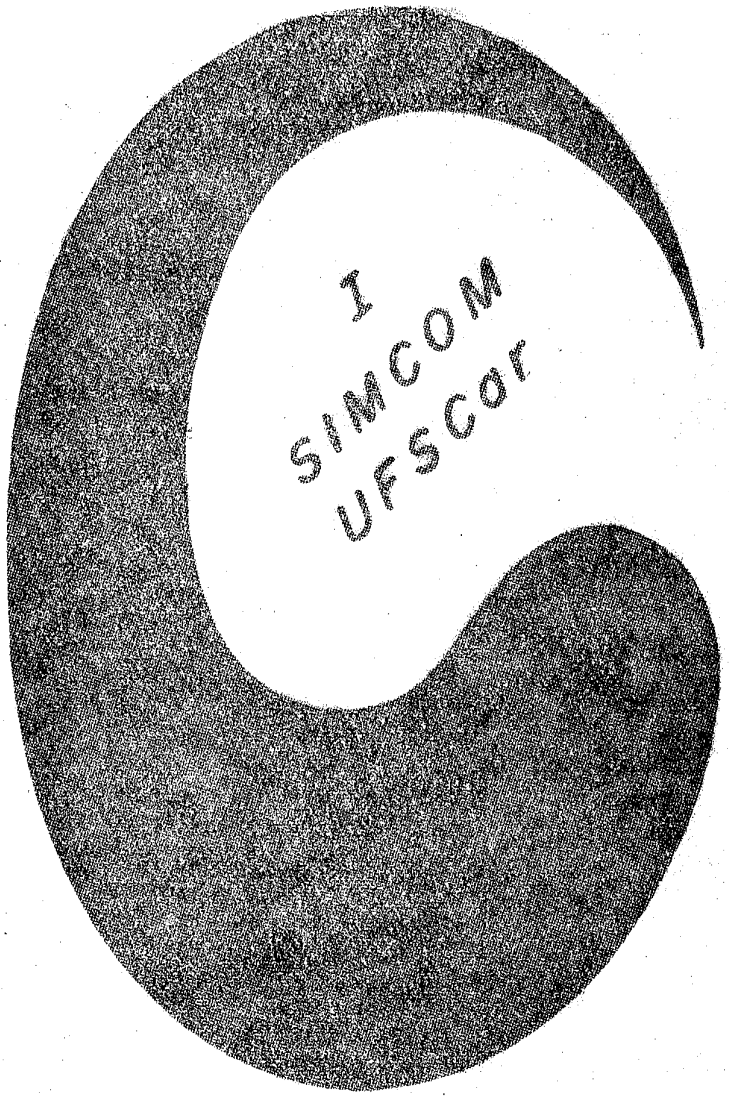
NIVERSIDADE FEDERA DE SÃO CARLOS

DEPARTAMENTO DE COMPUTAÇÃO E ESTATÍSTICA



I SIMPÓSIO DE COMPUTAÇÃO:
SISTEMAS OPERACIONAIS

ARLOS (SP), 15 a 18 de SETEMBRO

A ARQUITETURA DO NÚCLEO DE UM SISTEMA OPERACIONAL

Michael Stanton

Departamento de Informática
PUC/RJ

RESUMO

Durante os últimos dez anos, os conceitos básicos de arquitetura de sistemas operacionais sofreram modificações profundas, com o objetivo de construir sistemas de maior confiabilidade e segurança.

Em particular, a parte central, ou núcleo, de um sistema operacional moderno difere radicalmente do sistema tradicional. Este artigo considera algumas das características da arquitetura do núcleo, com um exemplo.

1. INTRODUÇÃO

Para combater complexidade de Software o remédio é estruturá-lo, ou seja, segmentá-lo em componentes de baixa complexidade individual. Na área de sistemas operacionais, o conceito essencial de estruturação é o processo, entidade da qual foi eliminado todo vestígio do paralelismo, que caracteriza o sistema como um todo. Para ordenar as interações entre processos foram introduzidas funções primitivas de sincronização e intercomunicação. Entre os mecanismos criados para permitir estas funções citamos semáforos, troca de mensagens e monitores.

O suporte para o conceito de processo, junto a implementação de comunicação entre processos, se encontra no núcleo (em inglês, *hernel*) do sistema operacional. O núcleo implementa as operações primitivas usadas pelos processos, através de rotinas ativadas por *"traps"* (SVCs). Durante execução do núcleo a seu pedido, o processo tem sua execução suspensa, voltando a ser reiniciada, quando o processo for novamente despachado pelo núcleo.

Outra função fundamental do núcleo é controlar os diversos

periféricos, que têm suas atividades iniciadas através de instruções especiais *(Iniciar E/S)* executadas pelo núcleo. Ao terminar a execução do comando dado, o periférico ativa novamente o núcleo, através de uma interrupção, que causa a execução de uma rotina apropriada do núcleo.

Deve-se notar a simetria entre processos e periféricos — o processo chama o núcleo através de um *"trap"*, e é reativado por ser despachado; o periférico chama o núcleo através de uma interrupção, e é reativado por execução de *Iniciar-E/S*.

Adicionalmente, o núcleo pode ser ativado através de interrupções de um relógio.

O núcleo é o único componente do sistema que toma conhecimento de interrupções. Fora dele, todo Software é executado por processos, cada um com seu processador virtual monoprogramado. Para estes processos, somente a velocidade do seu processador virtual é afetada pela ocorrência de interrupções.

Foi logo compreendido que a redução da complexidade do núcleo facilitaria a sua confiabilidade da sua implementação.

Uma vez feito o núcleo, com as necessárias facilidades básicas, ele serviria de fundação para erguer uma estrutura complexa de funções do sistema operacional, e apoio para o usuário, implementados na forma de processos.

Além de proporcionar maior confiabilidade no desenvolvimento de sistemas operacionais, a estruturação deste Software frequentemente tem sido utilizada para criar sistemas seguros, isto é, sistemas que garantem que acesso a dados sob sua responsabilidade serão controlados e protegidos. Neste caso, além de prover as funções primitivas necessárias para o suporte de processos e para E/S, o núcleo também assume total responsabilidade para a segurança do sistema, inclusive dos componentes construídos fora do núcleo.

É notória a falta de segurança encontrada em sistemas tradicionais. Os mecanismos para a proteção de informação ou inexistem, ou são construídos em cima de uma base insegura, ou são muito particulares para resolver o problema genérico.

O objetivo de segurança tem algumas consequências para a arquitetura de um sistema operacional. Para ser confiável, exigimos que a base, da qual depende a segurança, seja pequena. Como conse

quência, funções incluídas no núcleo para aumentar desempenho ou para facilitar a programação em níveis superiores não contribuem para o objetivo de segurança, e deverão ser excluídas do núcleo.

Por outro lado, se quisermos que nenhuma decisão sobre segurança dependa de Software externo ao núcleo, somos forçados a incluir dentro do núcleo funções tradicionalmente implementadas em nível superior.

2. CONSIDERAÇÕES TÉCNICAS

Para permitir especificar adequadamente um núcleo, será necessário examinar as restrições técnicas impostas a ele. Serão discutidas a política de segurança, as primitivas a serem implementadas, o Hardware disponível e o desempenho.

Política de Segurança

O objetivo mais comum é de segurança de informação, significando que um usuário somente pode ter acesso aquela informação para a qual este acesso foi explicitamente permitido. É claro que quanto mais sofisticada for a política adotada, mais numerosos e complexos serão os mecanismos necessários no núcleo.

Pode-se identificar três características de uma política de segurança:

i) *Quais objetos são protegidos?*

Algumas alternativas incluem processos, segmentos, páginas, arquivos, mensagens, periféricos, ou várias destas.

ii) *Qual é o tamanho do objeto protegido?*

O domínio de um processo pode ficar constante, ou mudar com uma chamada de procedure.

iii) *Qual é a precisão da política?*

Será possível identificar precisamente os usuários permitidos a ter acesso um dado arquivo?

Primitivas Implementadas

O núcleo suportará operações sobre os objetos protegidos. Será fixo o número destes objetos, ou será possível criar e destruí-los? As operações serão rudimentares ou sofisticadas? A escolha de primitivas neste nível pode eliminar ou dificultar a provisão de certas facilidades para processos. Por exemplo, a ausência de primitivas de E/S assíncrona no núcleo exclui a possibilidade de um só processo implementá-lo. Por outro lado, é possível implementar monitores usando semáforos, mas pouco conveniente fazê-lo desta maneira. A escolha determinará parcialmente a complexidade do núcleo.

Características do Hardware

A complexidade do núcleo é também determinada em grande parte pela complexidade do Hardware que ele coordena. Por exemplo,um sistema de processadores múltiplos poderá precisar de coordenação dos processadores dentro do núcleo, enquanto um processador simples não precisa dela. Frequentemente os acessos a memória feitos por processadores centrais são condicionados por mecanismos de proteção implementados em Hardware (mapas de páginas, chaves de proteção, registradores de relocação). Tipicamente, os controles são acionados por Software privilegiado, e então podemos executar Software arbitrário, confrantes que não serão ultrapassados os limites registrados nos controles.

O caso de canais, ou processadores de E/S, é tipicamente bem menos sofisticado, sendo os controles de acesso menos convenientes, onde eles existem. Por exemplo, canais normalmente não utilizam mapeamento de endereços por Hardware, portanto exigindo uma verificação pelo núcleo dos endereços absolutos usados na transferência.

Se é primitivo o apoio de Hardware para controle de acesso por canais à memória principal, no caso de memória secundária normalmente inexiste qualquer apoio. Portanto, o núcleo terá a incumbência de verificar endereços em memória secundária e comandos de E/S para garantir qualquer proteção à informação ali guardada.

Finalmente, a falta de uniformidade das interfaces dos diversos tipos de periférico requer que uma boa parte do núcleo seja dedicada aos diversos controladores necessários, um para cada interface

distinta.

Desempenho

Se a restrição de bom desempenho for crítica, a complexidade do núcleo poderá aumentar bastante. Por exemplo, decisões sobre o escalonamento de processos não são diretamente ligadas à segurança e portanto poderão ser tomadas por um processo fora do núcleo, e interrogado por este. Porém, a cada troca de processo, haveria a necessidade de chamar o escalonador, o que poderia ocasionar um custo inaceritável. Para resolver o problema, a resposta comum é de integrar escalonamento dentro do núcleo.

3. GERÊNCIA DE RECURSOS

As restrições técnicas impostas no núcleo afetarão o tamanho e complexidade deste. Examinaremos agora algumas considerações que podem simplificar o núcleo dentro das restrições impostas.

Uma maneira eficaz de reduzir a complexidade do núcleo é dentro do possível deslocar do núcleo para processos as funções que gerenciam um dado tipo de recurso. Identificamos três aspectos da gerência de recursos que podem ser assim deslocados para fora do núcleo: integridade de tipo, a política de gerência, e denominação. Discutimos cada um a seguir.

Integridade de Tipo

A ação mais radical é eliminar totalmente do núcleo tanto o recurso como seu suporte em Software. Consequentemente, o núcleo não mais protege este recurso, e qualquer proteção oferecida é a responsabilidade do Software onde ele é usado. O efeito desta decisão é de descentralizar a gerência do recurso. Em vez do sistema administrar todos os recursos de um dado tipo para todos os processos, estes recursos agora são divididos em conjuntos menores, administrados em separado.

Os custos e benefícios desta decisão são fáceis de avaliar. A retirada do recurso do núcleo efetua uma simplificação des

te, compensada por um desempenho pior. Pode-se mostrar que a administração descentralizada de um número fixo de recursos de um tipo é geralmente menos eficiente, devido a sub-utilização de alguns conjuntos, e esgotamento de outros. Para manter o mesmo nível de serviço, maior número de recursos será necessário.

A Política de Gerência

Às vezes, onde não for possível excluir totalmente do núcleo suporte para um tipo de recurso, poderá ser possível retirar a gerência do recurso, permanecendo dentro do núcleo somente o Software responsável para sua integridade. Isto significa que os mecanismos que manipulam o recurso fazem parte do núcleo, enquanto a política, que decide quais operações devem ser feitas no recurso, é implementada em Software fora do núcleo. Por exemplo, o escalonamento de processos ou a gerência de memória principal poderão ser confiados a processos. Estes processos executam primitivas do núcleo para despachar processos, trocar páginas, e assim em diante. Neste esquema, é possível um alto grau de sofisticação nos algorítmos de escalonamentos empregados, sem portanto introduzir complexidade no núcleo.

Denominação

Dentro de um sistema operacional, um mesmo recurso pode ser conhecido por vários nomes, em níveis diferentes de abstração. O sistema precisa não somente gerenciar estes nomes, como também conservar os mapeamentos entre os diferentes nomes do mesmo objeto. Por exemplo, em Multics, um segmento é especificado pelo usuário por um nome simbólico, por um programa em execução pelo número de segmento, e na tabela de segmentos pelo endereço físico da sua tabela de páginas. Isto é, estamos utilizando três espaços distintos de nomes.

Poderemos simplificar a gerência de nomes dentro do núcleo através da retirada de parte dos mecanismos mencionados. Por exemplo, núcleo poderá suportar o nível mais baixo de nomes, ficando para o processo do usuário os outros níveis e o Software de mapeamento. Isto geralmente envolve a partição entre os processos dos espaços de nomes de níveis superiores, com partições diferentes gerenciadas por processos diferentes. Porém há dificuldades, se for desejado o com

partilhamento de objetos, por causa da necessidade de coordenar as referências aos objetos compartilhados.

4. SEGURANÇA DISTRIBUÍDA

Assumimos até agora, que a arquitetura do nosso sistema coloca todo o Software responsável para segurança no núcleo do sistema, executando diretamente no Hardware, e o resto do sistema implementado em processos fazendo chamadas ao núcleo.

Uma alternativa viável é mudar parte das funções de segurança fora do núcleo para *"processos de confiança"*. Estes processos são indistinguíveis de outros processos, exceto que a informação passada por eles para o núcleo é usada por este para tomar ações ligadas a segurança.

Por exemplo, os terminais livres de um sistema, poderão ser controlados por um processo que admite e autentica a entrada de usuários, passando controle para o Software solicitado. O núcleo do sistema poderá aceitar a autenticação feita por este processo, e tomar decisões utilizando-a.

Frequentemente, estes processos de confiança têm partes que não são relevantes para segurança, e poderemos estruturá-los para que consistam de um núcleo, contendo toda a parte sensitiva, e outra Software, irrelevante para segurança. O resultado final de uma extensão desta análise é um sistema composto de níveis de núcleos. Para que funcione corretamente, cada núcleo depende da correção da sua implementação, e também dos núcleos de níveis mais baixos em que este depende.

Um exemplo é dado pela implementação como processo de um gerente de arquivos. Para manter integridade dos arquivos, o núcleo deste processo deve conter as funções de diretório de arquivos, que contém o mapeamento dos nomes simbólicos para blocos em disco. Posicionamento de ponteiros dentro do arquivo não ameaça a integridade, e pode ser feito fora do núcleo.

Com a remoção para núcleos de mais alto nível de muitas das funções de segurança, o núcleo básico se torna muito simples, podendo ser implementado em microcódigo, com consequente aumento de desempenho.

5. A ESTRUTURA DO NÚCLEO

Discutimos aqui aspectos da estrutura interna do núcleo. De maneira geral, a construção de um núcleo confiável não difere radicalmente da construção de outro Software confiável. Porém, mencionamos duas características particulares a esta aplicação: paralelismo e o uso de tipos abstratos.

Paralelismo

Atualmente, a certificação ou verificação de programação sequencial é menos difícil que de programação paralela. Portanto, o trabalho de garantir a confiabilidade do núcleo é simplificado quando este executa sem sofrer interrupções. Por isto especificamos a não interrupção da execução do núcleo.

Tipos Abstratos

A organização de Software pode ser facilitada pelo uso de tipos abstratos de dados. Aliás, o sistema operacional poderá ser visto como implementando os tipos abstratos de processos, memória virtual e arquivos. Qual tipo é o mais básico? Num sistema de memória virtual, a ilusão de um grande espaço de endereços é implementado através de Software que reconhece faltas de páginas, e executa as operações de E/S necessárias de maneira transparente ao processo. Poderemos muito bem querer implementar este Software como um processo. Também seria conveniente escrever o Software de gerência de processos usando um grande espaço de endereços. Não é óbvio qual abstração, seja processos ou seja memória virtual, é a mais básica, servindo para a implementação da outra.

6. EXEMPLOS

A. Um Núcleo para um Sistema com um Processador

Apresentamos nesta secção uma descrição detalhada de um núcleo simples. Este núcleo executará em Hardware com as seguintes características:

- um único processador central

- interface simples de E/S (Iniciar-E/S e Interrupção)

- relógio de tempo real.

O núcleo suportará o tipo abstrato, processo, dos quais há um número fixo. Comunicação e sincronização entre processos são feitas através de monitores. Comunicação processo/periférico é feita através de uma primitiva de E/S síncrona.

O núcleo consiste de um conjunto de rotinas que são chamadas de três maneiras:

primitivas - chamadas por processos, solicitando serviços.

interrupções de periféricos - indicando o final da operação solicitada.

interrupção do relógio - indica o fim da fatia do tempo de um processo.

São as seguintes as primitivas para suportar monitores e E/S síncrona:

ENTRA-MONITOR(M): o processo está querendo entrar em *M*, e precisa de exclusão mútua.

SAI-DO-MONITOR(M): o processo está saindo do monitor *M*, e outro processo pode entrar.

ESPERA(C): o processo está dentro de um monitor, e espera a condição *C*.

AVISA(C): o processo está dentro de um monitor e avisa a condição *C*. Se outro processo, estiver esperando por *C*, ele deve ser acordado.

EXECUTA-ES (PERIFÉRICO, COMANDO): o comando deverá ser executado pelo periférico especificado. O processo será bloqueado até completar a operação de E/S.

Um dos objetivos principais do núcleo é de compartilhar o único processador entre os diversos processos. Internamente o núcleo mantém um descritor de processo para cada processo no sistema onde é guardada a informação necessária para dar sequência a sua execução. Normalmente, esta informação inclui o conteúdo de registradores do processador, contador de instruções, e registradores de estado. Um processo é despachado (o processo adquire um processador) através da carga dos registradores do processador a partir do descritor do processo. Isto automaticamente passa controle para o processo.

O núcleo mantém uma fila de processos prontos para executar, através do encadeamento dos descritores dos processos que poderiam executar, se tivesse um processador disponível. Adicionalmente, o núcleo mantém filas de processos bloqueados. Para cada motivo de bloqueio (condição dentre de um monitor, operação incompleta de E/S) o núcleo mantém uma fila separada.

A principal atividade do núcleo é de manipular as diversas filas de processos. Para garantir a integridade desta informação, o núcleo executa ininterrupta, através da inibição de interrupções

No núcleo que estamos descrevendo, por simplicidade, exclusão mútua em monitores também é implementada através da inibição de interrupções. Observa que isto exclui ativação simultânea de dois monitores. Assumimos que o descritor do processo inclui o estado de inibição de interrupções.

No que segue, damos a especificação das rotinas do núcleo. Usamos um ponteiro chamado "executando" para localizar o descritor de processo do processo em execução. Logo o comando "retira executando de prontos" significa remover um descritor de processo da fila de processos prontos, e colocar o endereço deste descritor no ponteiro "executando". "Despacha executando" significa restaurar a partir do descritor apontado por "executando" os registradores do processador, e também o estado de inibição de interrupções.

ENTRA-MONITOR: Inibe interrupções:
Marca executando não interrompível;

SAI-DO-MONITOR: Marca executando interrompível;
Desinibe interrupções;

```
ESPERA(COND): Salva estado de executando;
              Insere executando em COND;
              Retira executando de prontos;
              Despacha executando;

AVISA(COND): IF ¬ vazio(COND) THEN
                Do; /* despacha processo esperando */
                  Salva estado de executando;
                  Insere executando em prontos;
                  Retira executando de COND;
                  Despacha executando;
                  END;

TROCA : /* interrupção do relógio */
        Inibe interrupções e salva estado de executando;
        Insere executando em prontos;
        Retira executando de prontos;
        Despacha executando;

EXECUTA-ES(PERIFÉRICO,COMANDO): /* primitiva de E/S */
        Inibe interrupções e salva estado de executando;
        Insere executando na fila PERIFERICO;
        Inicia-E/S (PERIFERICO,COMANDO);
                /* instrução do hardware */
        Retira executando de prontos;
        Despacha executando;

ES-COMPLETA(PERIFERICO): /* interrupção de periférico */
        Inibe interrupções e salva o estado de executando;
        Insere executando em prontos;
        Retira executando da fila PERIFERICO;
        Despacha executando;
```

Observações

1. Foi assumida alguma disciplina justa para organizar as filas.

2. O caso da fila "prontos" estar vazia significa que todos os processos estão bloqueados. Para não parar o processador, poderemos absorver seu potencial excedente para executar um processo não urgente, que só executa neste caso. Chamamos este processo, o processo oscioso.

B. Um Núcleo para um Sistema Multiprocessador

O seguinte núcleo também suporta um número fixo de processos e monitores, mas é mais apropriado para um sistema com mais de um processador tendo acesso a uma memória comum. Neste caso, exclusão mútua não pode ser garantida inibindo interrupções, e precisamos usar bandeiras para controlá-la.

Como antes, o estado dos processo é alterado somente pelo núcleo, e então temos que garantir que somente um processador execute o núcleo em determinado instante. Quando for solicitado um serviço do núcleo, através de trap ou interrupção, a rotina apropriada só será executada depois do núcleo terminar de executar o pedido anterior.

A implementação de monitores, utiliza uma fila de entrada para cada monitor, uma fila para cada condição e uma bandeira para indicar se o monitor é ativo.

As rotinas que implementam monitores são:

ENTRA-MONITOR(M):

```
IF ocupado(M) THEN
   Insere usuário em M;
ELSE DO;
    ocupado(M):= TRUE;
    Insere usuário em prontos;
    END;
```

SAI-DO-MONITOR:

```
IF  vazio(M) THEN
    ocupado(M):= FALSE;
ELSE DO;  /* transfere processo de M para prontos */
    Retira p de M;
    Insere p em prontos;
```

```
        END;
    Insere usuário em prontos;
```

ESPERA(C):

```
    IF vazio(M) THEN
        ocupado(M):= FALSE;
    ELSE DO;
        Retira p de M;
        Insere p em prontos;
        END;
    Insere usuário em C;
```

AVISA(C):

```
    IF vazio(C) THEN
        Insere usuário em prontos;
    ELSE DO;  /* transfere processo de C para prontos */
        Retira p de C;
        Insere p em prontos;
        Insere usuário em M;
        END;
```

Para implementar E/S as rotinas são:

EXECUTA-ES(PERIFERICO,COMANDO):

```
    Inicia-E/S (PERIFERICO,COMANDO);
        /* instrução de hardware */
    Insere usuário em PERIFERICO;
```

ES-COMPLETA(PERIFERICO)

```
    Retira p de PERIFERICO;
    Insere p em prontos;
```

As rotinas dadas acima especificam os algorítmos do núcleo. O que falta é dar as rotinas que garantem exclusão mútua de ativações do núcleo. Assumimos que temos n processadores idênticos, cada um com seu relógio independente. Qualquer processador pode executar qualquer processo. Cada periférico pode interromper qualquer processador e escolhe arbitrariamente qual será interrompido. Quando

o núcleo for ativado será executado no processador que recebeu a interrupção, ou executou o trap.

As rotinas que seguem compartilham os processos prontos entre os processadores. Elas utilizam duas rotinas *ENTRA-NUCLEO* e *SAI-DO-NUCLEO* para salvar o estado dos processos, obter exclusão mútua, e despachar os processos. Estas serão discutidas depois.

```
TRAP(NUM-TRAP): /* processo chama núcleo */
     ENTRA-NUCLEO;  /* obtém exclusão mútua */
     Chama rotina apropriada;
     Retira executando de prontos;
     SAI-DO-NUCLEO; /* fim de exclusão mútua */

TROCA:  /* interrupção do relógio */
     ENTRA-NUCLEO;
     Insere executando em prontos;
     Retira executando de prontos;
     SAI-DO-NUCLEO;

INTERRUPÇÃO-ES(PERIFERICO):
     ENTRA-NUCLEO;
     CALL ES-COMPLETA(PERIFERICO);
     Retira executando de prontos;
     SAI-DO-NUCLEO;
```

Nestas rotinas, cada processador deve ter variáveis particulares (registradores) chamadas *NUM-TRAP*, executando e *PERIFERICO*.

Se tiver somente um processador, podemos implementar *ENTRA-NUCLEO* e *SAI-DO-NUCLEO* simplesmente.

```
ENTRA-NUCLEO:  /* para 1 processador */
     Inibe interrupções;
     Salva estado de executando;

SAI-DO-NUCLEO:  /* para 1 processador */
     Restaura estado de executando;
     Desinibe interrupções;
```

Esta solução tem a vantagem sobre a anterior porque permite ativação simultânea de monitores, que podem ser interrompidos.

Para processadores múltiplos, usamos a instrução *TEST & SET* para garantir exclusão mútua dentro do núcleo. Uma solução é dada abaixo:

```
ENTRA-NUCLEO:  /* mais de 1 processador */
    Inibe interrupções;
    Salva estado de executando;
    TEST & SET(NUCLEO-OCUPADO);
    DO WHILE (NUCLEO ESTAVA OCUPADO);
         TEST & SET(NUCLEO-OCUPADO);
         END;

SAI-DO-NUCLEO:  /* mais de 1 processador */
    NUCLEO-OCUPADO:= FALSE;
    Restaura estado de executando;
    Desinibe interrupções;
```

O núcleo que acabamos de descrever independe do número de processadores usados. Portanto, continuará funcionando se alterar este número, embora com vazão diferente de serviço.

Bibliografia

A discussão de núcleos para sistemas seguros é baseada no trabalho de Popek e Kline. Monitores foram introduzidos por Dykstra, Brinch Hansen e Hoare, e usados nas implementações de Modula (with) e Pascal Concorrente (Brinch Hansen). Os núcleos descritos aqui vêm do livro de Holt.

BRINCH HANSEN,P. - *Operating System Principles* - Prentice Hall 1973.

_____ - *The Arquitecture of Concurrent Programs* - Prentice Hall 1973.

DYKSTRA, E.W. - *Cooperating Sequential Processes,* em Genuys (ed.).*Programming Languages.*

HOARE, C.A.R. - *Monitors: an Operating System Structuring Concept.* CACM 17 (10) - 549 - 557, 1974.

HOLT, R.C. & Outros - *Structured Concurrent Programming* - Addison Wesley - 1978.

POPEK, G.J. & KLIN, C.S. - *Issues Kernel Design em Bayer (ed) Operating Systems: an Advanced Covise* - pp. 209 - 227 - Springer - 1978

WIRTH, N. - *Modula: A Language for Modular Programming* - Software, Practice and Experience 7 (1) - 3 - 35 - 1977.