Sistemas Operacionais

Sincronização de Processos

Mecanismos de Sincronização

- Os mecanismos básicos para obtenção da exclusão mútua, também chamados de mecanismos de sincronização são:
  - Protocolos de acesso (protocolos em software puro)
  - Spin-lock
- Mecanismos de mais alto nível para obtenção da exclusão mútua são implementados a partir destes 3 mecanismos e são:
  - Semáforos
  - Mutex
  - Monitores

Protocolos de Acesso

Protocolos de Acesso

- Os protocolos de acesso ou soluções em software puro são:
  - soluções para apenas 2 tarefas;
  - consistem em códigos implementados (sem utilizar chamadas ao sistema) antes da entrada na seção crítica e na saída da seção crítica, fazendo o controle de acesso aos dados compartilhados.

- Esses protocolos possuem espera ocupada (*busy waiting*) para entrada na seção crítica, fazendo com que mesmo que o processo esteja esperando permaneça utilizando o processador. Assim, são soluções aplicaveis a seções críticas pequenas.

Protocolos de Acesso

- Consiste em métodos ou funções que definem:
  - Entrada_SC( )
    Somente entra se tiver permissão
  - Saída_SC( )
    - Saída da Seção Crítica
  - Seção Crítica
    - Locais onde cada processo ou thread realiza sua seção crítica

Protocolos de Acesso

- Ao todo, são 3 formas de implementar os Protocolos de Acesso
  - Algoritmo 1
  - Algoritmo 2

  } propostos por E. W. Dikstra

  - Algoritmo 3 → conhecido como Algoritmo de Petterson (1981)

- Não incluem chamadas ao Sistema Operacional

Algoritmo 1

- As *threads* compartilham uma variável inteira *x* inicializada em 0 ou 1.
- Se x = i, então T_i pode executar sua seção crítica.
- Garante que apenas uma *thread* de cada vez esteja executando sua seção crítica.
- Se turn = 0 e T_1 estiver pronto para entrar em sua seção crítica, não poderá fazê-lo, mesmo que T_0 possa estar em sua seção não crítica.

Algoritmo 1

```
Entrada_SC(i){
    while (x != i){
     //espera
    }
}

Saida_SC(i){
    x = 1 – i;
}
```

Algoritmo 2

- O Algoritmo 1 (Protocolo 1) não retém informações suficientes sobre o estado de cada *thread*. Somente lembra qual *thread* tem permissão para entrar em sua seção crítica.

- No Protocolo 2 a variável *x* é substituída pelo vetor booleano *flag.* Este vetor é compartilhado entre as *threads*:

boolean flag[2] = {*false*, *false*}

Algoritmo 2

- Se *flag*[i] = *true*, indica que T_i está pronta para entrar em sua seção crítica.

- O requisito de Exclusão Mútua é atendido, mas os demais ainda não:
  - T_0 define flag[0]=true indicando que quer entrar em sua seção crítica
  - Antes de entrar no while, ocorre troca de contexto e T_1 define flag[1]=true.
  - Ambas *threads* entrarão em um laço infinito.

Algoritmo 2

```
Entrada_SC(i){
    int outro;
    outro = 1 – i;
    flag[i] = true;
    while (flag[outro] == true){
                //espera
          }
}

Saida_SC(i){
    flag[i] = false;
}
```

Algoritmo 3

- Combinação dos Algoritmos 1 e 2 atendendo os requisitos de uma solução ao Problema da Seção Crítica.
- As *threads* compartilham a variável inteira e o vetor booleano

```
int x; //Pode ser inicializado com 1 ou 0
boolean flag[2] = {false, false}
```

Algoritmo 3

- Para entrar em sua seção crítica, T_i
  - primeiro define flag[i]=true
  - e declara que é a vez da outra *thread* entrar também (x = outro)
- Se ambas *threads* tentarem entrar ao mesmo tempo, x é definido como i e como j praticamente ao mesmo tempo
- Apenas uma dessas atribuições perdura
- O valor final de x decide qual das duas *threads* terá permissão para entrar em sua seção crítica primeiro

Protocolo 3

```
Entrada_SC(i){
    flag[i] = true;
    outro = 1 – i;
    x = outro;
    while ((flag[outro] == true) && (x == other)){
                //espera
    }
}

Saida_SC(i){
    flag[i] == false;
    X = 1 – id;
}
```

Desabilitar interrupções

Desabilitar interrupções

- A solução mais simples para proteção da Seção Crítica é fazer cada processo desativar todas as interrupções para entrar na seção crítica e reativá-las imediatamente após sair dela.
- Desabilitar interrupções ao entrar na Seção Crítica e habilitar as interrupções ao sair
- Pode ser usado
  - Sistemas embarcados
- Não é um método genérico, pois não é aconselhável dar a processos de usuário a permissão para desabilitar interrupções
  - Um processo de usuário pode desabilitar as interrupções e não habilitá-las

Desabilitar interrupções

- Não usado em máquinas multiprocessadas, pois apenas a CPU que realiza a instrução é afetada e as demais continuarão executando e poderão acessar a memória compartilhada.
- Essa técnica é útil no Sistema Operacional em si, pois o Sistema desativa interrupções enquanto atualiza variáveis ou listas, por exemplo.
- Para processos de usuário, como mecanismo de exclusão mútua, não é apropriada.

Spin-lock

Spin-Lock

Spin-lock ; Test-and-set

Hardware de Sincronização

Algumas máquinas fornecem *hardware* especial:

- permitem testar e modificar o conteúdo de uma palavra ou trocar o conteúdo de duas palavras (**de forma atômica**)

Spin-Lock

- Instrução de Máquina que executa de forma atômica

- Instrução SWAP – trocar conteúdo de uma posição de memória com o conteúdo de um registrador, sem interrupções

```
Swap(reg, mem)
    [mem] → aux
    reg → [mem]
    aux → reg
```

- A Seção Crítica estará protegida por uma variável em [mem] = *lock* (fechadura)
- *Lock* = zero → Seção Crítica livre
- *Lock* = 1 → Seção Crítica ocupada

Spin-Lock

- Antes de entrar na Seção Crítica, um processo precisa "fechar a porta" → *lock* = 1

- Somente pode fazer isso se *lock* = zero, ou seja, Seção Crítica livre

```
faça {
   reg = 1;
   swap(reg, lock);
} enquanto (reg==1);

Seção Crítica;

Lock = 0;
```

Entrada na Seção Crítica
(o *lock* deve ser zero para entrar na Seção Crítica)

Saída da Seção Crítica

Spin-Lock

Vantagens

→ Simplicidade

→ Instrução de máquina presente na maioria dos processadores

Desvantagens

→ *Busy-waiting*: processo no laço de espera ocupa o processador

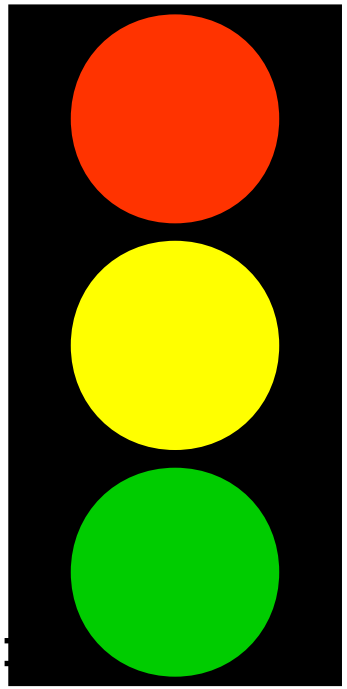
→ Se existem vários processos, pode haver postergação indefinida

→ Uso limitado a problemas com seção crítica pequena

Semáforos

Semáforo

→ É uma Ferramenta de Sincronização



→ Criado pelo matemático holandês *E. W. Dijkstra*, em 1965

→ É um tipo abstrato de dado que possui:

- » um valor inteiro
- » uma fila de processos
- » duas operações sobre o semáforo:

P = *proberen* (testar – wait – down – wait)

V = *verhogen* (incrementar – signal – up – post)

Operações sobre Semáforos

Operação P

→ Decrementa em um o valor do semáforo

e

→ Testa o valor do semáforo

- Se o valor é negativo, o processo é bloqueado e colocado no fim da fila do semáforo

Operação V

→ Incrementa em um o valor do semáforo

e

→ Se existe processo na fila do semáforo, sinaliza-o

- Retira o 1º processo da fila do semáforo e
- Acorda o processo

P e V são operações atômicas

Semáforo - Implementação

```
struct Semáforo{
   int valor;
   int *PCB;
}
Semáforo S;
```

- A implementação das operações P e V são feitas por meio de chamadas ao sistema. Há necessidade de desabilitar interrupções e de ter acesso aos descritores de processo.

Operação P(S)

```
S.valor = S.valor – 1;
Se S.valor < 0
    bloqueia o processo que executou
    a operação P(S);
    coloca o processo na fila de S;
Senão
    continua execução;
```

Operação V(S)

```
S.valor = S.valor + 1;
Se S.valor <= 0
    retira um processo da fila de S;
    acorda o processo que foi
    removido da fila;
Senão
    continua execução;
```

Tipos de Semáforos

Semáforo de Contagem

→ Pode assumir qualquer valor

Semáforo Binário

→ Valor 0 e 1

→ O Semáforo Binário é usado somente para controlar o acesso à seção crítica em processos ou threads

→ O Semáforo de Contagem pode ser usado para controlar o acesso à seção crítica e também para estabelecer a precedência de execução de operações em processos concorrentes

Uso de Semáforos para Proteção da Seção Crítica

- Valor inicial do semáforo deve ser igual a 1
- A Seção Crítica estará protegida pelo uso das operações P, antes da seção crítica, e V após a seção crítica.
- Exemplo:

```
Semaforo S = 1;

P(S);
Seção Crítica;
V(S);
```

Uso de Semáforos para Estabelecer a Precedência de Operações

- Para o estabelecimento da Precedência de Operações entre *threads*, para cada par de *thread*, um semáforo inicializado com zero deve ser utilizado.
- A thread que precisa esperar deve efetuar a operação P
- A thread que deve executar primeiro, executa a operação V após a operação
- Exemplo: ….

Uso de Semáforos para Estabelecer a Precedência de Operações

- Exemplo: Suponha que a Thread 1 deve executar a operação Consulta() após a Thread 2 executar a operação AtualizaBD(). A solução para com semáforos é:

```
Semaforo S = 0;
```

```
//Thread 1
P(S);
Consulta();
```

```
//Thread 2
AtualizaBD();
V(S);
```

Mutex

Mutex

- É uma versão simplificada do semáforo, ou seja, não possui a capacidade de contar

- Mutexes:
  - são usados para proteção da seção crítica (fazer a exclusão mútua de recursos compartilhados entre processos ou threads cooperativos)
  - são fáceis de usar e eficientes

Mutex

- Um mutex é uma variável que pode ter dois estados: livre ou ocupado
  - Por isto, apenas um bit é necessário para representá-lo
  - O valor zero representa o estado livre
  - Valores diferentes de zero representam o estado ocupado

Mutex

- Antes de entrar na seção crítica, é preciso chamar mutex_lock
  - Se o mutex está livre, o processo entra em sua seção crítica

  Se o mutex está ocupado, o processo que fez a chamada é bloqueado até que o processo que se encontra na sua região crítica termine e chame mutex_unlock
- Ao sair da seção crítica, é preciso chamar mutex_unlock
  - Se vários processos estiverem bloqueado no mutex, um deles será escolhido aleatóriamente e poderá entrar na região crítica.

Monitores

Monitores

- Os semáforos são mecanismos de sincronização eficientes, porém seu uso incorreto por parte dos programadores pode causar comportamentos imprevisíveis nos processos/threads cooperativos
  - Espera indefinida
  - Deadlock
  - Condição de corrida

- Brinch Hansen (1975) e Hoare (1974) propuseram uma primitiva de sincronismo de mais alto nível, para tornar mais fácil a sincronização de processos: **os monitores**

Monitores

- Um monitor é um conjunto de
  - Rotinas
  - Variáveis e
  - Estruturas de dados

todas agrupadas em um tipo especial de módulos ou pacotes.

- Os processos podem chamar as rotinas presentes em um monitor sempre que quiserem, mas não podem acessar diretamente as estruturas de dados internas do monitor a partir das rotinas declaradas fora dele.

Monitores

- Os monitores têm uma propriedade importante que os torna úteis para obter a exclusão mútua:
  - A qualquer instante, apenas um processo pode estar ativo em um monitor
- Os monitores são uma construção de linguagem de programação, onde o compilador sabe que são especiais e pode manipular chamadas às rotinas do monitor de forma diferente de outras chamadas de procedimentos

Monitores

- De modo geral, quando um processo chama uma rotina do monitor:
  - Suas primeiras instruções verificam se algum outro processo está ativo dentro do monitor.
    - Se sim: o processo que faz a chamada fica suspenso até que o outro processo tenha saído do monitor
    - Se nenhum outro processo estiver usando o monitor, o processo que fez a chamada poderá entrar.

Monitores

- O compilador implementa a exclusão mútua em entradas de monitor, mas uma maneira comum é utilizar um mutex ou um semáforo binário.
- Como é o compilador e não o programador que faz preparativos para a exclusão mútua, é muito menos provável a ocorrência de erros.

- **Operações concorrentes implementadas dentro de um monitor sempre serão executadas de forma a ter exclusão mútua.**

Monitores

- Para estabelecer a precedência de operações, com o uso de monitores, é preciso usar **Variáveis de Condição** junto com duas operações sobre elas: **wait e signal**
- Quando uma rotina do monitor não pode continuar, ela executa uma operação wait na variável de condição.
  - Isto causa o bloqueio do processo
- Para acontecer o desbloqueio, outro processo deve executar a operação signal na variável de condição que está esperando.

Monitores

- Após a operação signal, o que pode acontecer conforme:
  - Hoare: deixar o processo recentemente desbloqueado executar, suspendendo outro;
  - Brinch Hanse: refinar o problema, exigindo que um processo que execute uma operação signal deve sair do monitor imediatamente. Se uma operação signal é executada em uma variável de condição em que vários processos estão esperando, apenas um deles é desbloqueado (determinado pelo escalonador do Sistema Operacional).

Monitores

- Se uma variável de condição é sinalizada sem ninguém esperando nela, o sinal é perdido.

- A Linguagem Java suporta monitores
  - Adicionando a palavra synchronized em uma declaração de método, Java garante que qualquer *thread* que estiver começando a executar este método, nenhuma outra *thread* poderá realizar qualquer outro método synchronized nessa classe.

Monitores

- Os métodos sincronizados em Java não possuem variáveis de condição.
- Tem dois métodos: *wait* e *notify*
  - Semelhantes à *sleep* e *wakeup*
  - Quando usados dentro do monitor, não estão sujeitas às condições de corrida

Monitores

- Semáforos e monitores foram projetados para ambientes de memória compartilhada (com uma ou mais CPUs)

- Em Sistemas Distribuídos → memória distribuída
  - Esses mecanismos não são aplicáveis
  - Uso de Passagem de Mensagens:
    - Comunicação síncrona x assíncrona
    - RPC – Remote Procedure Call
    - RMI – Remote Method Invocation

Bibliografia

- OLIVEIRA, Rômulo Silva de; CARISSIMI, Alexandre da Silva; TOSCANI, Simão Sirineo. **Sistemas operacionais.** Porto Alegre: Bookman, 2010.

- SILBERSCHATZ, Abraham; GALVIN, Peter; GAGNE, Greg. **Sistemas operacionais: com Java.** Rio de Janeiro: Campus, 2008.

- DEITEL, Harvey M; DEITEL, Paul J; CHOFFNES, David R. **Sistemas operacionais.** São Paulo: Pearson Pretince Hall, 2005.

- TANENBAUM, Andrew S. **Sistemas operacionais:** projeto e implementãção. Porto Alegre: Bookman, 2008.

Exercícios

- Realize uma pesquisa para buscar a implementação do Problema do Produtor-Consumidor com Monitores

- Na bibliografia básica da disciplina, há a descrição do Problema do Jantar dos Filósofos e o Problema dos Leitores e Escritores
  - Caracterize a seção crítica dos problemas
  - Identifique os pontos de sincronização entre processos
  - Proponha uma solução para cada problema com o uso de:

    Semáforos

    ou

    Monitores