Fundamentos de Sistemas de Operação

Unix Windows NT Netware MacOS DOS/VS Vax/VMS
Linux Solaris HP/UX AIX Mach
Chorus

Programação Concorrente:

Sincronização: Enriquecendo a API

Semáforos Generalizados

- *Tipo abstracto de dados, proposto por E. W. Dijkstra*
  - *"Contentor" para valores inteiros não-negativos*
  - *Definida uma operação de inicialização e duas de manipulação, P (de "proberen", testar) e V (de "verhogen", incrementar)*
  - **`P(s)`***: Se* **`s > 0`** *s é decrementado e o "processo" invocador continua a sua execução; senão, se* **`s = 0`***, o invocador é bloqueado.*
  - **`V(s)`***: incrementa o valor do semáforo* **`s`***; quando tal acontecer, se houver processo(s) bloqueado(s) em* **`P(s)`***, um deles consegue progredir, continuando a execução, enquanto os outros continuam à espera da sua oportunidade ☺*

Aplicações de Semáforos (1)

- *Os semáforos são uma solução generalista que pode ser usada para resolver muitíssimos (?todos?) os problemas de sincronização de processos concorrentes; exemplos:*
  - *Semáforo para resolver problemas de Exclusão Mútua*
    - *Inicializado a 1, "oscila" entre 0 e 1*
  - *Semáforo "contador"*
    - *"Oscila" entre 0 e N (podendo ser inicializado a N e decrescer/crescer entre N e 0 ou, pelo contrário, ser inicializado a 0 e crescer/decrescer entre 0 e N)*
  - *Outros (que não abordamos agora ☺)*

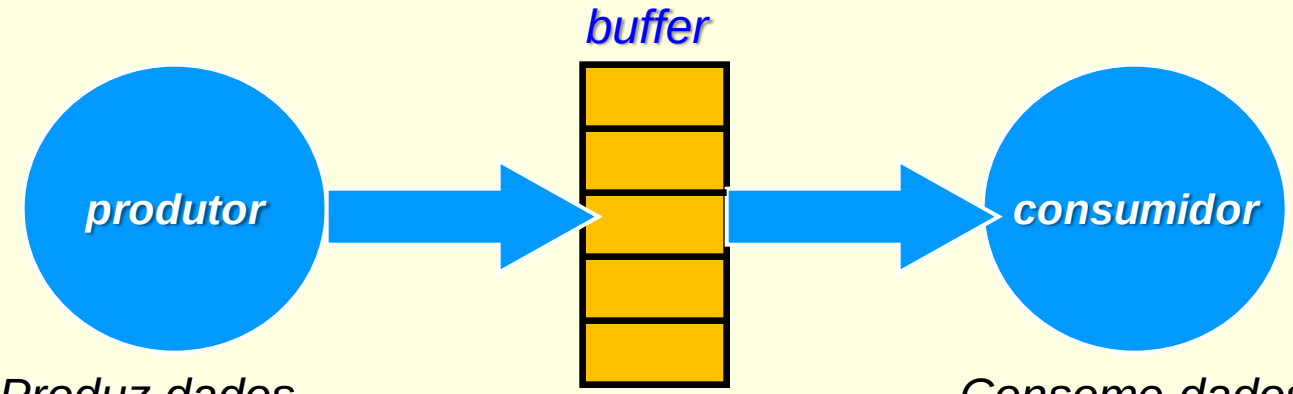
Aplicações de Semáforos (2)

- *Exclusão mútua / Região Crítica*
  - *Seja* **s** *um semáforo inicializado a 1*
  - *O seguinte fragmento realiza uma RC, ou zona de Exclusão Mútua:*

```
P(s)
// Código a executar em EM
V(s)
```

Aplicações de Semáforos (3)

- *Produtor/Consumidor com Buffer de capacidade N*



Produz dados
Coloca-os no buffer

Consome dados
Retira-os do buffer

SINCRONIZAÇÃO

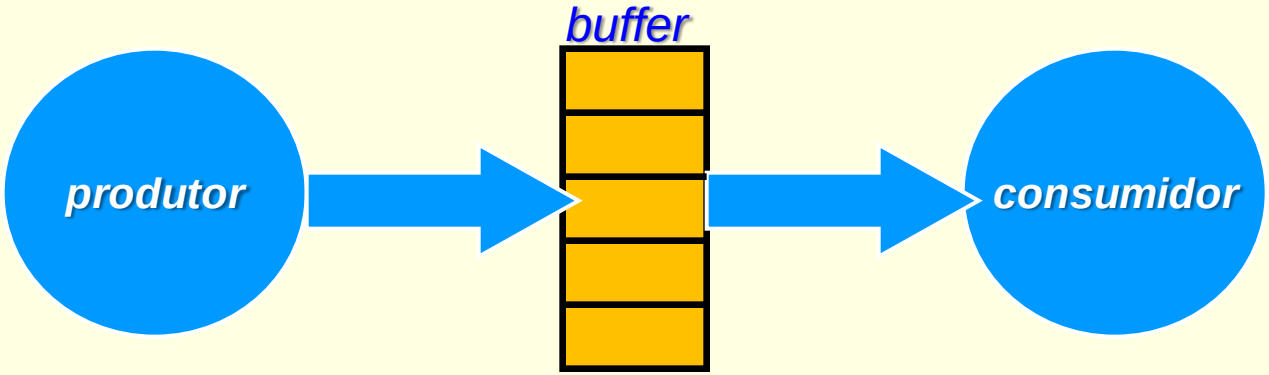
Não pode colocar se buffer cheio! Bloquear!

Não pode retirar se buffer vazio! Bloquear!

Acesso ao buffer em Exclusão Mútua!

Aplicações de Semáforos (4)

- *Produtor/Consumidor com Buffer de capacidade N*

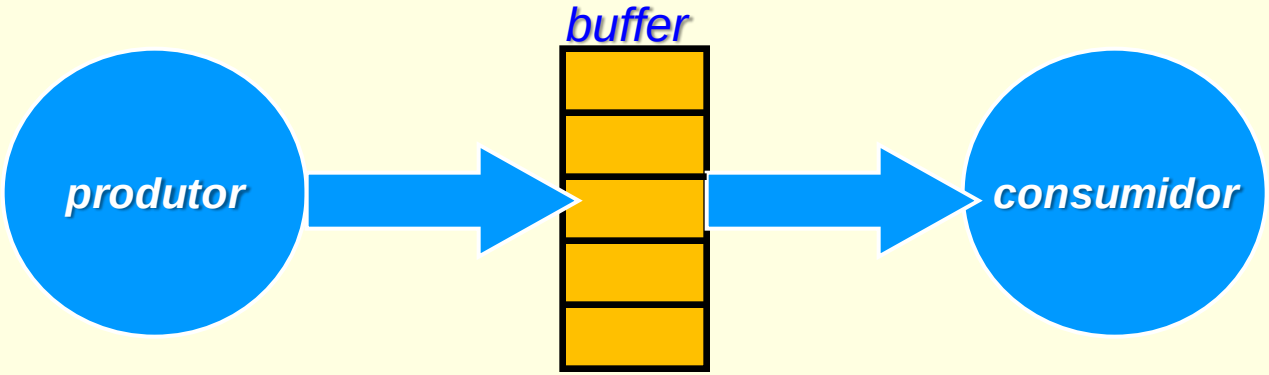



```
Sem slotsCheios= 0;

    Produzir();
    P(slotsVazios);
RC: PorNoBuffer();
    V(slotsCheios);
```

```
Sem slotsVazios= N;

    P(slotsCheios);
RC: TirarDoBuffer();
    Consumir();
    V(slotsVazios);
```

Aplicações de Semáforos (5)

- *Produtor/Consumidor: exclusão mútua no acesso ao buffer*




```
Sem exmut= 1;
```

```
...
P(exmut);
  PorNoBuffer();
V(exmut);
...
```

```
...
P(exmut);
  TirarDoBuffer();
V(exmut);
...
```

APIs de Semáforos

- *Em sistemas Unix estão disponíveis duas APIs*
  - *Semáforos System V (ou Unix IPC)*
    - *Vectores de semáforos generalizados*
    - *Operações de inicialização e atómicas do tipo P/V sobre vectores de semáforos*
  - *Semáforos POSIX*
    - *Semáforos generalizados "à la Dijkstra"*

Semáforos POSIX (1)

- *Declaração*
  - **`sem_t semEM;`**
- *Inicialização*
  - **`int sem_init(sem_t *s; int pshared, int v);`**

  Inicializa um semáforo **`s`** *com o valor* **`v`**. *O argumento* **`pshared`** *indica se o semáforo é local (partilhado apenas entre threads do processo), ou global (partilhado com outros processos); semáforos globais têm de ser declarados de forma partilhada: por* **`fork()`** *ou em "shared memory" (não estudada).*

Semáforos POSIX (2)

- *P*
  - **`int sem_wait(sem_t *s);`**

  A operação **`sem_wait`** *respeita integralmente a semântica definida por Dijkstra para* **`P`**.

- *V*
  - **`int sem_post(sem_t *s);`**

  A operação **`sem_post`** *respeita integralmente a semântica definida por Dijkstra para* **`V`**.

Semáforos POSIX (3)

- *Remoção*
  - **`int sem_destroy(sem_t *s);`**

  A operação **`sem_destroy`** *torna o semáforo inacessível a outras operações que não (um novo)* **`sem_init`**.

- *Notas finais*
  - *A norma POSIX define outras operações para além das propostas por Dijkstra; por exemplo,* **`sem_getvalue()`** *e* **`sem_trywait()`**. *Tais "extensões" da proposta de Dijkstra não serão aqui abordadas.*
  - *A API apresentada designa-se semáforos POSIX "sem nome" (unnamed); existe uma outra, semáforos POSIX "com nome", que não estudamos aqui.*

Para fazer…

- *Reescrever o programa de incremento de contador por 2 threads*
  - *Substituir o mutex por um semáforo*
- *Reescrever o programa de sincronização de threads*
  - *Substituir a variável condicional por semáforo(s)*
- *Reescrever o Produtor/Consumidor com Buffer[N]*
  - *Substituir as variáveis condicionais e o mutex por semáforo(s)*