

Semáforo

Volnys Borges Bernal
volnys.bernal@usp.br

Depto. de Eng. de Sistemas Eletrônicos
Escola Politécnica da USP

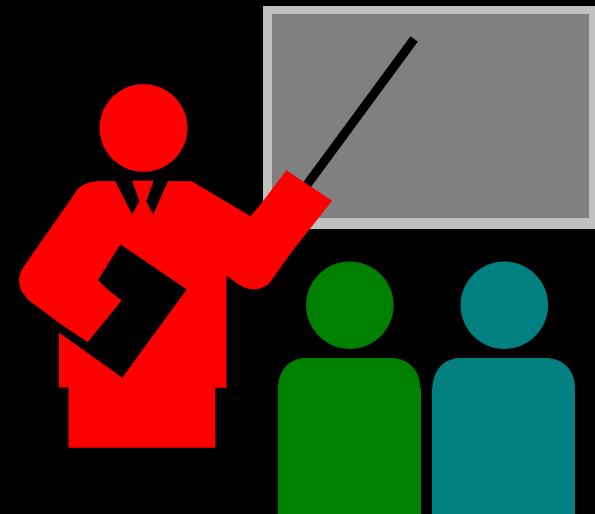


Semáforo

□ Sumário

- ❖ Semáforo
- ❖ Problema produtor-consumidor usando semáforo
- ❖ Semáforo binário
- ❖ Interface pthreads para semáforo

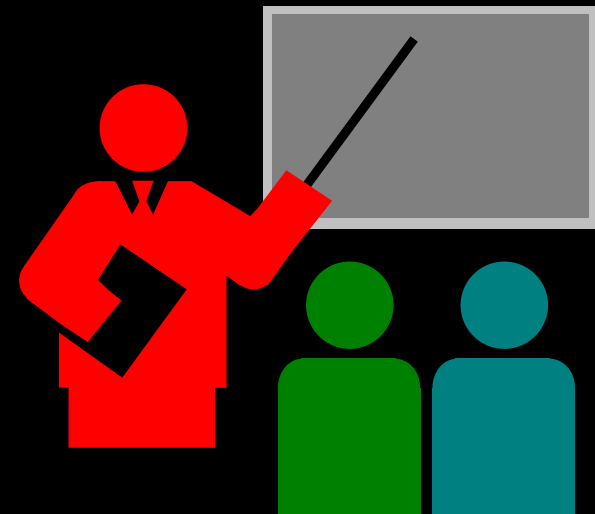
Semáforo



Semáforo

- ❑ **Método de sincronização que permite a contagem de recursos disponíveis**
 - ❖ Contador geralmente representa a quantidade de recursos disponíveis
- ❑ **Primitivas**
 - ❖ **Up(semáforo)**
 - Incrementa o contador do semáforo.
 - Se existirem entidades bloqueadas neste semáforo, uma delas é desbloqueada
 - ❖ **Down(semáforo)**
 - Decrementa o semáforo
 - Se o resultado for menor que zero, a entidade fica bloqueada neste semáforo.
 - ❖ **Init(semáforo,valor)**
- ❑ **Estas primitivas são atômicas (indivisíveis)**
- ❑ **O semáforo deve ser iniciado com um valor inteiro, geralmente associado à quantidade de recursos disponíveis.**

Problema do produtor-consumidor com semáforo



Problema do produtor-consumidor

- ❑ O problema do produtor consumidor possui 3 necessidades de sincronização (espera):

1. Uso da região crítica

- ❖ Tanto produtor quanto consumidor pode, eventualmente, precisar esperar para entrar na região crítica
- ❖ Quantidade de recursos disponíveis no início = 1
 - ❖ No início, somente 1 thread pode usar a região crítica

2. Espera por itens na fila

- ❖ Consumidor espera por itens na fila quando fila está vazia
- ❖ Quantidade de recursos disponíveis no início = 0
 - ❖ No início não existe nenhum item na fila

3. Espera por slots livres

- ❖ Produtor espera por slots livres quando fila está cheia
- ❖ Quantidade de recursos disponíveis no início = N (tamanho da fila)
 - ❖ No início existe N slots livres na fila

Problema do produtor-consumidor

- ❑ Cada necessidade de sincronização pode ser representada por um semáforo
- ❑ São necessários 3 semáforos:
 - ❖ Região crítica → bloqueia entidade (produtor ou consumidor) caso a região crítica esteja ocupada
 - ❖ Itens na fila → bloqueia consumidor caso fila esteja vazia
 - ❖ Slots livres → bloqueia produtor caso fila esteja cheia
- ❑ Quantidade de recursos disponíveis inicialmente:
 - ❖ Região crítica → 1 (permitido 1 thread por vez)
 - ❖ Itens na fila → 0 (inicialmente nenhum item na fila)
 - ❖ Slots livres → N (N=tamanho da fila)

Problema do produtor-consumidor

```
semaforo mutex = 1;  
semaforo itens = 0;  
semaforo slots = N;
```

Produtor

Repetir

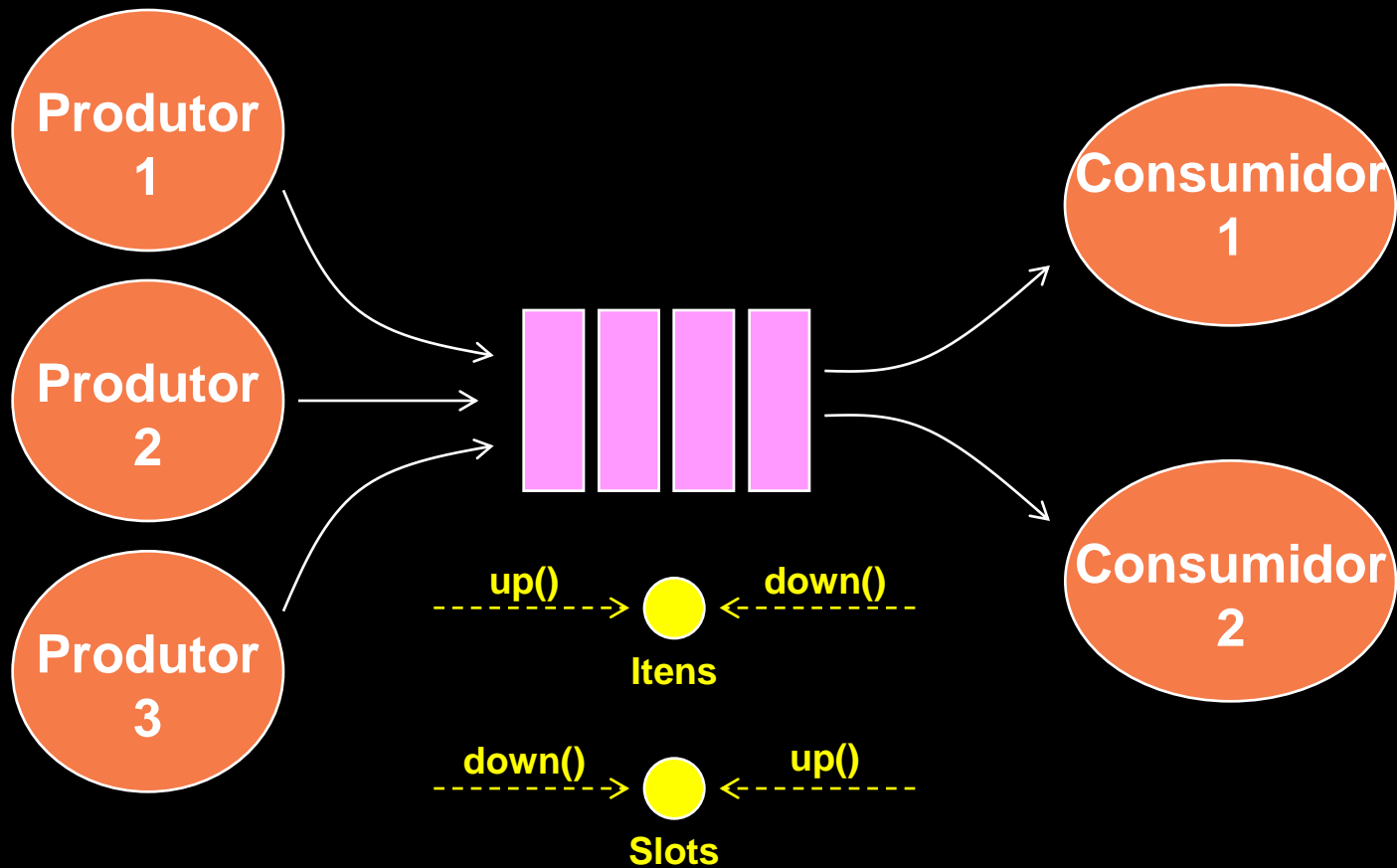
```
    Produzir(E) ;  
    Down(slots) ;  
    Down(mutex) ;  
    InserirFila(F,E) ;  
    Up(mutex) ;  
    Up(itens) ;
```

Consumidor

Repetir

```
    Down(itens) ;  
    Down(mutex) ;  
    E=RetirarFila(F) ;  
    Up(mutex) ;  
    Up(slots) ;  
    Processar(E) ;
```


Problema do produtor-consumidor



Exercício

(1) O que ocorre caso seja invertida a ordem de utilização dos semáforos no exercício anterior, ou seja:

Produtor

Repetir

 Produzir(E) ;

Down (mutex) ;

Down (slots) ;

 InserirFila(F,E) ;

Up (itens) ;

Up (mutex) ;

Consumidor

Repetir

Down (mutex) ;

Down (itens) ;

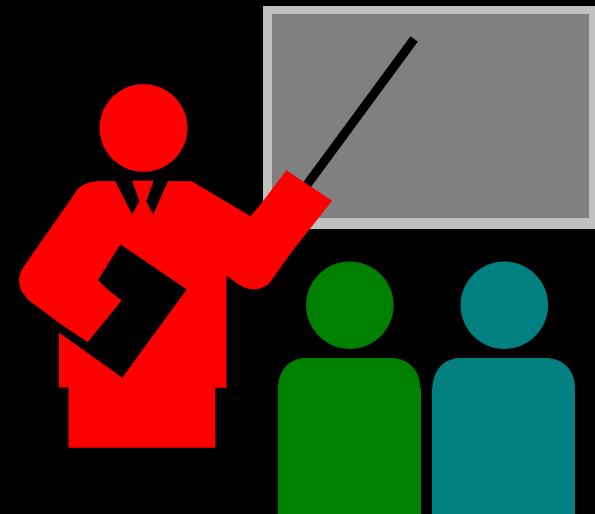
 E=RetirarFila(F) ;

Up (slots) ;

Up (mutex) ;

 Processar(E) ;

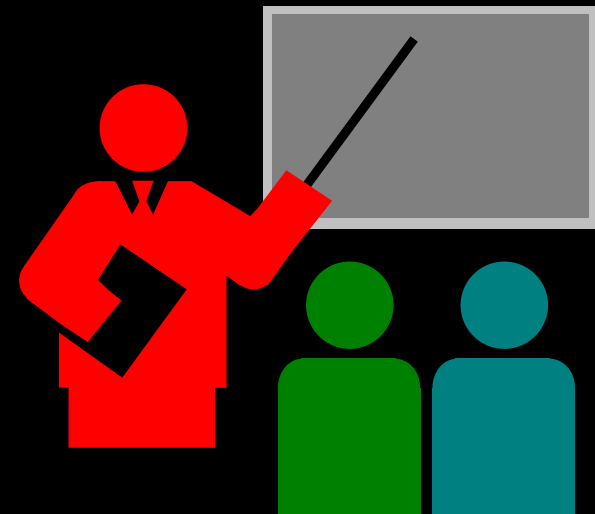
Semáforo Binário



Semáforo Binário

- ❑ Caso particular de semáforo no qual é iniciado com valor 1 e cujo valor nunca ultrapassa 1
- ❑ Pode ser utilizado para implementação de exclusão mútua:
 - ❖ `lock()` == `down(semáforo_binário)`
 - ❖ `unlock()` == `up(semáforo_binário)`

Interface pthreads para semáforo



Interface pthreads para semáforo

□ Tipos de dados

Tipo	Descrição
<code>sem_t</code>	Representa o tipo de um semáforo.

Interface pthreads para semáforo

□ Primitivas

Primitiva	Descrição
<code>sem_init</code>	Iniciação da variável de um semáforo.
<code>sem_wait</code>	Down. Decrementa o semáforo. Se o valor resultante for menor que zero a entidade de processamento é bloqueada.
<code>sem_trywait</code>	Variante de Down. Caso o valor do semáforo seja igual ou menor que zero, retorna. Caso contrário, decrementa o semáforo.
<code>sem_post</code>	Up. Incrementa o semáforo. Se existirem entidades de processamento bloqueadas neste semáforo, uma delas é desbloqueada.
<code>sem_get_value</code>	Retorna o contador do semáforo.
<code>sem_destroy</code>	Destrói um semáforo.

Interface pthreads para semáforo

□ Sintaxe das primitivas

```
#include <semaphore.h>
```

```
int  sem_init      (sem_t *sem, int pshared, unsigned int value)  
int  sem_wait      (sem_t *sem)  
int  sem_trywait   (sem_t *sem)  
int  sem_post      (sem_t *sem)  
int  sem_getvalue  (sem_t *sem, int *sval)  
int  sem_destroy   (sem_t *sem)
```

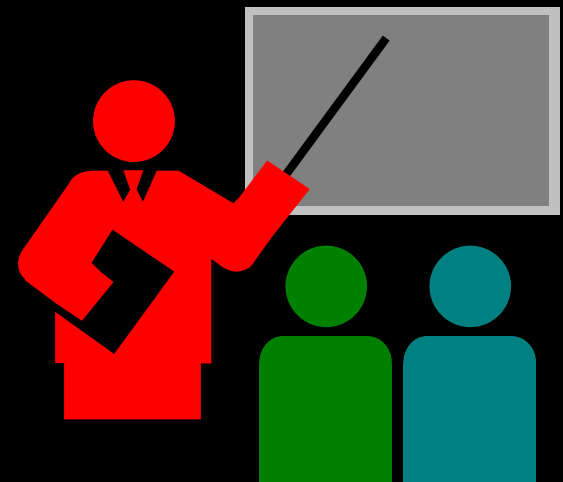

Interface pthreads para semáforo

□ Exemplo de uso:

```
#include <semaphore.h>

...
sem_t slots;
...
status = sem_init(&slots,0,10);
...
status = sem_wait(&slots);
...
status = sem_post(&slots);
...
```

Sincronização na cooperação entre threads



Sincronização na cooperação entre threads

- ❑ Uma das necessidades de sincronização é na cooperação das atividades entre threads.
- ❑ A cooperação pode gerar dependência da ordem de execução das atividades.
- ❑ Dependência entre atividades:
 - ❖ “y” \leftarrow “x”
 - ❖ Atividade “y” depende da execução da atividade “x”
 - ❖ Atividade “y” pode somente ser executada após a execução da atividade “x”
- ❑ O uso de semáforo é voltado principalmente para sincronização por recursos, porém pode também ser utilizado na sincronização de cooperação.

Sincronização na cooperação entre threads

❑ Exemplo:

```
thread_A()  
  work_a()  
end
```

```
thread_B()  
  work_b()  
end
```

```
thread_C()  
  work_c()  
end
```

```
main()  
  thread_create(thread_A)  
  thread_create (thread_B)  
  thread_create(thread_C)  
  thread_join(thread_A)  
  thread_join(thread_B)  
  thread_join(thread_C)  
end
```

Sincronização na cooperação entre threads

□ Exemplo:

❖ Suponha que, no exemplo anterior exista a seguinte dependência de execução de atividades:

- $c \leftarrow a$ (“c” depende de “a”)
- $c \leftarrow b$ (“c” depende de “b”)

Sincronização da cooperação entre threads

$c \leftarrow a$ (“c” depende de “a”)

$c \leftarrow b$ (“c” depende de “b”)

sem_t ac

sem_t bc

thread_A()

work_a()

sem_up(ac)

end

thread_B()

work_b()

sem_up(bc)

end

thread_C()

sem_down(ac)

sem_down(bc)

work_c()

end

main()

sem_init(ac,0)

sem_init(bc,0)

thread_create(thread_A)

thread_create (thread_B)

thread_create(thread_C)

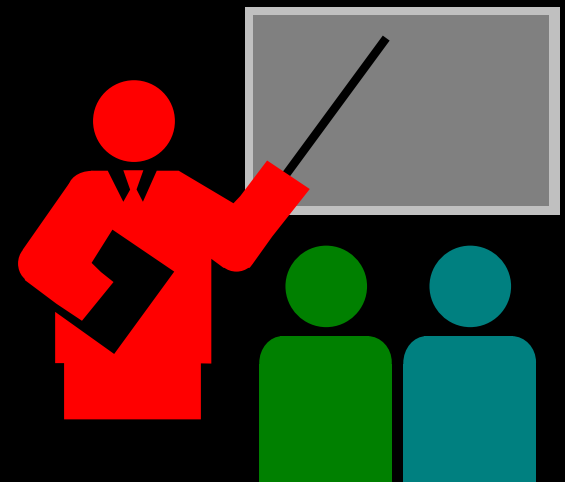
thread_join(thread_A)

thread_join(thread_B)

thread_join(thread_C)

end

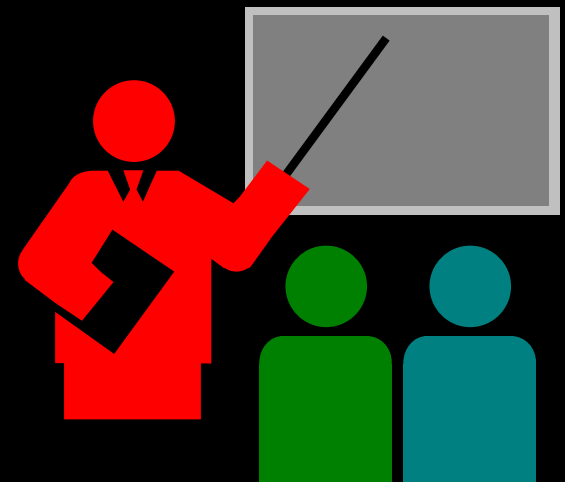
Exercício



Exercício

- (1) Adicione sincronização utilizando primitivas de semáforo de forma a garantir a execução dos threads A, B, C, D e E de acordo com as seguintes dependências:
 - ❖ $B \leftarrow A$
 - ❖ $C \leftarrow B$
 - ❖ $D \leftarrow B$
 - ❖ $E \leftarrow B$
 - ❖ onde $Y \leftarrow X$ significa que Y depende de X, ou seja, Y pode executar somente após a execução de X.

Referências Bibliográficas



Referências Bibliográficas

- ❑ **ANDREW S. TANENBAUM; Sistemas Operacionais Modernos. Prentice-Hall.**
 - ❖ Capítulo 2

- ❑ **ANDREW S. TANENBAUM; Sistemas Operacionais. Prentice-Hall.**
 - ❖ Capítulo 2