

## Sincronização

- Processos Concorrentes:
  - acessam informações comuns
  - dados compartilhados:
    - na memória principal
    - em um arquivo de dados
    - etc...
  - não é possível determinar a ordem em que os eventos vão ocorrer.
- Alguma forma de *sincronização* entre processos é necessária.

Diagrama de sincronização:

```

    P1 --(a:=1;)--> [memória compartilhada] <--(a:=11; a++;)-- P2
  
```

## Sincronização entre Processos

- Garantir a sequência adequada para a execução de eventos.
  - um processo espera para adquirir o recurso compartilhado (WAIT)
  - o processo quando termina de usar o recurso compartilhado emite um sinal (SIGNAL)

## Exemplo: a Tarefa de Impressão de Arquivos

Diagrama de fluxo de impressão:

```

    Processo A --(chamada do sistema)--> Sist. Operacional --(ARQ1)--> Lista de Impressão --(ARQ1)--> Impressora
    
```

Detalhes do sistema de impressão:

- Processo A:** Envia uma chamada do sistema para o Sistema Operacional.
- Sist. Operacional:** Gerencia a fila de impressão.
- Lista de Impressão:** Fila de arquivos a serem impressos (ARQ1).
- Impressora:** Recebe arquivos da fila para impressão.
- Daemon do Servidor de Impressão:** Gerencia a comunicação com a impressora.
- Variáveis compartilhadas:**
  - livre = 1
  - proximo = 1

## Exemplo (cont.): Necessidade da Sincronização

Dois Processos acessando dados compartilhados:

Diagrama de acesso compartilhado:

```

    Processo A --(livre = 7)--> [dados compartilhados] --(proximo = 4)--> Processo B
  
```

Lista de Impressão:

1	
2	
3	
4	abc
5	prog. c
6	prog. n
7	

## Condições de Corrida

- Condições de Corrida:
  - situação onde dois ou mais processos estão lendo ou escrevendo algum dado compartilhado

O diagrama mostra um processo executando várias instruções. Um trecho de código é destacado com uma bracejada e rotulado 'Trecho sem Condições de Corrida', contendo as seguintes instruções: 'Leia X', 'Enquanto x<10', 'x = x + 1' e 'Imprima X'. Outro trecho, também rotulado com uma bracejada, é rotulado 'Trecho com Condições de Corrida (Região Crítica)' e contém as instruções: 'Leia Livre', 'Prox\_Posic = Livr' e duas instruções vazias. O nome 'Processo' está escrito na base do diagrama.

## Exclusão Mútua

- Exclusão Mútua
  - Maneira de se evitar que dois processos entrem nas suas regiões críticas correspondentes ao mesmo tempo
- Exemplo:
  - Sistemas Operacionais: Acesso a Recursos
- Propostas para atingir a Exclusão Mútua:
  - desabilitação de interrupções
  - espera ocupada
  - Semáforos
  - Monitores

## Interrupção

- O Controlador do Dispositivo interrompe a CPU quando um dispositivo termina seu trabalho.
- Quando ocorre a interrupção a CPU executa uma rotina de tratamento de interrupção

O diagrama ilustra o fluxo de uma interrupção. Uma seta horizontal entra na CPU no ponto 't1', com o texto 'Processo pede que o controlador escreva um caractere'. Dentro da CPU, há um retângulo rotulado 'A CPU passa este tempo sendo usada por outro processo'. Uma seta horizontal sai da CPU no ponto 't2', com o texto 'O controlador interrompe a CPU, indicando que o caractere foi escrito'. Acima da CPU, há um retângulo rotulado 'O controlador escreve o caractere'.

## Desabilitação de Interrupções

- Interrupções podem ser desabilitadas durante a execução da seção crítica.
- Não ocorrendo interrupções, o processo corrente não será substituído por outro.
- Solução usada a nível de sistema operacional.
- Não é adequada para o nível usuário!
  - Interrupções de dispositivos não podem ser ignoradas!
  - Processos que não estão envolvidos na seção crítica também são penalizados!

## Espera Ocupada

- Uma variável pode ser usada para criar alternância estrita.
  - Um dos processos pode ser muito mais rápido que o outro!
  - Solução pouco geral!
  - Espera ocupada - desperdício de recursos!

```

PROCESSO 1
while (TRUE)
{
  while (vez != 0); /* verifica repetidamente a condição
  secao_critica();
  vez = 1;
  secao_nao_critica();
}

PROCESSO 2
while (TRUE)
{
  while (vez != 1); /* verifica repetidamente a condição
  secao_critica();
  vez = 0;
  secao_nao_critica();
}
  
```

## Espera com Bloqueio

- É interessante que um processo que está a espera de um recurso não desperdice tempo de CPU.
- Criação de mecanismos de *bloqueio*.
  - Processo “dorme” até que o recurso que ele requer se torne disponível.

## Idéia de Proposta para Exclusão Mútua

- Um Processo Coordenador (C)
- Processo X quer entrar na região crítica -> Solicitação ao Coordenador
- Processo X sai da região crítica -> Comunicação da Liberação ao Coordenador

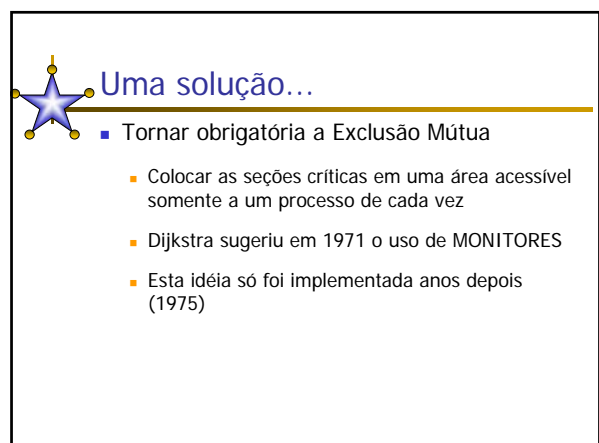
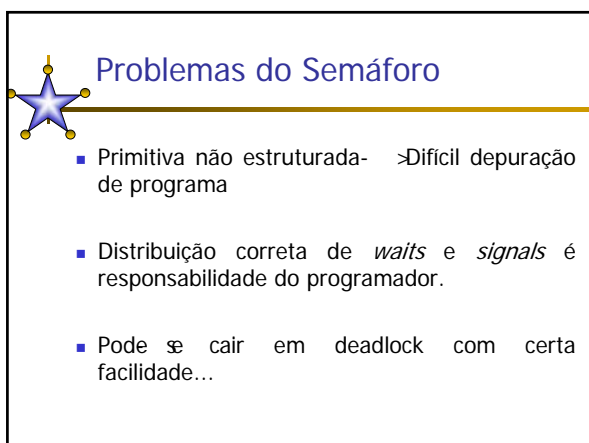
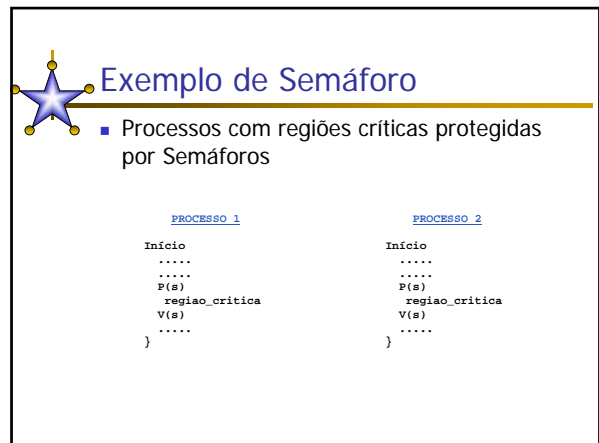
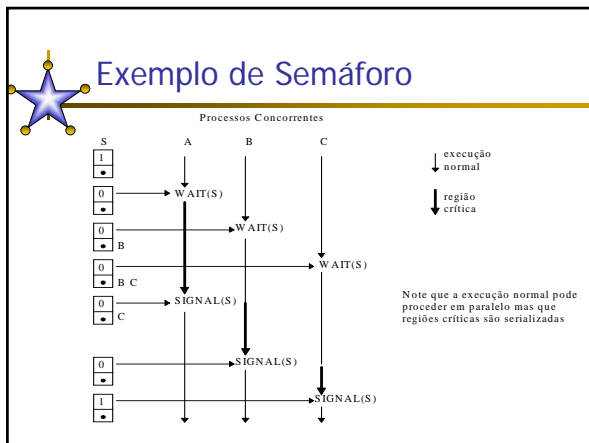
(a) Solicitação OK

(b) Solicitação Espera

(c) Liberação OK

## Semáforos

- Sincronização entre os processos que usam o semáforo
- S -> Variável inteira que contém duas operações associadas
  - WAIT (S): Se  $S > 0$  então  $S := S - 1$  e o processo continua a execução  
Senão a execução do processo corrente é bloqueada e o processo é colocado na fila de S
  - SIGNAL(S): Se há processo bloqueados na fila de S, o primeiro deles deve ser desbloqueado  
Senão  $S := S + 1$
- Wait(s) ou P(s)      Signal(s) ou V(s)
- Operações são *atômicas*!
- Facilidade oferecida por diversos sistemas operacionais!



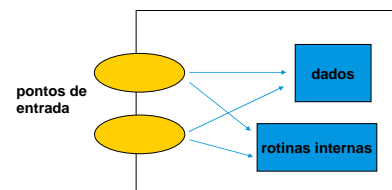


## Monitores

- Primitiva estruturada- idéia surgida com conceito de TAD.
- Comunicação entre processos passa a ser concentrada em pontos específicos do programa.
- Coleção de procedimentos e rotinas de acesso agrupados em um módulo
- Apenas um processo pode estar ativo no monitor por vez.



## Monitores



Dois procedimentos dentro de um mesmo monitor não podem estar ativos simultaneamente!



## Monitores

- Ao invés de codificar a seção crítica dentro de um processo, deve-se codificá-la como um procedimento de um monitor
  - Evita duplicação de código
  - Sincroniza processos que querem usar a mesma seção crítica
- Quando um processo quer usar um recurso compartilhado, ele chama um procedimento do monitor (o monitor garante a exclusão mútua)



## Monitores

**Monitor** *nome-do-monitor*  
*Variáveis globais*

Procedure p1  
begin  
.....  
end

Procedure p2  
begin  
.....  
end

## Exemplo Monitor


- Processos usando os procedimentos **p1** e **p2** do Monitor

<u>PROCESSO 1</u>	<u>PROCESSO 2</u>
Início	Início
.....	.....
P1	P2
.....	.....
}	}

O monitor será o responsável por implementar a Exclusão Mútua entre P1 e P2!!!


## Problemas relacionados a exclusão mútua

- Deadlock**: um conjunto de processos fica bloqueado, cada um a espera de um recurso que o outro detém.
- Starvation**: alguns processos são repetidamente preteridos, enquanto outros ficam com o acesso ao recurso desejado.
- Exemplo: Considere uma estrada com uma ponte por onde só passa um carro de cada vez.

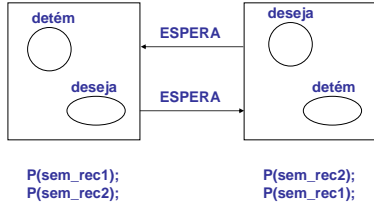


## Deadlock

- Um carro começa a atravessar de cada um dos lados, e os dois se encontram no meio da ponte.
- Nenhum dos dois tem como continuar...



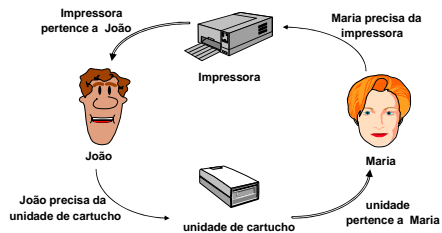
## Deadlocks com semáforos



```

graph LR
    subgraph P1 [P(sem_rec1); P(sem_rec2)]
        D1((detém))
        W1((deseja))
    end
    subgraph P2 [P(sem_rec2); P(sem_rec1)]
        D2((detém))
        W2((deseja))
    end
    W1 -- ESPERA --> D2
    W2 -- ESPERA --> D1
  
```

## Deadlock em um sistema de Computação

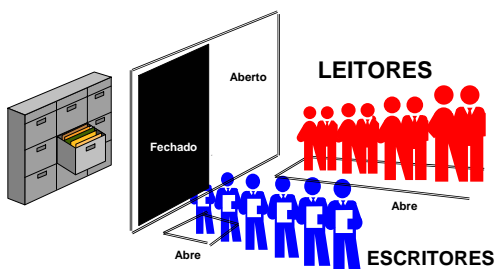


## Starvation

- Se, ao chegar à ponte, há um carro já atravessando na direção desejada, o carro que chegou atravessa também.
- Se há alguém vindo em direção oposta, o carro que chegou espera até que este tenha acabado de cruzar.
- Pode nunca chegar a sua vez...

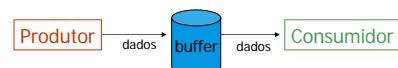


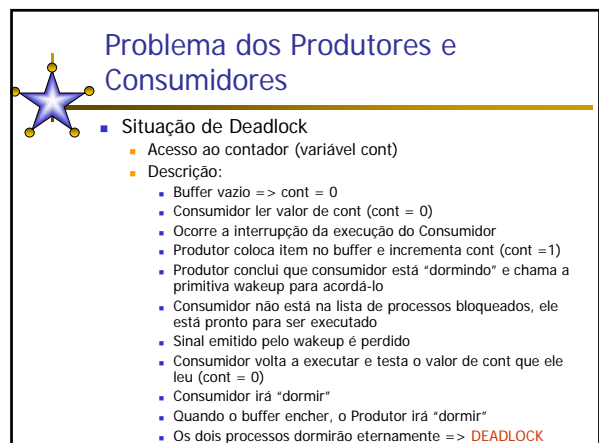
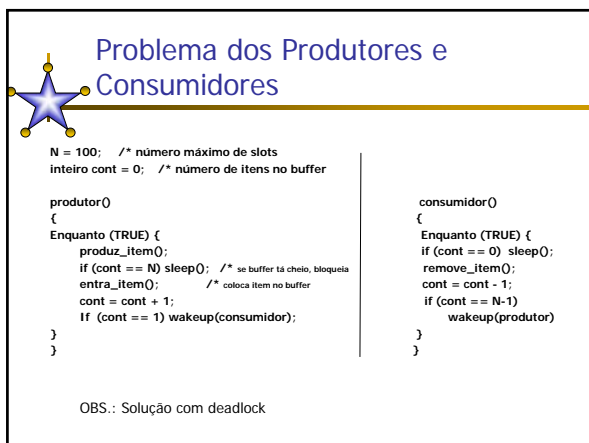
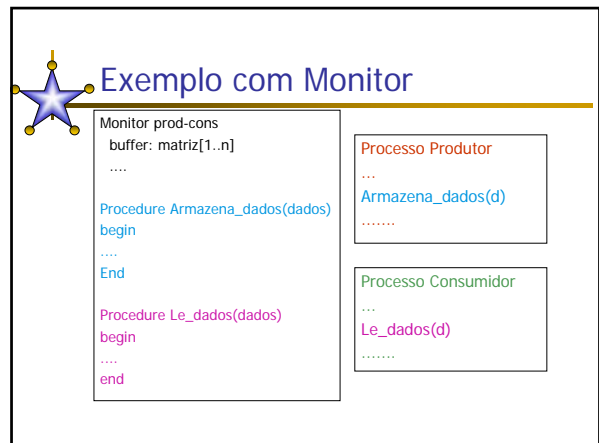
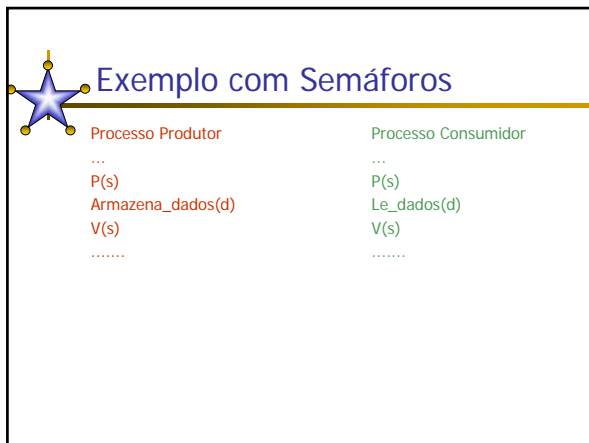
## Starvation em um sistema de Computação



## Problema dos Produtores e Consumidores ou do Buffer Limitado

- Dois processos concorrentes compartilham um buffer de tamanho fixo:
  - Produtor: coloca dados em um buffer
  - Consumidor: ler dados de um buffer
- Ordem dos eventos:
  - Produtor coloca os dados
  - Consumidor ler os dados









## Problema dos Leitores e Escritores

semáforo scl = 1; /\* semáforo para acesso exclusivo a variável compartilhada cont\_leitores  
semáforo sbd = 1; /\* semáforo para acesso exclusivo ao banco de dados  
inteiro cont\_leitores;

```
leitor()
{
    Enquanto (TRUE) {
        wait(scl); /* acesso exclusivo a variável cont_leitores
        cont_leitores = cont_leitores + 1;

        Se (cont_leitores == 1) wait(sbd); /* quando é o primeiro leitor
        signal(scl);
        ler_banco_de_dados();
        wait(scl);
        cont_leitores = cont_leitores - 1;
        Se (cont_leitores == 0) signal(sbd);
        signal(scl);
    }
}
```

```
escritor()
{
    Enquanto (TRUE) {
        wait(sbd);
        escreve();
        signal(sbd);
    }
}
```

OBS.: Solução com prioridade para os leitores, ocasionando Starvation para escritores.