M3P: UM MULTIPROCESSADOR FRACAMENTE ACOPLADO COM BALANCEAMENTO DE CARGA E MIGRAÇÃO DE PROCESSOS

Roque Luís Scheer[1]
Evandro Bender[2]
Philippe Olivier Alexandre Navaux[3]

Pós-graduação em Ciências de Computação-Instituto de Informática
Universidade Federal do Rio Grande do Sul
Av. Bento Gonçalves, 9500, Bloco IV, Caixa Postal 15064-91501 - Porto Alegre, RS
Fax: (051) 336-5576 Fones: (051)336-8399, R. 6165

RESUMO

Este artigo descreve a implementação do sistema operacional MinixM, utilizado no M3P, dando ênfase à distribuição e ao balanceamento da carga entre os processadores. No M3P, um sistema multiprocessador fracamente acoplado, o balanceamento de carga é obtido através do emprego da migração de processos e de heurísticas para a alocação de processos a processadores. Estas soluções são eficientemente implementadas com o uso de um recurso existente no hardware deste sistema que é a transferência do conteúdo da memória entre os processadores por DMA.

ABSTRACT

This article describes the implementation of the MinixM operating system, used on the M3P, especialy the load distribution and balancing between processors. In the M3P, a loosely-coupled multiprocessor system, the load balancing is obtained with the use of process migration and heuristics to assign processes to processors. These solutions are efficiently implemented using a resource present in the hardware, the transfer of memory contents beetween processors through DMA.

1 - Bacharel em Ciências de Computação; Mestrando do Curso de Pós-graduação em Ciências de Computação da Universidade Federal do Rio Grande do Sul. E-mail: *rscheer@inf.ufrgs.br*

2 - Engenheiro Elétrico; Mestrando do Curso de Pós-graduação em Ciências de Computação pela Universidade Federal do Rio Grande do Sul.

3 - Doutor em Informática (Grenoble, 1979); Professor do Instituto de Informática e Curso de Pós-graduação em Ciências de Computação da Universidade Federal do Rio Grande do Sul. E-mail: *navaux@inf.ufgrs.br*

1. INTRODUÇÃO

O M3P (Minix Multi-Microprocessador) é um projeto desenvolvido no Curso de Pós-graduação em Ciências de Computação da UFRGS com o objetivo de adquirir experiência no desenvolvimento de hardware e software para multiprocessadores. O sistema operacional adotado no projeto, ao invés de ser uma implementação totalmente nova, consiste na adaptação de um sistema já disponível para monoprocessadores, o Minix /TAN87/. Isto simplificou o desenvolvimento do sistema, tornando-o realizável com os recursos humanos e materiais disponíveis, permitindo canalizar os esforços no desenvolvimento de recursos avançados, como o controle do balanceamento de carga entre os processadores.

O sistema descrito neste artigo é um multiprocessador do tipo fracamente acoplado. Os sistemas deste tipo, em virtude da relativa facilidade com que podem ser implementados, têm sido freqüentemente usados como alternativa para obtenção de computadores com maior poder de processamento. Este tipo de arquitetura possui, no entanto, algumas limitações, das quais a principal é a dificuldade de distribuir equilibradamente as tarefas entre os processadores, o que impede a utilização plena de toda a capacidade de processamento disponível. Este trabalho procura mostrar como estas limitações foram minimizadas no M3P.

Este artigo inicia apresentando uma descrição sucinta da arquitetura do sistema, na seção 2. Na seção seguinte serão vistos o funcionamento e a implementação do sistema operacional propriamente dito, explicando as alterações efetuadas para tornar o sistema Minix Multiprocessado. Finalmente, na seção 4, serão descritas as caraterísticas avançadas deste sistema - a distribuição da carga entre os processadores e a manutenção do seu balanceamento.

2. A ARQUITETURA DO M3P

O hardware do sistema M3P é construído a partir de um microcomputador do tipo IBM-PC /IBM83/, chamado de hospedeiro, no qual são instaladas até três placas de expansão (chamadas placas processadoras) contendo os processadores adicionais para compor o multiprocessador, além dos componentes associados para permitir que estes processadores se comuniquem entre si /CAR89/. A configuração completa do sistema pode ser visualizada na figura 2.1. O barramento do hospedeiro é usado como barramento compartilhado no qual as placas de expansão são conectadas. É através dele que os processadores se interligam. O hospedeiro provê ainda para o sistema o seus controladores de DMA e de interrupções, timer, memória e ainda os controladores dos periféricos conectados ao sistema.

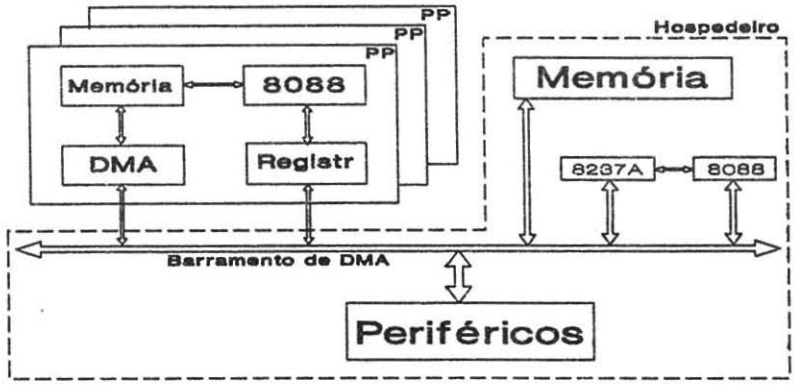

Figura 2.1 - Configuração do Hardware do M3P

As Placas Processadoras são construídas em torno de um microprocessador Intel 8088 /INT87/ igual ao do hospedeiro (o sistema portanto é homogêneo). Este processador possui associado a ele um controlador de interrupções e um timer, 512 kbytes de memória RAM, 32 kbytes de EPROM, um gerador de endereços para as operações de DMA, e um registrador de comunicação entre processadores, usado para o envio e recepção de mensagens. Todos estes componentes se interligam através de um barramento privado existente na placa, chamado de barramento local. Este barramento é controlado diretamente pelo processador da placa. A comunicação com as demais partes do sistema é feita através do barramento do hospedeiro utilizando transferências por DMA e através da passagem de mensagens pelo registrador de comunicação.

2.1 O Registrador de Comunicação

O Sistema operacional utilizado no M3P é implementado como uma coleção de processos que se comunicam através da passagem de mensagens de tamanho fixo. Quando dois processos comunicantes residem em processadores diferentes é necessário que exista um meio de passar o conteúdo da mensagem entre os processadores. O mecanismo atualmente implementado para esta passagem consiste simplesmente de registradores bi-direcionais de 1 byte existentes entre o hospedeiro e cada processador escravo. Estes registradores, ao serem escritos por um dos processadores (hospedeiro ou escravo) causam uma interrupção no segundo, avisando-o para que leia o conteúdo do registrador. Ambos os processadores podem escrever simultaneamente sem que um sobreponha o conteúdo escrito pelo outro (o mecanismo na verdade é implementado com o uso de dois registradores, um para cada direção de comunicação). Existe um destes mecanismos entre cada par de processadores comunicantes, ou seja, entre cada

processador escravo e o mestre. Os processadores escravos não se comunicam entre si diretamente.

Este mecanismo extremamente simples é suficiente para implementar as primitivas de comunicação entre processadores necessárias ao sistema operacional. Através destes registradores é implementado um diálogo entre os processadores de origem e destino para a passagem do conteúdo de uma mensagem entre eles, como será visto adiante. Devido ao tamanho das mensagens ser bastante reduzido (menos de 25 bytes) o desempenho obtido é aceitável para o protótipo implementado. Em versões anteriores do M3P /NAV90/ existia um mecanismo mais complexo para a passagem de mensagens, utilizando microcontroladores dedicados para esta tarefa, mas que, apesar de aparentemente mais sofisticado, fornecia um desempenho inferior a esta solução, que é bem mais simples e econômica.

2.2 O Mecanismo de DMA

Muitas vezes, além da passagem de mensagens, é necessário também transferir grandes volumes de dados entre as memórias dos processadores; estas transferências são feitas de maneira bastante eficiente no M3P com o uso de DMA. O barramento através do qual se processam estas transferências é o próprio barramento do hospedeiro, no qual as placas processadoras estão diretamente conectadas. O controle da transferência é feito pelo controlador de DMA existente no hospedeiro, o 8237A /INT87/ Este controlador gera os endereços de memória para o hospedeiro, controla o número de bytes a serem transferidos e define a direção da transferência (leitura ou escrita).

A memória local de cada processador escravo pode ser acessada pelo próprio processador através do barramento local da placa processadora ou pelo barramento de DMA. Para os acessos feitos pelo barramento de DMA existe um gerador de endereços que deve ser inicializado com o endereço inicial da área de origem ou destino dos dados na memória da placa processadora, antes de ser iniciada a transferência. A transferência de DMA propriamente dita se processa de maneira transparente para os processadores, e durante a mesma eles podem se dedicar a outras tarefas, simultaneamente. O fim da transferência e sinalizado por uma interrupção. As transferências só podem ser realizadas entre uma das placas processadoras e o hospedeiro, e num dado momento apenas uma única transferência pode estar em andamento. Quando for necessário transferir dados entre duas placas processadoras estes dados deverão ser transferidos inicialmente para a memória do hospedeiro e depois para a placa destinatária.

3. O SISTEMA OPERACIONAL MINIX MULTIPROCESSADO

Grande parte dos sistemas operacionais atualmente existentes, inclusive o Unix /THO78/ /BAC86/, é estruturada de forma monolítica, ou seja, eles são construídos como uma coleção de rotinas que chamam livremente outras rotinas à medida que necessitem dos serviços implementados por elas, e o controle da concorrência no acesso a regiões críticas é feito geralmente utilizando primitivas do tipo semáforos ou monitores. Todas as rotinas são geralmente ligadas formando um único código objeto.

O Sistema Minix, que foi portado para a arquitetura M3P, em oposição a este esquema, é estruturado em quatro níveis, cada nível composto por módulos com funções específicas e que funcionam de maneira relativamente independente. Cada módulo é um processo que possui seu próprio estado de execução, contador de programa, registradores e pilha. Os módulos se comunicam entre si utilizando como único método de comunicação e sincronização entre processos o envio de mensagens, ou seja, cada processo integrante do sistema requisita um serviço de outro através do envio de uma mensagem, recebendo de volta o resultado da mesma forma. Para obter a realização de algum serviço, os processos de usuário também fazem uso de mensagens para solicitar que os processos do sistema se encarreguem de executar as tarefas desejadas.

O nível 1 é o único que não é composto por um processo propriamente dito. Ele é uma coleção de rotinas que implementam para os demais níveis os mecanismos que permitem que estes tenham a forma de processos seqüenciais comunicantes. Este nível trata as interrupções, faz o chaveamento de contexto entre os processos salvando e restaurando o estado de execução dos mesmos, e implementa os mecanismos para o envio e recepção de mensagens.

O nível 2 contém processos especializados no tratamento de periféricos, existindo um destes processos para cada tipo de periférico conectável ao sistema. Estes processos também são conhecidos como `device drivers` ou `tasks`, que é o termo que usaremos para referí-los ao longo do texto. Existem duas tasks que não tratam de nenhum periférico, a System Task e a Clock Task. A primeira é responsável por funções especiais que serão vistas em mais detalhes adiante. A outra conta os *ticks* de relógio, controlando as atividades que envolvem o tempo, como a manutenção de temporizadores e o reescalonamento dos processos de usuário em intervalos regulares. O nível 1 e as tasks do nível 2 formam um único código objeto, denominado Kernel do Sistema Minix, sendo ligados no mesmo espaço de endereçamento. Desta forma todas as rotinas e tasks têm acesso direto aos dados globais deste kernel. Porém fora isto, todas as tasks executam independentemente, cada uma rodando uma porção diferente deste código objeto.

O nível 3 contém apenas dois processos cuja função é tratar as mensagens enviadas por programas de usuário para fazer chamadas ao sistema. O Memory Manager (MM) trata as funções envolvendo a criação e o encerramento de processos, além de outras funções que também envolvem a alocação de memória. O File System (FS) trata do gerenciamento de arquivos, sua organização em diretórios, da alocação do espaço em disco e também das demais operações de entrada e saída envolvendo outros periféricos. Os processos do nível 3 fazem a interpretação das chamadas de sistema dos usuários, requisitando para os

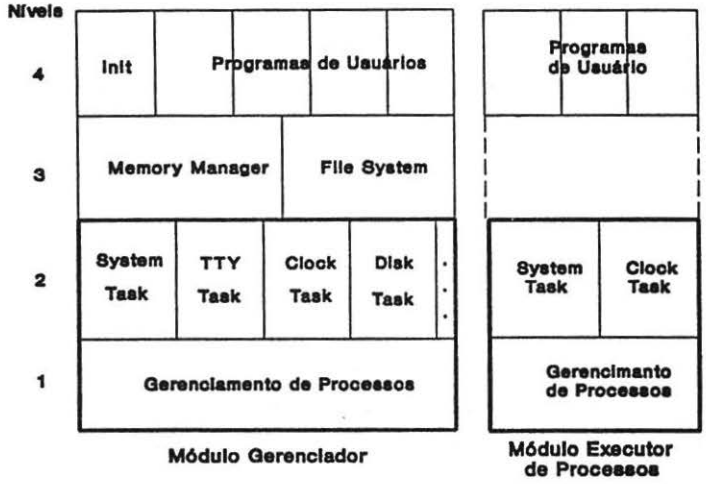

Figura 3.1 - Estrutura em níveis do Sistema Minix

processos do nível 2 a manipulação dos recursos necessários para implementar o serviço desejado, ou seja, o nível 3 implementa as políticas de utilização dos recursos, enquanto que o nível 2 implementa os mecanismos.

Finalmente o nível 4 é composto pelos processos de usuário (programas e utilitários). Neste nível os processos são criados e encerrados dinamicamente, enquanto que nos demais níveis eles são estáticos, sendo criados na inicialização do sistema e jamais terminam de executar (a não ser é claro quando o sistema é desativado).

Ao ser portado para a arquitetura multiprocessada do M3P o sistema Minix foi decomposto em dois tipos de módulos distintos, que chamaremos de Subsistemas. O Módulo

Gerenciador, que roda no processador hospedeiro (mestre), e os Módulos Executores de Processos, que rodam nos processadores escravos (Figura 3.1).

3.1 O Módulo Gerenciador

Como o próprio nome diz, este subsistema é que gerencia as principais atividades do sistema. É ele que faz a gerência dos processos do sistema, controlando a criação e o encerramento dos mesmos, bem como a distribuição da carga de trabalho entre os processadores. Como o hospedeiro é o único processador que tem acesso aos periféricos, também é neste módulo que se encontram as tasks responsáveis pela realização de todas as operações de entrada e saída.

Os servidores (MM e FS) residentes no MG são responsáveis pelo atendimento de todas as chamadas de sistema feitas por processos de usuário, não importando se estes processos de usuário residam no próprio hospedeiro ou em um processador escravo.

3.2 O Módulo Executor de Processos

Este subsistema, que possui uma cópia rodando em todos os processadores escravos, é bastante simples se comparado com o MG, podendo ser considerado um subconjunto deste. Os processadores escravos não possuem periféricos conectados e portanto não são necessárias tasks para tratá-los. As únicas tasks existentes neste subsistema são versões simplificadas das tasks especiais, a Clock Task (para escalonar os processos de usuário residentes no processador escravo) e uma System Task Local.

O MEP não possui nenhum processo servidor (nível 3). As chamadas de sistema feitas por processos de usuário (nível 4) residentes nos processadores escravos são redirecionadas de modo totalmente transparente para os servidores no hospedeiro, como será visto adiante.

3.3 Comunicação entre Processos

A comunicação e sincronização entre processos no sistema Minix é feita trocando mensagens de tamanho fixo. Na implementação do Minix para o IBM-PC o tamanho das mensagens é de 24 bytes. Ao portar o sistema para máquinas com tamanho de palavra diferente este tamanho pode mudar. Existem três primitivas que são usadas para a troca das mensagens:

a) `send( nro_proc, &mensagem)`
b) `receive( nro_proc, &mensagem)`
ou `receive( ANY, &mensagem)`
c) `sendrec( nro_proc, &mensagem)`.

A primitiva *send* usada para enviar uma mensagem recebe como parâmetros a identificação do processo destino e o endereço do buffer contendo a mensagem. A primitiva *receive* usada para a recepção de mensagens recebe como parâmetros o número do processo do qual se deseja receber a mensagem (um valor especial *ANY* pode ser usado indicando que estão sendo aceitas mensagens provenientes de qualquer processo) e o endereço do buffer para colocar a mensagem recebida. A primitiva *sendrec* é uma união das duas anteriores. Uma mensagem é enviada para o processo indicado e imediatamente é aguardada uma resposta deste mesmo processo, sendo que a mensagem recebida é colocada sobre o buffer da mensagem enviada.

Os processos são sincronizados pelo princípio do *rendezvous*, ou seja, o conteúdo da mensagem só é copiado quando tanto o emitente quanto o destinatário tiverem executado as primitivas de comunicação (*send* e *receive*, respectivamente). O processo que executar a sua primitiva primeiro fica bloqueado até que o outro também o faça. Desta forma o conteúdo das mensagens pode ser copiado diretamente da área de dados do emitente para a do destinatário sem passar por buffers intermediários.

Quando os processos comunicantes residem em processadores diferentes a passagem de mensagens entre eles é feita através dos registradores de comunicação. O fato deste hardware de comunicação entre os processadores ser bastante simples faz com que o software tenha que se encarregar de muitos detalhes envolvidos nos procedimentos de envio e recepção de mensagens. Mesmo assim esta solução é razoavelmente eficiente, devido ao tamanho bastante reduzido das mensagens, além de ser extremamente econômica em termos de hardware. A passagem de uma mensagem entre dois processadores é feita basicamente escrevendo-se sucessivamente cada byte do conteúdo da mensagem no registrador de comunicação existente entre os processadores comunicantes.

Obviamente a passagem do conteúdo da mensagem propriamente dito deve ser precedida de um diálogo entre os processadores para saber se os processos emitente e destinatário podem trocar mensagens naquele momento (por exemplo, o processo destinatário pode ainda não ter executado a primitiva *receive*, situação em que a transferência do conteúdo da mensagem deve ser adiada até que ele o tenha feito, para não ser necessário armazenar a mensagem em buffers no processador destino). Este diálogo também se dá através de escritas de valores específicos nos registradores de comunicação.

3.4 A System Task

Cada task no Minix é um processo especializado no tratamento de um periférico ou recurso específico. A estrutura de todas as tasks é bastante semelhante. Elas ficam

repetidamente aguardando a chegada de uma mensagem requisitando algum serviço sobre o recurso que controlam, dão o tratamento correspondente e se for o caso, enviam uma resposta com os resultados para o processo requisitante. Como as tasks são componentes do kernel do sistema elas podem acessar diretamente todos os recursos existentes no equipamento. Uma task pode realizar diretamente instruções de entrada e saída, copiar ou mover dados para qualquer posição da memória e alterar o estado de outros processos.

No Minix várias tarefas do sistema operacional são executadas pelos servidores no nível 3, o MM e o FS. Estes servidores conceitualmente não fazem parte do kernel do sistema, e portanto não podem acessar livremente qualquer recurso. Contudo para desempenhar suas atividades, tanto o MM quanto o FS precisam consultar e atualizar certos dados armazenados em tabelas internas do kernel e ler e escrever em áreas de memória pertencentes a processos de usuário. Para resolver este problema existe no kernel a System Task. Quando o MM ou o FS necessitam ter realizada uma operação restrita ao kernel, eles mandam uma mensagem para esta task requisitando para ela o serviço desejado.

A estrutura da System Task é semelhante a das outras tasks. Ela espera uma mensagem, executa a operação solicitada na mesma e envia de volta outra mensagem com os resultados. Existem mensagens para inserir e deletar dados nas tabelas do kernel quando um processo é criado ou encerrado, para atualizar ou consultar dados referentes a um processo específico, e finalmente para solicitar a movimentação de dados de ou para a área de memória de processos de usuário. Esta última é largamente utilizada tanto pelo MM quanto pelo FS, pois diversas chamadas do sistema requerem o acesso à área de dados do processo que efetuou a chamada. No caso de ser necessário fazer uma movimentação de dados entre as memórias dos processadores a System Task faz uso de operações de DMA. Para realizar as operações que requeiram a manipulação do estado de processos de usuário que residam em um processador escravo, existe uma System Task Local em cada MEP destes processadores, que pode assim atuar sobre estes processos diretamente.

3.5 Realização de Chamadas ao Sistema

Um processo de usuário efetua chamadas ao sistema enviando uma mensagem requisitando o serviço desejado para o servidor apropriado (MM ou FS). Este servidor então aciona as tasks apropriadas para que o serviço solicitado seja efetuado. Na figura 3.2 pode ser vista a implementação de uma requisição de leitura de disco.

O processo de usuário envia a mensagem requisitando o serviço. O FS, após verificar se os parâmetros informados pelo usuário na chamada eram válidos, requisita à task que controla o periférico de disco, através de mensagens, a leitura do bloco apropriado, cujo

conteúdo é lido para um pool de buffers interno ao FS. Após o conteúdo ter sido lido é necessário mover os dados solicitados para o buffer interno do processo de usuário. Como o FS não pode fazê-lo diretamente (apenas processos do kernel podem modificar áreas de memória não pertencentes ao próprio processo) o FS requisita à System Task que faça a movimentação. Feito isto a chamada está completada e uma última mensagem é enviada para informar ao processo de usuário o término da operação requisitada, ou um indicador de erro, se fosse o caso.

Caso o processo de usuário esteja residente num dos processadores escravos a única diferença no atendimento de uma chamada de sistema é que as mensagens de requisição e de resultado (mensagens 1 e 7) seriam mensagens interprocessadoras. A movimentação dos dados, que ainda seria efetuada pela System Task do hospedeiro, seria feita utilizando DMA.

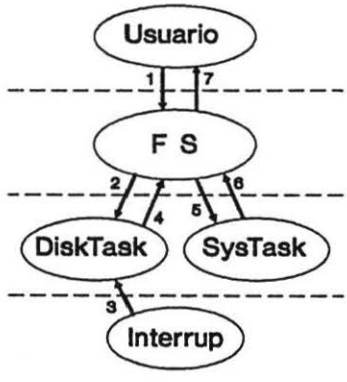

Figura 3.2 - Implementação de uma leitura de disco

Algumas chamadas, como o tratamento de sinais, não requerem a movimentação de dados como no exemplo acima, mas necessitam da manipulação do estado de execução dos processos (alteração do conteúdo da pilha, do *program counter*). Neste caso esta manipulação é feitas pela System Task Local do processador no qual o processo que precisa ter seu estado alterado reside.

4. GERENCIAMENTO DE PROCESSOS

O gerenciamento de processos se constitui numa das funções mais importantes de qualquer sistema operacional. Em arquiteturas multiprocessadas esta atividade ganha uma importância ainda maior pois a alocação adequada de processos a processadores é o principal fator determinante do desempenho geral do sistema.

4.1 A Criação de Processos

A criação de um novo processo no sistema MinixM, em função da compatibilidade com o Unix, é feita através da primitiva `fork`. O maior custo envolvido na criação do processo filho é representado pela criação de uma cópia da imagem do processo pai. Em muitos sistemas fracamente acoplados a cópia de uma imagem de um processo de um processador para outro possui um custo alto se for comparada a cópia da imagem para outra posição da memória associada ao mesmo processador. Por isto muitos sistemas optam por fazer a criação dos processos filhos sempre no mesmo processador. Alguns ainda permitem a criação num processador diferente apenas como procedimento alternativo no caso de não haver mais espaço disponível na memória do mesmo processador.

Na arquitetura M3P existe uma facilidade que auxilia enormemente a criação da imagem do processo filho num processador diferente do pai, o mecanismo de DMA. No M3P a realização de uma operação de DMA entre as memórias de dois processadores é tão eficiente quanto a cópia de dados de uma parte para outra na memória do mesmo processador. Por outro lado o DMA possui a limitação de que as transferências só podem ser realizadas entre o hospedeiro e um processador escravo, e nunca diretamente entre dois processadores escravos.

Em virtude disto um processo filho pode ser criado tanto no próprio processador do pai como num processador para o qual ele possa ser transferido via DMA com a mesma facilidade. Assim sendo, se o pai reside no hospedeiro, o filho poderá ser facilmente criado em qualquer processador. Se o pai já se encontrar num dos processadores escravos, por outro lado, o filho será criado mais eficientemente no mesmo processador ou no hospedeiro. Neste último caso a criação do filho em outro processador escravo implicaria na transferência da imagem do filho inicialmente para o hospedeiro e depois para o processador de destino, duplicando o tempo necessário para a criação da nova imagem. Mesmo assim este procedimento deve ser utilizado caso a criação do processo em outro escravo for necessária para que a carga do sistema fique uniformemente distribuída entre os processadores, como veremos a seguir.

Como um dos fatores determinantes da performance do sistema é o balanceamento equilibrado da carga entre os diversos processadores, o primeiro critério adotado para a escolha do processador onde o filho residirá consiste em usar o processador que estiver menos carregado no momento. O segundo critério usado na alocação de um processo baseia-se no comportamento de execução do processo, isto é, se o processo usa intensivamente a CPU (CPU-bound) ou realiza freqüentemente operações de entrada e saída (IO-bound). Como no M3P apenas o processador hospedeiro tem acesso direto aos periféricos, as operações de entrada e saída são mais eficientes quando realizadas por processos residentes neste processador. Por outro lado, para um processo CPU-bound é interessante residir num dos processadores escravos, pois estes processadores se dedicam exclusivamente a execução de processos, não sendo

interrompidos a todo instante para a realização de operações de entrada e saída. O comportamento de um processo a ser criado pode ser obtido de duas formas: o sistema o deduz baseado no comportamento prévio do pai, já que o filho é uma cópia deste, ou o usuário informa ao sistema o comportamento esperado, através de um indicador no código objeto dos programas, por exemplo.

No sistema MinixM, como em qualquer sistema compatível com o Unix, após um processo filho ter sido criado pela chamada `fork` freqüentemente este filho muda a sua imagem para executar um outro programa, através da chamada `exec`, descartando a imagem anterior, que muitas vezes teve um custo alto para ser criada. Isto faz com que o esforço feito para decidir a melhor alocação do processo filho as vezes seja em vão. Por isto é necessário que os critérios adotados sejam facilmente avaliáveis para não gerar um `overhead` que consuma a capacidade de processamento inutilmente. Estes mesmos critérios podem ser usados na chamada `exec`, onde os resultados para a performance são bem mais efetivos, já que o processo após o `exec` provavelmente passará a executar algum trabalho útil.

4.2 A Chamada `exec`

A chamada `exec` muda a imagem de um processo para que este passe a executar um outro programa. A imagem antiga do processo é quase totalmente descartada e uma imagem nova é montada a partir de um código objeto lido de disco. O fato de a criação da nova imagem requerer um número bastante elevado de operações de leitura de disco para carregar o código objeto determina que esta criação será mais eficiente se for feita no hospedeiro, pois assim o código objeto pode ser lido diretamente para dentro da área alocada para a nova imagem. A criação da nova imagem em um processador escravo implica em que cada bloco de código objeto lido de disco tenha que ser transferido para as placas processadoras em operações de DMA.

Para minimizar o problema acima é dada ao usuário a opção de identificar o código objeto dos programas como sendo CPU-bound ou IO-bound. A nova imagem será totalmente montada no hospedeiro caso o programa esteja explicitamente identificado como IO-bound. Caso o código objeto esteja explicitamente identificado como CPU-bound a imagem será montada diretamente em um processador escravo (o que estiver com menos carga), fazendo-se a transferência de cada bloco do código objeto lido de disco imediatamente para a placa processadora. Esta criação diretamente no processador escravo requer o disparo de uma operação de DMA para cada bloco do código objeto lido de disco, o que é um pouco ineficiente, mas é compensado pelo ganho na performance por causa da criação do processo CPU-bound no processador escravo. Caso o usuário não tenha especificado o comportamento do processo, ele será criado preferentemente no hospedeiro, por esta criação ser mais eficiente. O fato desta última decisão gerar uma má distribuição da carga será contornado com a migração de processos.

4.3 Migração de Processos

Os cuidados tomados para obter um balanceamento do sistema durante o procedimento de criação dos processos não são suficientes para manter o balanceamento da carga ao longo do tempo, pois alguns processos podem encerrar sua execução ou mesmo mudarem de comportamento (passarem a efetuar mais operações de entrada e saída ou usar mais intensamente a cpu). Para resolver isso foi incluída no sistema a possibilidade de fazer a migração de processos.

O esquema de migração de processos atualmente implementado utiliza o seguinte critério. De tempos em tempos o MM verifica a ociosidade de cada processador do sistema. Caso seja detectado algum desbalanceamento significativo, um dos processos mais pesados do processador mais ocupado (mas que não use mais de 50% da capacidade de processamento daquele processador) é migrado para o processador mais ocioso. A migração de um processo consiste basicamente da transferência da sua imagem para a memória de outro processador via DMA, juntamente com a transferência do seu estado de execução, armazenado em tabelas no processador de origem, para as tabelas do processador destino. Este esquema, apesar da simplicidade, se mostra bastante efetivo no sentido de manter a ocupação de todos os processadores relativamente homogênea e evitar a ociosidade de algum processador.

O desempenho proporcionado pela migração de processos depende largamente da natureza dos programas que serão executados no sistema. Se os processos tiverem vida curta ou um comportamento variável ao longo do tempo será necessário fazer realocações com maior freqüência . Um outro fator determinante é a atividade de entrada e saída dos processos. Caso o número de operações de E/S seja grande, o barramento de DMA estará freqüentemente ocupado com as transferências de dados, interferindo com a migração de processos que também utiliza o DMA. Contudo os testes experimentais demonstraram uma ocupação deste barramento inferior a 20% durante a execução de processos típicos do sistema Minix, tendendo para zero com a execução de processos puramente computacionais.

O desempenho de uma operação de DMA é de cerca de 500 kbytes transferidos por segundo. Como a maior imagem de um processo no MinixM é de no máximo 128 kbytes (o tamanho típico é a metade disto) um processo pode ser facilmente migrado em menos de 250 milissegundos. A verificação do balanceamento da carga a cada 2 segundos, mesmo com a utilização das heurísticas de realocação bastante simples desta primeira versão do sistema, já é suficiente para evitar a existência de processadores ociosos enquanto outros estejam sobrecarregados, por períodos significativos de tempo, sem gerar um *overhead* sensível. Caso o sistema esteja executando um número mínimo de processos é possível manter-se a carga praticamente equilibrada entre os processadores, com diferenças sempre inferiores a 20% entre a carga de um processador para outro. Apenas quando o número de processos em execução é

pequeno não é possível manter todos os processadores ocupados, podendo ocorrerem diferenças maiores entre a carga dos mesmos.

Este primeiro protótipo do sistema ficou pronto apenas recentemente (agosto de 1992). Com o mecanismo de migração de processos implementado, pretende-se a partir de agora utilizar o sistema como uma ferramenta para a experimentação e avaliação de diversos algoritmos e critérios mais sofisticados de realocação dinâmica de processos, fazendo uma análise estatística dos mesmos. Também serão estudadas formas de melhor utilizar a capacidade de execução paralela de tarefas, proporcionada por este sistema.

5. CONCLUSÃO

Apesar da simplicidade do hardware da arquitetura M3P, as dificuldades enfrentadas no projeto são as mesmas existentes na construção de sistemas de maior desempenho. Em função disto as várias técnicas e funções de gerenciamento e alocação de processos descritas neste artigo podem ser perfeitamente estendidas para outros sistemas.

As alterações do Minix para torná-lo multiprocessado implicaram em alterações bastante complexas em algumas partes deste sistema, especialmente no gerenciamento e na criação de processos. Muitas funções precisaram ser estendidas, como as primitivas de comunicação, que passaram a incorporar a passagem de mensagens entre processadores. Outras funções novas, até então inexistentes, foram incorporadas, notadamente o balanceamento de carga. Todas estas alterações foram cuidadosamente estudadas de forma a manter total compatibilidade com o sistema original monoprocessado, a nível de software. Também foi tomado o cuidado para que as alterações feitas em cada módulo do sistema não interferissem em outras partes do mesmo. Com isto muitos módulos puderam ser utilizados sem necessitar nenhuma adaptação. Como estas partes do sistema já estavam amplamente testados na versão monoprocessada, isto permitiu que o MinixM continuasse bastante confiável, simplificando os procedimentos de teste.

O sistema resultante, na forma descrita neste artigo, se encontra totalmente implementado e funcional, tanto em termos de hardware quanto software, permitindo ao usuário executar um número maior de tarefas simultaneamente, utilizando de maneira racional toda a capacidade de processamento disponível, de forma totalmente transparente. Apesar disto o sistema continua evoluindo. Estão sendo pesquisados novos critérios de alocação e realocação de processos bem como melhorias e otimizações nos algoritmos atualmente implementados.

BIBLIOGRAFIA

/BAC86/ BACH, M. J. **The Design of the Unix Operating System**, Englewood Cliffs, Prentice Hall, 1986.

/DUN90/ DUNCAN, R. "A Survey of Parallel Computer Architectures", **Computer**, 23(2), Fevereiro 1990.

/CAR89/ CARRERAS, M. S. **Projeto de implementação da Arquitetura M3P**. Trabalho Individual, Porto Alegre, PGCC UFRGS, 1989.

/IBM83/ IBM Corporation, **Personal Computer Technical Reference Manual**, Abril, 1983.

/INT87/ INTEL Corporation, **Microprocessor and Peripheral Handbook**. Volume 1, 1987.

/NAV90/ NAVAUX, P. O. A. et alii, "M3P - Máquina e Sistema Operacional Multiprocessadores", in **XIII Congresso Nacional de Informática**, SUCESU, 1990.

/TAN87/ TANENBAUM, A. S. **Operating Systems - Design and Implementation**. Englewood Cliffs, Prentice-Hall, 1987.

/THO78/ THOMPSON, K. "Unix Implementation", **Bell Systems Technical Journal**, volume 57, pp. 1931-1946, julho-agosto 1978.