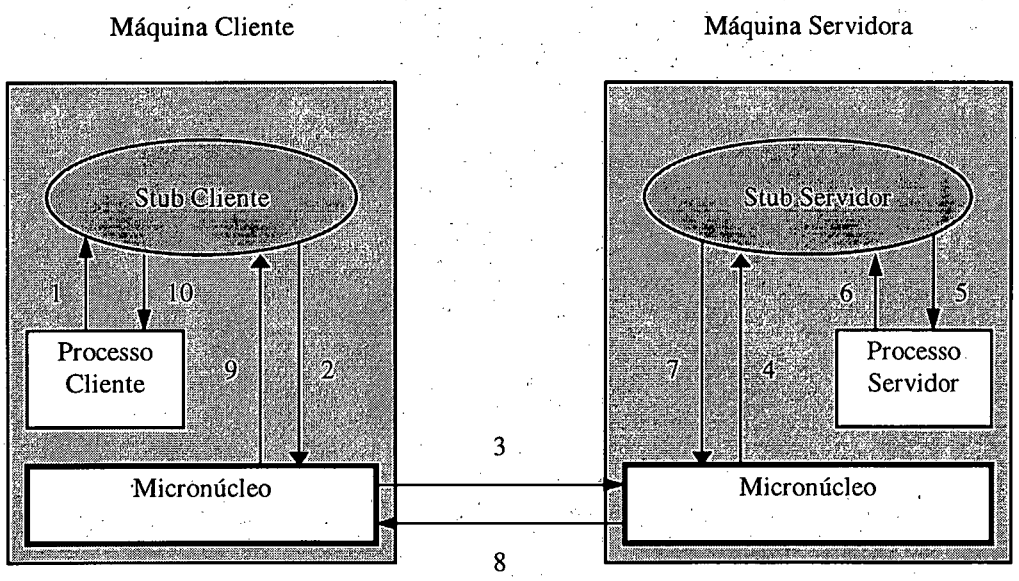

***Figura 3-3*: Chamada de Procedimento Remoto**

A Figura 3-3 apresenta as etapas que envolve uma chamada de procedimento remoto

1. O processo cliente requisita serviço ao processo servidor fazendo uma chamada de procedimento.
2. O stub cliente empacota a requisição e seus parâmetros em uma mensagem e pede para o micronúcleo enviá-la.
3. O micronúcleo envia a mensagem.
4. Do outro lado da linha, o micronúcleo do nó onde se encontra o processo servidor recebe a mensagem e a envia para o stub servidor.
5. O stub servidor desempacota a mensagem, formando uma chamada de procedimento ao servidor.
6. O servidor executa o serviço e retorna o resultado para o stub servidor.
7. O stub servidor empacota a resposta em uma mensagem e pede para o micronúcleo enviá-la.
8. O micronúcleo envia a mensagem.
9. O micronúcleo do nó onde se encontra o processo cliente recebe a mensagem e a envia para o stub cliente.
10. O stub cliente desempacota a resposta e a retorna para o processo cliente.

CAPITULO 4

4. Arquitetura do Multicomputador Nó//

4.1 Introdução

Neste capítulo é apresentada a arquitetura do multicomputador Nó// proposta em [COR93].

A especificação do multicomputador Nó // esta descrita em [CAM95] e sua implementação esta a cargo do grupo de Computação Paralela e Distribuída do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Santa Catarina.

Conforme pode ser observado na Figura 4-1, a arquitetura do Nó // é constituída de diversos nós processadores homogêneos, conectados entre si por intermédio de dois dispositivos distintos: um comutador de conexões do tipo *crossbar* e um barramento de serviços.

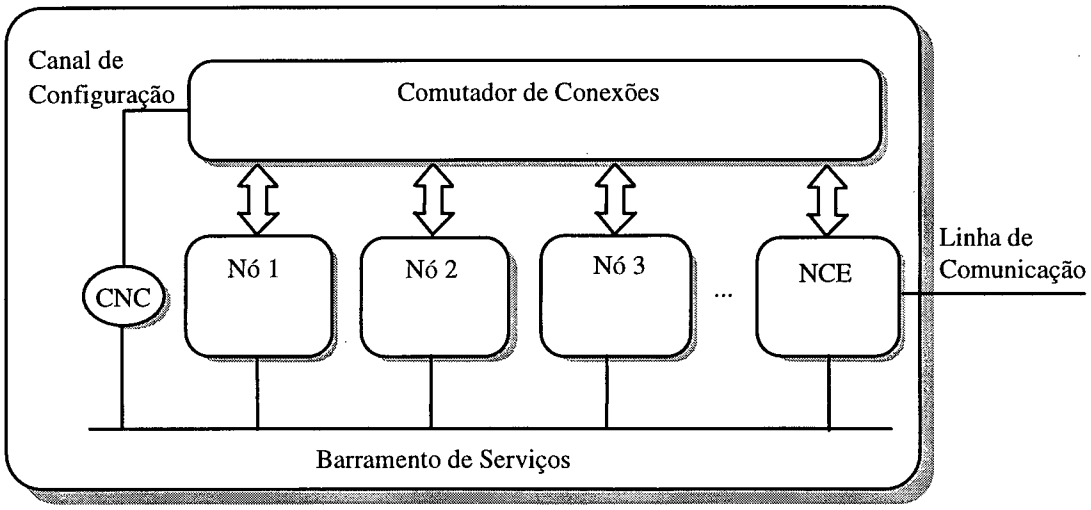

Figura 4-1: **A arquitetura do Nó //.**

4.2 Os Nós

Cada nó dessa arquitetura possui um processador, memória RAM privativa e uma pequena memória ROM, com seu respectivo microcódigo. Na Figura 4-2, que

apresenta detalhadamente um nó, pode ser observado que todos eles possuem também um ou vários canais de comunicação bidirecionais conectados ao comutador de conexões, para troca de mensagens. Além disso, cada nó possui uma interligação com o barramento de serviços, cuja função será descrita em detalhes mais adiante.

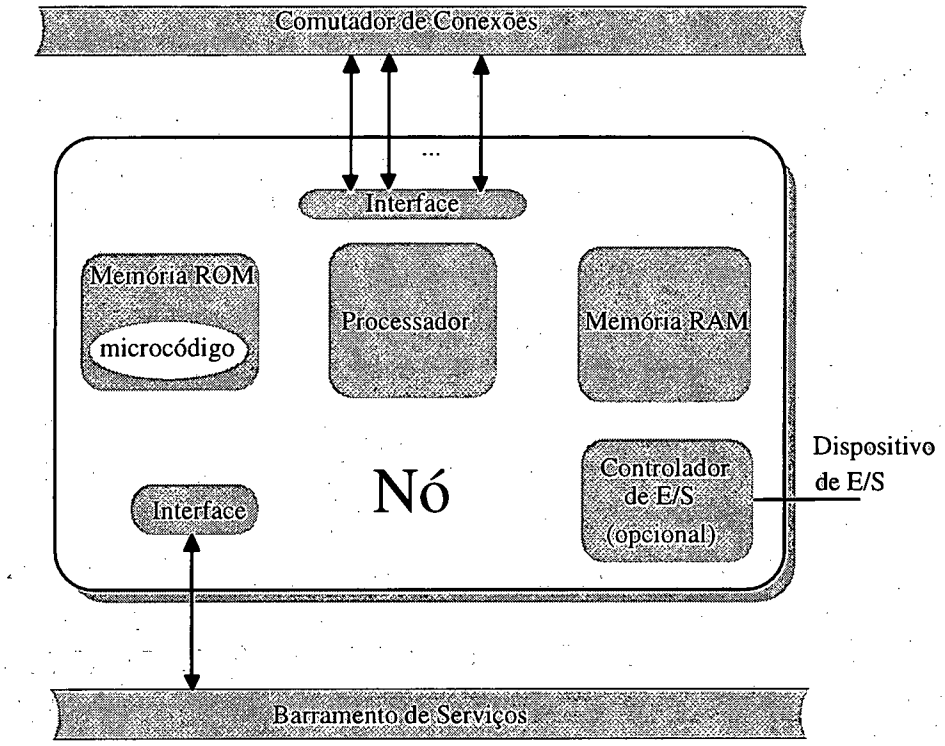

Figura 4-2: **A estrutura interna de um nó.**

4.3 O Comutador de Conexões

O comutador de conexões do Nó // é do tipo *crossbar*. Ele estabelece uma rede dinâmica de interconexão entre os nós processadores. Dessa forma, cada nó pode ser conectado a qualquer outro dinamicamente através de canais de comunicação bidirecionais.

Esses canais de comunicação permitem a transmissão eficiente de mensagens volumosas. Processos colocados em nós distintos precisam requisitar conexões entre si ao comutador de conexões para trocar mensagens através desses canais.

4.4 O Controlador de Nós e Conexões (CNC)

Um dispositivo inteligente, denominado *Controlador de Nós e Conexões* (CNC), é responsável no Nó // por comandar o comutador de conexões através de um *canal de configuração* (Figura 4-1). Todos os demais nós podem requisitar conexões bipontuais entre si se comunicando com o CNC através do barramento de serviços, que funciona como via de comunicação entre um nó qualquer e esse dispositivo.

De forma semelhante a um nó qualquer da arquitetura, o CNC é constituído de processador, memória RAM e memória ROM com o seu microcódigo.

Entretanto, a capacidade de comandar o comutador de conexões o distingue dos demais nós. Devido a essa importante atribuição, ele é dedicado exclusivamente às tarefas relacionadas com essas funções, não possuindo outras conexões com o comutador de conexões e, conseqüentemente, não se comunicando com outros nós através desse dispositivo.

O código que executa em seu processador é armazenado em ROM e os dados em RAM, o código é compacto e otimizado para atender às mensagens com as requisições de conexões entre nós e alocação / liberação de nós. Ele gerencia diversas filas mantidas para o controle dos pedidos de alocação e liberação de nós pelos processos, e de conexão e desconexão de canais de comunicação entre nós.

4.5 O Barramento de Serviços

Principalmente pelo fato de ser uma via compartilhada por todos os nós, o barramento de serviços é concebido para ser utilizado apenas para troca de mensagens curtas e pré-estabelecidas entre um nó qualquer e o CNC. Colisões no acesso ao barramento de serviços não acontecem porque é o CNC que determina com quem vai ser feita a comunicação, verificando, um a um, quais os nós que desejam se comunicar com ele.

As mensagens que trafegam por esse barramento e que são recebidas e tratadas pelo CNC são as seguintes:

Mensagem	Parâmetro	Retorno
CONNECT	*nid*	*nada*

Essa mensagem é uma requisição ao CNC para que ele estabeleça uma conexão bipontual entre o nó do processo que pediu a conexão e o nó *nid*. A conexão só é realmente estabelecida quando o processo no nó *nid* também pede uma conexão e, enquanto isso não ocorre, a requisição permanece numa fila de requisições de conexões geridas pelo CNC. Somente quando a conexão é realmente efetivada é retornado uma mensagem de confirmação para ao processo que requisitou a conexão.

Mensagem	Parâmetro	Retorno
CONNECT_ANY	*nenhum*	*nid*

Essa mensagem é semelhante à anterior com a diferença que é solicitada uma conexão com qualquer nó que deseja se comunicar com o nó onde se encontra o processo que pediu a conexão. Quando a conexão é efetuada, é retornado o identificador do nó com o qual foi efetuada a conexão.

Mensagem	Parâmetro	Retorno
DISCONNECT	*nid*	*nada*

Essa mensagem é um pedido ao CNC para que ele desfaça a conexão entre o nó do processo que pediu a desconexão e o nó *nid* especificado. A conexão física do canal de comunicação entre os nós é imediatamente desfeita e o processo que executou a chamada fica liberado para requisitar outras conexões. O processo que executa no outro nó conectado precisa, também, posteriormente, pedir a desconexão, que nesse caso é apenas lógica, já que a conexão física já foi desfeita.

Mensagem	Parâmetro	Retorno
ALLOCATE	*nid*	*nada*

Essa mensagem é um pedido para o CNC reservar o nó *nid* para o processo que enviou a mensagem. Após reservar o nó, o CNC estabelece, uma conexão entre o nó reservado e o nó onde se encontra o processo que pediu a alocação.

Se o nó requisitado já está alocado, essa chamada retorna uma mensagem de erro.

Mensagem	Parâmetro	Retorno
ALLOCATE_ANY	*nenhum*	*nid*

Essa mensagem é semelhante a anterior com a exceção que nela não se especifica qual nó se deseja alocar. O CNC reserva um nó qualquer segundo uma fila interna de nós disponíveis que ele possui, e retorna o identificador do nó para o processo que pediu a alocação. Se não há nós disponíveis para alocar, essa chamada retorna uma mensagem de erro. De forma semelhante à mensagem anterior, uma conexão entre os nós é estabelecida.

Mensagem	Parâmetro	Retorno
DEALLOCATE	*nid*	*nada*

Essa mensagem é um pedido ao CNC para que ele libere o nó onde executa o processo que enviou a mensagem.

4.6 O Nó de Comunicação Externa (NCE)

Um nó especial dessa arquitetura possui a tarefa de fazer a comunicação com o "mundo externo" através de uma linha de comunicação. Esse nó é denominado Nó de Comunicação Externa (NCE)

Máquinas com esta arquitetura podem ser usadas como nós independentes em redes locais (e é esta a vocação das mesmas), conforme o modelo de *pool de processadores* [TAN92 - pág. 531]. Os processadores podem ser alocados por demanda, segundo a necessidade dinâmica dos processos dos programas paralelos.

CAPITULO 5

5. Simulador do Multicomputador Nó //

5.1 Introdução

Neste capítulo é apresentado o simulador do multicomputador Nó //, apresenta-se a configuração mínima da máquina simulada e, descreve-se o simulador implementado.

Apesar da arquitetura do Nó // ser de concepção simples se comparada com outras existentes ou propostas, ela é uma solução inovadora. Esse fato torna importante o desenvolvimento de um ambiente que apresente as mesmas características da máquina física, sendo assim possível o desenvolvimento de software para o Nó // antes da conclusão do harware propriamente dito. Isto também possibilita a avaliação da proposta da máquina. Este ambiente é o Simulador do Multicomputador Nó//.

5.2 Descrição do Simulador

Ao se implementar o simulador, optou-se por fazê-lo simulando uma configuração mínima do Nó //. Essa configuração que mantém as propriedades essenciais do modelo proposto, apresenta 32 nós, cada qual com 64 *Kbytes* de memória, e apenas um canal de comunicação ligando-os ao comutador de conexões. Além disso, considera-se o Nó // fisicamente conectado a uma estação de trabalho autônoma, conforme mostra a Figura 5-1.

Figura 5-1: **Nó // conectado a uma estação de trabalho.**

Essa simplificação mantém a propriedade principal do modelo: uma tarefa disparada na estação de trabalho pode ser subdividida em processos que podem ser alocados em nós do Nó // e executados paralelamente entre si.

O simulador e composto por dois microcomputadores IBM-PC i486, um para executar o simulador sobre o sistema MS-DOS [DUN86] e o outro para servir de estação de trabalho utilizando o sistema operacional Linux [KIR94].

O simulador foi construído tomando-se como base o sistema XINU [COM84]. Este sistema foi modificado para que pudesse ser usado para os propósitos específicos de simular o multicomputador Nó // .

As necessidades principais do simulador são:

- ***Simulação dos Nós Processadores***

Existe a necessidade de simular cada nó processador do Nó //, incluindo o NCE e o CNC, que podem ser abstraídos como nós especiais. Além disso, é necessário a simulação do paralelismo real de processamento existente entre os nós processadores.

- ***Simulação da Comunicação***

A comunicação entre os elementos processadores do Nó //, tanto pêlos canais ligados ao comutador de conexões quanto pelo barramento de serviços necessitam ser simulados.

O código-fonte do XINU é escrito em linguagem C, com pequenos trechos em linguagem *assembly*, e encontra-se disponível para várias plataformas tanto à nível de máquinas (IBM-PC, Macintosh) quanto de compiladores (Microsoft C, Borland C).

Diversas camadas de serviços oferecidas pelo XINU são desnecessárias ao simulador e foram retiradas do código, gerando um código final mais enxuto.

5.3 Implementação do Simulador

As principais partes do XINU utilizadas pelo simulador são brevemente descritas a seguir:

- ***Gerente de Processos***

Processos XINU são semelhantes às funções das linguagens de programação convencionais com exceção de possuírem fluxos de execução que executam de forma concorrente entre si.

O gerente de processos do XINU oferece serviços para criar (*create*), remover (*kill*), suspender (*suspend*) e reativar (*resume*) processos. Ele foi alterado para gerar os processos que vão simular todos os nós do Nó //. Dessa forma, a simulação de cada nó é feita através da criação de um processo (*create*) no XINU, conforme pode ser observado na Figura 5-2.

```
// cria os nós do Nó //
for( no = 0 ; no < TOTAL_NOS ; no++ )
        create( CodigoNo, TAM_PILHA, INITPRIO, "NO", 0);
```

Figura 5-2: **Trecho de código que cria processos XINU.**

O escalonamento preemptivo de processos oferecido pelo XINU é utilizado para simular o paralelismo real de processamentos existentes entre nós do Nó //.

- ***Gerente de Memória***

Cada nó da máquina possui sua memória privativa. Para simular a memória de cada nó foram alterados os serviços *getmem* e *freemem* que o XINU oferece. Os

processos XINU requisitam memória através da chamada *getmem* para simular as memórias privativas dos nós.

A Figura 5-3, mostra um trecho do código que simula um nó, e nele pode ser visto a alocação de memória para simular a memória do nó.

```
/ Codigo do nó
int CodigoNo(void)
{          ...
// aloca a memória para simular a memória do nó
mem = getmem(TAM_NO);
```

***Figura 5-3*: Trecho de código que simula um nó**

- **Comunicação entre Processos**

O XINU possui duas funções *send* e *receive* que servem para comunicação entre seus processos. A alteração dessas funções permitiu simular o envio e o recebimento de *bytes* pelos nós através do barramento de serviços e do comutador de conexões.

A Figura 5-4, esquematiza a utilização do XINU como base para a confecção do simulador e mostra o relacionamento entre os diferentes componentes do projeto Nó//.

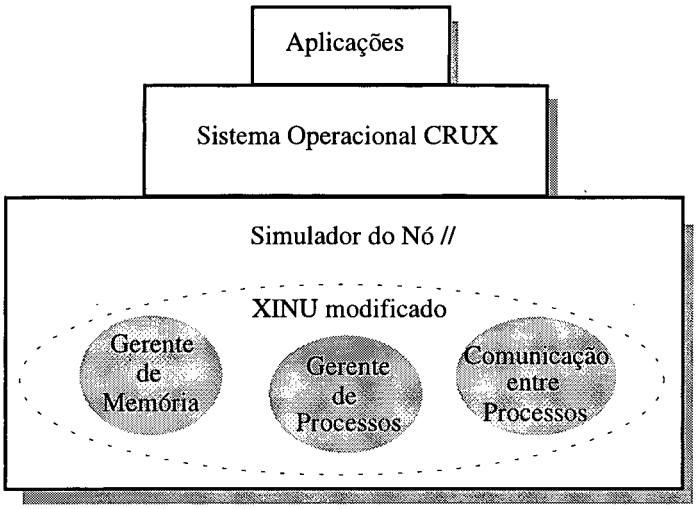

***Figura 5-4:* O XINU utilizado como base para o simulador do Nó //.**

É importante ressaltar que embora a confecção do simulador se constituísse num trabalho relevante, durante o seu desenvolvimento sempre se manteve em consideração o seu objetivo principal que é o de oferecer o mais rapidamente possível um ambiente completo de programação equivalente ao do Nó //.

5.4 Alterações no XINU

Embora se mostrasse adequado para servir de base para o simulador, o XINU necessitou de diversas alterações, principalmente na forma de execução das chamadas de sistema e na gerência de memória. Elas são descritas a seguir.

- ***Chamadas de sistema Implementadas através de Traps***

Na implementação original do XINU, os programas de usuário interagem diretamente com as camadas mais internas do sistema operacional. Os processos XINU após compilados são ligados ao próprio código do sistema XINU. Por isso, os processos conhecem o endereço das rotinas que implementam as chamadas de

sistema. Toda chamada de sistema é efetuada como qualquer rotina do XINU, através de um ***call***.

Essas características inviabilizariam a carga dinâmica de processos, necessária ao simulador. Para servir de base para o simulador foi necessário então implementar um outro esquema de chamadas de sistema, através de interrupções de ***software***.

Nesse esquema, é gerada uma biblioteca de chamadas de sistema. Essa biblioteca é ligada junto com as aplicações que desejam fazer as chamadas. Ao ser feita a chamada, a rotina respectiva da biblioteca armazena os valores necessários em determinados registradores da máquina e gera uma interrupção pré-estabelecida.

O XINU captura essa interrupção, sua rotina de tratamento recebe o controle, verifica através dos valores dos registradores qual o serviço requisitado e quais os parâmetros necessários, executa a função e retorna valores nos registradores.

A rotina da biblioteca, por sua vez, interpreta os valores retornados pelos registradores, e retorna os valores da forma padrão da chamada de sistema.

- ***Alteração no Gerente de Memória***

O gerente de memória original do XINU libera apenas 64 *Kbytes* para serem alocados entre todos os processos, o que é insuficiente para o simulador trabalhar com o número de nós que se pretende simular. Essa limitação sem razão aparente se deve, provavelmente, a motivos históricos. Portanto, um esquema alternativo necessita ser implementado para contornar esse problema. Duas alterações são propostas a seguir:

A primeira é alterar o gerente de memória para ele utilizar toda a memória abaixo dos 640 *Kbytes* disponíveis existente. Essa memória é denominada memória real, em contraste a memória estendida que reside acima do primeiro Megabyte. Essa alteração iria liberar uma quantidade de memória ainda insuficiente para simular todos os nós pretendidos.

A segunda alteração, que pode ser concebida como uma segunda etapa da primeira, consiste em fazer permutas (*swapping*) de processos que não estão em execução. A maneira mais simples seria fazer permutas em disco, porém a velocidade

da simulação seria muito prejudicada. Pretende-se, por isso, utilizar a memória acima do primeiro *megabyte*, através de um gerenciador de memória expandida EMM386.SYS, existente no MS-DOS [DUN86 - pág. 185]. Esse gerenciador irá criar bancos de 64 *Kbytes* que serão utilizados para se efetuar permutas em memória. Esse esquema é extremamente veloz, principalmente pelo fato do gerenciador não efetuar cópias de memória, e sim, utilizar de forma totalmente transparente, a memória virtual e mapeamentos de memória.

CAPITULO 6

6. O Sistema CRUX

6.1 Introdução

O sistema operacional é responsável pela administração dos recursos físicos e lógicos dos computadores. Ele pode ser visto globalmente como uma montagem de sub-sistemas concebidos como servidores construídos em torno de cada tipo de recurso para aplicar métodos de utilização específicos. O sistema operacional ***Crux,*** bem como cada um de seus sub-sistemas, pode ser visto tanto como uma **rede de processos comunicantes** quanto como uma **hierarquia de camadas de *software*** [COR93].

6.2 Processo CRUX

Os processos do *Crux* são programas em execução nos nós do Nó //. Um processo (Figura 6-1) é constituído pelo seu espaço de endereçamento, dados, pilha, valores dos registradores e outras informações necessárias para sua execução [MON95].

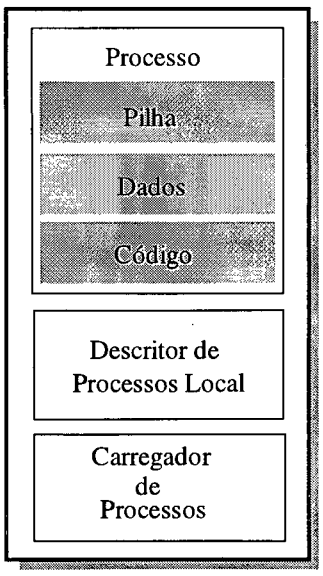

***Figura 6-1*: Um processo colocado na memória de um nó.**

Conforme pode ser observado na Figura 6-1, existem três componentes principais descritos a seguir:

- ***Carregador de Processos***

Os processos do *Crux* são criados através da chamada *fork*, que é uma chamada de sistema compatível com UNIX. Neste caso, o processo criado é enviado para um nó disponível que o recebe e o instala. O código responsável por receber e instalar o processo e seu descritor na memória é o carregador de processos.

- ***Descritor de Processos Local***

O sistema operacional possui uma estrutura na memória de cada nó própria para armazenar o contexto do processo. Essa estrutura, denominada *descritor de processos local*, armazena localmente diversas informações que servem para descrever o processo para o sistema operacional.

- ***Processo***

Um processo colocado na memória é composto de seu código, sua área de dados e pilha. A área de código consiste em uma seqüência de padrões de *bytes* que o processador interpreta como instruções de máquina. A área de dados corresponde à seção de dados inicializados e não inicializados existentes no arquivo executável. A área de pilha é criada e pode ser alterada dinamicamente durante a execução do programa.

Na implementação inicial do *Crux* utilizou-se um formato de programa que não necessita de relocação de seus endereços durante sua carga. Seu formato é semelhante aos dos programas .COM existentes nos sistemas MS-DOS. Essa característica permitiu uma grande simplificação nas tarefas dos carregadores de processos.

6.3 As Camadas do Sistema

A visão completa do *Crux* como uma hierarquia de camadas é mostrada na Figura 6-2.

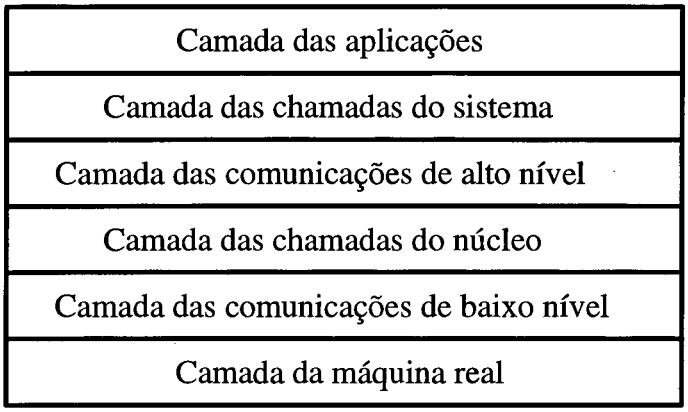

***Figura 6-2*: As camadas do sistema.**

A camada das comunicações de baixo nível, das chamadas do núcleo e das comunicações de alto nível compõem o **micronúcleo distribuído** (Figura 6-3) ou rede de serviços do núcleo do sistema operacional ***Crux***. A camada das chamadas de sistema é acessada através de procedimentos intermediários (*stubs*) relativos aos serviços habituais do sistema operacional e dos serviços de comunicação por troca de mensagens.

Essas camadas são examinadas a seguir em ordem descendente, com exceção da camada das aplicações (constituída das aplicações desenvolvidas sobre o ambiente proposto) e da camada da máquina real (constituída pelo multicomputador Nó //) [COR93].

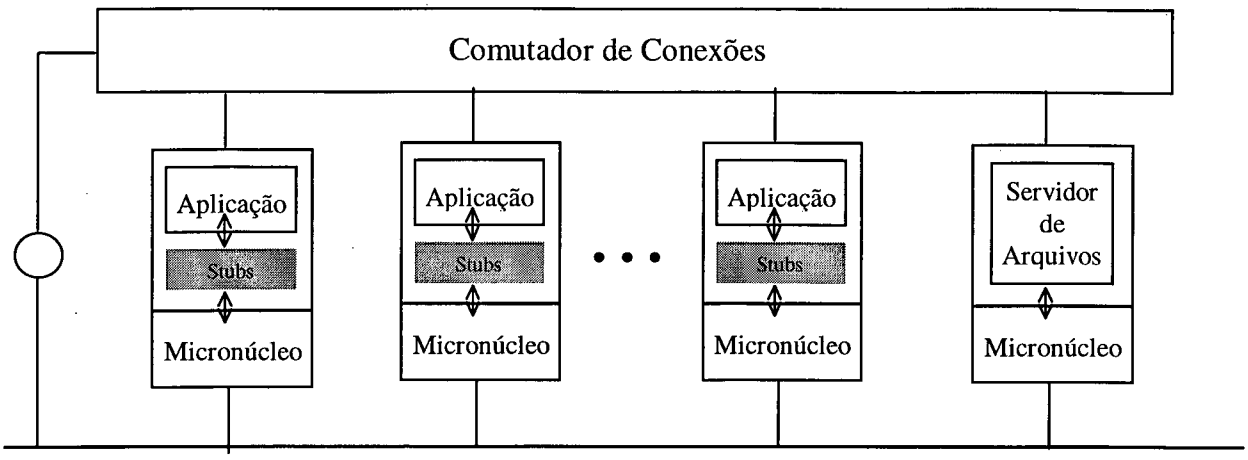

Figura 6-3: **Arquitetura do Crux.**

6.3.1 Camada das chamadas do sistema

Os serviços do sistema operacional são oferecidos através de uma interface de programação de tipo Unix. Ela é estendida, com serviços de comunicação síncrona direta, para criar um ambiente para programação paralela.

6.3.2 Camada das comunicações de alto nível

Na implantação da interface de programação Unix, através de chamadas de procedimentos remotos, as trocas de mensagens entre os intermediários dependem de um serviço de comunicação. O objetivo da camada das comunicações de alto nível é de prover esses serviços que servem ao transporte de mensagens volumosas transitando pêlos canais da rede de interconexão dinâmica.

Comunicações síncronas diretas podem ser efetuadas através dos seguintes operadores:

- ***Send*** (process, message, length)

Um processo colocado em um nó qualquer solicita, através do operador *Send,* o envio de uma mensagem de endereço *message* e de tamanho *length* a outro processo *process* colocado em outro nó.

- ***Receive*** (process, message, length)

Um processo colocado em um nó qualquer solicita, através do operador *Receive,* a recepção de uma mensagem no endereço *message,* de tamanho máximo *length,* de outro processo *process* colocado em outro nó.

- ***ReceiveAny*** (process, message, length)

Um processo colocado em um nó qualquer solicita, através do operador *ReceiveAny,* a recepção de uma mensagem no endereço *message,* de tamanho máximo *length,* de outro processo cuja identificação é devolvida em ***process.***

No caso do operador ***ReceiveAny***, a política de escolha de uma comunicação entre as alternativas possíveis é uma decisão de implementação cuja execução fica a cargo do núcleo do sistema operacional.

Todos os três operadores são utilizados nas comunicações que seguem uma disciplina de tipo cliente-servidor. Assim, um cliente executa o operador ***Send*** (para o envio de uma requisição de serviço) seguido, imediatamente, de ***Receive*** (para a recepção da resposta) e um servidor executa o operador *ReceiveAny* (para a recepção de uma requisição de serviço) e *Send* (para o envio da resposta). Dois desses operadores respondem de forma imediata às exigências das comunicações simétricas que caracterizam uma disciplina de tipo produtor-consumidor. Assim, um produtor executa o operador *Send* (para o envio de uma mensagem) e um consumidor executa o operador *Receive* (para a recepção de uma mensagem).

6.3.3 Camada das chamadas do núcleo

Os serviços oferecidos pelo núcleo se referem ao estabelecimento e rompimento de canais da rede de interconexão dinâmica necessárias às comunicações de alto nível e à alocação e liberação de nós de trabalho associadas ao mecanismo de criação dinâmica de processos.

A realização das comunicações de alto nível se apoia sobre os seguintes operadores de conexão e desconexão de canais:

- ***Connect*** **(node)**

Um processo colocado em um nó qualquer, solicita através do operador *Connect,* sua conexão através de um canal direto ao nó *node*.

– ***ConnectAny*** **(node)**

Um processo colocado em um nó qualquer solicita, através do operador *ConnectAny,* sua conexão através de um canal direto a um nó qualquer cuja identificação é devolvida em *node*.

– ***Disconnect*** **(node)**

Um processo colocado em um nó qualquer solicita, através do operador *Disconnect,* a desconexão do canal direto que o conecta ao nó *node*.

Os operadores *Connect*, *ConnectAny* e *Disconnect* são usados pelos operadores da camada das comunicações de alto nível.

Para permitir a criação e a remoção de processos, são necessários os seguintes operadores para a alocação e liberação de nós de trabalho:

– ***Allocate*** **(node)**

Um processo colocado em um nó qualquer solicita, através do operador *Allocate,* a alocação do nó de trabalho *node*.

– ***AllocateAny*** **(node)**

Um processo colocado em um nó qualquer solicita, através do operador *AllocateAny,* a alocação de um nó de trabalho livre qualquer cuja identificação é devolvida em *node*.

– ***Deallocate*** ()

Um processo colocado em um nó qualquer solicita, através do operador *Deallocate,* a liberação do nó de trabalho por ele ocupado.

Os operadores *Allocate*, *AllocateAny* e *Deallocate* são necessários ao mecanismo de criação dinâmica de processos oferecido pela interface do sistema Unix.

Eles são usados somente na implementação da camada das chamadas do sistema.

Os operadores de conexão e desconexão envolvem a aplicação de ações sobre o comutador de conexões. Os operadores de alocação e liberação de nós de trabalho envolvem a manipulação de sinais de controle do barramento de serviço. Como esses dispositivos só podem ser comandados pelo nó de controle, o núcleo do sistema operacional é colocado obrigatoriamente sobre esse nó.

6.3.4 Camada das comunicações de baixo nível

O objetivo da camada das comunicações de baixo nível é de prover serviços de comunicação, que permitem trocas de mensagens compactas, para as chamadas de procedimento remoto através do barramento de serviço.

As comunicações pelo barramento de serviço tem sempre o nó de controle como origem ou destino e, todas elas, se desenvolvem sobre o comando desse nó. Para suportar essas comunicações, operadores diferentes, para o nó de controle e para os nós de trabalho, evidenciam a assimetria no seu comportamento:

- ***W_SendReceive*** **(request, reply)**

Um processo colocado em um nó de trabalho qualquer solicita, através do operador *W_SendReceive,* o envio ao nó de controle de uma mensagem de requisição de serviço *request* e a recepção de uma mensagem de resposta em *reply*.

- ***C_ReceiveAny*** **(node, request)**

O processo colocado no nó de controle solicita, através do operador *C_ReceiveAny,* a recepção de uma mensagem de requisição de serviço em *request* de um nó de trabalho qualquer cuja identificação é devolvida em *node*.

- ***C_Send*** **(node, reply)**

O processo colocado no nó de controle solicita, através do operador *C_Send,* o envio de uma mensagem de resposta *reply* a um nó de trabalho identificado por *node*.

Os operadores são orientados exclusivamente para a disciplina de comunicação de tipo cliente-servidor. Assim, um cliente executa o operador *W_SendReceive* (para o envio de uma requisição de serviço e a recepção da resposta) e um servidor executa o operador *C_ReceiveAny* (para a recepção de uma requisição de serviço) e *C_Send* (para o envio da resposta).

No caso do operador *C_ReceiveAny*, a política de escolha de uma comunicação entre as alternativas possíveis é uma decisão de implementação cuja execução fica a cargo do núcleo do sistema operacional .

6.4 O Servidor CRUX

Em sua primeira versão, o sistema *Crux* considera o Nó // apenas como um *pool* de nós processadores, estando os dispositivos de entrada e saída (teclado, monitor, disco, etc.) e o sistema de arquivos, localizados na Estação de Trabalho. A Estação de Trabalho estará executando o sistema Linux, responsável por prover acesso aos dispositivos e ao sistema de arquivos [MON95].

Os processos *Crux* acessam os dispositivos e arquivos através de chamadas de procedimentos remotos, que fornecem uma interface compatível com a do sistema UNIX, tornando transparente a comunicação com a máquina hospedeira.

A interface entre o Nó // e a Estação de trabalho (Figura 6-4) será realizada por dois processos : o processo representante do **servidor de arquivos** (RSA) e o processo de acesso ao sistema Linux (AL).

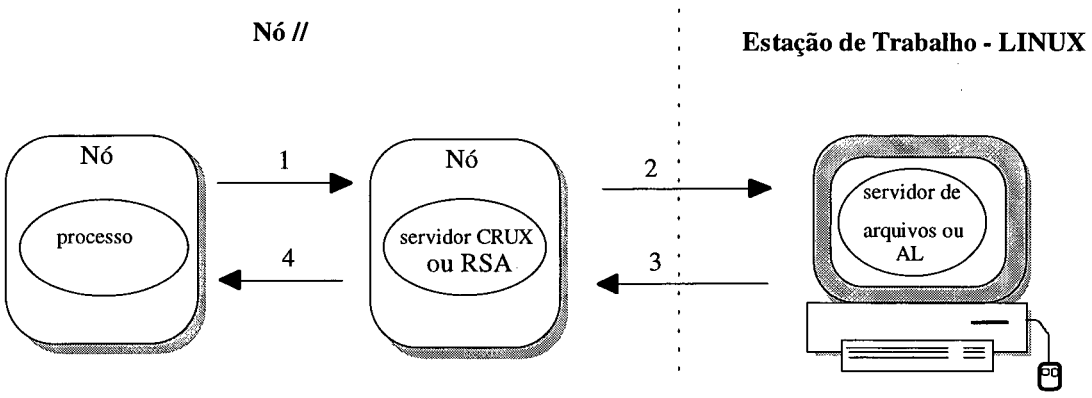

***Figura 6-4*: Utilização de um servidor de arquivos externo ao Nó //.**

- RSA consiste de um processo *Crux*, que está localizado no NCE, e recebe todas as requisições endereçadas ao sistema Linux, enviando-as à máquina hospedeira, pelo canal externo. Do ponto de vista dos processos *Crux*, o RSA constitui um

servidor de arquivos, que executa todos os serviços recebidos, muito embora, na realidade, as requisições sejam passadas à máquina hospedeira para execução.

O RSA executa, basicamente, os seguintes passos :

* Recebe de algum processo *Crux*, uma requisição de serviço endereçada à máquina hospedeira;
* Envia a requisição à máquina hospedeira, através do canal externo;
* Permanece esperando pela resposta do serviço;
* Recebe, pelo canal externo, o resultado do serviço;
* Retorna o resultado do serviço ao processo *Crux* requisitante.

O processo AL consiste de um processo Linux, que está localizado na Estação de trabalho, responsável por processar as requisições enviadas pelo RSA, através da execução da chamada Linux correspondente ao pedido. O AL executa, basicamente, os seguintes passos :

* Recebe do RSA uma requisição de serviço, através do canal externo;
* Analisa a mensagem recebida e processa o pedido através da execução da chamada Linux correspondente;
* Monta uma mensagem de resposta, enviando-a ao RSA através do canal externo.
* Resumindo, os passos de uma chamada ao sistema de arquivos (Figura 6-4) :
* Um processo requisita um serviço ao servidor *Crux* fazendo uma chamada de procedimento remoto (1);
* O servidor constata que a chamada se refere a serviços de arquivo e faz uma chamada de procedimento remoto ao servidor de arquivos (2);
* O servidor de arquivos recebe a mensagem, executa a tarefa e responde ao servidor *Crux* (3);
* Este último recebe a resposta e envia ao processo que requisitou o serviço (4).

CAPITULO 7

7. Sistema de Arquivos PYXIS

7.1 Introdução

O PYXIS [FRÖ 94] tem como base o modelo cliente-servidor. Neste caso, um ambiente próprio à execução do PYXIS pressupõe a existência de um núcleo que suporte o modelo, bem como de servidores específicos para cada tipo de dispositivo periférico presente. No projeto original do PYXIS não estão incluídos o núcleo e os servidores de dispositivos.

Na maioria dos sistemas operacionais, o sistema de arquivos é implementado por um único servidor. No caso do PYXIS, levou-se em conta a complexidade de um sistema de arquivos e, com o propósito de aproveitar o seu paralelismo implícito, decidiu-se subdividi-lo em três servidores:

- Um servidor encarregado de traduzir nomes de arquivos em identificadores internos, implementando as estruturas de diretórios.
- Um servidor responsável pela gerência de memória secundária, implementando o conceito de arquivo e estendendo o mesmo aos dispositivos de entrada e saída.
- Um servidor para armazenar, temporariamente, dados de memória secundária, visando otimizar o acesso a eles.

Uma visão parcial de um sistema computacional executando PYXIS é mostrado na Figura 7-1.

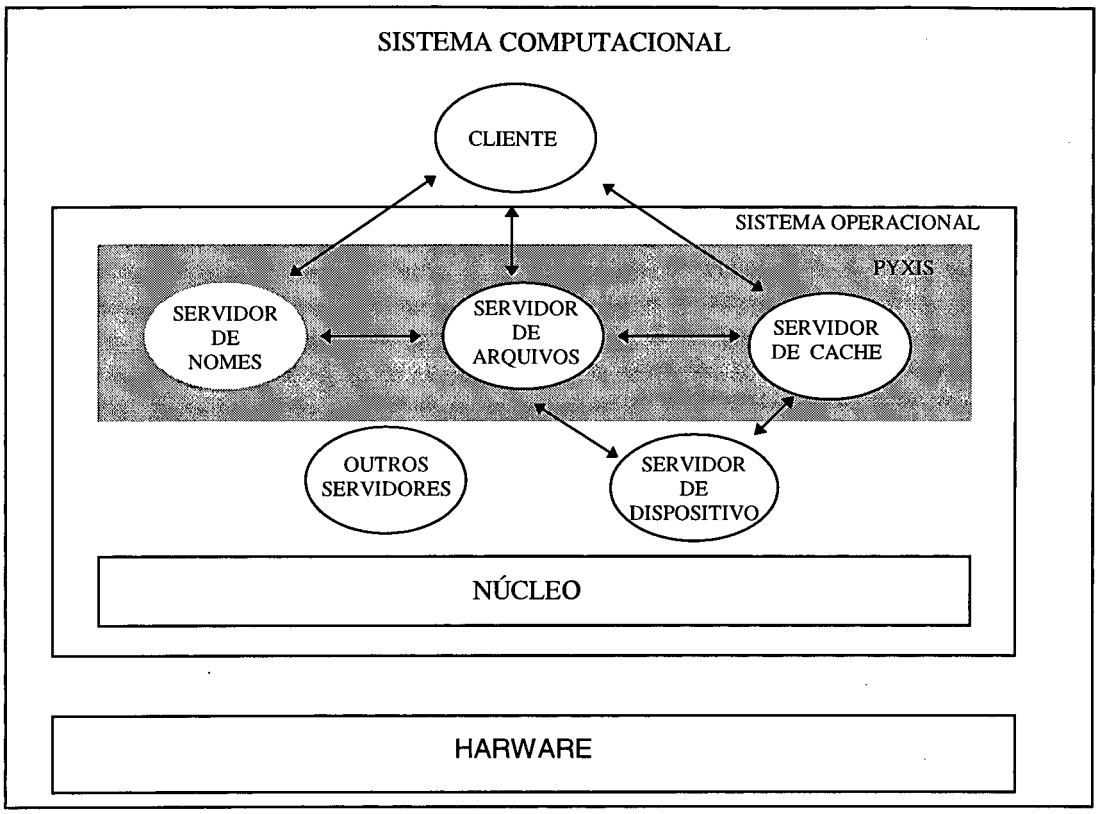

***Figura 7-1*: O PYXIS em um sistema computacional**

7.2 Identificação das Entidades do Sistema

Um sistema de arquivos envolve uma série de entidades que devem ser identificadas, tanto perante o conjunto de usuários quanto perante o sistema de arquivos, como por exemplo, os próprios usuários, os nodos formadores do domínio, os arquivos, entre outras. Para os usuários, tais entidades estão associadas a nomes, porém, o sistema de arquivo faz uso de estruturas internas especiais. A seguir pode-se ver como essas entidades estão definidas no PYXIS.

7.3 Identificação Perante o Usuário

Todas as entidades do sistema de arquivos são referidas pelos usuários através de nomes alfanuméricos quaisquer, contudo, os nomes de nodos e de arquivos seguem convenções específicas.

7.3.1 Nomes de nodos

Um dos objetivos definidos para o projeto é a adequação à execução sobre a INTERNET, portanto, os nomes de nodos de um domínio PYXIS segue a convenção estabelecida pelo Network Information Center da Internet, cujo programa de sintaxe é apresentado na Figura 7-2.

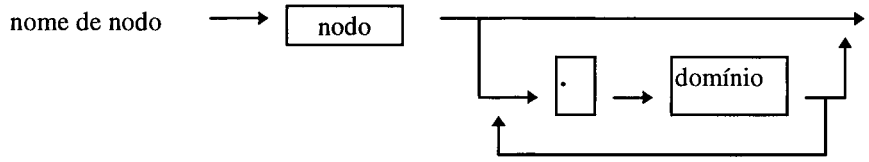

Figura 7-2: **Diagrama de sintaxe de um nome de nodo**

Se o ultimo elemento de um nome de nodo for um dos sub-domínios oficiais da **Internet**, então o nome é absoluto e designa um nodo perante todo o mundo; caso contrário, ele é local ao domínio. informações complementares sobre a convenção de nomenclaturas de nodos na INTERNET podem ser obtidas em [POS82], [MOC 87a] e [MOC 87b].

7.3.2 Nomes de arquivos

A Figura 7-3 mostra, através de um diagrama de sintaxe, a definição de um nome de arquivo

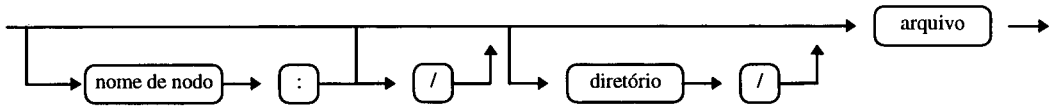

Figura 7-3: **Diagrama de sintaxe de um nome de arquivo**

Se um nome de arquivo começar com um nome de nodo, então o arquivo é remoto, salvo quando especificado o nodo local. se começar com "/", ele é relativo à raiz da árvore de diretório local. Caso contrário ele é relativo ao diretório de trabalho corrente.

7.4 Identificação Perante o Sistema de Arquivo

A fim de identificar as entidades de forma eficiente, o sistema de arquivos faz uso de estruturas especiais, das quais, as mais importantes são descritas a seguir.

7.4.1 Identificadores de Usuários

Usuários são agentes que manipulam os arquivos do sistema, logo, é importante que o sistema os identifique, possibilitando assim a validação de acesso. Um usuário qualquer é identificado perante o sistema por uma estrutura de 64 Figura 7-4) com três campos:

- Identificador de nodo: 32 bits equivalente ao número IP (INTERNET PROTOCOL) da Internet [POS 81a], que identificam um nodo perante o mundo.
- Identificador de grupo: 16 bits que identificam um grupo de usuários em um nodo.
- Identificador local de usuários: 16 bits que identificam um usuário dentro de um determinado grupo.

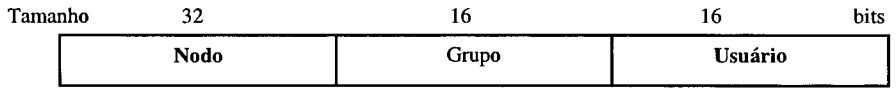

***Figura 7-4*: Identificador de usuário..**

7.4.2 Identificadores de Caixas Postais

As caixas postais utilizadas para a comunicação entre processos são identificadas globalmente por uma estrutura de 48 bits (Figura 7-5) formada por dois campos:

- Identificador de nodo: 32 bits equivalente ao número PI da INTERNET;
- Identificador local de caixa postal: 16 bits que identificam uma caixa postal em um nodo.

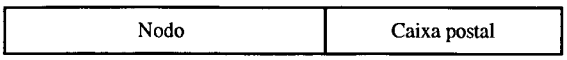

Figura 7-5: **Identificador de caixa postal**

7.4.3 Identificadores de Arquivos

Os identificadores de arquivos adotados pelo PYXIS são estruturas de 96 bits (Figura 7-6) formadas por tres campos descritos como segue:

- Identificador de nodo: 32 bits equivalente ao número PI da Internet;
- Identificador de dispositivo: 32 bits que identificam um dispositivo em um nodo (os 16 bist de mais alta ordem identificam a classe do dispositivo e os 16 restantes a unidade)
- Identificador de arquivo: 32 bits que identificam o descritor de um arquivo em um dispositivo.

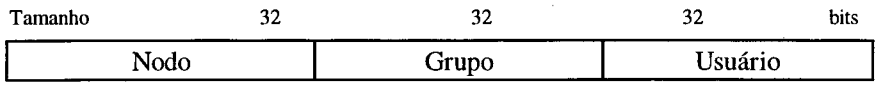

Figura 7-6: **Identificador de arquivo**

7.5 Comunicação e Localização dos Servidores

A comunicação entre clientes e servidores, bem como entre os próprios servidores está baseada no conceito de caixa postal. Quando um processo deseja se comunicar com outros, ele cria uma ou mais caixas postais que serão usadas para receber e enviar mensagens. Desta forma, a entidade identificada na comunicação é a caixa postal e não o processo que a criou.

O uso de caixas postais para a comunicação entre processos permite que se defina um espaço "bem conhecido" de caixas postais, de forma similar aos *Well Known Services [WKS]* da INTERNET [POS 87]. Esse espaço engloba todos os serviços básicos do sistema, de forma que, quando um cliente deseja um serviço, ele conheça a caixa postal correspondente. Da maneira como os *WKS* são implementados na INTERNET, se um dado serviço está disponível em um nodo, então o respectivo servidor também está disponível naquele nodo. Como essa é uma meta que contraria as definidas para o projeto, por não permitir que os servidores do PYXIS executem em nodos distintos, decidiu-se expandir este mecanismo para que uma caixa postal local possa estar associada a um servidor remoto.

Nessa primeira versão do PYSIS, o sistema de comunicação inclui uma tabela de localização com uma entrada para cada "serviço bem conhecido". Cada entrada da tabela identifica até três servidores aptos a executar os serviços associados à caixa postal.

Para proceder a entrega da mensagem, o sistema de comunicação consulta a tabela de localização em busca de um par (nodo,caixa postal) que identifica globalmente um servidor. Caso as três referências sejam nulas, uma mensagem indicando a indisponiblidade do serviço é retornada ao cliente. Caso alguma das entregas sejam válidas, o sistema de comunicação procederá o envio da mensagem à primeira caixa postal. Cada servidor tem associado um tempo máximo de execução do serviço, assim, o sistema de comunicação pode, decorrido o limite de tempo sem que se tenha recebido uma resposta do servidor, proceder ao envio da mensagem ao próximo servidor da lista. Um exemplo de tabela de localização de servidores pode ser visto na Figura 7-7.

	SERVIDOR 0			SERVIDOR 1			SERVIDOR 2		
Serviço	Nodo	CP	Exp	Nodo	CP	Exp	Nodo	CP	Exp
0	local	0	5	local	4	5	-	-	-
1	local	1	5	local	1	50	juno	8	100
2	vega	2	80	-	-	-	-	-	-
:	:	:	.	:	:	.	:	:	:
n	local	n	5	vega	n	50	baco	n	60

CP= Caixa postal *Exp= Tempo de expiração*

Figura 7-7: **Exemplo da tabela de localização dos servidores**

O mecanismo de localização de servidores descrito permite a replicação de servidores idempotentes, bem como a definição de servidores equivalentes, pois uma caixa postal pode ser redirecionada para outra no mesmo nodo. A tabela de localização de servidores pode ser alterada por meio de chamadas ao núcleo do sistema operacional, permitindo que, futuramente, um servidor externo faça reconfigurações dinâmicas na tabela, otimizando o tempo de resposta dos serviços. Na Figura 7-7 o serviço 0 da caixa postal 0 é equivalente ao serviço caixa postal 4; o serviço 1 está, também, disponível nos nodos "vega" e "juno"; o serviço 2 está disponível apenas remotamente no nodo "vega".

A tabela de localização de servidores permite que um cliente localize um servidor de forma transparente, porém, o próprio sistema de arquivo está dividido em mais de um servidor, sendo necessário definir-se um mecanismo para que os servidores se localizem entre si. Como o PYXIS não permite a replicação transparente de arquivos, definiu-se que sempre que um nodo contiver um arquivo, ele conterá também toda a coleção de servidores correspondentes a ele. Desta forma, pode se tirar do próprio identificador do arquivo a informação sobre a localização dos servidores, ou seja, assim que o servidor de nomes obtém o identificador de um arquivo, as mensagens relativas a ele passam a ser enviadas ao nodo que o contém.

Um caso especial ocorre quando cada um dos servidores do PYSIS executa em um processador particular de um multicomputador. Neste caso, o multicomputador é

visto como um único nodo, sendo que o espaço de "serviços bem conhecidos" é distribuído pelos processadores. O núcleo do sistema operacional ou o controlador de comunicações é responsável pela associação de caixas postais a processadores.

Para manter a interface do sistema de comunicação homogênea, todo o endereçamento de mensagem compreende o par (nodo, caixa postal), sendo que quando o campo "nodo" for nulo, o sistema tentará defini-lo através de uma pesquisa na tabela de localização de servidores.

CAPITULO 8

8. Um Sistema de Arquivos para o Nó //

8.1 Introdução

O sistema de arquivos PYXIS [FRO94] foi proposto com a intenção de ser incorporado por um projeto de âmbito maior, como o projeto do Multicomputador Nó// que está em desenvolvimento no CPGCC/UFSC [COR93]. Entretanto, a migração deste sistema para o multicomputador Nó// requer uma série de adaptações e modificações em relação à proposta original.

O trabalho envolve uma etapa de definição de um suporte básico para a migração. Este suporte básico inclui estratégias de comunicação que mais se adaptem a estrutura do Nó//.

8.2 O suporte básico de comunicação do PYXIS

Para que a migração do sistema PXYIS seja bem sucedida, é necessário inicialmente a definição de como alguns conceitos básicos deste sistema serão suportados no ambiente Nó//. A definição envolve o conceito de ***Caixa Postal***, que deve ser implementado no micronúcleo existente.

Um mecanismo de comunicação apropriado para suportar o modelo cliente servidor, são as caixas postais (Figura 8-1). Com este mecanismo, as mensagens são endereçadas a caixas postais e não a processos. Para receber mensagem, um servidor solicita ao núcleo que crie uma caixa postal com um certo endereço. As mensagens que chegam referindo este endereço são colocadas na caixa postal e lá permanecem até que o servidor solicite ao núcleo que retire uma mensagem. Cliente e servidor podem por convenção, definir um domínio de caixas postais, onde cada uma delas está previamente associada a um serviço.

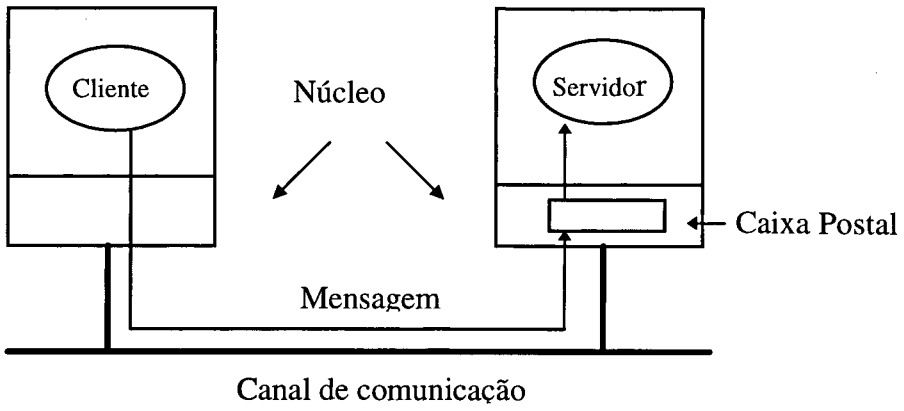

***Figura 8-1:* Comunicação com caixa Postal**

Para implementar as caixas postais, o núcleo mantém um conjunto de *buffers* para armazenar as mensagens e fornecer pelo menos, duas primitivas: "send" e "receive". A primeira envia uma mensagem a uma caixa postal, liberando o processo assim que a mensagem for enviada. A segunda recebe uma mensagem da caixa postal, bloqueando o processo se ela estiver vazia. Caso o núcleo não mais disponha de *buffers* para armazenar uma nova mensagem, ela poderá ser descartada e o mecanismo de confirmação de mensagens deverá detectar a falha.

A implementação de caixas postais dentro do micronúcleo é um enfoque adotado no sistema CHORUS versão 3 [BRI91].

8.2.1 O Micronúcleo

O micronúcleo do CRUX foi estruturado em um esquema de três camadas onde as camadas inferiores fornecem serviços às superiores, através de uma interface bem definida (Figura 8-2 Micronúcleo do CRUX).

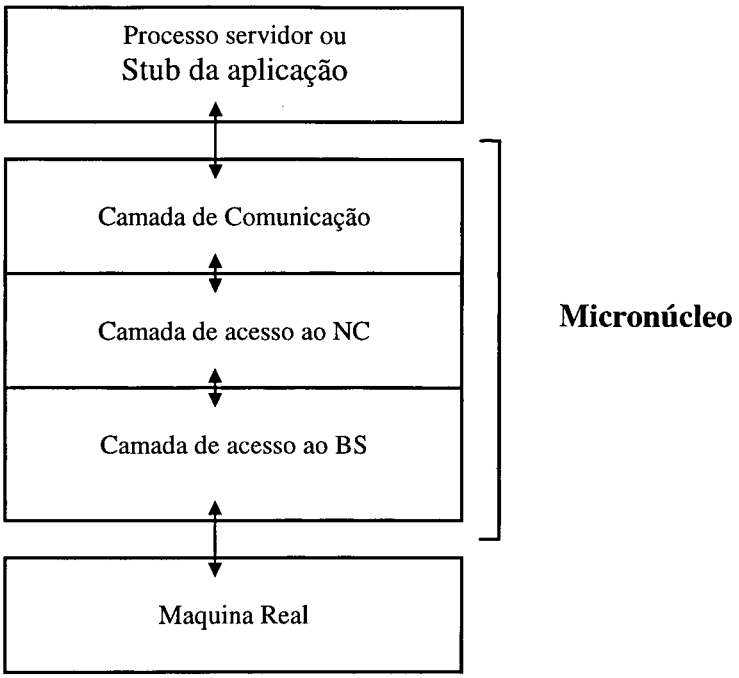

Figura 8-2 Micronúcleo do CRUX

Camada de acesso ao BS

Esta camada provê acesso ao barramento de serviço (BS), permitindo envio e recepção de mensagens ao nó de controle (NC), abstraindo as camadas superiores do controle de interrupções e registradores.

Os serviços oferecidos são adaptados às necessidades da camada superior, que utiliza o **BS** para comunicação com o NC, desenvolvendo uma disciplina cliente-servidor. Este serviços são agrupados em uma única primitiva:

BS_SendRec (request, reply) : Esta primitiva envia, ao NC, uma mensagem *request*, permanecendo bloqueada até que o NC efetue o processamento necessário e envie uma mensagem *reply* de resposta.

Camada de Acesso ao NC

Esta camada provê acesso aos serviços de alocação e conexão de processadores, oferecidos pelo NC, através de requisições de serviços, enviadas na forma de mensagens pelo BS, com formato bem definido, reconhecido pelo NC. Desse modo, esta camada pode ser considerada como um *stub* cliente, sendo a

execução do serviço efetuada remotamente no NC, que retorna, na forma de mensagem, o resultado da operação. O mecanismo descrito caracteriza, então, uma chamada de procedimento remoto.

As primitivas de acesso aos serviços de conexão do NC são:

- *Conect (node) :* Conecta o nó requisitante ao nó de endereço *node.* Se o nó *node* estiver com seu canal ocupado, o nó requisitante ficará bloqueado na primitiva *BS_SendRec*, esperando pelo resultado da operação, que será enviada pelo NC, quando o canal do nó destino estiver desconectado.

- *ConnectAny* : Conecta o nó requisitante a um nó qualquer, cujo endereço é retornado em *node().* Se não houver nó com a intenção de estabelecer conexão com o nó requisitante, este ficará bloqueado na primitiva *BS_SendRec*, esperando pelo resultado da operação, que será enviado pelo NC quando a operação for completada.

- *Disconnect ()* : Desconecta o canal do nó requisitante.

As primitivas de acesso aos serviços de alocação de processadores do NC são

- *Allocate(node) :* Aloca um nó específico no endereço *node.* Se o nó solicitado estiver ocupado, o nó requisitante ficará bloqueado na primitiva *BS_Send* esperando pelo resultado da operação, que será enviado pelo NC quando o nó for libertado.

- *AllocateAny (node)* : Retorna, em *node*, o endereço de um nó desocupado qualquer. Se todos os nós do multicomputador estiverem ocupados, o nó requisitante ficará bloqueado na primitiva BS_SendRec, esperando pelo resultado da operação, que será enviado pelo NC quando algum nó for liberado.

- Deallocate () : Desaloca o nó requisitante.

As primitivas para conexão e desconexão, são utilizadas, na primeira versão do CRUX, somente pela camada de comunicações, não estando disponíveis para acesso direto pelos processos de aplicação. As primitivas de alocação de processadores , porém, são utilizadas diretamente pelas chamadas de criação de processos do UNIX (camada de stubs).

Camada de comunicação

Esta camada implementa os serviços para a comunicação entre NTs, pelos canais ligados ao comutador de conexões, utilizando as primitivas de acesso ao NC para o estabelecimento de conexões físicas entre os processadores.

As primitivas permitem o envio e recepção de mensagens de tamanho variável, de maneira síncrona, com endereçamento direto :

- *Send (node, me ssage, length) :* envia ao NT *node* , a mensagem *messagem* de tamanho *length* . O nó requisitante permanecerá bloqueado até que o NT *node* realize uma chamada para recepção de mensagens.

- *Receive (node, message, length) :* recebe uma mensagem do NT *node*, armazenando-a em *message*, com tamanho l*ength.* O nó requisitante permanecerá bloqueado até o NT *node* lhe envie uma mensagem.

- *ReceiveAny (node, message, length) :* recebe uma mensagem de um nó NT qualquer. O conteúdo da mensagem será armazenado em *message*, o seu tamanho em length e o endereço do NT emissor em *node.* O nó requisitante permanecerá bloqueado até que algum NT envie uma mensagem.

A implementação destas chamadas consiste, basicamente, da seguinte seqüência de passos :

- Pedido de conexão, através das chamadas de acesso ao NC.

- Envio/recepção da mensagem, através dos canais ligados ao comutador de conexões. Objetivando informar o processo receptor do número de *bytes* que devem ser recebidos, para cada mensagem, o micronúcleo do nó emissor adiciona um cabeçalho, contendo o tamanho da mensagem enviada.

- Pedido de conexão.

As chamadas *Receive* e *ReceiveAny* possuem um funcionamento análogo ao da chamada *Send.* Na chamada *ReceiveAny*, porém, o serviço utilizado para conexão é *ConnectAny.*

8.2.2 O Mecanismo de Caixa Postal Proposto

Embora as primitivas de comunicação providas pelo CRUX sejam suficientes para o desenvolvimento de aplicações ***cliente/servidor***, a rigidez inerente ao endereçamento directo torna estas primitivas inadequadas às necessidades dos sistemas modernos, referentes à transparência de localidade e ao atendimento de um serviço por diversos servidores .

Uma solução para este problema consiste em adicionar mecanismos que permitam o endereçamento directo através de caixas postais. Uma caixa postal consiste de um endereço lógico, formado por um número escolhido aleatoriamente, que especifica um serviço oferecido, utilizado pêlos processos servidores para receber requisições de serviços dos processos clientes, não servindo como estrutura para armazenamento intermediário de mensagens.

Nesta primeira versão cada servidor pode possuir apenas uma caixa postal que é levada consigo independentemente de sua localização. Cada caixa postal pode pertencer a diversos servidores, permitindo que um mesmo serviço seja oferecido por mais de um servidor.

Para a implementação desse mecanismo, propomos a construção de um processo Servidor de Caixa Postal (SCP), e de uma camada de caixas postais, localizados acima da camada de comunicação no micronúcleo. A Figura 8-3 mostra a configuração do micronúcleo antes e depois da introdução do SCP.

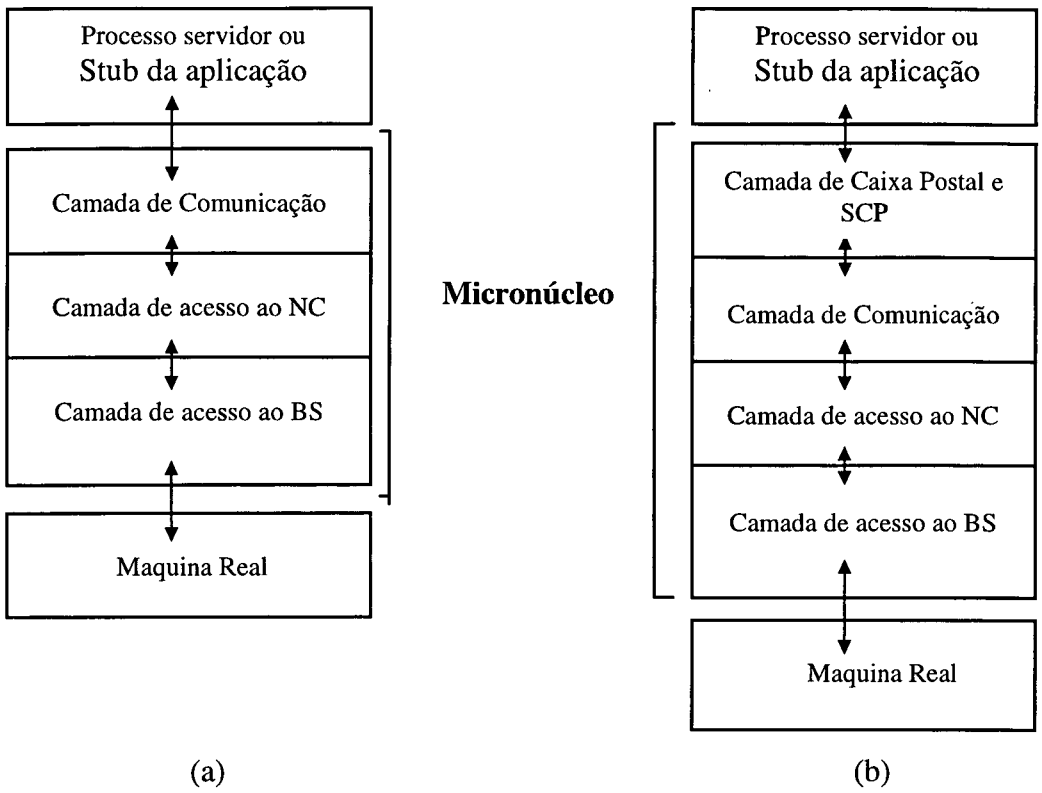

Figura 8-3 : Configuração de um micronúcleo antes (a) e depois (b) da introdução do SCP, e Camada de Caixas Postais.

8.2.3 O Servidor de Caixa Postal

A principal função do servidor de Caixa Postal consiste em manter o mapeamento entre as caixas postaise a localização dos servidores que as atendem

Em relação aos servidores, o SCP proverá serviços para registro e remoção do servidor requisitante a uma determinada caixa postal. A implementação deste esquema implica em manter no SCP uma tabela de endereços, onde cada entrada é formada pelo número do endereço e por uma estrutura que mantém os servidores que atendem essa caixa postal (Figura 8-4).

A implementação do serviço de registro tem os seguintes passos:

- Um processo servidor envia ao SCP uma requisição de registro, contendo o seu *pid* e o número do endereço da caixa postal que deseja utilizar.

- O SCP recebe a requisição e verifica se a caixa postal já existe. Caso afirmativo o *pid* do processo servidor é colocado na lista da caixa postal correspondente. Do contrário, o SCP aloca uma entrada na tabela, armazenando o número da caixa postal e colocando o pid do processo servidor na lista.

- Retorna o resultado da operação ao processo servidor.

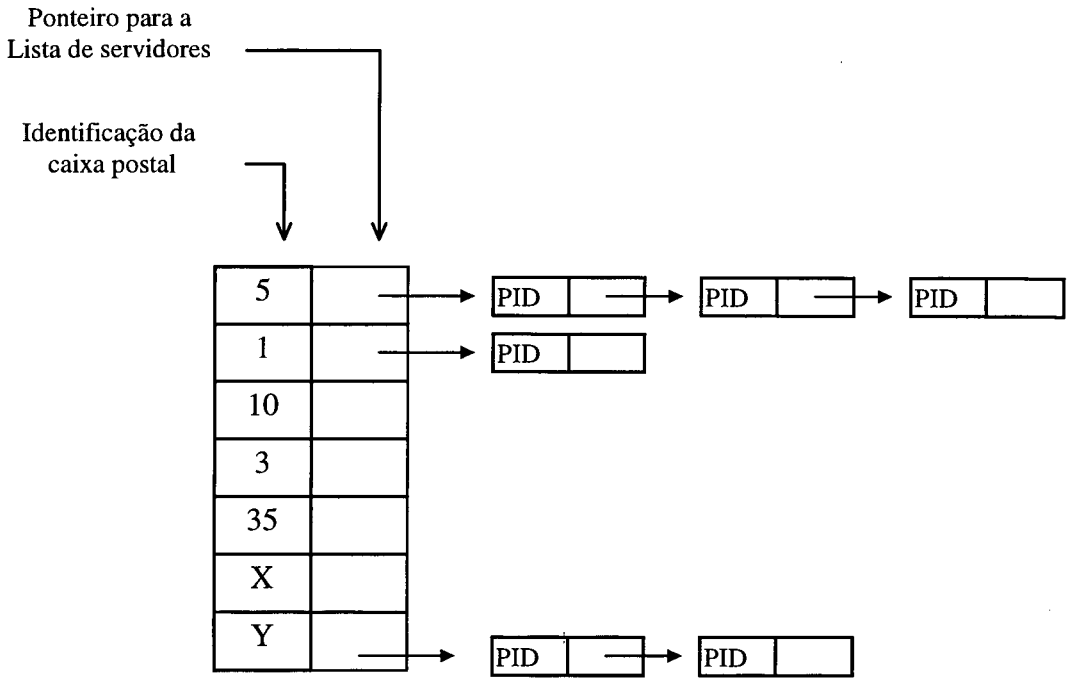

Figura 8-4 : Tabela de caixas postais mantidas pelo SCP

Do ponto de vista dos processos clientes, o SCP proverá serviços para localização de um servidor ligado a determinada caixa postal. A localização será retornada em forma de *pid*, permitindo ao cliente a comunicação com o servidor através da camada de comunicação. Se houver mais de um servidor ligado à caixa postal requisitada, o SCP escolherá o que teve menos requisições.

Camada de Caixa Postais

Os serviços oferecidos por esta camada podem ser divididos em dois conjuntos: serviços de acesso ao SCP e serviços de comunicação por caixa postal.

O primeiro conjunto provê rotinas de acesso aos serviços do SCP, utilizando os mecanismos de comunicação oferecidos pela camada de comunicação entre processos. As rotinas são:

- *Register (box)* : Informa ao SCP que o processo chamador deseja atender a caixa postal *box.*

- *Unregister ()* : informa ao SCP que o processo chamador não mais atenderá a caixa postal que atendia até então.

- *GetServ (box) :* Requisita ao SCP o *pid* de um dos processos servidores que esta registrado na caixa postal *box.*

O segundo conjunto provê os serviços efetivos para transmissão de mensagens através de caixas postais. Os serviços oferecidos são orientado à disciplina cliente-servidor :

- *SendRec (box,, request, len_req, reply, len_reply) :* Esta primitiva é utilizada por processos clientes, que desejam algum serviço de um processo servidor,. Sua função é enviar a mensagem *request*, de tamanho *len_req*, a um dos processos servidores que esteja esperando na caixa postal *box.* O processo é bloqueado até que o processo servidor envie a resposta do serviço solicitado, que será armazenada em *reply*, com tamanho *len_rep.*

- Receive (request, len_req) : esta primitiva é utilizada pelo processo servidor para receber, através da caixa postal em que está registrado, requisições de serviços enviadas pelos processos clientes. O processo servidor permanece bloqueado até que uma mensagem seja recebida e armazenada em *request.*

- *Reply (reply, len_rep)* : Esta primitiva é utilizada pelo processo servidor para devolver ao processo cliente a resposta a um serviço requerido.

A implementação da chamada *SendRec* implica nos seguintes passos :

* Requisita, através da rotina *GetServer*, o *pid* de um processo servidor, registrado na caixa postal *box.*

* Através da chamada *SendP*, localizada na camada de comunicação entre processos, envia uma mensagem *request* ao processo servidor cujo *pid* foi retornado em *GetServer.*

* Através da chamada *ReceiveP*, permanece esperando pela resposta do servidor, armazenando o resultado em *reply* e *len_rep*.

A implementação da chamada *Receive* implica nos seguintes passos :

♦ Através da chamada *ReceiveAnyP*, permanece esperando por uma requisição de um processo cliente qualquer, armazenando o resultado em *request* e *len_req*.

♦ Armazena o *pid* do processo que requisitou o serviço, retornado pela chamada *ReceiveAnyP*, para que a chamada *Reply* retorne a resposta da requisição.

A implementação da chamada Reply consiste em enviar uma mensagem reply, através da chamada SendP, ao processo cliente cujo pid foi armazenado pela chamada Receive.

Objetivando minimizar o número de acessos ao SCP, a chamada *SendRec* manterá, localmente ao nó do processo cliente, um mapeamento entre a caixa postal requisitada e o pid retornado pela chamada *GetServer*. Desse modo repetidos acessos a um mesmo servidor poderão ser realizados sem exigir comunicação com o SCP, considerando que, a partir do primeiro acesso, a localização do processo servidor destino ficará disponível localmente

Esta otimização, porém, traz problemas em relação à atualização das informações mantidas nos processos cliente. No caso de um processo servidor realizar uma chamada *Unregister*, este fato deve ser informado a todos os processos clientes que mantém a localização deste servidor.

Uma solução para esse problema seria o processo servidor difundir a cada chamada *Unregister*, uma mensagem a todos os NTs que, desse modo, atualizariam as informações. Em nosso modelo, porém, as primitivas para difusão de mensagens não estão previstas.

Outra alternativa seria não utilizar as informações dos processos clientes, deixando a própria chamada *SendRec* descobrir a invalidade da informação e realizar uma chamada *GetServer* para localizar outro servidor que esteja atendendo àquela caixa postal.

A detecção da invalidade da informação depende do estado em que o processo servidor se encontra.

•*O processo servidor está esperando por mensagens em outra caixa postal.*

Neste caso o processo cliente seria conectado ao processo servidor através de um protocolo muito simples o processo servidor avisaria que não mais atende àquela caixa postal.

•*O processo servidor já terminou ou está esperando por conexões .*

Na atual implementação das chamadas de comunicação do micronúcleo, este caso levaria o processo cliente a permanecer bloqueado indeterminadamente, esperando por uma conexão. A solução para este impasse implicaria em criar uma primitiva para conexão na qual, se ultrapassado um determinado intervalo de tempo, o processo chamador seria desbloqueado, sendo retornado um código de erro.

A partir destas modificações acreditamos que o PYXIS poderá ser migrado para o Nó // de forma natural, mantendo um mecanismo de comunicação homogêneo

8.3 Migração do PYXIS para o Nó//

Para análise da distribuição do PYXIS em um multicomputador, vamos considerar que cada servidor executa em um processador distinto, como na Figura 8-5. Um multicomputador é visto como um único nodo. Desta forma qualquer solicitação de serviço será local, porém, envolvendo a troca de mensagens entre os processadores.

A principal alteração no atual sistema de arquivos diz respeito exatamente aos mecanismos de intercomunicação entre processos (IPC). Todo o sistema PYXIS deverá utilizar como forma de comunicação entre servidores, as primitivas de comunicação disponíveis para o Nó //. Entretanto como o Nó // não tem primitivas de comunicação do tipo caixa postal, é necessária a definição destas primitivas .

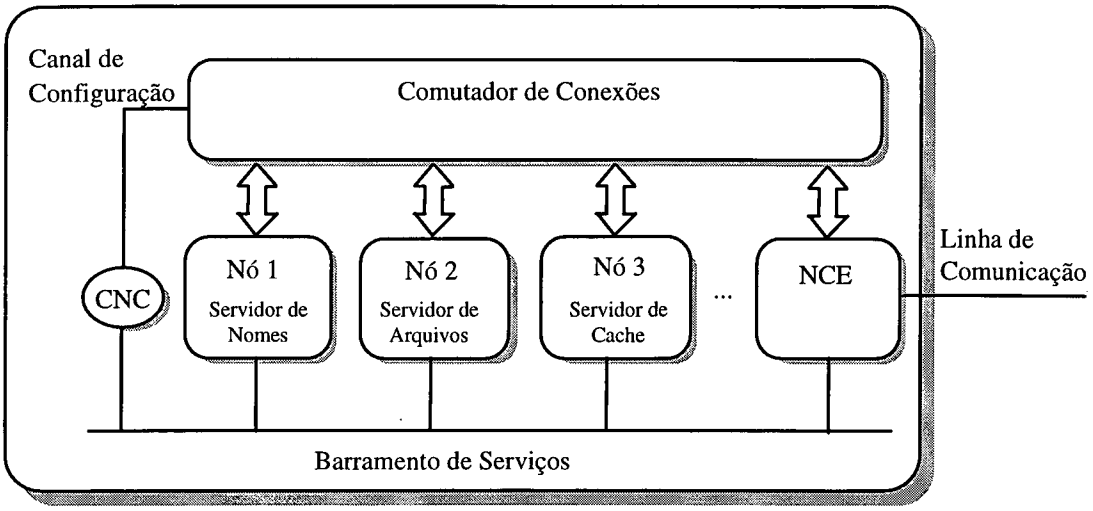

***Figura 8-5*: O PYXIS em um Multicomputador**

A versão atual do CRUX considera o Nó // apenas como um *pool* de processadores, estando os dispositivos de entrada e saída (teclado, monitor, disco, etc.) e o sistema de arquivos, localizadas na estação de trabalho.

As chamadas padrão, de sistemas de arquivos, *open, close, lseek, read e write*, são tratadas pelo servidor CRUX, que, para tal, se comporta como um servidor de arquivos, repassando as requisições de procedimentos remotos para a estação de trabalho, pegando a resposta da máquina hospedeira e repassando para o processo requisitante.

Com a implementação do sistema de arquivos no Nó //, as chamadas requisitadas a este sistema serão tratadas como chamadas locais, sendo resolvidas no próprio nó de trabalho, não sendo necessário acesso a servidores do sistema.

As chamadas a outros dispositivos de entrada e saída continuarão na estação de trabalho e seguirão o mesmo esquema de comunicação, através das chamadas de procedimento remoto.

Desta forma o funcionamento do Sistema de Arquivos teria as seguintes etapas.

♦ Carga e Inicialização do S. A.

Para melhor entendimento primeiro veremos rapidamente o estado do Nó // após sua inicialização. O processo de inicialização do Nó // é suportado por um conjunto de EPROMs com programas especiais. Cada nó possui uma EPROM, cujo código será executado no momento da inicialização de cada processador, sendo o responsável pela carga do nó.

O nó NCE é o responsável por receber todo o código de inicialização do Nó //. Este código é recebido por um canal de comunicação externo, ligado à uma estação hospedeira (Figura 2-1). O código da EPROM do nó NCE segue o seguinte algoritmo.

* Espera pelo canal externo o tamanho do programa a ser carregado no NCE.
* Recebe o programa, carregando-o em determinada posição de memória.
* Executa o programa recebido.

Todos os NTs possuem a mesma EPROM, que executa o seguinte algoritmo:

* Espera pelo canal ligado ao *crossbar* o tamanho do programa a ser carregado.
* Recebe o programa, carregado-o em determinada posição da memória.
* Executa o programa recebido.

NC é equipado com uma EPROM, que contém todo o código necessário para o seu funcionamento (serviços de conexão e alocação). Desse modo, o NC executa um código imutável, não necessitando de carga através dos canais ou de outros dispositivos.

estado do Nó // após a etapa de inicialização é o seguinte:

* O nó NCE espera, pelo canal externo, o programa para carregar no sistema.
* O NC espera, pelos BS, pedidos de conexão ou alocação.
* Os NTs esperam, pelos canais ligados ao crossbar, os programas de sistema e das aplicações.

Então de forma bem simplificada a carga do Sistema de arquivos, se dá através da máquina hospedeira, que envia uma chamada de procedimento remoto ao NCE, este recebe um a um os servidores para serem carregados no sistema. O NCE faz um pedido de alocação ao NC pelo barramento de serviço , e este por sua vez verifica na sua tabela descritora qual NT que se encontra ocioso, manda o tamanho e o arquivo que deseja alocar. O NT recebe o tamanho do servidor e carrega-o num espaço determinado de memória, e o executa .

♦ Os serviços dos servidores do Sistema de Arquivo são:

* Um servidor encarregado de traduzir nomes de arquivos em identificadores internos (servidor de nomes).

* Um servidor responsável pela gerência de memória secundária, implementando o conceito de arquivo (servidor de arquivos).

* Um servidor para armazenar, temporariamente, dados de memória secundária, visando otimizar o acesso a eles (servidor de cache).

Um exemplo de serviço, que podemos considerar é a obtenção de um descritor de arquivos:

* Primeiro o processo usuário (cliente) envia uma mensagem através do canal externo para o servidor de nomes solicitando a tradução de um nome para um identificador interno.

* O NCE retira da sua caixa postal a mensagem, converte ela numa mensagem local e a manda para o NC através do BS.

* O NC retira da sua caixa postal a mensagem e identifica e repassa a mesma para a caixa postal do servidor de nomes.

* O servidor de nomes retira a mensagem de sua caixa postal, pede conexão ao NC pelo BS com o servidor de arquivos, para efetuar a tradução, o servidor de nomes solicita a leitura de diretórios que são implementados como arquivos. Estabelecida a conexão, manda a mensagem.

* O servidor de arquivos retira a mensagem de sua caixa postal e executa a tarefa, retorna a resposta com uma mensagem para a caixa postal do servidor de nomes.

* Ambos servidores pedem a desconexão ao NC, e o servidor de nomes manda a mensagem de resposta do serviço solicitado que faz o caminho inverso até o usuário (cliente).

9. CONCLUSÃO

Neste trabalho, foi apresentada uma proposta de adaptação de um sistema de arquivos ao ambiente Nó //. Para isto, foi necessária a definição de mecanismos de comunicação mais apropriados para o tipo de arquitetura considerado.

A partir da definição e implementação do suporte necessário a migração do sistema de arquivos para o Nó // será possível a implementação definitiva de um sistema de arquivos para o Nó //, de forma distribuída.

O projeto do mecanismo de Caixas Postais foi escolhido principalmente porque o Sistema de Arquivos PYXIS é suportado pelo conceito de caixas postais, que é um mecanismo adequado para suportar o modelo cliente/servidor.

Através do mecanismo de Caixas Postais proposto, contatou-se a possibilidade de construção de diferentes mecanismos de comunicação fundamentando-se em comunicação sícrona com endereçamento direto.

A implementação completa do trabalho proposto, ou seja, a possibilidade de execução do sistema de arquivos PYXIS no ambiente Nó// nos permite avançar em considerações importantes:

- Comprovar a viabilidade de reconfigurações dinâmicas do multicomputador, como migrações de processos, no que diz respeito ao Sistema de Arquivos, redefinindo-se as tabelas de localização de servidores em cada um dos processadores ou no processador de controle.
- Possibilidade de replicação dinâmica dos servidores do PYXIS. O sistema de comunicação do multicomputador poderá ser programado para disparar, automaticamente, réplicas de servidores sobrecarregados e eliminá-las quando ociosas por um determinado tempo.

Para a continuidade e melhoramento deste trabalho, podemos sugerir:

- A adoção de threads se constitui numa proposta bem interessante. Durante o projeto de Caixas postais percebeu-se que a utilização de threads facilitaria muito a programação de eventos paralelos, abstraindo-os como eventos

puramente seqüenciais. Sem duvida, uma ótima opção seria a implementação threads, complementada com um esquema de endereçamento direto básico através de caixas postais.

- As chamadas padrão de sistemas de arquivos como, por exemplo, open, close, lseek, read e write, são tratadas pelo servidor CRUX que se comporta como um servidor de arquivos, uma vez migrado o Sistema de Arquivos PYXIS, estas chamadas ao servidor de arquivos terão que ser reconhecidas e enviadas ao PYXIS, para isto terão que ser reconfiguradas as tabelas de localização do NCE e do NC.
- Implementação do projeto de comunicação de Caixas Postais, proposto, como mecanismo básico de comunicação no ambiente Nó// elaborado neste trabalho.

10. Referências Bibliográficas

[ACC86] Accetta, M. et alli, Mach: a New Kernel Foundatipon for UNIX Developement, Pittsburgh: Proccedings of the summer 1986 USENIX Conference, p. 93-112, julho de 1986.

[AUS91] Austin, P. & Murray, K. & Wellings, K., *The Design of an Operating System for a Scalable Parallel Computing Engine*, Software - Practice and Experience, vol. 21, p. 989-1013, outubro de 1991.

[BRI91] Bricker, A & et al., Arquitectural Issues in Microkernel-based Operating Systems: The CHORUS experience, Computer Comunications, vol. 14, n. 6, p. 347-357, agosto de 1991.

[CAM95] Campos, R. A., *Um Sistema Operacional Fundamentado no Modelo Cliente-Servidor e Um Simulador Multiprogramado para Multicomputador*, Dissertação de Mestrado, CPGCC-UFSC, Florianópolis, 1995.

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

[COR93] Corso, T. B., *Ambiente para Programação Paralela em Multicomputador*, Depto de Informática e Estatística, UFSC – Florianópolis, Relatório Técnico, 1993.

[COU88] Coulouris, G. F. & Dollimore, J., *Distributed Systems: Concepts and Design*, Addinson-Wesley, 1988.

[DUN86] Ducan, R. Advanced MS-DOS, Microsoft Pres, 1986.

[FEN81] Feng, T., *A Survey of Interconnection Networks*, Computer, p. 12-27, dezembro de 1981.

[FRÖ94] Fröhlich, A. A., *PYXIS – Um Sistema de Arquivos Distribuído*, Dissertação de Mestrado, CPGCC-UFSC, Florianópolis, 1994.

[HWA93] Hwang, K, *Advanced Computer Architeture*, McGraw-Hill, 1993.

[KIR94] Kirck, O. *The Linux Network Administrator's Guide*, 1994.

[MER95] Merkle, C. & Boing, H., *Simulação do Nó //,* Depto de Informática e Estatística, UFSC – Florianópolis, Relatório Técnico a ser publicado, 1995.

[MOC87a] Mockapetris, P., Domain Names - Concept and Facilities, Menlo Park: Network Information Center, 1987 (RFC 1034).

[MOC87b] Mockapetris, P., Domain Names - Implementation and Specification, Menlo Park: Information Center, 1987 (RFC 1035)

[MUL90] Mulleder, S., et alli, Amoeba: a Distributed Systems for the 1990s, IEEE Computer, p. 44-53, maio de 1990.

[POS81a] Postel, J., Internete Protocol, Marina del Rey: USC, 1981 (RFC 791)

[POS82] Postel, J. & SU, Z., The Domain Naming Convention for Internet User Aplications, Menlo Park: Network Information Center, 1982(RFC819).

[POS87] Postel, J., & Reynolds, J., Official Internet Protocols, Marina del Rey: USC, 1987 (RFC 1011).

[REE87] Reed, D. A. & Fujimoto, R. M., *Multicomputer Networks: Message-Based Parallel Processing*, MIT Press, 1994.

[TAN92] Tanenbaum A. S., *Modern Operating Systems*, Prentice-Hall, 1992.

[BRO83] Broomell, George, Heath, J. Robert, *Classification Categories and Historical Development of Circuit Switching Topologies.* Computing Surveys, Vol. 15, No. 2, junho de 1983.

[HAN94a] Brinch Hansen, P., *SuperPascal - a publication language for parallel scientific computing.* Concurrency - Practice and Experience, vol. 6(5), p. 461-483, agosto de 1994.

[HWA87] Hwang, Kai, *Advanced Parallel Processing with Supercomputer Architectures.* Proceedings of the IEEE, Vol. 75, No. 10, outubro de 1987.

[INM84] INMOS Limited. *Occam Programming Manual.* London : Prentice-Hall, 1984.

[MON95] Montez, C. B., *Um sistema operacional com micronúcleo distribuído e um simulador multiprogramado de multicomputador.* Dissertação de Mestrado, CPGCC-UFSC, Florianópolis, maio de 1995.

[VAR87] Varma A., Sawchuk, A. A., Jenkins, B. K., Raghavendra, C. S., *Optical Crossbar Networks.* Computer, 0018-9162/87/0600-0050, IEEE, junho de 1987.