Sistemas Operacionais

Sincronização e Comunicação entre Processos

Aplicações Concorrentes

- Os mecanismos que garantem a comunicação entre processos concorrentes e o acesso a recursos compartilhados são chamados “mecanismos de sincronização”.
- No projeto de SO multiprogranáveis, é fundamental a implementação destes mecanismos para garantir a integridade e confiabilidade na execução de aplicações concorrentes.

Especificação de Concorrência em Programas

- Existem várias notações utilizadas para especificar a concorrência em programas, ou seja, as partes de um programa que devem ser executadas concorrentemente.
- A primeira notação para a especificação da concorrência em um programa foram os comandos FORK e JOIN, introduzidos por Conway (1963) e Dennis e Van Horn (1966).

Especificação de Concorrência em Programas

Program A

.

.

FORK B;

.

.

JOIN B;

.

.

END.

Program B

.

.

.

.

.

End.

- O programa A começa a ser executado e, ao encontrar o comando FORK, faz com que seja criado um outro processo para a execução do Programa B, concorrentemente ao Programa A. O comando JOIN permite que o Programa A sincronize-se com B, ou seja, quando o Programa A encontrar o comando JOIN, só continuará a ser processado após o término da execução do Programa B.

Problemas de Compartilhamento de Recursos

- A sincronização entre processos concorrentes é fundamental para a confiabilidade dos sistemas multiprogramáveis.
- Ex. Caixa Banco

Exclusão Mútua

- A solução mais simples para evitar os problemas de compartilhamento, é impedir que dois ou mais processos acessem um mesmo recurso simultaneamente.
- Para isso, enquanto um processo estiver acessando determinado recurso, todos os demais processos que queiram acessá-lo deverão esperar pelo término da utilização do recurso.

Exclusão Mútua

- Essa idéia de exclusividade de acesso é chamada “exclusão mútua (mutual exclusion)”.
- A exclusão mútua deve afetar apenas os processos concorrentes somente quando um deles estiver fazendo acesso ao recurso compartilhado.

Exclusão Mútua: Soluções de Hardware

- A exclusão mútua pode ser implementada através de mecanismos de hardware, por meio de:
  - Desabilitação de interrupções
  - Instrução test-and-set

Exclusão Mútua: Soluções de Hardware

- Desabilitação de interrupções: a solução mais simples para o problema de exclusão mútua é fazer com que o programa desabilite todas as interrupções antes de entrar em sua região crítica, e as reabilite após deixar a região crítica. Porém possuem limitações, por exemplo, a multiprogramação pode ficar seriamente comprometida, já que a concorrência entre processos tem como base o uso de interrupções.

Exclusão Mútua: Soluções de Hardware

- Instrução test-and-set: esta instrução permite ler uma variável, armazenar seu conteúdo em outra área e atribuir um novo valor à mesma variável, e tem como característica ser executada sem interrupção. Dessa forma, é garantido que dois processos não manipulem uma variável compartilhada ao mesmo tempo, possibilitando a exclusão mútua.

Exclusão Mútua: Soluções de Software

- Algoritmo de Dekker
- Algoritmo de Peterson

Sincronização Condicional

- É uma situação onde o acesso ao recurso compartilhado exige a sincronização de processos vinculada a uma condição de acesso. Um recurso pode não se encontrar pronto para uso devido a uma condição específica. Nesse caso, o processo que deseja acessá-lo deverá permanecer bloqueado até que o recurso fique disponível,

Semáforos

- O conceito de semáforos foi proposto por E. W. Dijkstra em 1965, sendo apresentado como um mecanismo de sincronização que permitia implementar, de forma simples, a exclusão mútua e sincronização condicional entre processos.

Semáforos

- De fato, o uso de semáforos tornou-se um dos principais mecanismos utilizados em projetos de SO e em aplicações concorrentes. Atualmente, a maioria das linguagens de programação disponibiliza rotinas para uso de semáforos.

Monitores

- São mecanismos de sincronização de alto nível que torna mais simples o desenvolvimento de aplicações concorrentes.
- O conceito foi proposto por C. A. R. Hoare em 1974, como um mecanismo de sincronização estruturado, ao contrário dos semáforos, que são considerados não-estruturados.

Monitores

- Monitores são considerados de alto nível e estruturados em função de serem implementados pelo compilador.

Troca de Mensagens

- É um mecanismo de comunicação e sincronização entre processos.

Deadlock

- É a situação em que um processo aguarda por um recurso que nunca estará disponível ou um evento que não ocorrerá. Essa situação é consequência, na maioria das vezes, do compartilhamento de recursos, como dispositivos, arquivos e registros, entre processos concorrentes onde a exclusão mútua é exigida.

Deadlock

- O problema do deadlock torna-se cada vez mais frequente e crítico na medida em que os SO evoluem no sentido de implementar o paralelismo de forma intensiva e permitir a alocação dinâmica de um número ainda maior de recursos.

Deadlock

- Para que ocorra um deadlock, quatro condições são necessárias simultaneamente:
  - Exclusão mútua: cada recurso só pode estar alocado a um único processo em um determinado instante;
  - Espera por recurso: um processo, além dos recursos já alocados, pode estar esperando por outros recursos;
  - Não-preempção: um recurso não pode ser liberado de um processo só porque outros processos desejam o mesmo recurso;
  - Espera circular: um processo pode ter de esperar por um recurso alocado a outro processo e vice-versa.

Deadlock

- Para prevenir a ocorrência de deadlocks, é preciso garantir que uma das quatro condições anteriores nunca se satisfaça.