Sistemas Operacionais

Sincronização de Processos

Sumário

- Programação Concorrente
- Processos cooperativos
- Problema do Produtor-Consumidor
- Condição de Corrida
- Seção Crítica
- Requisitos de uma solução à Seção Crítica
- Mecanismos de Sincronização

Programação Concorrente

- A programação concorrente caracteriza-se pela execução de vários processos que cooperam entre si para realização de uma determinada tarefa.

- Um programa executado por um único processo é dito programa sequencial.

- Um programa concorrente possui vários fluxos de execução de instruções (vários fluxos de controle) e há necessidade de trocas de dados por estes fluxos

Programação Concorrente

- Nas arquiteturas com múltiplos processadores na execução de um programa concorrente, acontece o paralelismo real
  - os vários processos ou threads são escalonados aos diferentes processadores, tornando a execução paralela (simultânea).

- Nas arquiteturas monoprocessador, a execução de um programa concorrente acontece alternadamente
  - os diferentes processos ou threads são escalonados à mesma CPU e não executam simultaneamente. Há um paralelismo aparente.

Programação Concorrente

- A motivação para o uso da programação concorrente se dá por ela:
  - permitir a exploração do paralelismo real existente em máquinas multiprocessadoras, consequentemente, o aumento de desempenho da aplicação;
  - permitir realizar operações simultâneas de processamento e E/S (entrada/saída), ou seja, enquanto um fluxo de execução de instruções realiza operações de CPU, outro(s) executam operações de entrada e saída.
    - Isso também possibilita a obtenção de desempenho nas aplicações, diminuindo a ociosidade do processador enquanto operações de E/S são realizadas.

Programação Concorrente

- Em relação à programação sequencial, a programação concorrente é mais complexa:
  - Há necessidade de projetar e implementar as operações dos diferentes threads/processos executando concorrentemente, de forma que os dados comuns a eles mantenham a consistência, os processos/threads mantenham a execução (não fiquem em espera indefinida) e o resultado final da aplicação seja correto, independente da velocidade de execução dos processos;

- A depuração de um programa concorrente é tarefa árdua comparada à execução sequencial.

Processos Cooperativos

- Processos Concorrentes que podem afetar ou ser afetados pela execução uns dos outros
- São processos que cooperam para a realização de determinada atividade
- Podem compartilhar diretamente o mesmo espaço de endereçamento lógico (dados e código)
- Por serem concorrentes, pode acontecer inconsistência ao serem acessados os dados compartilhados

Problema do Produtor-Consumidor

- A programação concorrente implica no compartilhamento de recursos, como variáveis, estruturas de dados, registros, arquivos, bancos de dados...

- O acesso aos recursos compartilhados deve ser feito mantendo-se o correto e coerente estado do sistema.

- O problema do compartilhamento de recursos pode ser exemplificado com um Problema Clássico de Sincronização em sistemas operacionais, chamado
  - Problema do Produtor-Consumidor ou
    Problema do Buffer Limitado (*bounded buffer*).

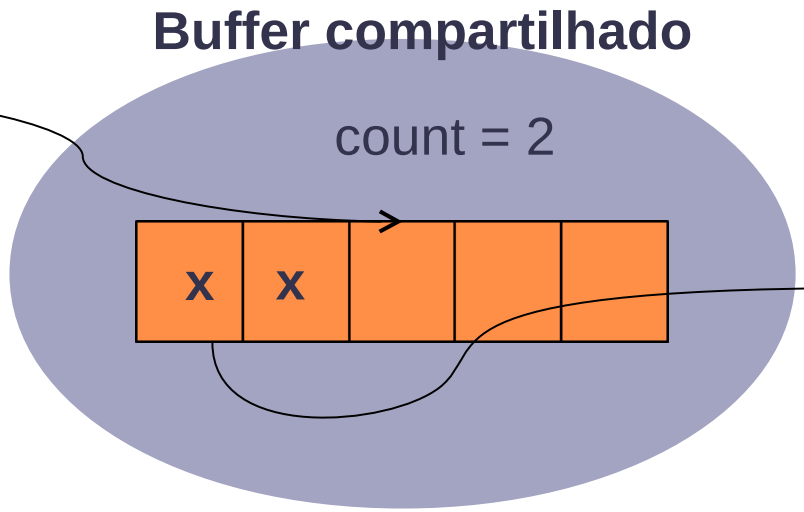
Problema do Produtor-Consumidor

- Segundo Silberschatz (2008), o Problema do Produtor-Consumidor possui:
  - um buffer de tamanho limitado compartilhado entre os processos;
  - há um processo produtor que insere um item no buffer a cada execução;
  - há um processo consumidor que remove um item do buffer a cada execução;
  - há uma variável inteira (*count*) compartilhada entre os processos que é utilizada para contar quantos itens existem no buffer.

Problema do Produtor-Consumidor

Processo Produtor

- Insere 1 item no buffer, a cada execução
- Atualiza cont: cont++;



Processo Consumidor

- Remove 1 item do buffer, a cada execução
- Atualiza cont: cont--;

Problema do Produtor-Consumidor

- Este problema mostra uma relação entre processos, bastante comum em sistemas operacionais.
- Como exemplo, pode-se citar um servidor de impressão.
  - Os vários processos usuários como editores de texto, planilhas, aplicativos (produtores) produzem impressões e as enviam para uma fila (o buffer limitado) do servidor de impressão.
  - O processo consumidor (servidor de impressão) organiza a fila, lê as impressões e as encaminha para a impressora.

Problema do Produtor-Consumidor

- Neste problema, pode-se identificar as seguintes situações de sincronização:
  - Quando o buffer estiver cheio:
    - o processo produtor não poderá executar e deverá esperar até que exista espaço no buffer (até que um item seja consumido);
  - Quando o buffer estiver vazio:
    - o processo produtor não poderá executar e deverá esperar até que um item seja inserido no buffer;
  - Quando um processo está atualizando dados compartilhados, nenhum outro poderá faze-lo.

Thread Produtor x Thread Consumidor

```
//variáveis compartilhadas
int BUFFER_SIZE=5;
int cont=0, in=0, out=0;
struct tipo_dado buffer[BUFFER_SIZE];
```

Thread Produtor

```
while(cont!=BUFFER_SIZE){
        //adiciona item no buffer
        cont = cont +1;
        buffer[in] = dado;
 }
```

Thread Consumidor

```
while(cont!=0){
        //remove item no buffer
        cont = cont-1;
        buffer[out] = 0;
```

Condição de Corrida (*race condition)*

- Como a execução dos processos é concorrente pode acontecer um erro com o valor da variável contadora
  - ao ser incrementada ou decrementada a variável *cont*, as instruções são traduzidas pelo compilador para:

```
//cont = cont +1;
MOVE cont, ACC
INC ACC
MOVE ACC, cont
```

```
//cont = cont – 1;
MOVE cont, ACC
DEC ACC
MOVE ACC, cont
```

Condição de Corrida

- Na execução sequencial, não há problema de concorrencia à variável **cont**.

- Porém, na execução concorrente pode acontecer situações em que ambos fluxos manipulam a variável **cont** concorrentemente e o valor final após a execução, pode ser inconsistente.

- Por exemplo, se **cont** = 2, tanto produtor, quanto consumidor executam 1 vez concorrentemente e a troca de contexto acontece justamente na manipulação da variável cont, como descrito a seguir, qual será o valor final da variável **cont**?

Condição de Corrida

```
Produtor: MOVE cont, ACC          //ACC = 2
Produtor: INC ACC                  //ACC = 3
Consumidor: MOVE cont, ACC        //ACC = 2
Consumidor: DEC ACC               //ACC = 1
Produtor: MOVE ACC, cont          //cont = 3
Consumidor: MOVE ACC, cont        //cont = 1
```

- No término da execução, o valor da variável cont = 1 e deveria ser 2, pois produtor produziu um item e consumidor consumiu um item.
- Se na execução concorrente, fossem invertidas as duas últimas linhas, ou seja, o produtor executasse a última instrução, o valor do count = 3.

- Em nenhuma das situações o valor estaria correto!

Condição de Corrida

- É a situação ilustrada anteriormente com a variável **cont.**

- A condição de corrida acontece quando o valor dos dados compartilhados depende da ordem de execução dos fluxos de instruções e geralmente o valor resultante desta execução é um valor incorreto.

Seção Crítica

- A Seção Crítica é uma seção (parte) de código de um processo ou thread onde acontece a manipulação de dados que são compartilhados com outros processos ou threads e deve executar de forma atômica
  - com as interrupções desabilitadas.

- Para evitar a Condição de Corrida, deve-se proteger a Seção Crítica do código.

- Quando vários threads possuem seção crítica, enquanto um deles estiver executando sua seção crítica, nenhum outro pode executar a sua, ou seja, o acesso à seção crítica é mutuamente exclusivo.

Requisitos de uma solução à Seção Crítica

- Uma solução ao problema da seção crítica deve satisfazer aos requisitos listados a seguir. Considere Ti a *thread* ou processo i:

1. Exclusão mútua

   Se Ti está executando sua seção crítica, qualquer Tj , onde j≠i, não poderá executar sua seção crítica. Isto significa que dois ou mais processos não podem estar em sua seção crítica ao mesmo tempo.

2. Progresso

   Somente threads executando sua seção não crítica participam da decisão de quem será o próximo a executar sua seção crítica; nenhum processo fora de sua seção crítica pode bloquear a execução de um outro processo;

Requisitos de uma solução à Seção Crítica

3. Espera limitada

   Os processos não podem esperar indefinidamente para entrar em sua Seção Crítica. Deve existir um limite para essa espera.

4. Solução independente da velocidade dos fluxos

   A solução deve ser independente da velocidade e quantidade de processadores existentes e da quantidade de *threads* e processos.

Soluções erradas ao problema da seção crítica apresentam a possibilidade de: *deadlock*, postergação indefinida ou inconsistência dos dados.

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:
  - Mutex
  - Semáforos
  - Monitores

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.