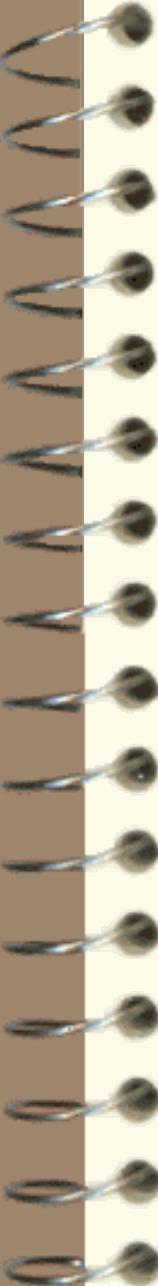


Problema do Produtor -Consumidor

Solução por semáforos



Autoria

✓ Autores

- alunos da disciplina SO II

✓ Local

- Instituto de Informática – UFRGS

✓ Versão

- v7
- agosto de 2008
- por C. Geyer



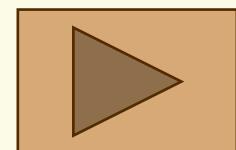
Índice

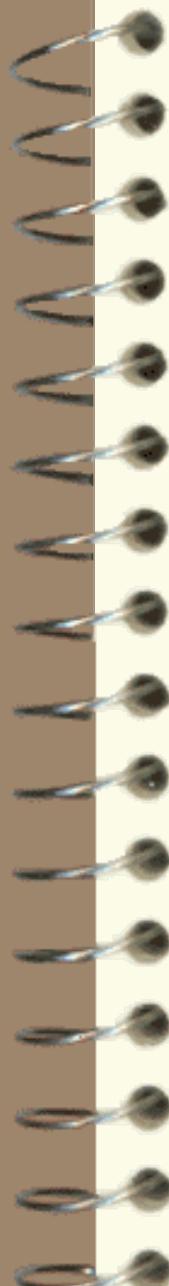
✓ Visão geral do assunto	2
✓ Conceito	3
✓ Conceito mais abrangente	6
✓ Exemplo.....	19
✓ Exercício.....	20



Visão Geral do Assunto

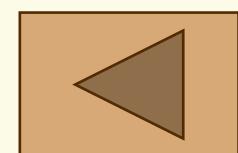
- ✓ Ao longo do texto em questão, será descrito
 - o problema do produtor/consumidor
 - bem como sua solução através do uso de semáforos.
- ✓ Semáforos constituem um mecanismo para
 - sincronização e sinalização entre processos,
 - voltado ao compartilhamento de recursos e dependência entre processos.

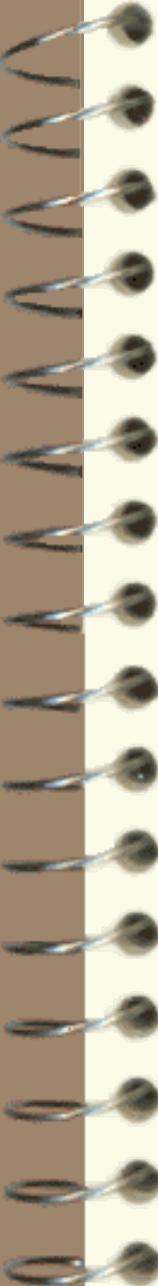




Visão Geral do Assunto

- ✓ A utilização de semáforos garante os dois tipos básicos de sincronização:
 - exclusão mútua
 - sinalização
 - execução de uma operação somente após o término de outra.

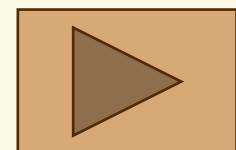


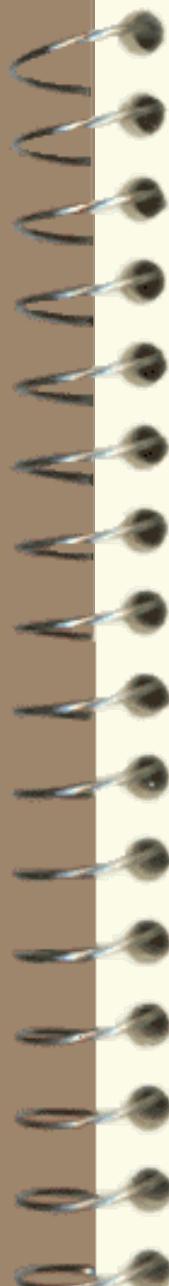


Conceito

- ✓ O problema do produtor/consumidor consiste de
 - um processo que produz dados
 - para serem consumidos por outro processo.

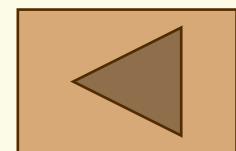
- ✓ Os dados - ou mensagens - são
 - armazenados temporariamente num buffer
 - enquanto esperam para serem utilizados.





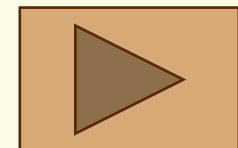
Conceito

- ✓ A solução que será apresentada aqui utiliza semáforos para
 - garantir a exclusão mútua
 - e a seqüência correta na execução das operações.



Conceito mais abrangente

- ✓ Sejam dois processos: um *Consumidor* e outro *Produtor*. O produtor produz dados e os envia ao consumidor; este recebe os dados e os processa.

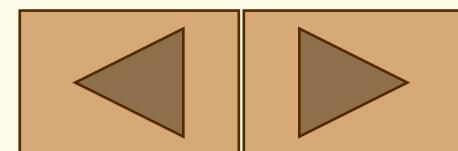




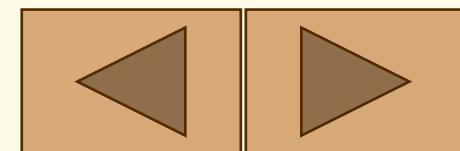
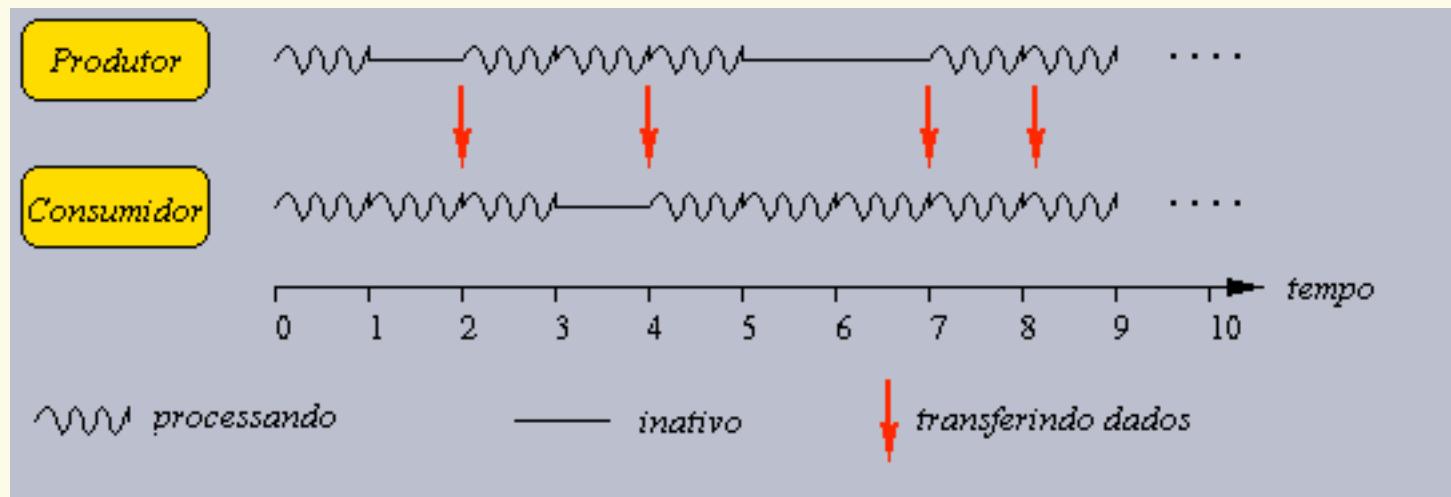
Conceito mais abrangente

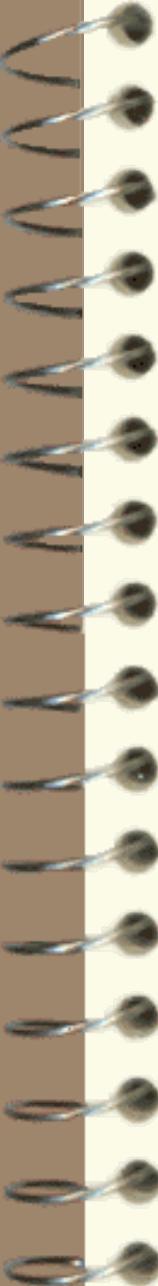
- ✓ O problema é que o produtor
 - só pode enviar cada dado quando o consumidor estiver pronto para recebê-lo;
 - logo, o produtor fica esperando sem fazer nada enquanto o consumidor não estiver apto a receber os dados.

- ✓ Da mesma forma, o consumidor
 - sempre precisa esperar um dado enviado pelo produtor
 - para continuar seu trabalho, também possuindo tempo ocioso.



Conceito mais abrangente

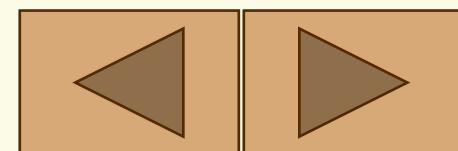




Conceito mais abrangente

✓ O problema é

- como transferir dados de forma que o produtor e o consumidor nunca fiquem inativos sem necessidade.
- Ou melhor: menos tempo ociosos

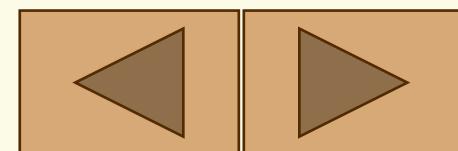


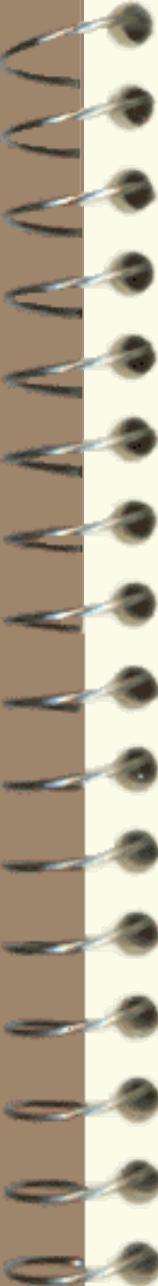


Conceito mais abrangente

✓ O primeiro passo

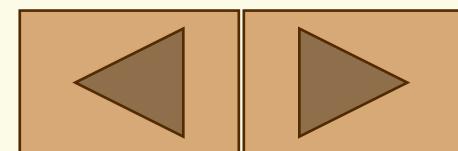
- é eliminar o obstáculo de um computador (ou processador) ter de esperar pelo outro antes de enviar ou receber.
- A solução é introduzir um *buffer*, onde os dados ficam armazenados temporariamente.





Conceito mais abrangente

- ✓ O produtor
 - coloca no buffer os dados que produz
- ✓ e o consumidor
 - retira-os do buffer.
- ✓ Agora, produtor e consumidor podem operar de forma independente um do outro.





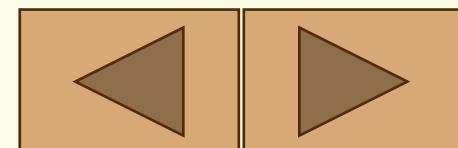
Conceito mais abrangente

- ✓ Primeiro problema relacionado ao buffer
 - Exclusão mútua no acesso ao buffer em escrita (e leitura) pelo produtor e consumidor
- ✓ Solução:
 - Um semáforo com uso do tipo mutex no acesso ao buffer



Conceito mais abrangente

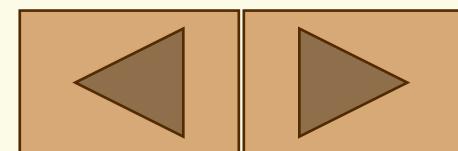
- ✓ Porém, há outro problema:
 - Imagine que o produtor deseja inserir um dado no buffer, mas este está cheio, esperando que o produtor retire dados.
- ✓ Da mesma forma, imagine que o consumidor tenta obter dados do buffer quando este está vazio
 - ou seja, todos os dados enviados pelo produtor já foram processados pelo consumidor.

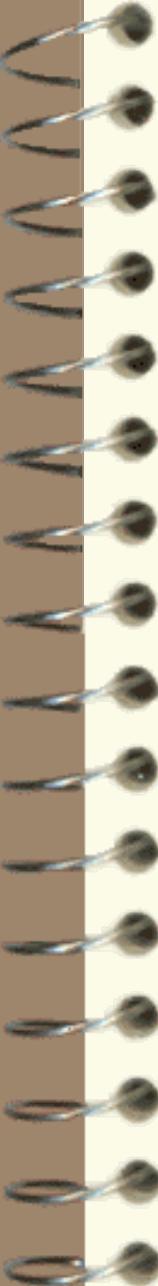




Conceito mais abrangente

- ✓ É necessária uma maneira de impedir os dois problemas:
 - o produtor enviar dados para um buffer cheio.
 - o consumidor receber dados de um buffer vazio.





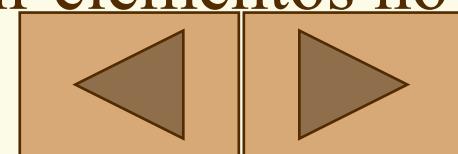
Conceito mais abrangente

✓ A solução é

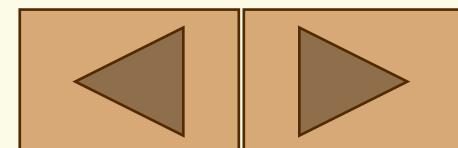
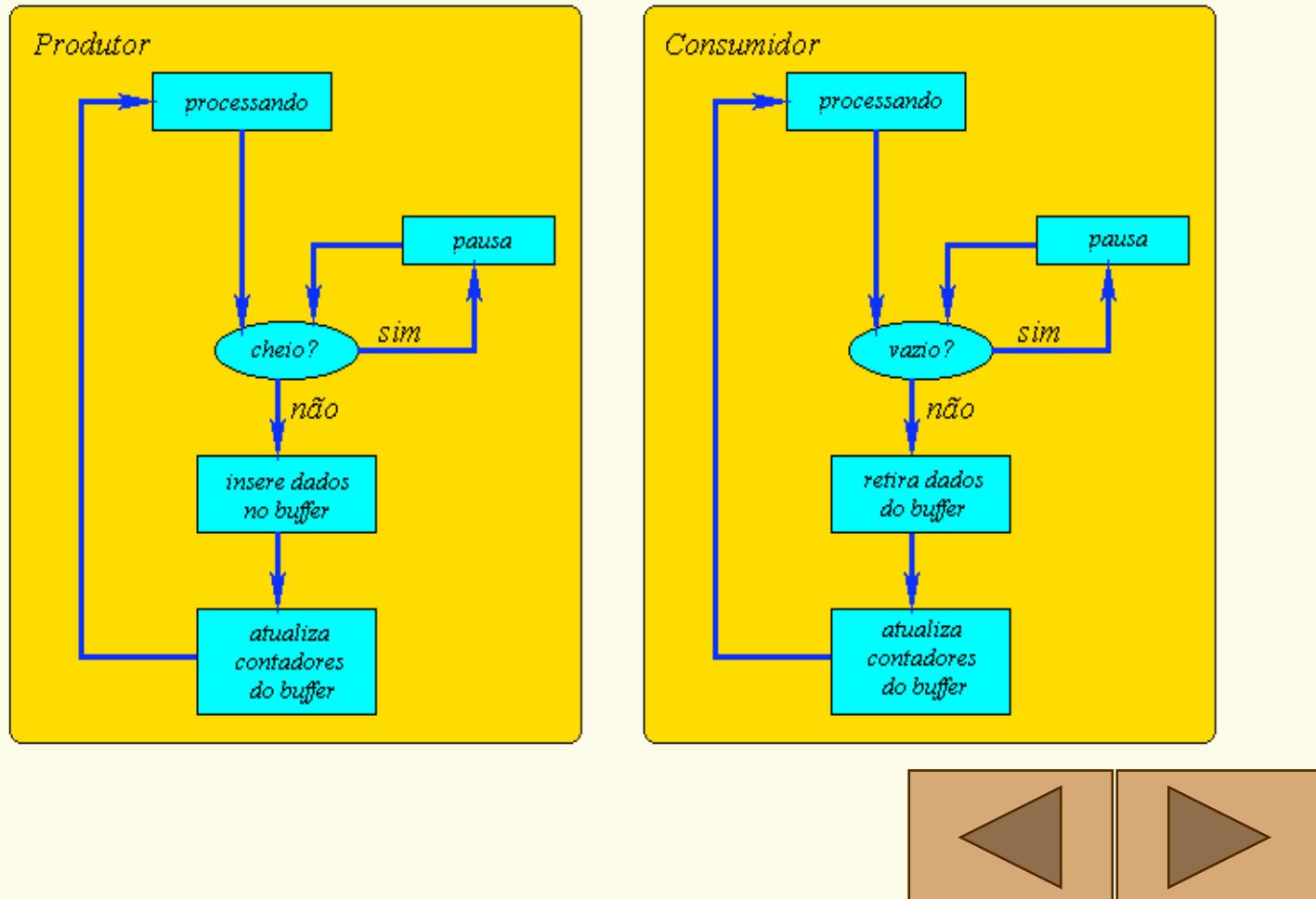
- fazer com que o produtor faça uma pausa quando o buffer estiver cheio,
- dando tempo para que o consumidor retire elementos do buffer.

✓ Similarmente, o consumidor

- deve parar quando o buffer estiver vazio, a fim de que o produtor possa inserir elementos no buffer.



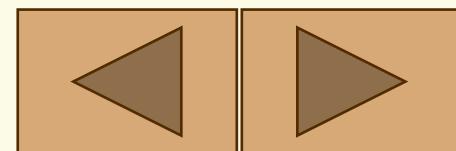
Conceito mais abrangente





Conceito mais abrangente

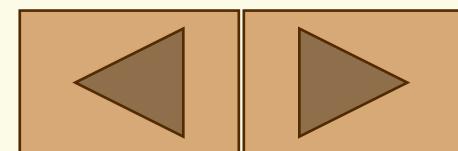
- ✓ Mas como saber quando o buffer está cheio ou vazio?
- ✓ Aqui entra a utilização de variáveis semáforos.

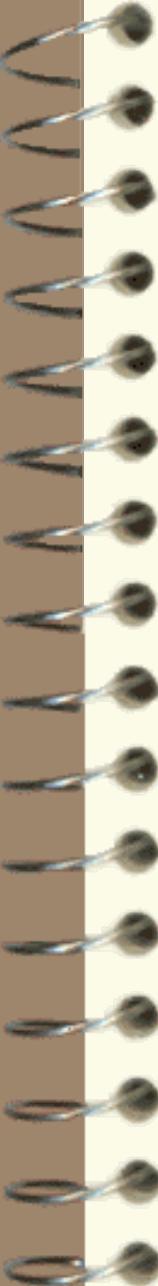




Conceito mais abrangente

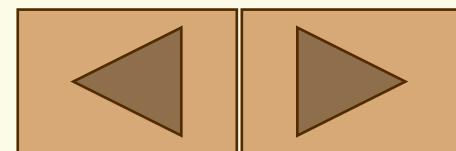
- ✓ São usadas três variáveis semáforas:
 - *full* para contar o número de espaços ocupados no buffer
 - *empty* para contar o número de espaços vazios no buffer
 - *mutex* para garantir que consumidor e produtor não acesssem o buffer ao mesmo tempo, ou seja, garantir a exclusão mútua.

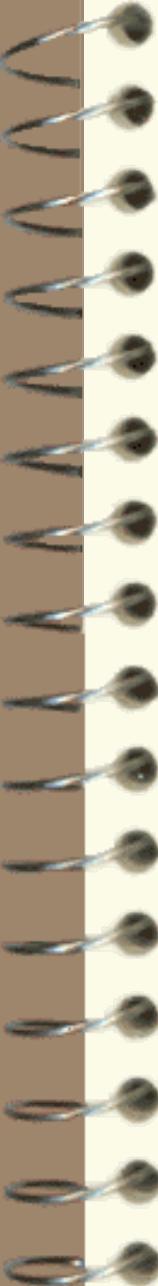




Conceito mais abrangente

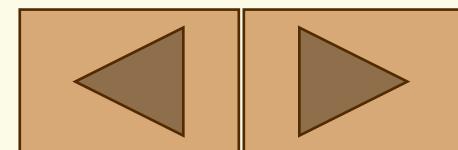
- ✓ Full inicia com o valor 0.
 - Espaços ocupados
- ✓ Empty inicia com o valor n .
 - Onde n é o número de espaços no buffer.
- ✓ Mutex começa com valor 1.

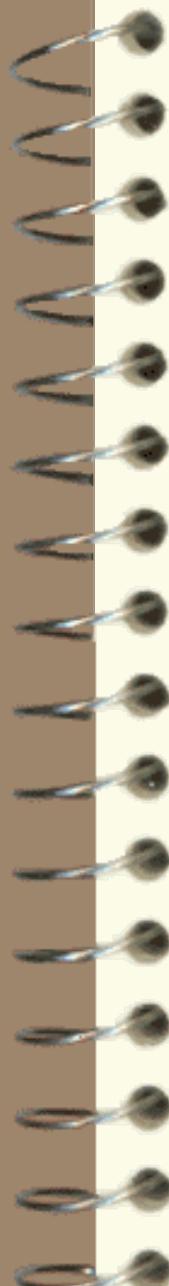




Conceito mais abrangente

- ✓ O semáforo mutex é usado pelos dois processos (produtor e consumidor) com a finalidade de garantir a exclusão mútua.
- ✓ Já as variáveis empty e full são usadas para garantir que certas seqüências de eventos ocorram ou não.
- ✓ Ou seja, garantir que o produtor pare quando o buffer estiver cheio e o consumidor pare quando o buffer estiver vazio.





Conceito mais abrangente

✓ Produtor

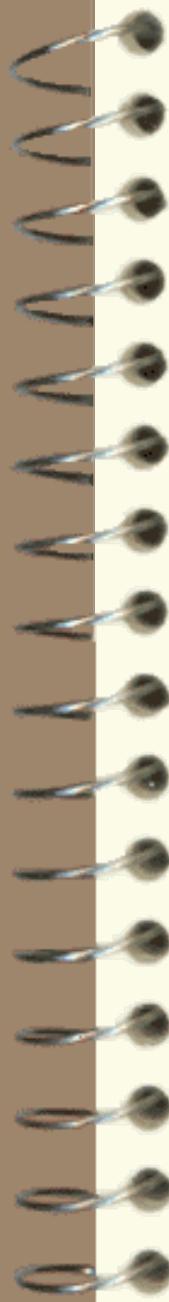
- Quando buffer cheio, espera que o consumidor consuma ao menos 1 e **sinalize**

✓ Consumidor

- Quando buffer vazio, espera que o produtor produza ao menos 1 e **sinalize**

✓ Questões de implementação

- Quem incrementa o full (contador de ocupados)?



Conceito mais abrangente

✓ Questões de implementação

- Quem incrementa o *full* (contador de ocupados)?
- Quem decrementa o *empty* (contador de livres)?
- Analogamente em qual semáforo o produtor bloqueia?
- E em qual o consumidor bloqueia?
- Lembrando: semáforo bloqueia quando semáforo igual a zero (ou menos)
 - P: bloqueia; V: incrementa



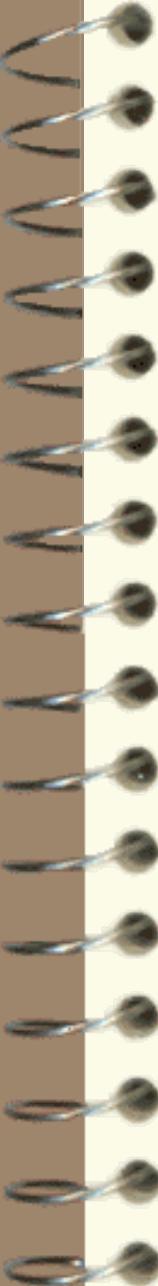
Conceito mais abrangente

✓ Produtor

- No estado inicial não bloqueia
 - Buffer vazio
- Bloqueia quando contador de espaços livres (empty) = zero

✓ Consumidor

- No estado inicial bloqueia
 - Buffer vazio
- Bloqueia quando contador de ocupados (full) = zero



Exemplo

```
/*número de espaços no buffer*/
#define N 100

/*semáforos são um tipo especial de inteiro*/
typedef int semaphore;

/*controla o acesso à região crítica*/
semaphore mutex = 1;

/*contador do número de espaços vazios no
   buffer*/
semaphore empty = N;

/*contador do número de espaços cheios no
   buffer*/
semaphore full = 0;
```



Exemplo

```
void producer (void)
{
    int item;

    while (TRUE) {
        produce_item (&item); /*gera algo para colocar no buffer*/

        P(&empty);          /*decrementa o contador empty; bloqueia se cheio*/
        P(&mutex);          /*entra na região crítica*/
        enter_item (item);   /*coloca um novo item no buffer*/
        V (&mutex);         /*sai da região crítica*/
        V (&full);          /*incrementa o contador full */
    }
}
```



Exemplo

```
void consumer (void)
{
    int item;

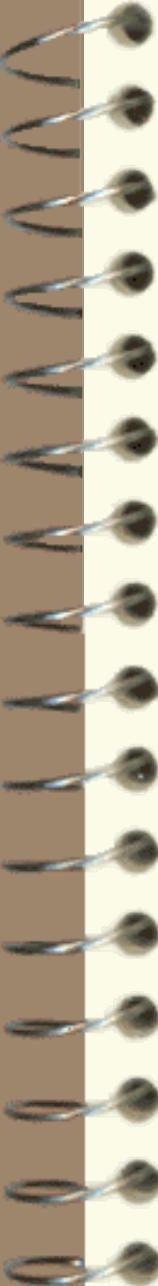
    while (TRUE) {
        P (&full);          /* bloqueia se vazio - zero*/
        /* decrementa o contador full */

        P (&mutex);         /*entra na região crítica*/
        remove_item (&item); /*retira um item do buffer*/

        V (&mutex);         /*sai da região crítica*/
        V (&empty);          /*incrementa o contador empty*/
        consume_item (&item); /*processa o item retirado do
                                buffer*/
    }
}
```

Exemplo

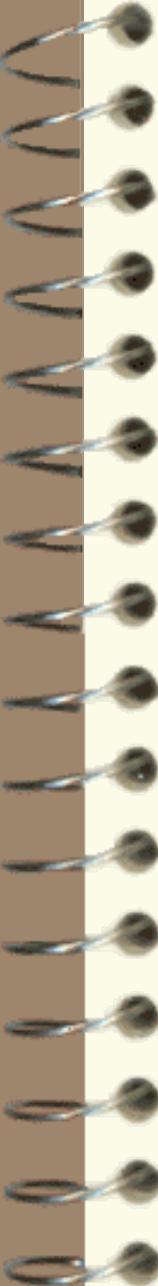
```
#define N 100                                     /*número de espaços no buffer*/  
  
typedef int semaphore;                          /*semáforos são um tipo especial de inteiro*/  
  
semaphore mutex = 1;                           /*controla o acesso à região crítica*/  
semaphore empty = N;                           /*contador do número de espaços vazios no buffer*/  
semaphore full = 0;                            /*contador do número de espaços cheios no buffer*/  
  
void producer (void){  
    int item;  
  
    while (TRUE) {  
        produce_item (&item);  
        P(&empty);  
        P(&mutex);  
        enter_item (item);  
        V (&mutex);  
        V (&full);  
    }  
}  
  
void consumer (void){  
    int item;  
  
    while (TRUE) {  
        P (&full);  
        P (&mutex);  
        remove_item (&item);  
        V (&mutex);  
        V (&empty);  
        consume_item (&item);  
    }  
}
```



Conceitos Adicionais

✓ [Andrews, 1991]

- Derivação da solução através de propriedades
- Propriedades
 - Relação entre contadores de:
 - Início de append (deposit, enter, ...) pelo produtor
 - Final de append ...
 - Início de retirada (get, ...) pelo consumidor
 - Final de retirada ...



Conceitos Adicionais

✓ [Andrews, 1991]

– 1^a solução

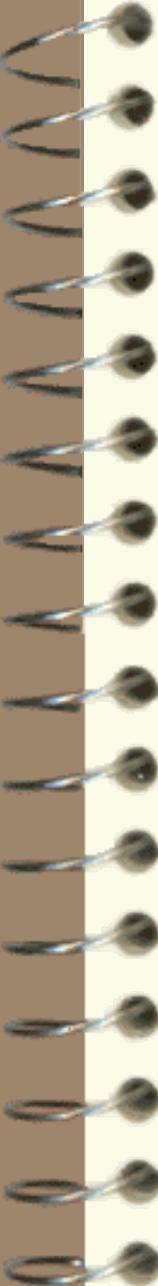
- Buffer com somente 1 posição
- Semáforos split: entre 0 e 1
- Alternância obrigatória entre produtor e consumidor
- Similar a “coorotinas”:
 - » rotinas que cooperam entre si para resolver o mesmo problema
 - » Modelo de PC onde há alternância em pontos bem definidos



Conceitos Adicionais

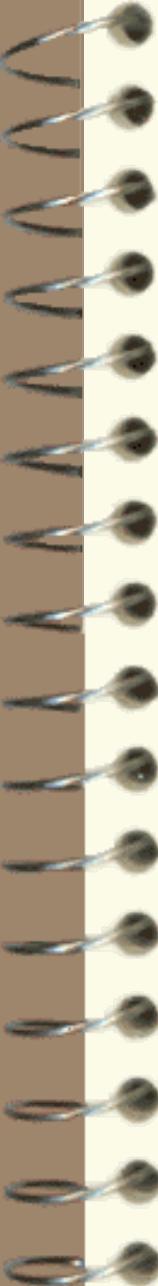
✓ [Andrews, 1991]

- 2^a solução
 - Buffer com tamanho n



Resumo

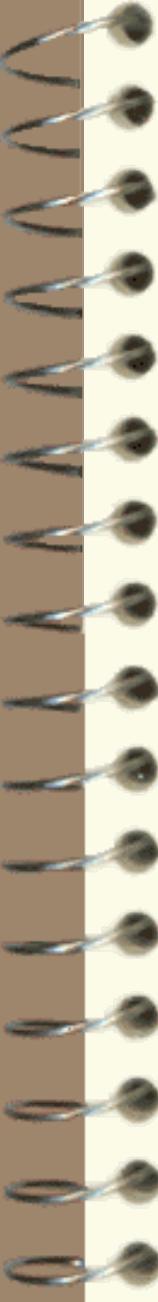
- ✓ Conceito de produtor/consumidor
- ✓ Problema derivado das velocidades distintas
- ✓ Buffer: solução para problema acima
- ✓ 3 novos problemas
 - Acesso exclusivo ao buffer
 - Buffer cheio
 - Buffer vazio
- ✓ Solução com semáforos



Resumo

✓ Solução com semáforos

- Semáforo mutex
- 1 semáforo de sinalização (ou recursos)
 - Buffer cheio: inicialmente = n
 - Isto é: há n posições livres (recursos)
- 1 semáforo de sinalização
 - Buffer vazio: inicialmente = zero
 - Isto é: há zero espaços ocupados



Exercício

- ✓ Abaixo é apresentada a solução para o problema do produtor/consumidor. Considere um buffer de tamanho N.

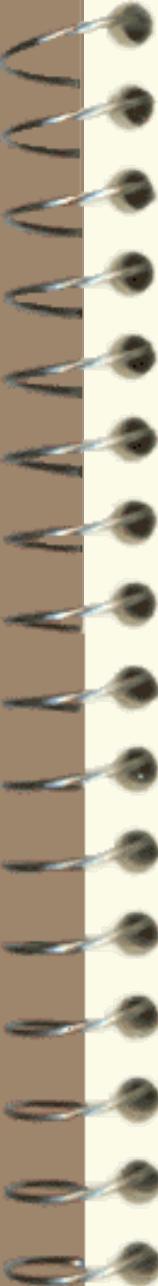
Producer:

```
Begin
  ...
  P (Empty);
  P (Mutex);
  <add item to
    buffer>
  V (Mutex);
  V (Full);
  ...
End.
```

Consumer:

```
Begin
  ...
  P (Full);
  P (Mutex);
  <remove item
    from buffer>
  V (Mutex);
  V (Empty);
  ...
End.
```

- ✓ Quais são os valores iniciais dos semáforos *Empty*, *Full* e *Mutex* listados acima?



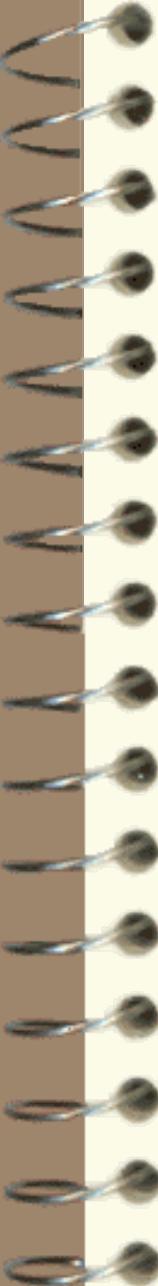
Exercícios

✓ Exercício A

- Verifique se é possível mudar a posição das chamadas à primitiva V para os semáforos Cheio ou Vazio
 - A) antes de V(Mutex); B) antes do tratamento do buffer; C) antes de P(Mutex)

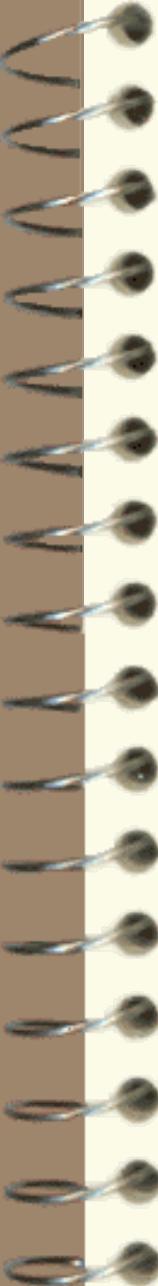
✓ Exercício B

- Considerando vários produtores e consumidores, e que os produtores são mais lentos que os consumidores, é possível ocorrer que um ou mais consumidores morram de fome?
 - explique



Revisão

- ✓ Especificação do produtor?
- ✓ Idem do consumidor?
- ✓ Como aumentar a concorrência entre os dois?
- ✓ Explique o problema de exclusão mútua
- ✓ Qual a situação de exceção para o produtor?
- ✓ E para o consumidor?



Revisão

- ✓ Como resolver o problema do acesso concorrente ao buffer com semáforos?
 - Quais semáforos, valores iniciais, uso das primitivas, ...
- ✓ Como resolver a exceção do produtor?
 - Idem ...
- ✓ Como resolver a exceção do consumidor?
 - Idem ...