Sistemas Operacionais

Prof. Robson de Souza

Aulas 15 e 16

Conteúdo: Comunicação entre processos

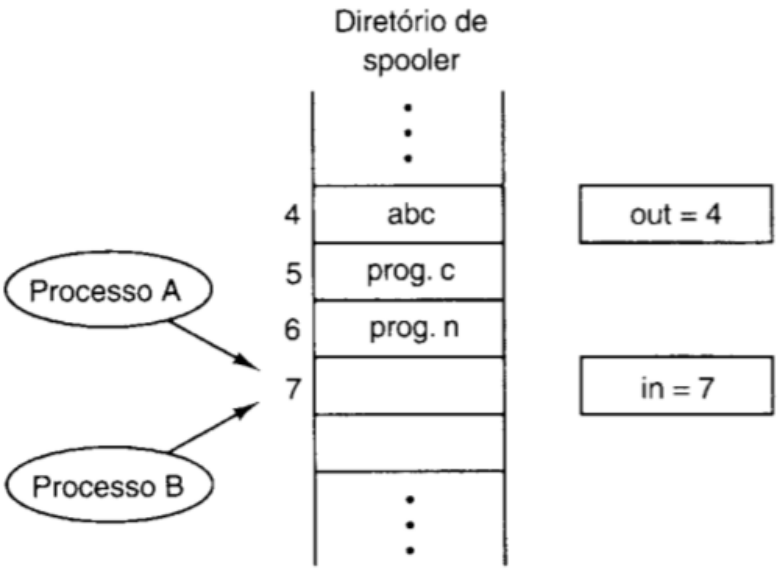
Os processos precisam se comunicar com outros processos, por exemplo, um processo A está calculando um valor que precisa ser passado como entrada para um processo B, logo, esses dois processos precisam ser capaz de se comunicar e trocar essas informações, para que quando o processo A terminar sua execução, o valor calculado será passado para o processo B iniciar sua execução.

A forma de comunicação entre processo preferencialmente deve ser feita sem interrupções, caso possível. Para que isso aconteça, a comunicação deve ser bem estruturada. Com relação à comunicação entre processos (CIP), existem três questões importantes:

- Como um processo pode passar as informações para outro.
- Certificar-se que dois ou mais processos não vão interferir um com o outro em atividades críticas.
- Sequenciamento adequado em caso de dependência (Por exemplo, se um processo A produz dados e um processo B os imprime, então B tem que esperar A terminar para começar a sua execução).

Condições de corrida

Em um Sistema Operacional pode ocorrer um problema com relação ao fato de existirem alguns recursos e espaços que os processos podem compartilhar. Por exemplo, supondo que para executar alguma atividade de impressão, o processo precise alocar o nome do arquivo em uma pilha de impressão, especificamente no próximo espaço livre da pilha. Supondo que em um determinado momento, dois processos precisem imprimir arquivos, com isso, o primeiro processo recebe o índice do local onde ele deve inserir e insere o nome do seu arquivo, mas o processador interrompe esse processo antes de atualizar o índice e libera a execução do segundo processo. Como o índice da pilha não foi atualizado, o segundo processo receberá o mesmo índice e vai alocar o nome do seu arquivo em cima do primeiro arquivo, logo, quando o primeiro processo retomar sua execução, ou ele vai apagar o nome do arquivo do processo 2 ou o seu próprio arquivo não será impresso. Casos desse tipo se chamam **condições de corrida** e devem ser tratados com algum mecanismo. Observe a figura abaixo:



Para evitar problemas desse tipo, onde se tem memória compartilhada, arquivos compartilhados e recursos compartilhados, é preciso encontrar alguma maneira de proibir que mais de um processo leia e grave os dados compartilhados ao mesmo tempo, isso se chama **exclusão mútua**, ou seja, certificar-se que se um processo está utilizando um arquivo ou variável compartilhada, os outros processos serão impedidos de fazer a mesma coisa.

A parte do programa em que a memória compartilhada é acessada é chamada de **região crítica** ou **seção crítica.** Condições necessárias para prover exclusão mútua:

- Nunca dois processos podem estar simultaneamente em uma região crítica
- Nenhum processo executando fora de sua região crítica pode bloquear outros processos
- Nenhum processo deve esperar eternamente para entrar em sua região crítica
- Nenhuma suposição pode ser feita sobre as velocidades ou sobre os números de CPUs.

Os mecanismos que implementam a exclusão mútua, utilizam protocolos de entrada e saída da região crítica. Existem várias propostas para obter exclusão mútua, algumas delas serão vistas a seguir.

Desativar interrupções

Quando um processo acessar uma região crítica, qualquer interrupção é desativada, desse modo, como a mudança de contexto de processos só pode ser realizada através de interrupções, o processo que as desabilitou terá acesso exclusivo garantido.

Essa solução não é das mais aconselháveis, pois isso pode dar poder de desabilitar interrupções a processos de usuário. Outro problema é que se o sistema é multiprocessado, desabilitar interrupções só vai afetar a CPU que executou o processo e nada impede que outra CPU acesse a região crítica.

Variáveis de bloqueio

Essa opção diz respeito a ter uma variável global única compartilhada (bloqueio) que inicialmente é setada com valor 0. Sempre que um processo quiser acessar a região crítica, ele verifica a variável, se ela for 0, o processo altera seu valor para 1 e entra na região crítica, quando algum processo verificar o valor da variável de bloqueio sendo 1, ele entende que a região crítica já está sendo acessada e espera até que ela esteja liberada. Ao terminar de acessar a região crítica, o processo sai e retorna o valor da variável de bloqueio para 0. Infelizmente, essa solução pode sofrer com o mesmo problema de impressão já exemplificado aqui.

As soluções anteriores, apresentam o problema da *espera ocupada*: Toda vez que um processo não consegue entrar em sua região crítica, por já existir um outro processo acessando o recurso, o processo permanece em looping testando uma condição, até que lhe seja permitido o acesso. Dessa forma, fica consumindo tempo do processador desnecessariamente.

Uma solução para isso é introduzir mecanismos de sincronização que coloquem o processo em estado de espera:

- Semáforos
- Monitores

Semáforos

Semáforo é uma variável inteira, não-negativa, manipulada por duas instruções (DOWN e UP) indivisíveis (a execução não pode ser interrompida).

O semáforo indica se o processo fica na fila de espera ou acessa a região crítica;

Semáforos contadores – assumem qualquer valor inteiro e positivo;

A utilização de semáforos evita a “espera ocupada”;

As instruções DOWN e UP funcionam como protocolos de entrada e saída da região crítica, onde DOWN coloca o processo em uma fila de espera e UP coloca algum processo que estava em espera para executar;

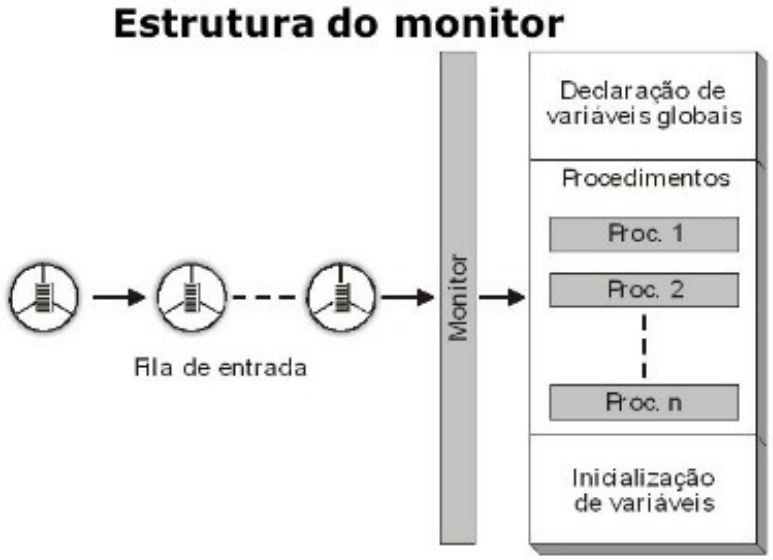
Mutexes

São semáforos binários, que assumem somente 0 ou 1 nos seus valores.
Implementam apenas exclusão mútua, são mais simples e "leves" que semáforos e são muito utilizados por threads. Um mutex pode estar em dois estados (unlocked ou locked).

A forma genérica de funcionamento de um mutex é bastante simples. Antes de entrar em uma região crítica, o processo ou thread executa uma chamada mutex_lock. Se outros processos ou threads tentarem acessar a mesma região crítica eles serão bloqueados. Ao sair de uma região crítica, o processo ou thread executa uma chamada mutex_unlock. Se houver um ou mais processos ou threads bloqueados neste mutex, um deles será liberado para executar.

Monitores

Monitores contém um conjunto de funções que manipulam um determinado recurso compartilhado. A exclusão mútua de um monitor se aplica a todas as funções dentro dele. É um módulo que possui procedimentos e variáveis para implementar a exclusão mútua.
Os monitores possuem as variáveis de condição, que são um mecanismo para sincronização de threads ou processos. Por exemplo, supondo que um thread ganhe acesso ao monitor, mas uma dada condição necessária a sua execução não é verdadeira (ainda). Se outro thread precisar de acesso ao monitor para tornar a condição verdadeira, a espera ocupada não é viável. A solução para isso é utilizar os comandos wait e signal, onde wait aguarda que uma determinada condição seja verdadeira e o processo ou thread aguardando fica bloqueado e permite que outros acessem o monitor. O signal avisa que determinada condição agora é verdadeira, com isso, o processo ou thread que precisava dessa condição, pode executar.



https://slideplayer.com.br/slide/325976/

Referências bibliográficas:

TANENBAUM, Andrew. 2ª ed. **Sistemas Operacionais Modernos,** Editora Pearson, 2003.

SILBERSCHATZ, Abraham. **Sistemas Operacionais com JAVA,** 6ª ed. Editora Campus

MACHADO, Francis B. **Arquitetura de Sistemas Operacionais,** 4ª ed, LTC, 2007.