Escalonamento de Processos em Sistemas de Tempo Compartilhado

Sincronização entre Processos

Pedroso

2025

Sincronização entre Processos

- A Sincronização é um mecanismo para gerenciar a concorrência entre processos (ou threads).
- Garante que processos que acessam recursos compartilhados o façam de forma ordenada e segura.
- Evita resultados imprevisíveis e incorretos.

Problemas a serem Resolvidos

Condição de Corrida (Race Condition)

- Ocorre quando múltiplos processos acessam e manipulam um recurso compartilhado simultaneamente.
- O resultado final depende da ordem de execução, que é imprevisível.
- Exemplo: dois processos incrementando a mesma variável. O valor final pode ser inconsistente.

Seção Crítica

- A seção crítica é a parte do código que acessa o recurso compartilhado.
- A sincronização visa garantir que apenas um processo por vez execute sua seção crítica.

Principais Soluções

- **Exclusão Mútua:** Garante que apenas um processo acesse o recurso compartilhado por vez.
- **Espera Ocupada:** Em hardware, instruções atômicas como `Test-and-Set` ou `Swap`, ou em alto nível com flags globais.
- **Soluções de Alto Nível:**
  - ✔ **Semáforos:** Variáveis inteiras que sinalizam a disponibilidade de um recurso.
  - ✔ **Monitores:** Construtores de programação que encapsulam dados compartilhados e procedimentos de sincronização.
  - ✔ **Mensagens:** Sincronização através da troca de mensagens entre processos.

Semáforos

O que são?

- Variáveis inteiras que podem ser acessadas apenas por duas operações atômicas: `wait()` e `signal()`.
- `wait(S)`: Tenta Decrementar o semáforo `S`. Se `S` for menor que zero, o processo é bloqueado neste semáforo.
- `signal(S)`: Incrementa o semáforo `S`. Se `S` for menor ou igual a zero, um processo bloqueado neste semáforo é desbloqueado.
- Ver a descrição detalhada na aula sobre Condições de Corrida.

Exemplo 1

Problema: Sequência ABABAB...

- Dois processos (ou threads) são executados simultaneamente.
- O **Processo A** imprime o caractere 'A' em loop.
- O **Processo B** imprime o caractere 'B' em loop.
- **Objetivo:** Garantir que a saída seja a sequência exata **ABABABAB...**
- Este é um problema de sincronização entre processos!

Problema 1: Pseudo-código

Processo A (Sem Sincronização)

```
loop:
    imprime('A')
```

Processo B (Sem Sincronização)

```
loop:
    imprime('B')
```

Resultado Inesperado

- A ordem de execução é imprevisível.
- Saídas possíveis: `AABBAB...`, `BAABA...`, `AAAAABBBB...`

Solução com Semáforos: ABAB...

Variáveis Globais

```
semaphore semA = 1;// Permite a entrada do Processo A primeir
semaphore semB = 0;// Bloqueia a entrada do Processo B
```

Pseudo-código do Processo A

```
loop:
    wait(semA)   // Tenta obter permissão para imprimir
    imprime('A') // Executa a operação
    signal(semB) // Libera a vez para o Processo B
```

Pseudo-código do Processo B

```
loop:
    wait(semB)   // Tenta obter permissão para imprimir
    imprime('B') // Executa a operação
    signal(semA) // Libera a vez para o Processo A
```

Exemplo 2:

Sequência AABAAB...

- Novamente, dois processos: um para 'A' e outro para 'B'.
- **Objetivo:** Garantir a sequência **AABAABAAB...**
- O Processo A deve imprimir 'A' duas vezes.
- O Processo B deve imprimir 'B' uma vez.

Problema 2: Pseudo-código

Processo A (Sem Sincronização)

```
loop:
    imprime('A')
    imprime('A')
```

Processo B (Sem Sincronização)

```
loop:
    imprime('B')
```

Resultado Inesperado

- A ausência de sincronização resulta em uma ordem aleatória.
- Exemplos: `AABBAAB...`, `BABAAB...`, etc.

Solução com Semáforos: AABAAB...

Variáveis Globais

```
semaphore semA = 2; // Permite 2 impressões para Processo A
semaphore semB = 0; // Bloqueia o Processo B inicialmente
```

Pseudo-código do Processo A

```
loop:
    wait(semA)
    imprime('A')
    signal(semB)
```

Pseudo-código do Processo B

```
loop:
    wait(semB)
    wait(semB)
    imprime('B')
    signal(semA)
    signal(semA)
```

Exemplo 3

Problema dos Produtores-Consumidores

- Suponha dois processos, Produtor e Consumidor, compartilham um buffer circular de tamanho fixo.
- **Produtor:** Insere itens no buffer.
- **Consumidor:** Retira itens do buffer.
- **Problema:** Condição de corrida na variável `itens` e nos ponteiros do buffer (`primeiro`, `ultimo`).
- **Problema:** O produtor não pode inserir itens no buffer além de MAXBUFFER.
- **Problema:** O consumidor não pode consumir itens se o buffer estiver vazio.
- **Estes dois últimos problemas referem-se à Sincronia entre Processos.**

Pseudo-código do Problema (sem Sincronização)

Variáveis Globais

```
int buffer[MAXBUFFER];
int itens = 0, int primeiro = 0, int ultimo = 0;
```

escreve(int valor)

```
buffer[ultimo] = valor;    /* escreve valor no buffer */
ultimo = (ultimo + 1) % MAXBUFFER; /* buffer circular */
```

int le()

```
int tmp;
if (itens==0)  /* ops. nenhum item disponivel */
    return -1; /* retorna codigo de erro  */

tmp = buffer[primeiro];   /* recupera o valor do buffer */
primeiro = (primeiro+1)%MAXBUFFER; /* incrementa indice */
return tmp;
```

Pseudo-código do Problema (sem Sincr.)... contin.

Função Produtor

```
int item;

loop:
    produzir(&item)
    itens++
    escreve_no_buffer(&item)
```

Função Consumidor

```
int item;

loop:
    item = le_do_buffer()
    itens--
    consumir(&item)
```

Solução com Semáforos (Pseudo-código)

Variáveis Globais

```
int buffer[MAXBUFFER];
int primeiro = 0;
int ultimo = 0;
```

Semáforos

```
semaphore mutex = 1;       // Exclusão mútua
semaphore cheios = 0;      // Contagem de itens cheios
semaphore vazios = MAXBUFFER; // Contagem de espaços vazios
```

Solução com Semáforos (Pseudo-código)... contin.

Função Produtor

```
loop:
    produzir(&item)
    wait(vazios)
    wait(mutex)
    escreve_no_buffer(&item)
    signal(mutex)
    signal(cheios)
```

Função Consumidor

```
loop:
    wait(cheios)
    wait(mutex)
    item = le_do_buffer()
    signal(mutex)
    signal(vazios)
    consumir(&item)
```

Código em C (sem Sincronização)

- O código apresentado possui duas threads produtores e um processo consumidor, compartilhando um buffer circular de tamanho fixo.
- Este é um exemplo específico, poderia haver um número arbitrário de threads produtoras e consumidoras
- **Ver:** `https://www.eletrica.ufpr.br/pedroso/2022/TE355/produtor-consumidor.c`

Código em C (com Sincronização)

Ver: `https://www.eletrica.ufpr.br/pedroso/2022/TE355/produtor-consumidor-solucao.c`