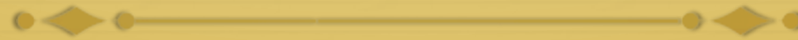


Problema Produtor-Consumidor



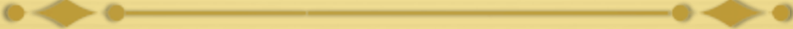
Problemas Clássicos

Caracterização



- ✦ Itens (ou objetos) são produzidos e consumidos por threads independentes, em paralelo.
 - ✦ **Produtor:** thread que produz itens
 - ✦ **Consumidor:** thread que consome itens
- ✦ Os itens são armazenados em um espaço (estrutura de dados) compartilhado entre todas as threads.
 - ✦ Requer acesso (inserir ou remover item) com exclusão mútua
 - ✦ Espaço pode ser *infinito* ou *finito*
 - ✦ Consumidor é bloqueado quando não houver um item disponível para consumo
 - ✦ Produtor é bloqueado quando não houver espaço livre para inserir o novo item produzido, no caso de espaço finito

Exemplo: Sistema baseado em eventos (IoT)



- ✧ *event-driven system*
- ✧ Evento é algo que ocorre no sistema. Exemplos: usuário pressiona uma tecla ou move o mouse, uma mensagem chega pela rede, uma operação de escrita em disco é concluída.
- ✧ Composto por threads que **produzem** eventos e por threads que **consomem** eventos.
 - ✧ Produtor: gera um objeto que representa o evento ocorrido e o insere no buffer de eventos.
 - ✧ Consumidor: remove um objeto do buffer de eventos e processa (trata) o correspondente evento.

Código básico

Produtor:

```
evento := esperarEvento( )  
buffer.inserir( evento )
```

Consumidor:

```
evento := buffer.remove( )  
evento.processar( )
```

Sincronização



- ✧ Acesso exclusivo ao buffer:

```
mutex := Semáforo( 1 )
```

- ✧ Número de itens no buffer:

```
itens := Semáforo( 0 )
```

- ✧ Capacidade do buffer (no caso de ser finito):

```
espaços := Semáforo( tamanho do buffer )
```


Solução com buffer infinito

Consumidor:

```
itens.esperar( )  
mutex.esperar( )  
    evento := buffer.remove( )  
mutex.sinalizar( )  
  
evento.processar( )
```

Produtor:

```
evento := esperarEvento( )  
  
mutex.esperar( )  
    buffer.inserir( evento )  
mutex.sinalizar( )  
itens.sinalizar( )
```

Solução com buffer finito

Consumidor:

```
itens.esperar( )  
mutex.esperar( )  
    evento := buffer.remove( )  
mutex.sinalizar( )  
espaços.sinalizar( )  
evento.processar( )
```

Produtor:

```
evento := esperarEvento( )  
espaços.esperar( )  
mutex.esperar( )  
    buffer.inserir( evento )  
mutex.sinalizar( )  
itens.sinalizar( )
```

Exemplo:

fila de tamanho limitado

- ✦ A fila pode ter, no máximo, 100 objetos
- ✦ Operações:
 - a. `inserir(objeto)`: insere um objeto na fila; caso a fila esteja cheia, deve aguardar até que algum objeto seja removido.
 - b. `remover ()`: remove um objeto da fila; caso a fila esteja vazia, deve aguardar até que algum objeto seja inserido.

INICIAÇÃO:

```
fila = { } // vazia  
itens = Semaforo( 0 )  
espaco = Semaforo( 100 )  
mutex = Semaforo( 1 )
```

Thread 1: PRODUTOR

inserir(objeto):

```
    espaco.esperar()  
  
    mutex.esperar()  
        fila.insere( objeto )  
    mutex.sinalizar()  
  
    itens.sinalizar()
```

Thread 2: CONSUMIDOR

remover(objeto):

```
    itens.esperar()  
  
    mutex.esperar()  
        fila.remove( objeto )  
    mutex.sinalizar()  
  
    espaco.sinalizar()
```

(THREADS PARALELAS)