Aula 11: Sincronização de Processos

Processo cooperativo é aquele que pode afetar outros processos em execução no sistema ou ser por eles afetado. Processos cooperativos tanto podem compartilhar diretamente um espaço de endereçamento lógico (isto é, código e dados) como compartilhar dados somente através de arquivos.

O acesso concorrente a dados compartilhados pode resultar em inconsistência dos dados. Nesta aula será discutido vários mecanismos para assegurar a execução ordenada de processos cooperativos que compartilham um espaço de endereçamento lógico, mantendo-se a consistência dos dados.

O Problema do buffer limitado

Chamado de Produtor e o Consumidor (também conhecido como o problema do buffer limitado), consiste em um conjunto de processos que compartilham um mesmo buffer. Os processos chamados produtores põem informação no buffer. Os processos chamados consumidores retiram informação deste buffer.

Esse é um problema clássico em sistemas operacionais, que busca exemplificar de forma clara, situações de impasses que ocorrem no gerenciamento de processos de um sistema operacional. Para quem está iniciando na programação, esse problema se torna complexo, pois trabalhar com programação concorrente nem sempre é tão simples. Como sabemos, precisamos nos preocupar com acessos ilegais a certos recursos que são compartilhados entre os processos, e manter sincronismo entre os mesmos.

Problemas: 1. O produtor insere em posição: Ainda não consumida. **2.** O consumidor remove de posição: Já foi consumida.

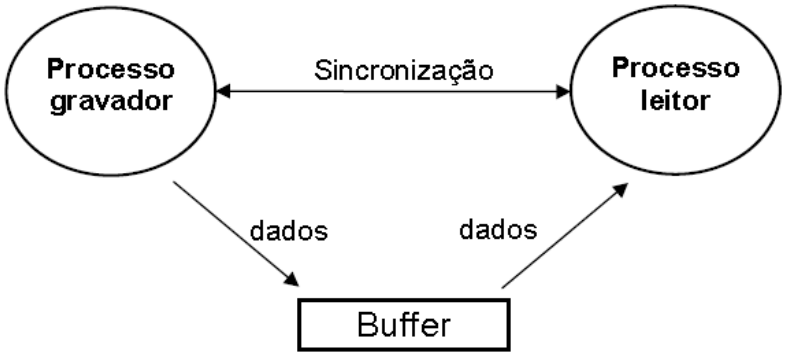


Figura 1: Exemplo de dois processos compartilhando memória.

Código Produtor

```
item nextProduced;
while (1) {
  while (counter == BUFFER_SIZE)
     /* buffer cheio */
     /* não faz coisa alguma */ ;

  buffer[counter] = nextProduced;
  counter++;
}
```

Código Consumidor

```
item nextConsumed;
while (1) {
   while (counter==0)
   /* buffer vazio */
   /* não faz coisa alguma */ ;

   nextConsumed = buffer[counter];
   counter--;
}
```

Figura 2: Código dos processos produtor e consumidor.

A variável contador guarda o número de dados contidos no buffer, ao passo que a variável n determina a capacidade do buffer. A implementação é de uma fila (FIFO), onde novo aponta para o fim da fila e prox para o início. Ao serem compilados os comandos contador++ e contador-- geram as sequências de instruções:

contador++	contador--
LOAD (#end), R1	LOAD (#end), R2
ADD 1,R1,R1	SUB 1,R2,R2
STORE R1, (#end)	STORE R2, (#end)

O exemplo a seguir ilustra uma possível execução, cujo resultado não é o esperado. Partindo do princípio que contador = 5 (armazenado em #end), após a execução das duas instruções, o contador deveria permanecer com o valor 5.

Processo executando	Instrução		(#end)	R1	R2
produtor	LOAD	(#end), R1	5	5	-
produtor	ADD	1,R1,R1	5	6	-
consumidor	LOAD	(#end), R2	5	-	5
consumidor	SUB	1,R2,R2	5	-	4
produtor	STORE	R1, (#end)	6	6	-
consumidor	STORE	R2, (#end)	4	-	4

Porém, ele indica 4, o que não corresponde a realidade. Chegou-se ao estado incorreto por que foi permitido que ambos processos manipulassem concorrentemente a variável *counter*. Tal situação, onde diversos processos acessam e manipulam os mesmos dados concorrentemente e o resultado da execução depende da ordem particular na qual o acesso ocorre, é chamada uma **condição de corrida**.

O Problema de Seção Crítica

Em programação concorrente, uma **seção crítica** é uma área de código de um algoritmo que acessa um recurso **compartilhado** que não pode ser acedido concorrentemente por mais de uma linha de execução. O objetivo é tornar a operação sobre o recurso compartilhado **atômica** (isto é, não pode ser interrompido durante sua execução). Uma seção crítica geralmente termina num tempo específico, e uma linha de execução ou processo só precisa esperar um tempo específico para entrá-la. Algum **mecanismo** de sincronização é necessário para implementar a entrada e a saída duma seção crítica para assegurar o uso exclusivo, como por exemplo um semáforo.

O acesso concorrente pode ser evitado ao controlar cuidadosamente quais variáveis são modificadas dentro e fora a seção crítica. Uma seção crítica é geralmente usada quando um programa multitarefa deve atualizar diversas variáveis relacionadas sem que outra linha de execução faça modificações conflitantes nos dados. Numa situação relacionada, uma seção crítica também pode ser usada para assegurar que um recurso compartilhado como uma impressora seja acessado por **somente um processo** a cada vez.

Os seguintes requisitos são necessários para solução do problema de seção crítica:

- **Exclusão mútua**: se um processo está na seção crítica, nenhum outro processo pode entrar na seção.
- **Progresso**: os processos participam na decisão sobre qual é o próximo processo a entrar na seção crítica.
- **Espera limitada**: se um processo deseja entrar na seção crítica, há um limite no numero de outros processos que podem entrar nela antes dele.

Soluções de Hardware

Muitos processadores possuem uma instrução especial onde um processo apenas lê o conteúdo de uma variável, e armazena seu valor em outra área podendo neste caso fazer todas as manipulações necessárias e devidas sem precisar de prioridades ou esperar que a variável original seja liberada.

Esta instrução é chamada de **test-and-set** e tem como característica ser executada sem interrupção, ou seja, trata-se de uma instrução invisível. Assim garante-se que dois processos não manipulem uma variável compartilhada ao mesmo tempo, possibilitando a implementação da exclusão mútua. O uso desta instrução especial oferece vantagens, como a simplicidade de implementação da exclusão mútua em múltiplas seções críticas e o uso da solução em arquiteturas com múltiplos processadores. A principal desvantagem é que ela não oferece espera limitada, pois a seleção do processo para o acesso ao recurso é arbitrária.

```
do {
   while (TestAndSet(&lock));
   critical section
   lock = false;
remainder section
} while (1);
```

Figura 3: Utilização de TestAndSet para uma seção crítica.

A Figura 3 ilustra uma implementação de código, onde há uma seção crítica, e através da função TestAndSet é garantido Exclusão mútua e Progresso. Entretanto, essa implementação não garante espera limitada. Para resolver este problema, pode-se criar uma vetor que armazenaria a intenção de cada processo para utilizar a seção crítica, conforme é ilustrado na Figura 4.

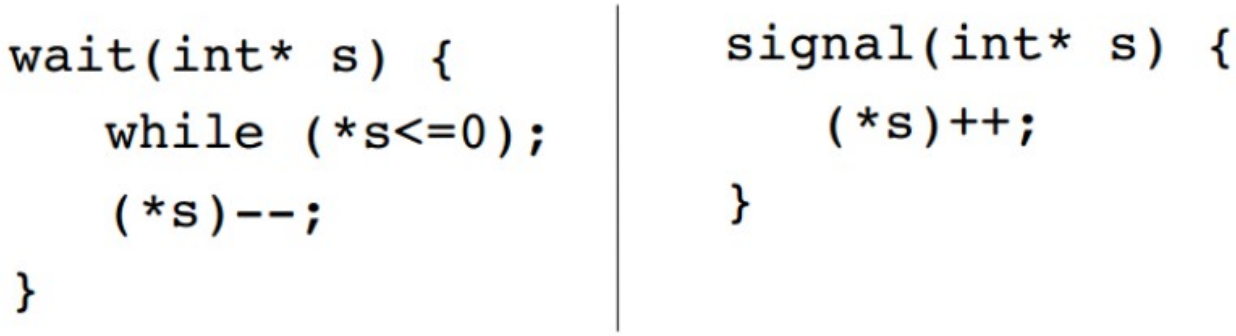
```
do {
   waiting[i] = true;
   while (waiting[i] && TestAndSet(&lock)); // Exclusao Mutua
   critical section
   j = next(i);
   while ((j!=i) && !waiting[j]) // procura o proximo
        j = next(j);  // Espera limitada
   if (j==i) { lock = false; } // ninguém esperando
   else { waiting[j] = false; } // passa a vez diretamente
   waiting[i] = false;  // Progresso
   remainder section
} while (1);
```

Figura 4: Utilização de uma fila (waiting) para garantir espera limitada.

Solução Através de Semáforos

O **semáforo** é uma variável que fica associada a um recurso compartilhado, indicando quando este está sendo acessado por um outro processo. Ela terá seu valor alterado quando o processo entra e quando sai da região crítica de forma que se um outro processo entrar em sua região critica ele possa checar antes este valor para saber se o recurso esta ou não disponível. Quando o processo tiver seu acesso impedido, ele será colocado em uma fila de espera associada ao semáforo aguardando sua vez de utilizar o recurso. Todos os processos da fila terão acesso ao recurso na ordem de chegada.

O semáforo pode ser usado também para implementar sincronizações condicionais. Isto consiste em um processo que necessita ser notificado sobre a ocorrência de um evento. Pode-se usar o semáforo para notificar este processo sobre a ocorrência deste evento.



```
wait(int* s) {
    while (*s<=0);
    (*s)--;
}
```

```
signal(int* s) {
    (*s)++;
}
```

Figura 5: Funções wait e signal para semáforos.

Transação Atômica

Transação Atômica é uma operação, ou conjunto de operações, em uma base de dados, ou em qualquer outro sistema computacional, que deve ser executada completamente em caso de sucesso, ou ser abortada completamente em caso de erro. Um exemplo prosaico que ilustra este conceito é o da gravidez. Não se diz que uma mulher está "meio grávida"; ou ela está grávida ou não está.

O exemplo clássico para a necessidade de uma "transação atômica" é aquele da transferência entre duas contas bancárias. No momento de uma transferência de valores de uma conta "A" para uma conta "B", que envolve pelo menos uma operações de ajuste no saldo para cada conta, se o computador responsável pela operação é desligado por falta de energia, espera-se que o saldo de ambas as contas não tenha se alterado. Neste caso são utilizados sistemas que suportam transações atômicas.