Sistemas Operacionais

4º período

Professora: Michelle Hanne

PUC Minas

Semáforos

Princípios da boa exclusão mútua

PUC Minas

1) Somente um processo pode estar na região crítica a cada momento
2) Não se pode fazer suposições sobre velocidade e ordem de execução de processos.

Obrigatoriamente ou não funciona

3) Nenhum processo fora da região crítica pode impedir outros processos de entrarem em suas regiões críticas
4) Um processo não pode esperar indefinidamente para entrar em sua região crítica

Desejadamente ou teremos algum problema

Espera ocupada (*busy waiting*)

PUC Minas

Sem ajuda ou interferência do SO

- Não há estado de bloqueio nem chamadas de sistema

Processo que não pode adentrar a região crítica gasta o tempo de processador inutilmente até o fim do quantum

Espera ocupada (*busy waiting*)

PUC Minas

Alternativa 1: desabilitar interrupções

- Processo desabilita interrupções ao entrar e habilita ao sair da região crítica
    - Espera indefinida
    - Segurança comprometida
- Pode ser usado pelo próprio SO

Espera ocupada (*busy waiting*)

PUC Minas

Alternativa 2: Variável de Bloqueio

- Usar variável booleana para indicar permissão de entrada

```
Debitar(Conta c, valor v, lock)  Creditar(Conta c, valor v, lock)
   while(lock);                     while(lock);
      lock = true;                     lock = true;
      saldo = c.saldo();               saldo = c.saldo();
      saldo -=v;                       saldo +=v;
      c.setSaldo(saldo);               c.setSaldo(saldo);
      lock = false;                    lock = false;
```

Espera ocupada (*busy waiting*)

PUC Minas

Alternativa 2: Variável de Bloqueio

- Usar variável booleana para indicar permissão de entrada

```
Debitar(Conta c, valor v, lock)  Creditar(Conta c, valor v, lock)
   while(lock);                     while(lock);
      lock = true;                     lock = true;
      saldo = c.saldo();               saldo = c.saldo();
      saldo -=v;                       saldo +=v;
      c.setSaldo(saldo);               c.setSaldo(saldo);
      lock = false;                    lock = false;
```

Seria ótimo, porém não funciona!!!

Espera ocupada (*busy waiting*)

PUC Minas

Alternativa 2: Variável de Bloqueio

```
Debitar(Conta c, valor v, lock)
   while(lock);
      lock = true;
      saldo = c.saldo();
      saldo -=v;
      c.setSaldo(saldo);
      lock = false;
```

```
Creditar(Conta c, valor v, lock)
   while(lock);
      lock = true;
      saldo = c.saldo();
      saldo +=v;
      c.setSaldo(saldo);
      lock = false;
```

O PROBLEMA SAIU DE *SALDO* E FOI PARA *LOCK*

Espera ocupada (*busy waiting*)

PUC Minas

Alternativa 3: Alternância Estrita

- Variável controla quem tem o direito de executar naquele turno
- Funciona, pois a ação é de ceder a vez, não de tomar

Espera ocupada (*busy waiting*)

PUC Minas

Alternativa 3: Alternância Estrita

```
P0(Conta c, valor v, turno)
   while(turno!=0);
      saldo = c.saldo();
      saldo -=v;
      c.setSaldo(saldo);
      turno = 1;
```

```
P1(Conta c, valor v, turno)
   while(turno!=1);
      saldo = c.saldo();
      saldo +=v;
      c.setSaldo(saldo);
      turno = 0;
```

turno
0

Espera ocupada (*busy waiting*)

PUC Minas

Alternativa 3: Alternância Estrita

```
P0(Conta c, valor v, turno)
   while(turno!=0);
      saldo = c.saldo();
      saldo -=v;
      c.setSaldo(saldo);
      turno = 1;
```

```
P1(Conta c, valor v, turno)
   while(turno!=1);
      saldo = c.saldo();
      saldo +=v;
      c.setSaldo(saldo);
      turno = 0;
```

Espera ocupada (*busy waiting*)

PUC Minas

Alternativa 3: Alternância Estrita

```
P0(Conta c, valor v, turno)
   while(turno!=0);
      saldo = c.saldo();
      saldo -=v;
      c.setSaldo(saldo);
      turno = 1;
```

```
P1(Conta c, valor v, turno)
   while(turno!=1);
      saldo = c.saldo();
      saldo +=v;
      c.setSaldo(saldo);
      turno = 0;
```

turno
1

Espera ocupada (*busy waiting*)

PUC Minas

Alternativa 3: Alternância Estrita

```
P0(Conta c, valor v, turno)
   while(turno!=0);
      saldo = c.saldo();
      saldo -=v;
      c.setSaldo(saldo);
      turno = 1;
```

```
P1(Conta c, valor v, turno)
   while(turno!=1);
      saldo = c.saldo();
      saldo +=v;
      c.setSaldo(saldo);
      turno = 0;
```

turno
1

Espera ocupada (*busy waiting*)

PUC Minas

Alternativa 3: Alternância Estrita

```
P0(Conta c, valor v, turno)
   while(turno!=0);
      saldo = c.saldo();
      saldo -=v;
      c.setSaldo(saldo);
      turno = 1;

P1(Conta c, valor v, turno)
   while(turno!=1);
      saldo = c.saldo();
      saldo +=v;
      c.setSaldo(saldo);
      turno = 0;
```

turno
0

- Funciona, porém...
  - Múltiplos processos e desempenho
  - Bloqueio desnecessário

Solução de Sincronismo via Software

PUC Minas

Algoritmos são implementados como ferramentas de sincronismo de comunicação entre processos.

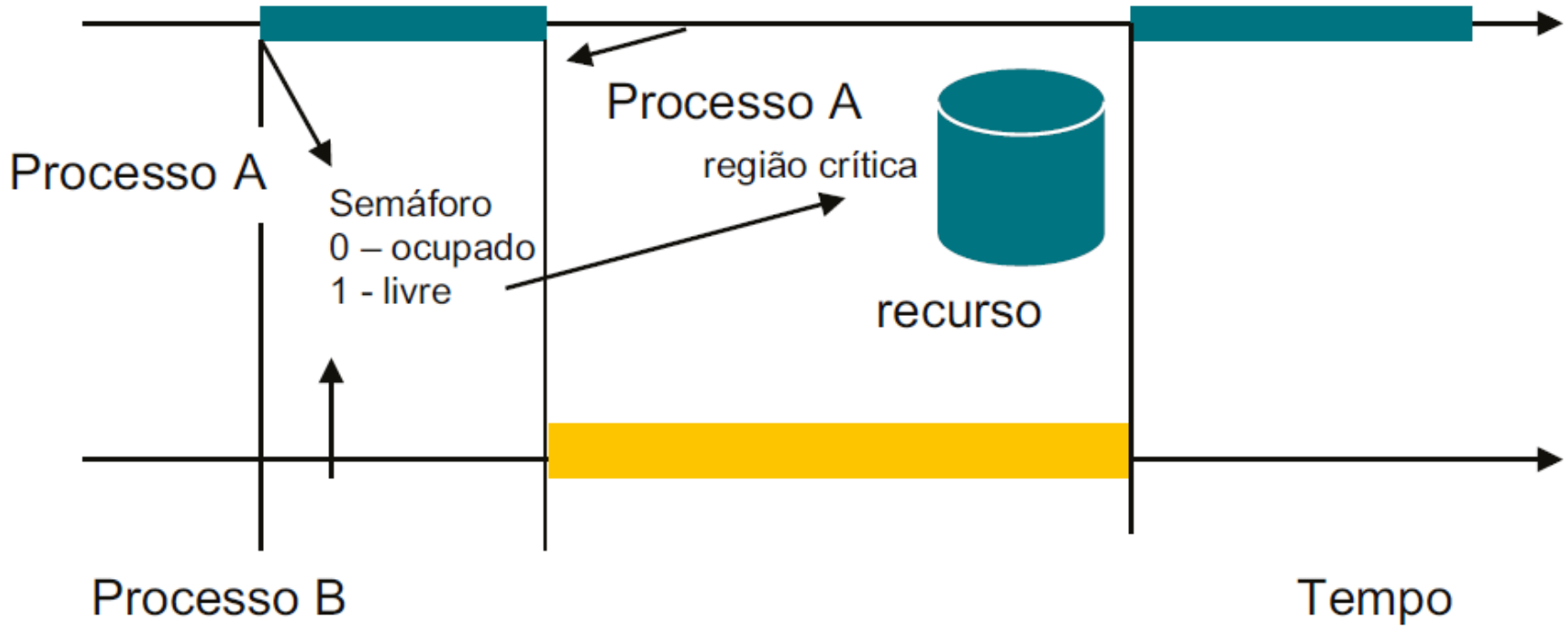
Semáforos

Semáforos é um conceito para problema de sincronismo entre processos concorrentes que faz uso de uma variável inteira não negativa (STUART, 2011). Essa variável só pode ser manipulada por duas instruções: **DOWN** e **UP**. Essas instruções funcionam como sinalizadores de entrada e saída de um processo em sua região crítica.

O semáforo fica associado a um recurso compartilhado e indica se o recurso está sendo acessado por um processo. Se o valor for **superior a 0,** indica que o recurso está **disponível**, caso contrário, o recurso fica **indisponível ao acesso por um processo.**

Semáforos

PUC Minas



O processo A acessa a sua região crítica na alocação do recurso devido o semáforo sinalizar 1, processo B aguarda.

Problema produtor/consumidor usado com semáforo.

Semáforos

PUC Minas

Proposta de Dijkstra (1965)

- Variáveis especiais para controlar o numero de sinais pendentes
- Permitem apenas duas operações indivisíveis:
  - **Up (originalmente V, *verhogen* → incrementar)**
  - **Down (originalmente P, *proberen* → testar)**
  - Oferecidos por meio de chamadas de sistema do SO
    - Operação em modo kernel
    - Interrupções desabilitadas
  - Disponibilizados por meio de bibliotecas de programação
    - Operação pelo processo, de acordo com a compilação
    - Processo controla suas threads e pedidos

Semáforos

PUC Minas

Comportamento geral:

- **Down:** tenta decrementar o contador. Se não for possível, manda o processo para bloqueio
- **Up:** incrementa o contador e libera processos pendentes

Um mutex (MUTual EXclusion) é um semáforo simplificado:

- **valores 0 ou 1 apenas**
- **“Semáforos binários”**
- **Úteis em cenários mais simples, quando apenas o bloqueio de uma região é necessário**

Semáforos – Exemplo Hipotético

```
Debitar(Conta c, valor v)
   down(c.semaforo);
      saldo = c.saldo();
      saldo -=v;
      c.setSaldo(saldo);
   up(c.semaforo);
```

```
Creditar(Conta c, valor v)
   down(c.semaforo);
      saldo = c.saldo();
      saldo +=v;
      c.setSaldo(saldo);
   up(c.semaforo);
```

CONTA
semáforo
1

```
Debitar(Conta c, valor v)
   down(c.semaforo);
      saldo = c.saldo();
      saldo -=v;
      c.setSaldo(saldo);
   up(c.semaforo);
```

```
Creditar(Conta c, valor v)
   down(c.semaforo);
      saldo = c.saldo();
      saldo +=v;
      c.setSaldo(saldo);
   up(c.semaforo);
```

CONTA
semáforo
0

Semáforos – Exemplo Hipotético

```
Debitar(Conta c, valor v)
   down(c.semaforo);
      saldo = c.saldo();
      saldo -=v;
      c.setSaldo(saldo);
   up(c.semaforo);
```

```
Creditar(Conta c, valor v)
   down(c.semaforo);
      saldo = c.saldo();
      saldo +=v;
      c.setSaldo(saldo);
   up(c.semaforo);
```

CONTA
semáforo
0

```
Debitar(Conta c, valor v)
   down(c.semaforo);
      saldo = c.saldo();
      saldo -=v;
      c.setSaldo(saldo);
   up(c.semaforo);
```

```
Creditar(Conta c, valor v)
   down(c.semaforo);
      saldo = c.saldo();
      saldo +=v;
      c.setSaldo(saldo);
   up(c.semaforo);
```

CONTA
semáforo
0

Semáforos – Exemplo Hipotético

```
Debitar(Conta c, valor v)
   down(c.semaforo);
      saldo = c.saldo();
      saldo -=v;
      c.setSaldo(saldo);
   up(c.semaforo);
```

```
Creditar(Conta c, valor v)
   down(c.semaforo);
      saldo = c.saldo();
      saldo +=v;
      c.setSaldo(saldo);
   up(c.semaforo);
```

CONTA
semáforo
0

```
Debitar(Conta c, valor v)
   down(c.semaforo);
      saldo = c.saldo();
      saldo -=v;
      c.setSaldo(saldo);
   up(c.semaforo);
```

```
Creditar(Conta c, valor v)
   down(c.semaforo);
      saldo = c.saldo();
      saldo +=v;
      c.setSaldo(saldo);
   up(c.semaforo);
```

CONTA
semáforo
1

Semáforos – Exemplo Hipotético

```
Debitar(Conta c, valor v)
   down(c.semaforo);
      saldo = c.saldo();
      saldo -=v;
      c.setSaldo(saldo);
   up(c.semaforo);
```

```
Creditar(Conta c, valor v)
   down(c.semaforo);
      saldo = c.saldo();
      saldo +=v;
      c.setSaldo(saldo);
   up(c.semaforo);
```

CONTA
semáforo
0

Produtor / Consumidor com Semáforos

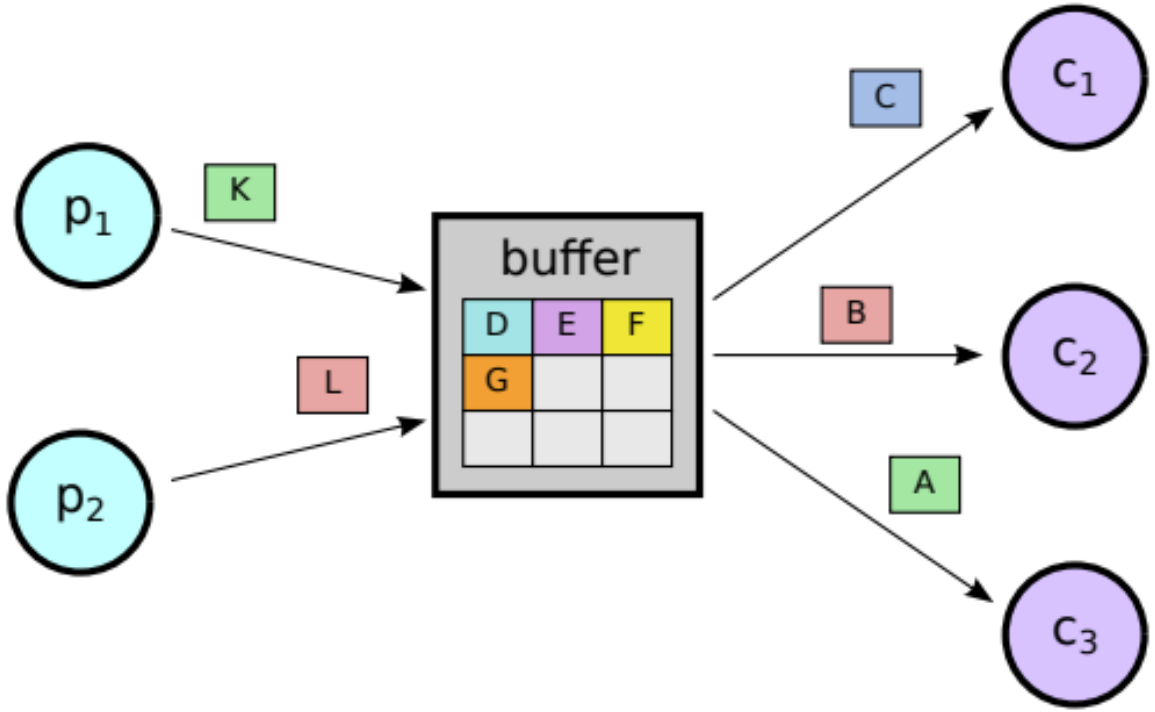
PUC Minas

Este problema também é conhecido como o problema do buffer limitado, e consiste em coordenar o acesso de tarefas (processos ou threads) a um buffer compartilhado com capacidade de armazenamento limitada ***a N itens (que podem ser inteiros, registros, mensagens, etc.).***

- **Produtor**: **produz e deposita um item no buffer, caso o mesmo tenha uma vaga livre.** Caso contrário, deve esperar até que surja uma vaga. Ao depositar um item, o produtor "consome" uma vaga livre.
- **Consumidor**: retira um item do buffer e o consome; caso o buffer esteja vazio, aguarda que novos itens sejam depositados pelos produtores. Ao consumir um item, o consumidor "produz" uma vaga livre no buffer

Fonte: https://wiki.inf.ufpr.br/maziero/lib/exe/fetch.php?media=socm:socm-12.pdf

Produtor / Consumidor com Semáforos



Deve-se observar que o acesso ao buffer é bloqueante, ou seja, cada processo fica bloqueado até conseguir fazer seu acesso, seja para produzir ou para consumir um item.

- **A exclusão mútua no acesso ao buffer, para evitar condições de disputa entre produtores e/ou consumidores que poderiam corromper o conteúdo do buffer.**

- **A suspensão dos produtores no caso do buffer estar cheio**: os produtores devem esperar até que surjam vagas livres no buffer.

- **A suspensão dos consumidores no caso do buffer estar vazio:** os consumidores devem esperar até que surjam novos itens a consumir no buffer

Fonte: https://wiki.inf.ufpr.br/maziero/lib/exe/fetch.php?media=socm:socm-12.pdf

Produtor / Consumidor com Semáforos

PUC Minas

```
1  mutex mbuf ;          // controla o acesso ao buffer
2  semaphore item ;      // controla os itens no buffer (inicia em 0)
3  semaphore vaga ;      // controla as vagas no buffer (inicia em N)
4
5  task produtor ()
6  {
7     while (1)
8     {
9        ...                    // produz um item
10       down (vaga) ;          // espera uma vaga no buffer
11       lock (mbuf) ;          // espera acesso exclusivo ao buffer
12       ...                    // deposita o item no buffer
13       unlock (mbuf) ;        // libera o acesso ao buffer
14       up (item) ;            // indica a presença de um novo item no buffer
15    }
16 }
17
18 task consumidor ()
19 {
20    while (1)
21    {
22       down (item) ;          // espera um novo item no buffer
23       lock (mbuf) ;          // espera acesso exclusivo ao buffer
24       ...                    // retira o item do buffer
25       unlock (mbuf) ;        // libera o acesso ao buffer
26       up (vaga) ;            // indica a liberação de uma vaga no buffer
27       ...                    // consome o item retirado do buffer
28    }
29 }
```

Solução é genérica, pois não depende do tamanho do buffer, do número de produtores nem do número de consumidores.

Fonte: https://wiki.inf.ufpr.br/maziero/lib/exe/fetch.php?media=socm:socm-12.pdf

Produtor / Consumidor com Semáforos

```
1  mutex mbuf ;                    // controla o acesso ao buffer
2  condition item ;                // condição: existe item no buffer
3  condition vaga ;                // condição: existe vaga no buffer
4
5  task produtor ()
6  {
7     while (1)
8     {
9        ...                       // produz um item
10       lock (mbuf) ;             // obtem o mutex do buffer
11       while (num_itens == N)    // enquanto o buffer estiver cheio
12          wait (vaga, mbuf) ;    // espera uma vaga, liberando o buffer
13       ...                       // deposita o item no buffer
14       signal (item) ;           // sinaliza um novo item
15       unlock (mbuf) ;           // libera o buffer
16    }
17 }
18
19 task consumidor ()
20 {
21    while (1)
22    {
23       lock (mbuf) ;             // obtem o mutex do buffer
24       while (num_items == 0)    // enquanto o buffer estiver vazio
25          wait (item, mbuf) ;    // espera um item, liberando o buffer
26       ...                       // retira o item no buffer
27       signal (vaga) ;           // sinaliza uma vaga livre
28       unlock (mbuf) ;           // libera o buffer
29       ...                       // consome o item retirado do buffer
30    }
31 }
```

SOLUÇÃO USANDO VARIÁVEL DE CONDIÇÃO:

- O problema dos produtores/consumidores também pode ser resolvido com variáveis de condição. Além do mutex para acesso exclusivo ao buffer, são necessárias variáveis de condição para indicar a presença de itens e de vagas no buffer.

Fonte: https://wiki.inf.ufpr.br/maziero/lib/exe/fetch.php?media=socm:socm-12.pdf

Produtor / Consumidor com Semáforos

PUC Minas

Produtores de itens (escrevem na memória)
Consumidores de itens (retiram da memória)

Precisamos proteger em três cenários:

- **Buffer cheio** (produtor)
- **Buffer vazio** (consumidor)
- **Escrita/retirada do buffer** (ambos)

• Dois processos compartilham um buffer de tamanho fixo • O produtor insere informação no buffer • O consumidor remove informação do buffer

Produtor / Consumidor com Semáforos

PUC Minas

```
int N = Buffer.CAPACIDADEMAXIMA;
Semaforo cheio = new Semaforo(0);
Semaforo vazio = new Semaforo(N);
Semaforo mutex = new Semaforo(1);
```

Produtor / Consumidor com Semáforos

PUC Minas

```
Produtor(){
   Item it;
   while(true){
      it = produzir();
      down(vazio);
      down(mutex)
         buf.inserir(it);
      up(mutex);
      up(cheio);
   }
}
```

```
Consumidor(){
   Item it;
   while(true){
      down(cheio);
      down(mutex)
         it = buf.retirar(it);
      up(mutex);
      up(vazio);
      consumir(it);
   }
}
```

Semáforos

PUC Minas

Características dos semáforos:

- É simples e sempre tem um valor *Integer* não negativo.
- Trabalha com muitos processos.
- Pode ter muitas seções críticas diferentes com diferentes semáforos.
- Cada seção crítica possui semáforos de acesso exclusivos.
- Pode permitir múltiplos processos na seção crítica de uma só vez, se desejável.

Os semáforos com limitação podem bloquear um processo indefinidamente, gerando *deadlock*.

Semáforo Binário

PUC Minas

O semáforo binário é um objeto de sincronização que possui somente dois estados: wait(), e signal().

- Os semáforos binários não possuem atributo de propriedade e podem ser liberados por qualquer ***thread*** ou manipulador de interrupção, independentemente de quem executou a última operação. **Devido a isso, os semáforos binários são frequentemente usados para sincronizar encadeamentos com eventos externos implementados, por exemplo, esperando por um pacote de uma rede ou esperando que um botão seja pressionado.**

Então, semáforo binário protege o acesso a um recurso compartilhado **ÚNICO**, sempre que você tiver um requisito para proteger o acesso a um recurso **ÚNICO** acessado por vários segmentos, o semáforo binário será mais adequado.