* RELATÓRIO TÉCNICO *

MULPLIX: UM SISTEMA OPERACIONAL TIPO UNIX PARA O MULTIPROCESSADOR MULTIPLUS

Gustavo Peixoto de Azevedo
Rafael Peixoto de Azevedo
Norival Figueira Ribeiro
Júlio Salek Aude

NCE 01/91

Universidade Federal do Rio de Janeiro
Núcleo de Computação Eletrônica
Caixa Postal 2324
20001 - Rio de Janeiro - RJ
BRASIL

MULPLIX: Um Sistema Operacional tipo UNIX para o Multiprocessador MULTIPLUS *

Gustavo Peixoto de Azevedo
Rafael Peixoto de Azevedo
Norival Figueira Ribeiro
Julio Salek Aude †

8 de janeiro de 1991

*Esta é uma versão revista e ampliada, pelos dois primeiros autores, do artigo apresentado no III Simpósio Brasileiro de Arquitetura de Computadores – Processamento Paralelo, realizado no Rio de Janeiro em novembro de 1990.

†Os autores agradecem ao CNPq e à FINEP o apoio ao desenvolvimento deste projeto.

MULPLIX:
Um Sistema Operacional tipo UNIX para o Multiprocessador MULTIPLUS

Resumo

Este trabalho é um relato do estágio atual de desenvolvimento do MULPLIX, sistema operacional que está sendo projetado para atuar no MULTIPLUS, um multiprocessador científico de alto desempenho em desenvolvimento no NCE/UFRJ.

Na sua versão inicial, o MULPLIX será resultado de extensões no PLURIX visando adequá-lo à arquitetura do MULTIPLUS a aos requisitos mínimos de aplicações científicas paralelizáveis.

As principais extensões se referem a possibilidade de criação de processos leves, modificação das políticas de escalação e gerência de memória, colocação de primitivas de sincronização disponíveis para o usuário e implementação mais eficiente das primitivas de sincronização do tipo espera ocupada.

Abstract

This paper describes the current development state of MULPLIX, an operating system which is being designed for MULTIPLUS, a high-performance scientific multiprocessor computer under development at NCE/UFRJ.

In its initial version, MULPLIX will be a result of extensions to PLURIX aiming at the adaption of PLURIX to MULTIPLUS architecture and the requirements of parallel scientific applications.

The main extensions are related to the possibility of creation of light-weight processes, changes in the scheduling and memory management policies, availability of synchronization primitives for the user and a more efficient implementation of busy waiting synchronization primitives.

Conteúdo

1 Introdução

Ao longo de sua história, o Núcleo de Computação Eletrônica da UFRJ (NCE/UFRJ) tem construído vários sistemas computacionais, desenvolvendo tanto o hardware, quanto o software. Destacam-se os projetos do COPPEFOR (monitor/compilador FORTRAN para o IBM-1130), do PPF (processador de ponto flutuante) para o IBM-1130, do POTI (microcomputador de 8 bits) e seu sistema operacional SOCO, da CPU compatível com a família PDP 11/XX da DIGITAL, do multimicroprocessador PEGASUS 32X e do sistema operacional PLURIX.

1.1 O Projeto MULTIPLUS

O projeto **MULTIPLUS** [3], atualmente em desenvolvimento no NCE/UFRJ, visa a concepção e construção de uma família de computadores paralelos de alto desempenho para aplicações científicas e de engenharia. A sua realização demanda pesquisa e desenvolvimento nas áreas de arquitetura de computadores, sistemas operacionais, microeletrônica e algoritmos paralelos.

O sistema operacional **MULPLIX** está sendo projetado para atuar no MULTIPLUS. Ele é uma evolução do **PLURIX** [6] visando propiciar ao usuário um ambiente para desenvolvimento e execução de aplicações intensamente paralelas. A sua compatibilidade com o PLURIX garante acesso a todo o conjunto de utilitários e aplicativos já desenvolvidos e/ou transportados para o PLURIX.

A arquitetura do MULTIPLUS está estruturada como um conjunto de **clusters** interligados através de uma *Rede de Interconexão* (**RI**) multiestágio do tipo n-cubo invertido. Cada cluster é normalmente composto de um número entre 1 e 8 de *Nós de Processamento* (**NP**), um *Processador de Entrada e Saída* orientado a *Blocos* (**PESB**), um *Processador de Entrada e Saída* orientado a *Caracteres* (**PESC**). e uma *Interface de Rede* (**IR**). Os NP, os PES e a IR são interligados através de um **barramento duplo**, composto de um barramento de dados e de um barramento de instruções

com 64 bits de largura. Ao todo o MULTIPLUS é capaz de suportar até 2048 NP.

Cada NP está equipado com um microprocessador **SPARC** [14], um co-processador de ponto flutuante, um módulo de memória com até 32 Mbytes, duas unidades de memória cache (uma para dados e a outra para instruções) de 64 Kbytes e hardware de suporte à gerência de memória.

A **memória do sistema é global** apesar de estar fisicamente distribuída pelos NP. Isto significa que os NP podem acessar transparentemente tanto o seu próprio módulo de memória, quanto os módulos de memória localizados em outros NP.

1.2 Os Sistemas Operacionais PLURIX e MULPLIX

O PLURIX é um Sistema Operacional com Filosofia Unix (SOFIX), inteiramente desenvolvido no NCE/UFRJ a partir de 1982. O PLURIX é capaz de controlar simetricamente computadores multiprocessados. Esta característica exige que o núcleo do PLURIX seja um programa paralelo e que a sua construção seja baseada em técnicas mais sofisticadas, normalmente não empregadas em sistemas monoprocessados.

A capacidade de multiprocessamento do PLURIX é transparente aos usuários, ou seja, o PLURIX oferece ao programador um ambiente multiusuário essencialmente igual ao UNIX. Entretanto, é possível acelerar até certo ponto determinadas tarefas, desde que elas possam ser decompostas em subtarefas de tamanho razoável e de interação bastante simples.

O utilitário make do PLURIX [4] é capaz de identificar e comandar operações que podem ser realizadas concorrentemente. Em computadores contando com mais de um processador esta concorrência se transforma em paralelismo. No caso da utilização típica do PLURIX em ambiente universitário - desenvolvimento de software - o exemplo mais claro de uso destas facilidades refere-se à aceleração da compilação de programas com fontes organizados em vários módulos. Esta tarefa envolve a compilação (concorrente) dos módulos cujos objetos são a seguir ligados. A compilação de cada módulo pode, por sua vez, ser composta por três fases (concorrentes)

interligadas por um *"pipe"*: pré-processamento, compilação propriamente dita e montagem.

O exemplo dado no parágrafo anterior sugere uma possibilidade de grande aumento da velocidade de processamento. Uma análise mais aprofundada, entretanto, revela que no caso mais otimista (compilação de muitos módulos e disponibilidade de muitos processadores) o tempo total de compilação tem um limite inferior dado pelo tempo de ligação dos módulos objetos, porque este processo é realizado seqüencialmente.

O grau de paralelismo que pode ser explorado pelo PLURIX é adequado à arquitetura multiprocessadora de pequena escala do PEGASUS [7]. Em casos específicos, ele proporciona ganhos de velocidade compatíveis com um pequeno número de processadores (até cerca de uma dezena). No caso mais geral, entretanto, os processadores são vistos no PLURIX como mais um recurso disponível para compartilhamento entre aplicações. Assim, a vantagem principal obtida através do uso de um número maior de processadores não é acelerar o processamento de uma aplicação, mas sim atender satisfatoriamente a um número maior de aplicações ou usuários.

Em termos de desempenho, o objetivo fundamental do MULTIPLUS e, conseqüentemente, do MULPLIX é viabilizar, através da exploração intensa de paralelismo, aplicações científicas e de engenharia que demandam uma enorme quantidade de processamento. Isto significa acelerar estas aplicações por um fator de dezenas ou centenas de vezes.

A maior parte das máquinas para processamento paralelo tem um escopo de utilização limitado devido a dificuldade para programá-las. Esta dificuldade decorre das limitações ou das particularidades dos modelos de programação paralela presentes nestas máquinas. Assim, normalmente as aplicações são programadas especificamente para uma máquina particular, com uma configuração de hardware pré-estabelecida.

Em relação à interação com usuários, o MULTIPLUS tem por objetivo se diferenciar das demais máquinas por prover um ambiente de programação mais genérico e confortável. Esta generalidade envolve uma maior independência do programador em relação à configuração de hardware disponível para utilização. Um nível mínimo de conforto já é atingido pela

disponibilidade de um SOFIX, com um ambiente para desenvolvimento de software ao qual a maioria dos programadores sofisticados está habituado, e pela existência de um mesmo ambiente para execução e desenvolvimento de aplicações paralelas.

Considerando os objetivos particulares de cada projeto e as diferenças de arquitetura entre as máquinas base para desenvolvimento (MULTIPLUS e PEGASUS), o MULPLIX representará em alguns aspectos uma evolução técnica do PLURIX. Deste modo, o MULPLIX será um ambiente mais completo para o desenvolvimento e execução de programas paralelos, utilizará técnicas mais sofisticadas para gerência de memória e escalação de processos e tornará disponíveis para o usuário mecanismos de sincronização. A definição de um sistema operacional como o MULPLIX ainda é objeto de pesquisa a nível internacional. Por isso, a melhor estratégia para desenvolvimento do MULPLIX é realizá-lo em etapas; a experiência de uso do sistema com certeza influenciará fortemente o seu próprio desenvolvimento.

A versão inicial do MULPLIX tem o objetivo principal de minimizar o tempo necessário a que o MULTIPLUS se torne operacional. Por operacional se entende que o MULTIPLUS possa ser utilizado como uma máquina SOFIX e possa ser demonstrado através da execução de aplicações paralelas científicas e/ou de engenharia. Assim a versão 0.0 do MULPLIX é essencialmente uma adaptação do PLURIX, contendo modificações decorrentes da mudança de hardware e do suporte à execução de programas paralelos.

1.3 Sumário deste Artigo

O objetivo principal deste artigo é divulgar o estado atual de desenvolvimento do MULPLIX. As seções a seguir apresentam as extensões e adaptações que serão realizadas no PLURIX para a preparação da versão 0.0 do MULPLIX. A seção 2 descreve o modelo de execução concorrente do PLURIX e apresenta as modificações necessárias à programação paralela. A seção 3 aborda as alterações sobre a gerência de memória do PLURIX decorrentes do novo conceito de processo e da arquitetura do MULTIPLUS. A seção 4 discute a necessidade da sincronização, analisa os métodos de sincronização no UNIX e no PLURIX e apresenta os mecanis-

mos disponíveis no MULPLIX. A seção 5 apresenta os diferentes objetivos e métodos para escalação em sistemas de tempo compartilhado, como o PLURIX. e em sistemas para programação paralela, como o MULPLIX. A seção 6 aborda as alterações previstas para o sistema de entrada e saída de memória de massa. Na seção 7 apresentamos o estágio atual de desenvolvimento e as perspectivas futuras para o MULPLIX.

Os apêndices complementam este trabalho, enfocando aspectos da implementação na arquitetura do MULTIPLUS das idéias propostas nas seções de 2 a 6. O apêndice A apresenta e discute a implementação dos semáforos binários. O apêndice B descreve a implementação das instruções atômicas FAI ("*Fetch and Increment*") e TAS ("*Test and Set*"), que não estão definidas na arquitetura SPARC.

2 Modelos de Execução Concorrente

Uma aplicação paralela se distingue de uma aplicação seqüencial pela sua estruturação em **unidades de execução concorrente**. Unidades de execução concorrente têm a capacidade de serem executadas simultaneamente. O **paralelismo** ocorre quando há mais de um processador e esta capacidade é exercida.

O desempenho de uma aplicação paralela depende fundamentalmente da qualidade da sua concepção e da adequação e eficiência do ambiente de execução provido pelo sistema operacional.

A estruturação em unidades concorrentes de uma aplicação paralela bem concebida atende aos seguintes objetivos:

1. Ser função da natureza do problema a ser resolvido e dos seus dados de entrada. Uma aplicação independente da arquitetura do computador para o qual foi inicialmente projetada pode aproveitar os melhoramentos de computadores paralelos mais poderosos. Por exemplo: um sistema composto por um número fixo de unidades de execução não poderá ser acelerado por um fator superior a este número.
2. Maximizar o grau de concorrência, que é o potencial para paralelismo. Isto significa decompor a aplicação em um número máximo de unidades concorrentes.
3. Minimizar a interdependência das unidades concorrentes. A interdependência está relacionada a necessidade de sincronização e comunicação, que são atividades que envolvem custos adcionais.

Os três objetivos acima podem acarretar dificuldades. A independência da arquitetura pode levar à ineficiência de execução em um computador com características discrepantes das requeridas pela aplicação. Há ainda um conflito entre os objetivos (2) e (3): a divisão de um sistema em um número maior de subsistemas normalmente implica em uma maior dependência entre os subsistemas.

Um sistema operacional adequado a aplicações paralelas deve prover um ambiente de execução que minore estas dificuldades. Por exemplo:

um número de unidades concorrentes bastante superior ao número de processadores pode ser eficientemente suportado através de uma adequada escalação destas unidades pelos processadores. Outro exemplo: mecanismos eficientes de sincronização e comunicação entre as unidades concorrentes possibilitam aumentar o grau efetivo de paralelismo da aplicação.

A subseção 2.1 analisa as características do ambiente provido pelo PLURIX para a execução de unidades concorrentes, que são decorrência da decisão de ser um sistema de uso geral, e mostra a sua inadequação para a programação paralela. A subseção 2.2 discute a solução adotada para o MULPLIX versão 0.0.

2.1 Modelo de Concorrência do PLURIX

A unidade de execução concorrente do PLURIX é o **processo**. As seguintes entidades, entre outras, estão associados a um processo:

- Espaço de endereçamento virtual composto no modo usuário de um segmento de texto, com o código em execução, um segmento de pilha e um segmento de dados.
- Contexto do processador, contendo os valores correntes dos registradores.
- Recursos alocados, entre eles os arquivos abertos.

O mínimo de operações efetivamente necessárias em qualquer sistema para que haja uma troca de contexto é o salvamento do estado atual do processador e a restauração do seu estado após a última instrução executada pela unidade concorrente que está retornando à execução. Uma troca de contexto no PLURIX envolve além destas operações, outras que podem ser extensas, como a substituição das informações da gerência de memória relativas ao espaço virtual de endereçamento do processo que sai pelo que entra. A realização de uma troca de contexto no PLURIX exige a execução de várias centenas de instruções.

As aplicações paralelas bem concebidas apresentam um grau de concorrência muito grande. Este grau de concorrência tipicamente implica em um número maior de unidades de execução do que de processadores

disponíveis e pode exigir a utilização mais freqüente de mecanismos de sincronização e comunicação, o que por sua vez tende a aumentar o número de trocas de contexto. Assim, aplicações bem concebidas demandam muitas trocas de contexto. Como no PLURIX uma troca de contexto é relativamente lenta, ela não é adequada à execução de aplicações paralelas.

A comunicação e sincronização entre processos no PLURIX pode se dar por **herança** (um processo novo recebe entre outros ítens uma cópia dos dados do processo que o originou), por envio e recepção de **sinais** (interrupções de software), por leitura e escrita de **fifos** e por leitura e escrita de **arquivos**. A comunicação por herança só ocorre no momento de criação de um novo processo. Os sinais são uma forma de comunicação muito restrita, sendo praticamente úteis apenas para informar a ocorrência de um evento. A troca de mensagens por leitura e escrita em fifos é restrita e relativamente lenta. As operações sobre arquivos, mesmo que realizadas em memória, também utilizam muitas instruções. Portanto os mecanismos para a comunicação entre processos disponíveis no PLURIX não permitem uma eficiente cooperação intensiva entre processos. Esta característica dificulta a utilização eficiente de um grau de concorrência maior. Desta forma, os mecanismos para comunicação e sincronização entre processos providos pelo PLURIX não são adequados à programação paralela.

2.2 Modelo de Concorrência do MULPLIX

A principal causa de ineficiência relacionada ao suporte provido pelo PLURIX para cooperação entre as unidades de execução concorrentes é que elas são processos distintos e portanto não compartilham recursos.

No MULPLIX o conceito de processo foi estendido para suportar mais de uma **linha de execução** (*"thread"*). Nesta nova organização, um processo é um ambiente onde podem coexistir várias linhas de execução. Estão associadas a um processo um conjunto de recursos providos pelo sistema operacional e compartilhados por suas linhas de execução, tais como o texto (instruções) de um programa, um espaço de endereçamento, arquivos abertos, etc... Uma linha de execução basicamente tem associado um ponto de execução do programa. Uma aplicação paralela corresponde a um processo

e o seu conjunto de linhas de execução.

O compartilhamento de recursos entre linhas de execução de um mesmo processo simplifica, e conseqüentemente acelera, a troca de contexto entre estas linhas de execução.

As seções 3 (Gerência de Memória) e 4 (Sincronização) analisam mais detalhadamente as facilidades para cooperação entre linhas de execução de um mesmo processo.

A pluralidade de linhas de execução em um mesmo processo é uma idéia também adotada no sistema operacional Mach [1] da Universidade Carnegie Mellon e em sistemas operacionais de tempo real [13].

3 Gerência de Memória

Em sistemas operacionais que provêem um ambiente de multiprogramação, a gerência de memória é responsável pelo controle da utilização e do compartilhamento de memória entre os diversos processos. Isto significa que cada processo não manipula diretamente a memória, mas recebe da gerência de memória um **espaço de endereçamento virtual**. Assim a programação de aplicações é baseada em endereços virtuais que são **mapeados** em tempo de execução (normalmente por hardware) nos endereços reais, de acordo com a **alocação** dos espaços de endereçamento virtual **na memória física**. realizada pelo sistema operacional. Além disto, qualquer tentativa por parte de um processo de leitura ou escrita em um endereço que não pertença ao seu espaço virtual de endereçamento pode ser detectada (normalmente por hardware) e impedida, sem que haja interferência no funcionamento dos outros processos.

Esta seção apresenta e discute a gerência de memória projetada para a versão 0.0 do MULPLIX. A subseção 3.1 define o espaço de endereçamento virtual proporcionado aos processos. A subseção 3.2 aborda a alocação de memória física.

3.1 Espaço de Endereçamento Virtual

O espaço de endereçamento virtual provido pelo PLURIX consiste de um conjunto de áreas contínuas denominadas **segmentos**. Há segmentos para instruções, dados e pilha, tanto para o modo usuário como para o modo supervisor. Este esquema tem a vantagem de agrupar no espaço virtual ítens logicamente relacionados.

O espaço de endereçamento virtual provido pelo MULPLIX representa uma extensão ao PLURIX. Os segmentos definidos abaixo estão associados em modo usuário a cada linha de execução de um processo:

Texto — Este segmento contém as instruções do programa preparado pelo usuário. Dè acordo com a definição de processo adotada pelo MULPLIX, o segmento de texto de usuário é compartilhado por todas

as linhas de execução do processo. (Pode-se incluir neste segmento dados apenas para leitura.)

Dados Globais — Este segmento contém as variáveis para uso por mais de uma linha de execução do processo. O segmento de dados globais de usuário é compartilhado por todas as linhas de execução do processo. (Pode-se incluir dados com valores iniciais estabelecidos.)

Dados Locais — Este segmento contém as variáveis para uso exclusivo por sua linha de execução. Naturalmente este segmento não é compartilhado pelas outras linhas de execução do processo.

Pilha — Este segmento é utilizado para a alocação dos registros de ativação e das variáveis automáticas. Este segmento é de uso exclusivo por sua linha de execução.

Os segmentos definidos para o modo supervisor são relacionados a seguir:

Texto — Instruções para o núcleo do sistema operacional. O segmento de texto de supervisor é compartilhado por todas as linhas de execução de todos os processos.

Dados Globais — Este segmento contém variáveis para uso do sistema operacional. O segmento de dados globais de supervisor é compartilhado por todas as linhas de execução de todos os processos.

Dados Locais — Este segmento contém as variáveis utilizadas exclusivamente pelo núcleo do sistema operacional em execução neste NP. O segmento de dados locais de supervisor é compartilhado por todas as linhas de execução ativas neste NP.

Pilha — Este segmento contém os registros de ativação e as variáveis automáticas para o modo supervisor. O segmento de pilha de supervisor é de uso exclusivo por sua linha de execução.

TCB — O bloco de controle para a linha de execução (*"Thread Control Block"* - TCB) contém um contexto de execução para a linha de execução. O TCB é de uso exclusivo por sua linha de execução.

PCB — O bloco de controle para o processo (*"Process Control Block"* - PCB) contém informações sobre o processo adicionais às informações mantidas nas estruturas de dados internas ao núcleo do sistema operacional. Estas informações podem estar fora do núcleo porque são

necessárias apenas no contexto de execução deste processo. Por outro lado, as informações no PCB são válidas para todas as linhas de execução do processo. O PCB é de uso compartilhado pelas linhas de execução do processo executando em modo supervisor.

Comunicação — São dois segmentos que permitem ao processo executando em modo supervisor ter acesso aos seus segmentos definidos para o modo usuário. Isto torna-se necessário, por exemplo no caso da transferência em blocos relativa às operações de E/S.

3.2 Alocação de Memória Real

O PLURIX adota um esquema de **alocação segmentada** para a memória física. O mapeamento de endereços virtuais em endereços físicos é (simplificadamente) realizado através de uma operação de soma do endereço virtual com um **endereço base** correspondente ao segmento virtual em questão [10].

A alocação segmentada exige que os segmentos virtuais sejam alocados em áreas contínuas de memória física. Esta exigência traz como conseqüência a necessidade de movimentação pela memória física de segmentos virtuais já alocados, quando não há um espaço livre contínuo para a alocação de um segmento que está sendo criado ou expandido.

O MULPLIX adota um esquema de **alocação paginada** para a memória física. Este esquema particiona a memória em um conjunto de unidades, denominadas **páginas**, com tamanhos iguais e potência de dois. Desta forma, um endereço virtual admite uma representação binária composta de duas partes:

1. o número da página — dado pelos bits mais significativos — e
2. um deslocamento dentro da página — dado pelos bits restantes.

O mapeamento de endereços virtuais em endereços físicos consiste em substituir os bits relativos ao número da página, mantendo-se o deslocamento dentro da página.

A alocação paginada é mais flexível do que a alocação segmentada porque não há restrições quanto às posições relativas das páginas de memória

física alocadas para um segmento. Esta maior flexibilidade resulta nas seguintes vantagens:

- A alocação de memória correspondente à criação ou à expansão de um segmento virtual nunca exige a alteração na alocação de outros segmentos, ou da parte já alocada do segmento em expansão.
- Possibilita o uso de técnicas mais sofisticadas, como por exemplo, a cópia na escrita (*"copy on write"*) para a realização de cópias virtuais. (Esta técnica torna a criação de processos muito mais eficiente.)
- A alocação paginada funciona nos casos em que há uma *"lacuna"* no espaço de endereçamento físico. Isto pode ocorrer, por exemplo, se algum NP estiver defeituoso (a área de memória correspondente ao seu módulo de memória não pode ser utilizada).

As seguintes regras valem para a alocação dos segmentos virtuais:

- O segmento de texto definido para o modo supervisor é replicado para cada NP.
- Os segmentos de texto, dados globais e dados locais definidos para o modo supervisor são pré-alocados, ocupando as páginas de endereço mais baixo em cada NP.
- Os segmentos de comunicação (modo supervisor) referem-se a áreas já alocadas em modo usuário, portanto não demandam alocação.
- Os segmentos de pilha, PCB e TCB para o modo supervisor e todos os segmentos para o modo usuário são alocados preferencialmente na área de memória relativa ao NP executando a criação ou expansão do segmento.
- O segmento de texto para o modo usuário é replicado para cada cluster em que há linhas de execução do processo ativas.

4 Sincronização

Esta seção define o termo sincronização, analisa a sincronização interna dos sistemas operacionais UNIX, PLURIX e MULPLIX, apresenta e discute os mecanismos de sincronização disponíveis para a programação de aplicações paralelas no MULPLIX. O apêndices A e B apresentam a implementação dos mecanismos de sincronização na arquitetura do MULTIPLUS.

4.1 Definição de Sincronização

Quando duas ou mais atividades concorrentes cooperam entre si, faz-se necessária uma *coordenação* para que esta cooperação se processe corretamente. O termo **sincronização** abrange os aspectos temporais desta coordenação, ou seja, atividades concorrentes são sincronizadas quando o seu progresso no tempo obedece a determinadas restrições.

Um sistema concorrente pode ser modelado no tempo como um conjunto de operações seqüenciais, sujeitas a uma relação de ordem parcial e a uma relação de exclusão mútua.

A relação de **ordem parcial** é uma relação transitiva, que define pares ordenados de operações na forma (a, b), significando que a operação b somente pode ser executada após a operação a. O adjetivo *"parcial"* indica que a relação não abrange todos os possíveis pares de operações [1].

A relação de **exclusão mútua** define pares de operações cuja execução não pode se sobrepor no tempo. A relação de exclusão mútua está diretamente associada ao compartilhamento de recursos.

No caso do modelo básico de programação paralela oferecido pelo MULPLIX, as operações de um sistema concorrente são agrupadas em diversas linhas de execução. As operações que compõem cada linha de execução são executadas seqüencialmente, assim estão totalmente ordenadas entre si e obviamente são mutuamente exclusivas e portanto já estão implicitamente

[1] Por isto o sistema é concorrente !

sincronizadas. A sincronização envolvendo operações em linhas de execução distintas é realizada através de mecanismos de sincronização. Quanto maior o número de linhas de execução, maior o potencial de paralelismo, mais intensa será a cooperação entre elas e, conseqüentemente, maior a necessidade de sincronização explícita.

4.2 Sincronização Interna ao Sistema Operacional

Em ambientes SOFIX a interação entre o núcleo do sistema e um processo ocorre através de chamadas ao sistema, que envolvem a alteração do estado do processador do **modo usuário** para o **modo supervisor**. Em modo supervisor o processo executa o núcleo (código e estruturas de dados). Assim, os processos executam não apenas o programa preparado pelo usuário, mas também o núcleo do sistema operacional.

O compartilhamento do núcleo pelos processos define linhas de execução concorrente para o mesmo e portanto existe a necessidade de sincronização do núcleo. Os parágrafos a seguir analisam as soluções para sincronização do núcleo adotadas para os sistemas UNIX, PLURIX e MULPLIX.

O problema de sincronização do núcleo do UNIX [5] é bastante simplificado em razão de duas restrições de seu projeto:

1. aplicação voltada para sistemas uniprocessadores e
2. incapacidade de preempção.

Em sistemas uniprocessadores as possibilidades de concorrência do núcleo são restritas aos diversos processos inativos em modo supervisor e ao processo ativo. A incapacidade de preempção significa que quando o núcleo é ativado por uma chamada ao sistema, ele normalmente executa continuamente até o término da chamada; o processo ativo em modo supervisor somente é substituído em situações bem definidas. Assim é necessário considerar a sincronização apenas nestas situações:

1. quando uma subrotina do núcleo decide substituir o processo ativo e
2. quando há uma interrupção para atendimento de um dispositivo de E/S.

A **substituição do processo ativo** exige do programador do núcleo a garantia de que todas as estruturas de dados manipuladas por este processo estão em um estado consistente. Do ponto de vista da engenharia de software, esta solução tem a desvantagem de exigir um conhecimento global do núcleo mesmo para alteração de uma característica específica do sistema.

No caso de **atendimento de interrupções**, é suficiente ordenar os diversos tipos de atendimentos de modo a desabilitar interrupções que poderiam trazer concorrência na manipulação de estruturas de dados.

O problema de sincronização do núcleo do PLURIX [8] foi resolvido de um modo mais completo do que no UNIX. O núcleo de PLURIX utiliza sincronização explícita baseada nos seguintes mecanismos:

1. semáforos binários em espera ocupada (família de primitivas spin),
2. semáforos binários preemptíveis (família de primitivas sleep),
3. semáforos generalizados (família de primitivas sema) e
4. eventos (família de primitivas event).

A situação de bloqueio fatal (*"deadlock"*) é eliminada através da ordenação dos recursos (para impedir circularidade na alocação) e, quando não é possível obedecer a esta ordem, através da liberação de todos os recursos alocados pelo processo e início de uma nova tentativa de alocação no caso da detecção da possibilidade de bloqueio fatal.

O método adotado para sincronização interna do PLURIX será também adotado para o MULPLIX. A implementação das primitivas será adaptada para a obtenção de um melhor desempenho no MULTIPLUS (veja o apêndice A).

4.3 Primitivas de Sincronização Oferecidas no MULPLIX

As primitivas de sincronização oferecidas pelo MULPLIX na versão 0.0 foram projetadas de modo a atender aos seguintes objetivos:

- **Suficiência** — Todas as diversas necessidades de sincronização por parte das aplicações paralelas devem ser atendidas direta ou indiretamente através das primitivas.
- **Simplicidade: generalidade e eficiência** — O núcleo deve implementar com eficiência um pequeno número de primitivas bastante genéricas. A utilização destas primitivas deve ser feita preferencialmente através de bibliotecas de subrotinas e compiladores, apesar do seu uso direto pelo programador mais experiente também ser possível.
- **Segurança** — As primitivas se mal utilizadas poderão levar uma aplicação a um estado de bloqueio fatal, sem que isto afete gravemente as outras aplicações ou o próprio sistema.

  Aplicações em estado de bloqueio fatal podem ser identificadas através da contagem do tempo em que os processos permanecem bloqueados (*"timeout"*). Assim haveria a definição de um prazo máximo para o processo permanecer bloqueado em espera ocupada e, caso este prazo fosse ultrapassado, outro prazo para o processo permanecer bloqueado e inativo. A expiração deste último prazo seria considerada um erro e causaria o cancelamento da aplicação.

Embora o sistema não restrinja o uso de mecanismos potencialmente perigosos, os programas de sistema que eventualmente os utilizarem devem ser implementados de modo a não permitir uma interferência danosa em seu funcionamento por parte de usuários.

As primitivas de sincronização oferecidas pelo MULPLIX são extensões às primitivas baseadas em semáforos e eventos utilizadas para sincronização do núcleo. Por simplicidade, as primitivas serão implementadas totalmente em modo supervisor, cabendo ao núcleo alocar as estruturas de dados necessárias aos mecanismos.

As primitivas baseadas em semáforos foram estendidas para fornecer diretamente a identificação do recurso liberado. As primitivas são as seguintes: semaalloc e semadealloc para alocação de um semáforo, semainit para o estabelecimento de um valor inicial para o semáforo, semalock para a obtenção de um recurso, semafree para a liberação de um recurso e sematest que obtém o recurso apenas se imediatamente disponível e retorna o sucesso da operação.

As primitivas baseadas em eventos foram estendidas para possibilitar a manipulação de eventos cuja ocorrência depende da sua sinalização por mais de uma linha de execução. As primitivas são as seguintes: **eventalloc** e **eventdealloc** para alocação de um evento, **eventinit** para estabelecimento do número de linhas de execução que precisarão sinalizar este evento para que ele seja considerado ocorrido, **eventcount** para a sinalização por uma linha de execução, **eventdone** para a sinalização forçada do evento (desconsiderando o valor estabelecido pela primitiva **eventinit**), **eventwait** para esperar a ocorrência de um evento e **eventtest** para verificar a ocorrência de um evento.

5 Escalação

Um escalador é um gerenciador de recursos. Ele é um mecanismo que implementa uma política que efetiva e eficientemente gerencia o acesso e o uso de recursos por consumidores. A realização de suas atribuições se faz pela alocação de linhas de execução de processos aos processadores.

5.1 Análise do Problema

Esta subseção analisa as diversas alternativas de funcionamento para um escalador de um computador paralelo. Um escalador pode ser classificado pelas opções de funcionamento que adota [15]. Assim a sua classificação facilita a identificação suscinta do seu modo de funcionamento.

Quanto ao seu escopo, um escalador pode ser local ou global. Ele é **local** se atua apenas na escalação de processos a um processador. Um escalador **global** decide em que processador cada processo deve executar, deixando a decisão de escalação dos processos em um processador para o escalador local.

Um escalador pode ser classificado segundo o momento de aquisição das informações relativas à utilização de recursos pelos processos. Se as informações utilizadas pelo escalador precisam estar disponíveis antes da execução dos processos, o escalador é **estático**. Se elas são obtidas durante a execução, o escalador é **dinâmico**. Normalmente os escaladores estáticos realizam a escalação *antes* da execução dos processos, durante a fase de geração deles. Os escaladores dinâmicos sempre realizam a escalação *durante* a execução dos processos, com base no comportamento apresentado pelo sistema.

As classificações a seguir são pertinentes aos escaladores globais e dinâmicos.

A responsabilidade pela realização de uma escalação, classifica os escaladores em distribuídos e não distribuídos. São **distribuídos** os escaladores onde todos os processadores participam das atividades de obtenção de informações sobre o sistema e de execução das decisões da escalação.

A autoridade para a tomada de decisões pelos escaladores distribuídos, os divide em descentralizados e centralizados. Nos escaladores **centralizados** as decisões da escalação são tomadas por apenas um processador. Nos escaladores **descentralizados** todos processadores estão autorizados a tomar decisões de escalação.

Os escaladores distribuídos podem ser classificados de acordo com o grau de autonomia na determinação de como seus recursos devem ser utilizados. Eles podem ser cooperativos e não cooperativos. Se cada processador realiza ações independentemente das ações realizadas pelos demais, se preocupando apenas com as conseqüências locais destas ações, o escalador é **não cooperativo**. Nos escaladores **cooperativos** cada processador realiza as ações que lhe competem quanto à escalação, mas todos os processadores trabalham de acordo com objetivos comuns quanto à escalação global e eventualmente sacrificam a seu desempenho individual em benefício do total.

Quanto a distribuição da carga de processamento entre os diversos processadores, um escalador pode ter uma estratégia de balanceamento de carga ou de divisão de carga. A estratégia de **balanceamento de carga** visa distribuir igualmente a carga pelos processadores. Seu princípio fundamental é que a minimização do tempo total de execução de uma aplicação, pode ser obtida através da minimização da diferença entre os tempos de término de cada processo da aplicação. Já a estratégia de **divisão de carga** visa apenas manter todos os processadores com carga. Seu princípio fundamental é que a minimização do tempo total de execução de uma aplicação pode ser obtida pela minimização do custo adicional da escalação e pela manutenção de todos os processadores ocupados durante a execução da aplicação, sem que necessariamente a carga se mantenha igualmente distribuída pelos processadores.

Os escaladores podem ainda ser classificados quanto à mobilidade dos processos nos processadores. Um escalador pode realizar uma atribuição estática ou dinâmica de processos aos processadores. Na atribuição **estática** os processos permanecem nos processadores de sua primeira atribuição. Na atribuição **dinâmica** os processos podem ter sua atribuição alterada ao longo do tempo.

5.2 Objetivos

Os escaladores dos sistemas multiprogramados interativos têm por objetivos prover a ilusão da existência de um processador para cada unidade de execução e maximizar a utilização dos processadores. Como forma de atingir estes objetivos, eles são dinâmicos e realizam um compartilhamento do tempo dos processadores entre as unidades de execução. As políticas adotadas para este compartilhamento visam distribuir com justiça o tempo de processador entre as unidades de execução, minimizar os seus tempos de reação aos comandos dos usuários e evitar esperas indefinidas.

O objetivo fundamental do MULTIPLUS é tornar viável a execução de aplicações científicas ou de engenharia que demandem muito processamento, através da exploração intensiva de paralelismo. A viabilidade está relacionada ao tempo real necessário à finalização das aplicações paralelas em execução, conseqüentemente o objetivo central da escalação no MULPLIX é minimizar este tempo.

Uma qualidade desejável do escalador é a **generalidade**. Deve ser possível a escalação de qualquer tipo de aplicação paralela. O escalador deve prover um ambiente de execução que independa da configuração disponível de hardware, mas que permita às aplicações paralelas utilizar de modo transparente e eficiente as características e os recursos do hardware disponível. O próprio escalador deve ser capaz de automaticamente se adaptar a qualquer configuração do MULTIPLUS. A generalidade do escalador deve prover ainda a existência de um mesmo ambiente para a execução de aplicações paralelas e o seu desenvolvimento.

5.3 Escalação no MULPLIX

Esta subseção apresenta as alternativas de funcionamento adotadas pelo escalador do MULPLIX e algumas heurísticas empregadas.

5.3.1 Características Gerais

O MULTIPLUS pode se apresentar sob a forma de diversas configurações diferentes. Ele pode ter desde alguns NP até milhares deles. Como forma de simplificar a adaptação do escalador a esta diversidade, o escalador é *simétrico*, ou seja o mesmo código do escalador é copiado em cada NP.

Apesar de dispor de uma pluralidade de processadores, a arquitetura MULTIPLUS refere-se a computadores paralelos unitários, portanto a sua escalação é *global*.

Os escaladores estáticos podem produzir uma escalação ótima e podem apresentar um mínimo de custo extra durante a execução, porque utilizam o conhecimento a priori das características de execução das aplicações. Entretanto, eles são inviáveis para muitas aplicações paralelas, para as quais a obtenção deste conhecimento é impossível ou demasiadamente onerosa. Os escaladores dinâmicos, por outro lado, podem se adaptar a qualquer carga de processamento, suportam eficientemente aplicações interativas e podem apresentar um desempenho próximo do ótimo quase sempre, mesmo utilizando heurísticas simples, que implicam num custo adicional desprezível. Assim a escalação *dinâmica* é mais adequada aos objetivos do MULPLIX e é a adotada.

Uma característica importante da arquitetura do MULTIPLUS é a sua organização com até milhares de NP separados em grupos. A utilização eficiente desta organização é fundamental para o desempenho geral do sistema. Os dois tipos de operações a seguir são ineficientes: as que envolvam simultâneamente muitos NP e aquelas que só possam ser realizadas por um pequeno subconjunto dos NP. Como forma de evitar as operações do primeiro tipo, o escalador do MULPLIX é *pouco cooperativo*. Para evitar as do segundo tipo, ele é *distribuído* e *descentralizado*.

Outra característica da arquitetura do MULTIPLUS que influencia o escalador é o compartilhamento de memória por todos os NP. A possibilidade de acesso a toda memória por qualquer NP, permite uma grande mobilidade dos processos e logo a sua atribuição dinâmica aos processadores. A atribuição dinâmica dá uma maior liberdade ao escalador, aumentando a sua adaptabilidade às condições de execução. Considerando as suas van-

tagens e a possibilidade de seu uso, a *atribuição dinâmica* de processos a processadores é adotada no MULPLIX.

5.3.2 Política de Particionamento do Tempo

A política de particionamento do tempo estabelece para cada processador em que momentos há oportunidade para a substituição da linha de execução ativa.

O desempenho das aplicações científicas paralelas é especialmente contemplado pelo escalador através do privilegiamento das linhas de execução destas aplicações em relação às linhas de execução de aplicações interativas.

O privilégio das linhas de execução de aplicações paralelas científicas se manifesta através do tratamento diferenciado à evolução das suas prioridades e na existência de um modo dedicado de funcionamento dos processadores. Esta diferenciação refere-se a não alteração das suas prioridades. As vantagens advindas desta diferenciação são a manutenção da prioridade inicial (*alta*) das linhas de execução privilegiadas, não considerando a utilização de processadores por elas, e a economia do processamento que seria necessário para o cálculo de sua evolução.

Quando um processador está funcionando em modo dedicado, a linha de execução que o utiliza sofre o mínimo de interferências externas. Ela executa sem interrupções, até que termine ou necessite de um recurso não disponível imediatamente.

A monopolização do MULTIPLUS pelas linhas de execução privilegiadas ocorreria se todos os processadores estivessem em modo dedicado ao mesmo tempo. Esta monopolização é incoveniente se há em execução processos interativos, ou de monitoração das aplicações paralelas em execução. Ela é impedida pela determinação de um número máximo de processadores simultaneamente em modo dedicado, que é controlado por um semáforo. Este número é definido durante a iniciação do MULPLIX e pode ser alterado posteriormente.

5.3.3 Política de Seleção

A **política de seleção** determina qual linha de execução é escalada quando um processador torna-se disponível.

A utilização eficiente da hierarquia de memória do MULTIPLUS depende da exploração da localidade de memória. A localidade de memória cresce a medida que a posição de memória a ser acessada por um NP está na memória local de um NP em outro cluster, na memória local de outro NP no mesmo cluster, na memória local do próprio NP, ou no cache do próprio NP. Assim a próxima linha de execução a ser escalada em um processador, pertence ao processo de maior prioridade e maximiza a localidade de memória. Do mesmo modo um processo é criado no cluster que contém o PESB que gerencia o disco de seu diretório preferencial de trabalho.

5.3.4 Implementação

Como forma de maximizar a localidade de memória, há uma fila de linhas de execução prontas por cluster. Cada uma destas filas pode ser acessada por todos os processadores, mas um processador disponível só procura uma linha de execução pronta para executar na fila de outro cluster se a fila do seu cluster estiver vazia.

5.4 Comparação com a Escalação no PLURIX

A escalação no PLURIX [12], apesar de mais flexível que o do UNIX, não é adequada a exploração de paralelismo e a uma arquitetura NUMA com muitos processadores, como o MULTIPLUS. Sua inadequação à exploração de paralelismo decorre do não tratamento diferenciado à execução de aplicações paralelas. A inadequação à arquitetura do MULTIPLUS advém da existência de um fila única de processos prontos e da não consideração da localidade de memória. A utilização de uma fila única de prontos em uma arquitetura com muitos processadores pode provocar uma contenção das vias de acesso à memória e a ela própria. Não há no PLURIX o objetivo de maximizar a localidade de memória. A escolha do pro-

cessador da próxima execução de um processo no PLURIX independe da localização de seus acessos à memória, ocasionando mesmo em arquiteturas UMA (*"Uniform Memory Access"*) o desperdício do cache nas trocas de contexto.

6 Gerência do Sistema de Arquivos

Um **sistema de arquivos** é uma forma de organizar e facilitar a utilização da memória de massa de um computador. Ele permite aos usuários organizar semanticamente seus dados, armazenando-os em **arquivos** e provê um mecanismo de nomeação dos arquivos, agrupando-os em **diretórios**. As operações sobre o sistema de arquivos permitem a manipulação de seus dados com independência do dispositivo físico que os armazena.

6.1 O Sistema de Arquivos dos SOFIX

Nos SOFIX os arquivos são estruturados sob a forma de uma seqüência de bytes. Eles ocupam uma seqüência de blocos não necessariamente contíguos, cada bloco com normalmente 512 bytes. Os endereços dos blocos disponíveis estão contidos em uma lista armazenada em blocos do próprio dispositivo.

Para cada arquivo há uma pequena tabela, conhecida por **inode**, com todas as informações relativas ao arquivo associado, com a exceção do seu nome. Estas informações abrangem a posse do arquivo, a sua proteção, a localização dos seus blocos, seu tamanho, número de nomes diferentes e as datas de criação, último acesso e última modificação. Os inodes de um sistema de arquivos ocupam uma seqüência de blocos do dispositivo físico, cuja localização é conhecida.

A estrutura de um diretório é muito simples. Um diretório é mais um arquivo, cujo conteúdo tem uma entrada para cada arquivo pertencente a ele. Cada entrada tem o nome do arquivo neste diretório e o número do seu inode. Esta estrutura possibilita que um arquivo tenha mais de um nome e permite o seu compartilhamento por processos diferentes. Um sistema de arquivos forma um grafo direcionado acíclico.

As operações sobre um sistema de arquivos são realizadas através de chamadas ao núcleo do sistema operacional. As principais operações sobre o

sistema de arquivos são a montagem e desmontagem de sistemas de arquivo e a abertura, leitura, escrita e fechamento de arquivos.

O PLURIX apresenta algumas melhorias em relação aos SOFIX, que aumentam o desempenho do sistema de arquivos. As principais são a possibilidade de montagem de um sistema de arquivos em memória e possibilidade de ter programas residentes em memória.

6.2 Entrada e Saída de Blocos no MULTIPLUS

Na arquitetura MULTIPLUS há componentes dedicados a controlar a operação dos dispositivos físicos que realizam operações com blocos, como unidades de disco e de fita. Eles são os Processadores de Entrada e Saída Blocada (PESB). Cada dispositivo está conectado a um PESB, que normalmente têm vários dispositivos conectados. Cada PESB tem um cache de blocos para acelerar as operações sobre os seus dispositivos. Os PESB se comunicam por uma rede it Ethernet, assim cada PESB tem acesso também aos blocos dos dispositivos conectados aos demais PESB.

A cada NP está associado um PESB, é aquele presente no mesmo cluster, ou se este não existir, é o presente em um cluster vizinho. Todas as operações de entrada e saída blocada são pedidas pelos NP aos seus PESB associados. As operações de transferência de blocos entre os PESB e a memória se utilizam de um mecanismo de transferência em "rajada". Este mecanismo pode envolver apenas o barramento de um cluster, ou até os barramentos de dois clusters e parte da RI.

6.3 Objetivos para o Sistema de Arquivos do MULPLIX

A arquitetura do MULTIPLUS e as aplicações científicas paralelas apresentam características, que determinam alguns objetivos para a implementação de sistemas de arquivos no MULPLIX, além da compatibilidade com os SOFIX. Estes objetivos são:

- Minimizar o tempo das operações sobre o sistema de arquivos. As aplicações científicas normalmente manipulam uma enorme quantidade de dados, devido a sua dimensão, eles são armazenados em sistemas de arquivos. O atendimento ao objetivo principal do MULTIPLUS de tornar viável a execução de aplicações científicas, exige a minimização do tempo gasto com a manipulação de dados por elas. Por outro lado há uma enorme capacidade disponível de processamento e para que o sistema esteja balanceado é necessário uma velocidade do sistema de arquivos correspondente a de processamento. Como os processadores tem evoluído mais rapidamente que os dispositivos de armazenamento de massa, para se atingir este balanceamento é necessário cada vez mais aprimorar a utilização destes dispositivos.
- Minimizar o tráfego entre os clusters e intracluster. Na arquitetura do MULTIPLUS, os principais pontos potenciais de gargalo são a rede de interconexão (RI) e os barramentos. A diminuição de suas utilizações colabora para o aumento da performance do sistema como um todo.
- Adaptação ao paralelismo. Os processos no MULPLIX podem apresentar mais de uma linha de execução. A gerência do sistema de arquivos tem então que prover um funcionamento, consistente e coerente, para a manipulação de arquivos pelas linhas de execução de um mesmo processo.

6.4 O Sistema de Arquivos do MULPLIX

Para atingir os dois primeiros objetivos, os sistemas de arquivos que sejam básicamente apenas de leitura, como aqueles com os comandos do sistema, as bibliotecas, etc, são replicados nos diversos PESB. Cuidados especiais tem então que ser empregados para a modificação (rara) destes sistemas de arquivos.

A implementação da replicação acima é simples. Nos SOFIX os sistemas de arquivos são ligados transparentemente constituindo um único sistema de arquivos global. Há uma tabela de montagem global que descreve a origem física de cada sistema de arquivos e a sua localização no sistema de

arquivos global. A replicação pode ser implementada através da existência de uma tabela de montagem diferente e apropriada para cada PESB.

O terceiro objetivo pode ser atendido pelo seguinte esquema. A abertura de um arquivo por uma linha de excecução garante a manipulação exclusiva (em relação às demais linhas de execução deste mesmo processo) do arquivo por ela. Esta solução garante a consistência, a coerência e a compatibilidade. A limitação imposta não implica em perda de performance nem em restrições severas ao programador, pois as operações sobre um arquivo já são centralizadas por um PESB.

7 Estágio Atual e Perspectivas Futuras

A versão 0.0 do MULPLIX está definida funcionalmente. A estratégia para implementação abrange duas fases. A fase inicial utilizará supermicros com o PLURIX como máquinas de desenvolvimento e abrangerá dois esforços paralelos: implementação das extensões ao PLURIX e adaptação do compilador para a linguagem *"C"* do PLURIX para os processadores SPARC. Na fase seguinte, após a disponibilidade de um protótipo do MULTIPLUS, o sistema será transportado e seu desenvolvimento prosseguirá neste ambiente fortemente influenciado pela realimentação decorrente da sua própria experiência de uso.

Ordem de implementação no EBC32010:

1. gerência de memória,
2. uma linha de execução para cada processo,
3. várias linhas de execução por processo,
4. sincronização para o usuário.

As perspectivas mais imediatas são de conclusão e apresentação pública da versão 0.0 ao final do segundo semestre de 1991. Já está em estudos a evolução do MULPLIX para melhor refletir o paralelismo da arquitetura do MULTIPLUS. Versões posteriores suportarão uma variedade de modelos de programação paralela, a nível de compiladores, bibliotecas de subrotinas ou diretamente pelo núcleo.

A Implementação do Semáforo Binário

Este apêndice descreve a implementação das primitivas de sincronização baseadas em espera ocupada de semáforos binários. Estas primitivas formam a base para a implementação das demais primitivas para sincronização em modo supervisor (entre linhas de execução do núcleo) e em modo usuário (entre linhas de execução de um mesmo processo).

A.1 Análise do Problema

A implementação eficiente dos mecanismos de espera ocupada é um dos fatores determinantes do desempenho global de sistemas multiprocessadores de larga escala com memória compartilhada.

Os parâmetros e critérios relacionados a seguir são fundamentais para a avaliação do desempenho de um algoritmo para a implementação dos mecanismos de sicronização:

- Latência mínima. Este parâmetro estabelece o tempo mínimo necessário para a alocação de um semáforo, considerando-se que não há contenção.
- Taxa de *"troughtput"*. Esta taxa fornece o número máximo de vezes que o semáforo pode ser alocado durante um dado intervalo de tempo, supondo-se que os outros processadores estão também estão sempre tentando alocar o semáforo.
- Interferência no desempenho dos outros processadores. Isto pode ocorrer por exemplo através da contenção das vias de acesso à memória. As vias de acesso à memória constituem um recurso de uso extremamente crítico no caso de configurações com um número elevado de processadores.

A.2 Idéias Básicas da Solução

O algoritmo *"Queueing in shared memory"* apresentado por Anderson em [2] foi utilizado como base para desenvolvimento de um novo algoritmo

mais adequado ao MULTIPLUS. As idéias básicas deste novo algoritmo são:

- Os processadores esperando pelo semáforo binário são enfileirados em um buffer circular.
- A liberação do semáforo é realizada pelo processador que o detém e inclui a sua transferência para o processador seguinte na fila (se houver).
- Um processador esperando um semáforo percebe a transferência do semáforo para si através da observação contínua (*"polling"*) de uma variável local no cache.

O enfileiramento dos processadores resolve a disputa pelo semáforo no momento da decisão da posição de cada um deles na fila (esta decisão é implementada por hardware através de uma instrução atômica, conforme será visto adiante). Este procedimento resulta em um esquema de *"round robin"*, em que o tempo máximo de espera é limitado (não há *"starvation"*).

A transferência direta do semáforo do processador que o detém para o processador seguinte na fila evita a interferência de/nos outros processadores.

A percepção da recepção do semáforo através da observação de uma variável local no cache mantém o barramento, a rede de interconexão e o módulo de memória local livres para acesso por outros processadores.

A.3 Definições Básicas e Estruturas de Dados

Os algoritmos operam com as seguintes variáveis e constantes:

buffer — Vetor para a implementação de uma fila circular. Cada posição da fila pode ser ocupada pela identificação de um processador ou por uma das constantes DOANDO e ESPERANDO. Há uma instância desta variável para cada semáforo.

atual — Posição ocupada pelo último processador a entrar na fila. Há uma instância desta variável para cada semáforo.

p — Posição ocupada por este processador na fila. Esta variável conserva o seu valor entre a obtenção e a liberação do semáforo. Há uma instância desta variável para cada processador.

id — Identificação de um outro processador na fila.

ID — Constante identificando este processador.

DOANDO — Constante que indica que houve um processador nesta posição da fila que já liberou o semáforo.

ESPERANDO — Constante que indica que o processador nesta posição da fila aguarda a liberação do semáforo.

As posições na fila são representadas por inteiros de 32 bits, que admitem uma variação muito mais abrangente do que a faixa de posições disponíveis na fila. A operação POS(x) realiza a conversão necessária, retornando a posição válida na fila correspondente ao seu argumento x. As operações ANT(x) e SUC(x) são similares à operação POS(x), com a exceção de retornarem respectivamente as posições anterior e posterior na fila em relação à posição retornada por POS(x). Estas operações admitem implementação direta através de máscaras de bits, se o tamanho da fila é uma potência de 2.

A instrução F&I (*"Fetch and Increment"*) é uma instrução atômica que retorna o conteúdo de uma variável e (imediatamente a seguir e sem interrupções) o incrementa. A implementação desta instrução é o tema do apêndice B.

A.4 Algoritmos

Os algoritmos para a obtenção, liberação e iniciação de um semáforo binário são explicados a seguir.

O algoritmo para a iniciação de um semáforo binário prepara a fila de processadores, indicando a disponibilidade do semáforo para o primeiro processador que executar o algoritmo para obtenção.

O algoritmo para obtenção de um semáforo binário realiza as seguintes operações: (1) indica em sua área local que está aguardando a liberação

```
Iniciação de um Semáforo Binário:
    para cada posição i de buffer
        buffer[i] := ESPERANDO
    atual := 0
    buffer[ANT (atual)] := DOANDO
```

do semáforo, (2) aloca uma posição na fila, (3) se coloca na fila, (4) se o processador que estava posicionado na fila imediatamente a sua frente já liberou o semáforo, então o semáforo já é seu; em caso contrário, aguarda a liberação do semáforo em sua área local.

```
Obtenção de um Semáforo Binário:
    local[ID] := ESPERANDO                          (1)
    p := POS (F&I (atual))                          (2)
    buffer[p] := ID                                 (3)
    se buffer[ANT (p)] <> DOANDO                    (4)
        espera (local[ID] = DOANDO)
```

O algoritmo para a liberação de um semáforo binário realiza as seguintes operações: (1) avisa na fila que o semáforo está liberado, (2) se já há algum outro processador imediatamente atrás na fila, avisa também em sua área local, (3) reinicia a posição imediatamente à sua frente (a sua própria posição é usada para indicar a liberação do semáforo).

```
Liberação de um Semáforo Binário:
    buffer[p] := DOANDO                             (1)
    se buffer[SUC(p)] = id                          (2)
        local[id] := DOANDO
    buffer[ANT(p)] := ESPERANDO                     (3)
```

B Implementação das Instruções Atômicas

Este apêndice descreve a implementação das operações F&I" (*"Fetch and Increment"*) e TAS (*"Test and Set"*).

B.1 Instrução F&I

A operação F&I é necessária para a implementação do semáforo binário (veja o apêndice A).

A operação F&I não é diretamente suportada pelo SPARC. As instruções atômicas definidas pelo SPARC são: LDSTUB (lê um byte sem sinal de uma posição de memória e escreve uma constante em seu lugar) e SWAP (lê uma palavra de uma posição de memória e escreve outra em seu lugar).

Considerando que o projeto MULTIPLUS também envolve o desenvolvimento de um novo microprocessador, segundo as definições do SPARC, a conveniência da implementação de uma instrução F&I foi analisada para este novo processador. Esta opção foi descartada por duas razões: (1) a disponibilidade do novo processador está prevista apenas para uma etapa posterior do projeto MULTIPLUS e (2) é importante conservar a compatibilidade deste processador com a definição do SPARC, de modo a não limitar a sua utilização ao MULTIPLUS.

Diante da impossibilidade de alteração do processador, foi adotada uma solução externa. Ela se baseia nos seguintes princípios:

- O programador usa LDSTUB como F&I de um byte e SWAP como F&I de uma palavra.
- Os pedidos de leitura originados por uma operação F&I são sempre enviados aos módulos de memória.
- Os barramentos e a rede de interconexão são responsáveis por transmitir também a informação de que estes são pedidos F&I.
- Os módulos de memória são responsáveis pela operação de incremento.

- Os circuitos externos ao SPARC ignoram os pedidos de escrita decorrentes de operações de F&I.

A solução adotada impede o uso no MULTIPLUS das instruções atômicas originais do SPARC. Esta perda não é considerada significativa porque a natureza extremamente específica destas instruções indica que elas não devem estar presentes em programas de usuário compilados em outros ambientes.

A viabilidade da solução adotada dependeu da resolução das seguintes questões:

- Capacidade de identificação das instruções atômicas através dos sinais gerados pelo SPARC. Resolvido pelo sinal LOCK.
- Interação correta com os caches. Solução: as variáveis operadas por F&I são armazenadas em espaços de endereçamento definidos como não possíveis de estar no cache.
- Serialização das leituras F&I. Isto é garantido porque as operações F&I são realizadas pelo módulo de memória em que o operando se encontra.

B.2 Instrução TAS

A instrução TAS pode ser implementada a partir da instrução F&I.

Referências

[1] Accetta, M., R. Baron, W. Bolosky, D. Golub, R. Rashid, A. Tevanian e M. Young; *Mach: A New Kernel Foundation for UNIX Development*: Proc. of the Summer 1986 USENIX Technical Conference and Exhibition, pp.93-112.

[2] Anderson, T. E.; *The Performance of Spin Lock Alternatives for Shared Memory Multiprocessors*; IEEE Trans. on Parallel and Distributed Systems, January 1990, Vol 1, Number 1, pp. 6-16.

[3] Aude, J. S. et al, *MULTIPLUS: Um Multiprocessador de Alto Desempenho*: Anais do X Congresso da Sociedade Brasileira de Computação, Julho de 1989, pp. 93-105.

[4] Azevedo, G. P., *Make, a Ferramenta Essencial de um SOFIX*; Boletim do PLURIX, Novembro de 1987, Ano 1 Número 2.

[5] Bach, M., *The Design of the UNIX Operating System*; New Jersey, Prentice-Hall, 1986.

[6] Faller, N., Azevedo, G. P., Azevedo, R. P., Barbosa, S. M. A., e Salenbauch, P., *O PLURIX Versão 2.0*; Boletim do PLURIX, Agosto de 1988, Ano 2 Número 5.

[7] Faller, N. et al; *O Projeto Pegasus-32X/Plurix*; Anais do XVII Congresso Nacional de Informática, 1984.

[8] Faller, N. e Salenbauch, P., *PLURIX, O Sistema Operacional Multiprocessador do NCE-UFRJ: (1) Sincronização de Processos*; Data News, 24 de setembro de 1985.

[9] Faller, N. e Salenbauch, P., *A Multiprocessing UNIX-like Operating System*; Proc. of the Second IEEE Workshop on Workstation Operating Systems, IEEE Computer Society Press. Washington, DC, E.U.A, pp. 29-36

[10] Figueira, N. R.; *Gerência de Memória e Memória Virtual para o Projeto MULTIPLUS*; Relatório Técnico 06/89, NCE/UFRJ, outubro de 1989.

[11] LeBlanc, T. J., Marsh, B. D. e Scott, M. L., *Memory Management for Large-Scale Multiprocessors*; Technical Report, Computer Science Department, University of Rochester, 1989.

[12] Salenbauch, P., *O escalador de processos do PLURIX*; Boletim do PLURIX, agosto de 1989, Ano 3 Número 9.

[13] Technical Committee on Operating Systems of the IEEE Computer Society; *POSIX: IEEE trial-use standard portable operating systems for computer environment*; New York, Wiley-Interscience, 1986.

[14] Cypress Semiconductor Corporation; *SPARC RISC USER'S GUIDE*; segunda edição, fevereiro de 1990.

[15] Casavant, T. L. e J. G. Kuhl, *A Taxonomy of Scheduling in General Purpose Distributed Computing Systems*; IEEE Transactions on Software Engineering, New York, V.14, N.2, pp.141-154, Feb 1988.