PRIMITIVAS DE SINCRONIZAÇÃO PARA UM SISTEMA OPERACIONAL ALTAMENTE PARALELO

Felipe Knop(*)
Laboratório de Sistemas Integráveis - DEE - EPUSP

RESUMO

Este trabalho trata das questões relativas ao desenvolvimento de um sistema operacional multiprocessador de alto desempenho, concentrando-se sobre o problema da proteção de dados.

É descrito neste trabalho o **LSI-SO.01**, o sistema operacional concebido para o **MS-8701** (um minissupercomputador de arquitetura multiprocessadora). O projeto deste sistema operacional tem, como meta, permitir um alto grau de paralelismo em sua execução. Para isso, são pesquisadas melhores organizações, e maneiras mais adequadas de se efetuar a proteção de seus dados.

Um conjunto eficiente de primitivas de sincronização de baixo nível é proposto, sendo mostrado como estas primitivas são empregadas na obtenção de um maior grau de paralelismo. O trabalho apresenta uma implementação de semáforo binário com "busy-waiting", que evita a sobrecarga no barramento, mesmo quando o grau de disputa for alto.

ABSTRACT

In this work, the issues concerning the development of a high-performance multiprocessor operating system are presented, and the problem of data protection is focused.

LSI-SO.01, an operating system for the **MS-8701** minisupercomputer (a multiprocessor machine), is described. The goal of the operating system project is to allow a high degree of parallelism in its execution. To achieve this goal, better organizations and better ways to protect the operating system data are searched.

An efficient set of low-level synchronization primitives is proposed. In addition, the way of using these primitives in achieving a better degree of parallelism is explained. In this work, an implementation of a busy-waiting binary semaphore is described. This implementation avoids bus overloading, even if there is a high degree of contention for the semaphore.

(*) Engenheiro eletrônico (EPUSP 1986); sistemas operacionais, arquitetura de computadores; LSI-DEE-EPUSP Caixa Postal 8174, CEP 05508, Telefone (011) 211-4574 e 815-9322 ramal 270. E-mail: fknop@vme131.lsi.usp.ansp.br

1. Introdução

A tecnologia relacionada aos sistemas multiprocessadores com memória compartilhada tem evoluído muito nos últimos anos. Isto vem possibilitando um aumento no número de processadores empregados nestes sistemas.

Entretanto, à medida em que se usa um número maior de processadores, torna-se cada vez mais difícil utilizá-los de forma efetiva. Este problema aparece de maneira acentuada nos sistemas operacionais, que, na maioria das vezes, acabam sendo responsáveis por um grande desperdício da capacidade de processamento disponível.

Neste trabalho, é descrito o LSI-SO.01 - um sistema operacional multiprocessador - e as providências tomadas no sentido de se obter um alto grau de paralelismo em sua execução. O aumento no grau de paralelismo permite que o sistema operacional possa ser executado, de modo eficiente, em uma arquitetura composta de dezenas ou até centenas de processadores.

O trabalho é organizado da maneira mostrada a seguir. A seção 2 descreve o LSI-SO.01, bem como a máquina para a qual o sistema operacional foi projetado. Na seção 3, são apresentadas as decisões de projeto adotadas para o LSI-SO.01. A seção 4 mostra as primitivas de sincronização criadas, e o modo como estas são utilizadas.

A implementação das primitivas é descrita na seção 5, enquanto que as facilidades incorporadas para auxílio à depuração são mostradas na seção 6. Por fim, a seção 7 faz uma análise dos problemas de contenção, provocados pelo uso de "busy-waiting".

2. O MS-8701 e o LSI-SO.01

O **MS-8701** [13,14] é um minissupercomputador de arquitetura multiprocessadora, projetado no Laboratório de Sistemas Integráveis da Escola Politécnica da USP (LSI-EPUSP). A sua arquitetura pode ser descrita como uma hierarquia de barramentos. Os processadores - até um máximo de 64 - são agrupados em aglomerados ("clusters") de quatro elementos. Cada aglomerado contém, além dos processadores, uma certa quantidade de memória, acessível também pelos processadores dos outros aglomerados.

A entrada/saída (E/S) é feita através de duas placas especializadas, uma para comunicação e outra para controle do armazenamento de massa. Existe ainda uma placa responsável pelo gerenciamento do sistema, cuja principal atribuição é a distribuição das interrupções entre os processadores. Esta placa contém uma memória, acessível por qualquer um dos processadores.

O MS-8701 foi concebido como um computador de "propósito geral", podendo ser usado em aplicações científicas ou comerciais. Assim, além de fornecer uma grande capacidade de processamento, o MS-8701 deve suportar uma alta taxa de comunicação através de terminais e rede, e uma grande capacidade (e velocidade) de armazenamento de massa.

A utilização do MS-8701 pode ser feita em um ambiente onde um número grande de usuários compartilha os seus recursos e, ocasionalmente, são executadas aplicações que fazem uso das dezenas de processadores do sistema.

O **LSI-SO.01** [11] é o sistema operacional desenvolvido para equipar o MS-8701. Como este último foi especificado para suportar diversos tipos de aplicação, o LSI-SO.01 deve apresentar um alto grau de paralelismo, de forma a permitir um bom desempenho em um ambiente onde o sistema operacional é usado intensamente. O grau de paralelismo atingido determina a taxa de aproveitamento efetivo dos processadores.

3. Decisões de projeto

Esta seção apresenta algumas das decisões tomadas no projeto do LSI-SO.01, no sentido de se obter um desempenho adequado.

3.1 - Estruturação do sistema operacional

Entre as diversas formas de se estruturar um sistema operacional multiprocessador (veja [8], por exemplo), escolheu-se a organização ***simétrica***, por ser aquela onde o grau de paralelismo é o mais elevado possível. Cada um dos processadores do MS-8701 executa uma cópia idêntica do sistema operacional, e todos eles trabalham sobre dados compartilhados.

Naturalmente, a organização adotada dá margem ao aparecimento de problemas de concorrência no acesso aos dados compartilhados. Torna-se necessária a escolha de mecanismos de sincronização, que possam resolver estes problemas, mas sem comprometer o desempenho do sistema operacional.

3.2 - Mecanismos de sincronização

Vários mecanismos de sincronização têm sido usados nos sistemas multiprocessadores. Alguns deles, como os ***monitores*** [7], apresentam a vantagem de propiciarem um sistema mais estruturado, isolando, em parte, o programador da arquitetura multiprocessadora. Entretanto, conforme apontado em [4], os monitores acabam desencorajando o aproveitamento de todo o paralelismo possível. Este problema, que não é grave quando se utiliza poucos processadores, tem inibido o uso de mecanismos de

alto nível na sincronização de sistemas com grande número de processadores.

Atualmente, a grande maioria dos sistemas operacionais multiprocessadores tem empregado mecanismos de sincronização de baixo nível, como semáforos e "eventcounts". O uso destes mecanismos dificulta a geração de um sistema correto, mas resulta (se corretamente usados) em um maior desempenho. **Mach** [1], **DG/UX** [9] e **Ultrix** [6] são exemplos de sistemas operacionais que se utilizam deste enfoque. O LSI-SO.01, devido aos seus requisitos de desempenho, também segue este caminho.

3.3 - Escolha das primitivas

Mesmo optando-se por mecanismos de baixo nível, existe uma certa liberdade em relação ao tipo de primitiva de sincronização usado.

A implementação de Bach e Buroff [5] utilizava, como primitiva básica de sincronização, um semáforo contador com suspensão dos processos. Este semáforo era usado não somente para proteger as estruturas de dados, como também para a sincronização de condição.

A vantagem deste procedimento é óbvia: existindo apenas uma primitiva de sincronização, o projeto do sistema operacional fica mais simples. Entretanto, os semáforos contadores e com suspensão dos processos apresentam um "overhead" (tanto de memória como de tempo de execução) que os tornam inadequados para a proteção de seções críticas curtas. Este inconveniente não é apenas "teórico". Em [10], é descrito o problema causado pela primitiva em um "benchmark", onde a máquina com dois processadores apresentava um desempenho inferior ao de uma configuração monoprocessadora.

O problema ocorria porque a primitiva obrigava a suspensão dos processos, causando um "overhead" adicional (devido à troca de contexto) sempre que um processo fosse acessar o semáforo. Este "overhead" aumentava o tempo de travamento da seção crítica, provocando um aumento no número de processos bloqueados na fila do semáforo. Neste caso em particular, uma das providências adotadas foi o uso de alguns semáforos com "busy-waiting".

A situação acima mostra que o uso de um único tipo de mecanismo de sincronização em um sistema complexo pode não ser muito apropriado. Como é natural a existência de seções críticas com características muito diferentes entre si, é mais adequada a criação de primitivas diversas, cada uma adaptada à seção crítica que irá proteger. A idéia é usar primitivas com baixo "overhead" para seções críticas simples, e outras mais elaboradas para a proteção de estruturas de dados mais complexas. Com isso, procura-se obter um equilíbrio entre "overhead" e contenção, e atingir um melhor desempenho.

Assim, são empregadas no LSI-SO.01 diversas primitivas de sincronização, a maioria delas sendo uma variação dos **semáforos** (veja descrição na seção 4).

3.4 - Nível de granularidade

A complexidade e o desempenho de um sistema operacional simétrico são fortemente determinados pela granularidade da proteção das estruturas de dados, ou seja, a quantidade de dados protegidos por um semáforo.

Por simplicidade, poderia ser usado um único semáforo para proteger *todas* as estruturas de dados do sistema operacional. Com isso, qualquer processador, ao entrar em modo supervisor - seja devido a uma chamada de sistema ou a uma interrupção - deveria disputar o acesso ao semáforo, para poder prosseguir em sua execução. E fácil verificar que um sistema desses tem um desempenho idêntico a um **mestre-escravo**, pois apenas um processador por vez pode executar o sistema operacional.

Um sistema mais eficiente poderia ser construído dividindo-se os dados em grandes grupos, cada um deles sendo protegido separadamente. Esta divisão seria feita de acordo com os módulos do sistema operacional. Assim, teríamos, por exemplo, um semáforo para o gerenciador de processos, outro para o gerenciador de arquivos, e assim por diante. Apesar de melhor que a alternativa anterior, esta também não é satisfatória, principalmente se for usado mais de 4 ou 5 processadores.

Pode-se perceber que o grau de paralelismo do sistema operacional aumenta, à medida em que diminui a quantidade de dados protegida de cada vez (maior número de grupos). Isto ocorre porque, com o aumento do número de semáforos (e a consequente diminuição na quantidade de dados protegida por cada um), diminuem as possibilidades de disputa por uma determinada estrutura de dados. Outro efeito importante é a diminuição no tempo em que um certo dado fica travado.

Para o LSI-SO.01, a política adotada é a de se elevar o nível de granularidade até um ponto onde um aumento adicional causaria um incremento descabido no número de semáforos ou na complexidade do sistema.

4. Descrição e uso das primitivas

O LSI-SO.01 oferece um conjunto de primitivas de sincronização, cada uma ajustada a uma determinada situação.

4.1 - Travas simples

As **travas simples** são basicamente semáforos de baixo "overhead" com "busy-waiting". Estas primitivas são usadas na proteção de seções críticas muito curtas e pouco disputadas.

Uma situação típica onde as travas simples são aplicadas acontece no incremento ou decremento de variáveis globais, ou na manipulação de mapas de bits. Para manipulações mais complexas, este mecanismo não é usado, pois não incorpora um suporte à depuração, nem meios de diminuir a carga imposta ao barramento.

4.2 - Operações atômicas

As **operações atômicas** fornecem uma maneira de se efetuar um incremento ou decremento indivisível em uma variável global. Apesar desta operação poder ser protegida por travas simples, as operações atômicas implicam em um "overhead" menor, além de serem mais visíveis ao programador. Este mecanismo possibilitaria uma redução mais significativa no "overhead", em arquiteturas com suporte **em hardware** para incremento e decremento atômicos.

4.3 - Semáforos não bloqueantes

Os **semáforos não bloqueantes** são semáforos binários com "busy-waiting" e política de escalonamento FCFS ("First-Come, First-Served"), usados na proteção de seções críticas mais complexas e/ou muito disputadas. A sua implementação evita a sobrecarga do barramento quando a contenção pela seção crítica for alta. A utilização da política FCFS evita o fenômeno da "starvation", cuja possibilidade de ocorrência aumentaria com o incremento no número de processadores.

Os semáforos não bloqueantes constituem-se nas principais primitivas de exclusão mútua do LSI-SO.01, sendo empregados principalmente na proteção de listas ligadas e tabelas.

As listas ligadas com acesso global devem ter sua manipulação protegida, pois inserções e retiradas simultâneas podem deixar as litas em um estado inconsistente. O método usado em sua proteção é o seguinte: um semáforo é alocado para a cabeça da lista, devendo ser obtido a cada operação de inserção, retirada ou varredura da lista.

Para se manipular os elementos de uma lista, é utilizado um semáforo para *cada* um deles. Isto permite que elementos diferentes de uma lista possam ser lidos ou alterados concorrentemente. Se a manipulação não envolver inserções, retiradas ou varreduras, não há a necessidade de obtenção do semáforo da lista.

A figura 1 mostra uma situação típica no LSI-SO.01: uma tabela organizada como uma lista duplamente ligada, protegida por semáforos.

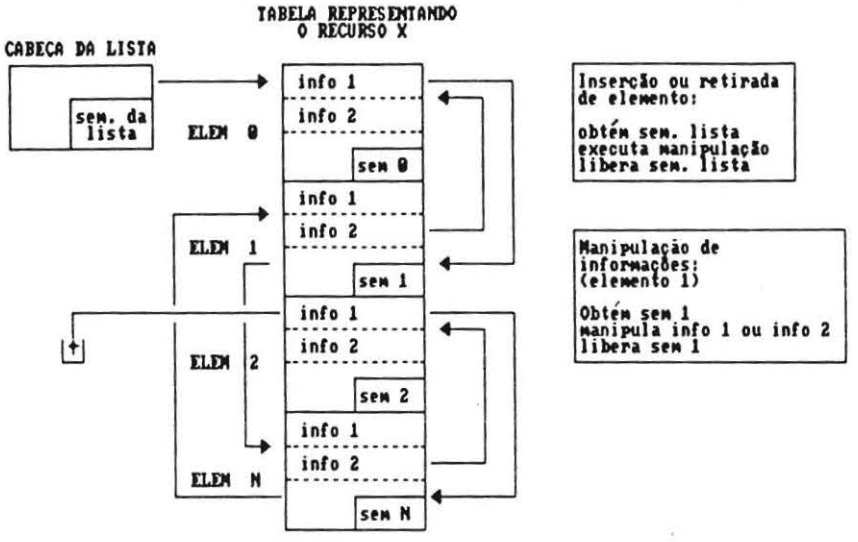

Figura 1 - Tabela do LSI-SO.01 organizada como uma lista ligada

Existem casos onde a proteção explícita dos dados pode ser dispensada. Diversas vezes, o algoritmo empregado na manipulação de algum dado impede que este seja ***escrito*** simultaneamente por vários processadores. Se isto ocorrer, não há necessidade de proteger o acesso, pois leituras concorrentes com ***uma*** escrita não se constituem em seções críticas(*).

A análise das situações onde o uso de semáforos é desnecessário torna-se importante, na medida em que contribui para o aumento no grau de paralelismo do sistema operacional.

4.4 - Semáforos bloqueantes

Os **semáforos bloqueantes** possuem também uma política de escalonamento FCFS, mas, ao contrário dos semáforos do item anterior, suspendem os processos que estiverem à espera pelo semáforo.

Pelas suas características, os semáforos bloqueantes impõem uma carga menor ao barramento (não há "busy-waiting"). Entretanto, existe, neste caso, o "overhead" da troca de contexto, que na

---

(*) Esta afirmação vale se o dado puder ser lido ou escrito com um único acesso à memória.

maioria dos sistemas (incluindo o LSI-SO.01) não pode ser desprezado.

No LSI-SO.01, as estruturas de dados são protegidas através dos **semáforos não bloqueantes**, a não ser quando o tempo de travamento depender de fatores que não apenas o tempo de processamento. Assim, se, durante uma seção crítica, houver uma espera pelo término de uma operação de E/S, a proteção será feita com um semáforo bloqueante. Este procedimento impede que um eventual "busy-waiting" prolongue-se por tempo demasiado.

Os semáforos bloqueantes também são empregados quando um outro semáforo bloqueante for obtido dentro de uma seção crítica. Com isso, há uma tendência deste tipo de semáforo ser mais utilizado à medida em que se aproxima das camadas mais externas do sistema operacional. Políticas semelhantes a esta são adotadas em [6,15,3].

Outro uso importante para os semáforos bloqueantes é na **sincronização de condição**, ou seja, na espera e sinalização de eventos.

4.5 - Semáforos de leitura/escrita

Os **semáforos de leitura/escrita** constituem-se em um caso mais geral dos semáforos não bloqueantes, permitindo o controle sobre o acesso a estruturas de dados apenas lidas em algumas situações e modificadas em outras. Este tipo de semáforo foi criado objetivando a obtenção de um melhor grau de paralelismo no acesso a algumas estruturas de dados.

Os semáforos de leitura/escrita são usados na substituição aos semáforos não bloqueantes, em casos onde a estrutura de dados protegida é alterada com baixa frequência. Sua aplicação, nesses casos, ajuda a diminuir a disputa pelo acesso à estrutura de dados, pois permite a execução concorrente de operações onde esta não é modificada.

As principais operações que se podem beneficiar dos semáforos de leitura/escrita são o acesso a variáveis ou conjunto de variáveis raramente modificadas, e listas frequentemente percorridas, mas com uma baixa taxa de inserções ou retiradas.

Devido ao maior "overhead", os semáforos de leitura/escrita só são utilizados quando a manipulação executada for longa, e a taxa de leitores for muito maior que a de escritores. Entretanto, à medida em que aumenta o número de processadores, o desempenho passa a depender mais do paralelismo, e os semáforos de leitura/escrita tornam-se mais vantajosos.

As primitivas implementadas no LSI-SO.01 possuem uma capacidade adicional: a possibilidade de "promoção" e "rebaixamento" dos semáforos, ou seja, alteração do tipo de

alocação de um semáforo após a sua obtenção. Isto é usado, por exemplo, em casos onde uma lista é percorrida, e dependendo de alguma condição, um elemento acaba sendo removido.

5. Implementação das primitivas

As primitivas foram implementadas de forma a apresentarem um "overhead" compatível com a sua utiliação. Assim, vale a seguinte relação entre os "overheads":

trava simples < semáforos não bloqueantes < semáforos de leitura/escrita < semáforos bloqueantes

As **travas simples**, **operações atômicas** e **semáforos bloqueantes** apresentam uma implementação imediata. São descritos a seguir aspectos da implementação dos **semáforos não bloqueantes** e **semáforos de leitura/escrita**.

5.1 - Semáforos não bloqueantes

O algoritmo que resulta em uma política FCFS é baseado em [2] (é também de onde vem o nome das variáveis descritas a seguir). Cada semáforo é composto por três variáveis: **np**, **next** e **avail**. As duas primeiras controlam o acesso FCFS ao semáforo (através de um sistema de "tickets"), enquanto a última indica se este está livre ou não (*).

As variáveis **np** e **next** são sempre incrementadas, mas o algoritmo é imune a possíveis "overflows", continuando a funcionar mesmo quando estas voltarem a zero.

A **obtenção** e **liberação** do semáforo são efetuadas como mostrado a seguir. **Prnp** é uma variável local por processo, usada para guardar o valor do "ticket" recebido.

(*) neste trabalho, os semáforos não bloqueantes são binários.

obtenção:

(1) Incrementa **np** atomicamente de 1, guardando o valor incrementado em **prnp**.
(2) Enquanto **prnp** <> **next** espera (em "busy-waiting").
(3) Enquanto **avail** < 1 espera (em "busy-waiting").
(4) Decrementa **avail** de 1.
(5) Incrementa **next** de 1.

liberação:

(6) Incrementa **avail** de 1.

No algoritmo acima, (1) representa a obtenção do "ticket". E neste ponto que é feita a arbitração da ordem de chegada dos processos.

Para evitar que o "busy-waiting" congestione o barramento, é necessário reduzir a taxa na qual os testes em (2) e (3) são feitos. Entretanto, esta taxa deve ser alta o suficiente para não elevar em demasia o tempo de latência das primitivas (tempo entre a liberação do semáforo e sua obtenção por outro processo).

A solução encontrada foi fazer os processos que estiverem mais à frente na "fila" testarem as variáveis com maior frequência, ou seja, quanto menor for a diferença entre **prnp** e **next**, maior deverá ser a taxa na qual o teste em (2) é feito. Assim, o barramento é mais ocupado pelos processos que estão na iminência de poderem seguir adiante no algoritmo. O parâmetro **ESPERA_NEXT** é multiplicado pela diferença entre **prnp** e **next**, controlando a taxa de acesso.

No caso do processo que estiver esperando em (3), a taxa de acesso a **avail** é fixa, sendo superior à efetuada em (2), pois este processo será o próximo a conseguir a posse do semáforo. Nesta etapa do algoritmo, a taxa de acesso é controlada pelo parâmetro **ESPERA_AVAIL**.

O "overhead" medido para as primitivas de obtenção e liberação foi de 16,9 microssegundos (MC68020 a 16Mhz), aproximadamente quatro vezes superior ao da **trava simples**.

5.2 - Semáforos de leitura/escrita

Uma decisão importante tomada na implementação dos semáforos de leitura/escrita foi a prioridade a ser dada aos acessos. Um esquema de prioridades explícitas poderia resultar em "starvation" quando a taxa de acesso ao semáforo fosse muito alta. A solução

encontrada foi novamente empregar-se os "tickets", garantindo um acesso FCFS ao semáforo.

A idéia básica é permitir o acesso concorrente dos leitores, mas só até o aparecimento do primeiro escritor. Quando isto acontecer, os leitores que vierem a seguir permanecerão na fila atrás do escritor. Após o escritor ter liberado o semáforo, todos os leitores da fila (até o próximo escritor) podem prosseguir. A figura 2 exemplifica o funcionamento desta política.

ORDEM DE CHEGADA

número do processo	3	2	1	5	6	4	7	8
tipo do acesso	leit.	leit.	leit.	escr.	escr.	leit.	leit.	escr.

ACESSO AO RECURSO

(3, 2, 1) (5) (6) (4, 7) (8)

Figura 2 - Exemplo de funcionamento dos semáforos de leitura/escrita

Como se pode notar, os leitores e escritores não ficarão indefinidamente sem acessar o recurso.

A implementação das primitivas de "promoção" e "rebaixamento" de semáforos requer alguns cuidados. Na "promoção", é necessário evitar a "starvation", causada pela chegada contínua de novos leitores, ou o "deadlock", provocado quando vários processos requerem a "promoção" simultaneamente. O procedimento básico da primitiva de "promoção" é o seguinte:

1) Marca semáforo como sendo de *escrita*, para inibir a chegada de novos leitores. Se não é o primeiro a se "promover", espera a sua vez.

2) Decrementa o número de leitores, pois o processo atual está deixando de ser leitor.

3) Espera o número de leitores chegar a zero.

6. Facilidades para depuração

Para contornar as dificuldades decorrentes do uso de mecanismos de sincronização de baixo nível, é fornecido um suporte para facilitar a utilização correta das diversas primitivas. Em cada operação de travamento, a identificação do processo correspondente é guardada na estrutura de dados que representa o semáforo. Ao mesmo tempo, o endereço do semáforo obtido é armazenado na estrutura de dados associada ao processo.

Este mecanismo de armazenamento permite que se descubra erros elementares na utilização dos semáforos. Assim, se um processo tentar travar um semáforo já travado, ou destravar um que não havia sido obtido, a primitiva detectará a situação.

Erros mais sutis podem ser descobertos através das **asserções**, expressões que comparam dinamicamente a situação de travamento dos semáforos com o desejado pelo projetista do sistema. Se a comparação resultar diferente, é sinal de que há erro, e a execução do sistema operacional é interrompida.

As asserções são usadas, por exemplo, em rotinas que são chamadas com o processo já de posse de algum semáforo. Através das asserções, verifica-se, a cada chamada, se o semáforo está mesmo travado. Isto é particularmente útil quando a rotina for chamada de vários pontos diferentes.

São fornecidos quatro tipos de asserções:

- verifica se o processo corrente possui um dado semáforo.
- verifica se o processo corrente já liberou um dado semáforo.
- verifica se o processo corrente possui **apenas** um dado semáforo.
- verifica se o processo não possui nenhum semáforo.

Naturalmente, a implementação das asserções só é possível devido ao mecanismo de armazenamento de semáforos.

7. Análise dos problemas de contenção

A disputa pelo acesso às estruturas de dados causa dois efeitos negativos no desempenho: o atraso dos processos que devem ficar à espera pela liberação das seções críticas, e a saturação do barramento, provocada pelas primitivas com "busy-waiting". A adoção de primitivas e nível de granularidade adequados colaboram na diminuição do problema, mas nem sempre o projetista tem condições de tomar as melhores decisões à priori.

O grau de disputa pelos semáforos é uma função dos programas que estão sendo executados, e muitas vezes pode ser bem diferente do inicialmente estimado. Deste modo, fica clara a necessidade de uma fase de sintonia, onde as primitivas são ajustadas, e as estruturas de dados mais acessadas e disputadas são descobertas.

É apresentado, a seguir, um programa desenvolvido para auxiliar na descoberta dos pontos de maior disputa. Também é descrita uma análise do impacto causado ao barramento pelas primitivas que fazem uso de "busy-waiting".

7.1 - Disputa pelas estruturas de dados

O programa ***trace*** [16] foi projetado para facilitar as fases de sintonia e análise de desempenho do sistema operacional. O que o ***trace*** faz é coletar e analisar eventos enviados pelo sistema operacional, fornecendo informações como o número de vezes que uma certa porção de código foi executada, ou o tempo médio de execução de uma rotina.

Através do **trace**, dados importantes podem ser obtidos:

- número de acessos a cada semáforo

- tempo médio entre a requisição e a liberação do semáforo

Estes dados fornecem subsídios para a determinação da contenção existente em cada estrutura de dados. O grau de disputa pode ser calculado, comparando-se o tempo entre a requisição e a liberação, nos ambientes mono e multiprocessador. Como este tempo engloba também a espera pelo semáforo, a diferença entre eles indicará o grau de contenção.

Para que os dados acima possam ser obtidos, são inseridas no sistema operacional rotinas para a "geração" de eventos *antes do travamento* e *depois da liberação* dos semáforos.

7.2 - Impacto do "busy-waiting" no barramento

Para verificar o comportamento das primitivas não bloqueantes com relação à contenção, foi utilizado o programa **LIDEX** [12], um simulador de arquiteturas. Simulou-se um sistema com memória compartilhada, onde os processadores eram conectados por um barramento (cujo árbitro implementa uma política "round-robin" de atendimento aos processadores). Os processadores foram projetados com um conjunto relativamente limitado de instruções, que incluia o "*test-and-set*".

Cada um dos processadores foi carregado com um programa que executava um protocolo de exclusão mútua e, a seguir, incrementava uma ou mais variáveis compartilhadas. O objetivo da simulação foi verificar o comportamento dos programas, quando se alterava a primitiva de sincronização e o comprimento da seção crítica.

Dois mecanismos foram comparados: a **trava simples** e o **semáforo não bloqueante**. No caso deste último, procurou-se medir o efeito da sintonia dos seus parâmetros **ESPERA_NEXT** e **ESPERA_AVAIL** na contenção e tempo de latência.

Os experimentos realizados avaliaram o tempo de execução do programa (dependente da contenção e da latência da primitiva), e a ocupação do barramento durante este tempo. A simulação foi feita para 10 processadores.

O gráfico 1 mostra o comportamento das primitivas quando é variado o comprimento da seção crítica. No eixo **Y**, temos o tempo de execução do programa (em número de ciclos, normalizado para o teste com a trava simples e seção crítica mais curta). O eixo **X** apresenta o comprimento da seção crítica, em número de incrementos nas variáveis compartilhadas.

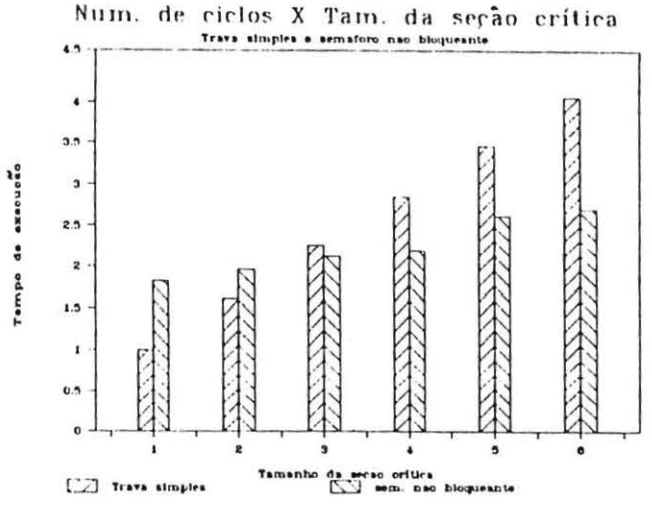

Gráfico 1 - Número ciclos X tamanho da seção crítica

Nos experimentos que resultaram no gráfico 1, notou-se que, para as travas simples, o barramento permaneceu ocupado, em média, 96%. Para os semáforos não bloqueantes, este número caiu para 68%, tendendo a diminuir para as seções críticas mais longas.

O gráfico 2 refere-se aos experimentos onde foi variado o parâmetro **ESPERA_NEXT**. Também neste caso, foi medido o tempo de execução do programa por todos os processadores. A taxa de ocupação do barramento variou entre 70% (**ESPERA_NEXT** = 3) e 63% (**ESPERA_NEXT** = 7).

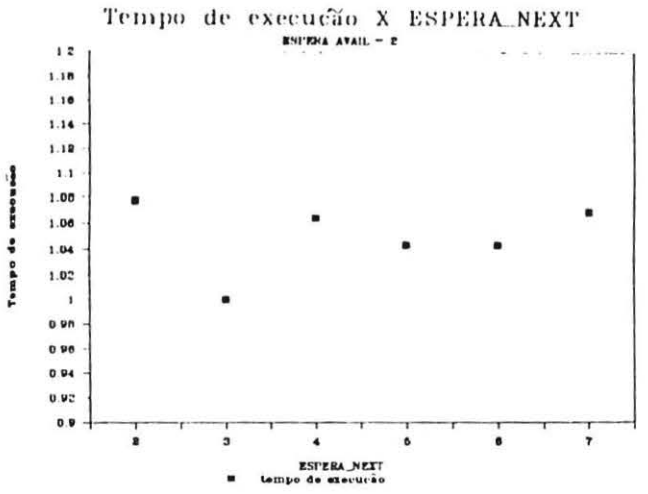

Gráfico 2 - Tempo de execução X ESPERA_NEXT (ESPERA_AVAIL=2)

As simulações permitem que sejam tiradas as seguintes conclusões:

- em estruturas de dados muito disputadas, as **travas simples** só são adequadas quando o acesso for muito rápido. Para os demais casos, os **semáforos não bloqueantes** devem ser usados, pois implicam em uma carga menor no barramento.
- a sintonia dos parâmetros **ESPERA_NEXT** e **ESPERA_AVAIL** é uma tarefa complexa. O acerto destes parâmetros depende da velocidade de processamento e do comprimento médio das seções críticas.

8. Conclusão

Este trabalho apresentou as técnicas usadas na sincronização do sistema operacional LSI-SO.01. Foi definido um conjunto de primitivas de sincronização, tendo sido descrita a utilização de cada uma.

O trabalho teve como objetivo obter um sistema operacional capaz de suportar - com um desempenho adequado - um grande número de processadores. Para isso, esforços foram feitos no sentido de se maximizar o grau de paralelismo em sua execução.

Foi apresentada uma implementação de semáforo binário com "busy-waiting" que, além de usar uma política de atendimento FCFS (mais "justa"), permite a redução da carga imposta ao barramento. Isto é obtido sem um aumento significativo no tempo de latência da primitiva.

9. Bibliografia

[1] ACCETTA,M. et alii. Mach: a new kernel foundation for UNIX development. In: USENIX CONFERENCE, Atlanta, Georgia, Summer 1986. **PROCEEDINGS** p. 93-112.

[2] AMIT,N. & HOFRI,M. A simple semaphore-queue management for multiprocessing systems. **Operating Systems Review**, New York, v.14 n.3 p.13-5, July 1980.

[3] ANDERSON,D.P. & TZOU,S.Y. **The Dash local kernel structure**. Berkeley, Computer Science Department of University of California, 1988. (Report no. UCB/CSD 88/463).

[4] ANDRE,F.; HERMAN,D.; VERJUS,J.-P. **Synchronization of parallel programs**. Cambridge, Massachusetts, MIT, 1986. 110 p.

[5] BACH,M.J. & BUROFF,S.J. Multiprocessor UNIX operating systems. **AT&T Bell Laboratories Journal** v.63 n.8 p.1733-49, Oct. 1984.

[6] HAMILTON, G. & CONDE, D. S. An experimental symmetric multiprocessor Ultrix kernel. In: USENIX CONFERENCE. Dallas, Texas, Winter 1988. **PROCEEDINGS**. p. 283-90.

[7] HOARE,C.A.R. Monitors: an operating system structuring concept. **Communications of the ACM**, New York, v.17 n.10 p.549-57, Oct. 1974.

[8] HWANG,K. & BRIGGS,F.A. **Computer architecture and parallel processing**. New York, Mc Graw Hill, 1984. 846 p.

[9] KELLEY,M.H. Multiprocessor aspects of the DG/UX kernel. In: USENIX CONFERENCE. San Diego, California, Winter 1989. **PROCEEDINGS**. p.85-99.

[10] LEE,T.P.; LUPPI,M.W.; MENNINGER,R.E. Solving performance problems on a multiprocessor system. In: USENIX CONFERENCE. Phoenix, Arizona, Summer 1987. **PROCEEDINGS**. p. 399-405.

[11] MIDORIKAWA,E.T.; KNOP,F.; BRANCO,R.D.; YU,W.K. LSI-SO.01: um sistema operacional multiprocessador. In: SIMPOSIO BRASILEIRO DE ARQUITETURAS DE COMPUTADORES, 2, Aguas de Lindóia, São Paulo, set. 1988. **ANAIS**, p. 3.A.1.1 - 6.

[12] MOREIRA,J.E. **LIDEX: um sistema para descrição, simulação e análise de arquitetura e organização de computadores**. São Paulo, Escola Politécnica da Universidade de São Paulo, 1990. Dissertação de mestrado.

[13] SANGIORGIO,C.A. Computadores paralelos com arquitetura de dutos. In: SIMPOSIO BRASILEIRO DE ARQUITETURAS DE COMPUTADORES, 2, Aguas de Lindóia, São Paulo, set. 1988. **ANAIS**, p. 11.B.5.1 - 9.

[14] TAKEDA,J.H. et alii. Projeto minissupercomputador: características gerais do sistema MS-8701. In: SIMPOSIO BRASILEIRO DE ARQUITETURAS DE COMPUTADORES, 2, Aguas de Lindóia, São Paulo, set. 1988. **ANAIS**, p.10.1.1-8.

[15] TEST,J.A. Multi-processor management in the Concentrix operating system. In: USENIX CONFERENCE. Denver, Colorado, Winter 1986. **PROCEEDINGS**. p. 173-82.

[16] TRAN,A.P. & SUZUKI,R.S. **Especificação funcional do utilitário Trace**. São Paulo, LSI-EPUSP, 1989.