Sistemas Operacionais

4º período

Professora: Michelle Hanne

PUC Minas

Comunicação entre Processos

Comunicação entre processos

PUC Minas

- Ocorre quando processos precisam compartilhar um recurso (informação?).
- O que pode ocorrer quando dois ou mais processos tentam acessar ou utilizar um mesmo recurso ao mesmo tempo?
- O sistema operacional precisa sincronizar os processos que estão sendo executados na unidade central de processamento com a finalidade de garantir o processamento total e a alocação de todos os recursos necessários a tarefa em execução.
- *Comunicação entre processos - Inter-Process Communication* (IPC).
- São recursos:
  - ✓ Arquivos.
  - ✓ Memória.
  - ✓ Registros.
  - ✓ Dispositivos de E/S.
  - ✓ Softwares (programas).

Comunicação entre processos

PUC Minas

Todos os sistemas que operam com diversos processos ao mesmo tempo na concorrência por um recurso como arquivos, memória, registros e dispositivos precisam que seus processos operem em sincronia, ou seja, **esses processos concorrentes ao compartilharem recursos podem gerar falhas que podem até comprometer o sistema operacional** (STUART, 2011)

Editores de texto e jogos são exemplos de sistemas de intensa interatividade. Eles possuem tarefas associadas as suas interfaces e **reagem a comandos do usuário, enquanto outras tarefas como renderização de imagens,** correção ortográfica, envio de dados pela rede, etc

Comunicação entre processos

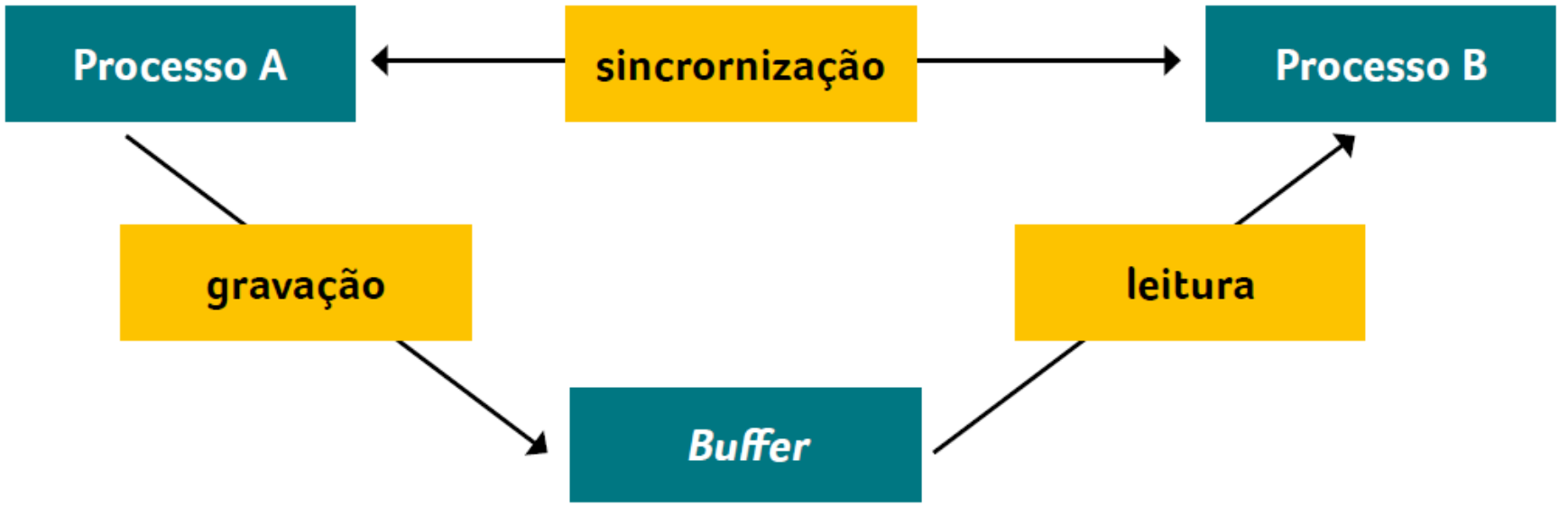
PUC Minas

Em sistemas **multitarefas** é fundamental a utilização desses mecanismos de sincronização para garantir integridade e a confiabilidade das operações realizadas pelo sistema operacional:

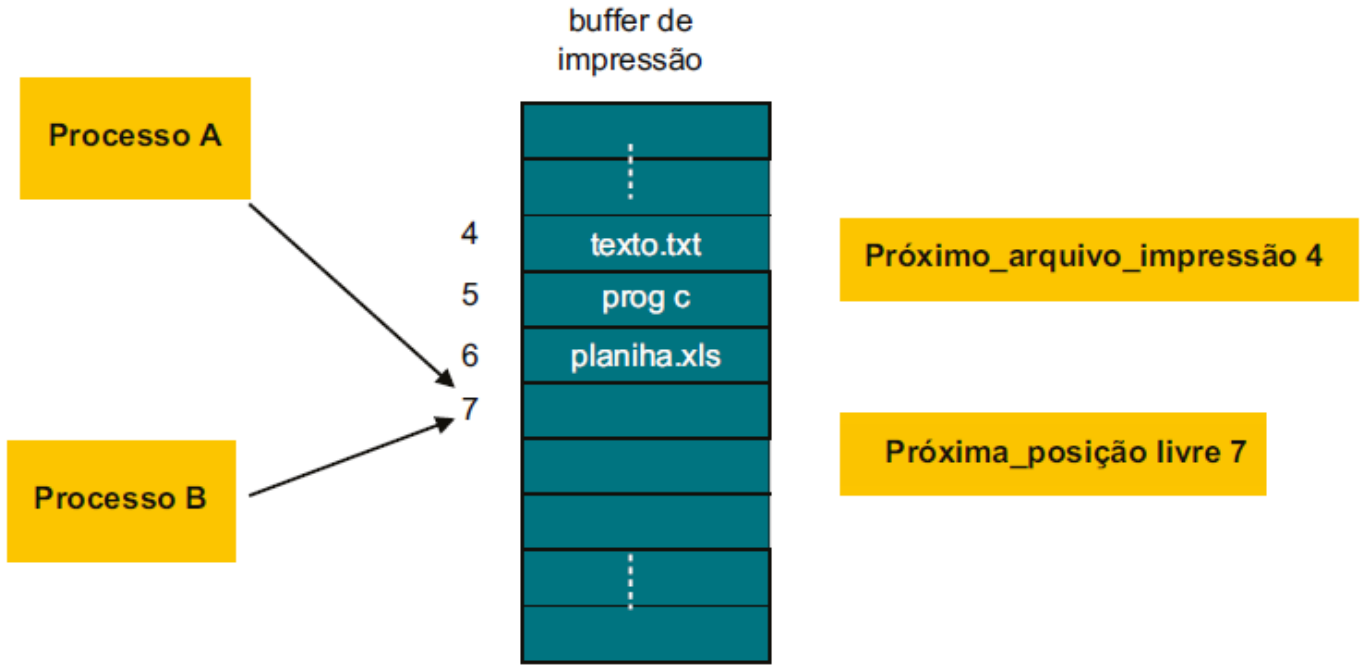
- 1ª – Como o processo envia informações a outro processo?
- 2ª - Como o sistema operacional evita que dois processos não acessem as suas regiões críticas ao mesmo tempo?
- 3ª – Qual a sequência adequada quando há dependências presentes, ou seja, um processo A produz dado que o processo B tem que imprimir?

PUC Minas

Comunicação/sincronização entre processos



Condição de Corrida

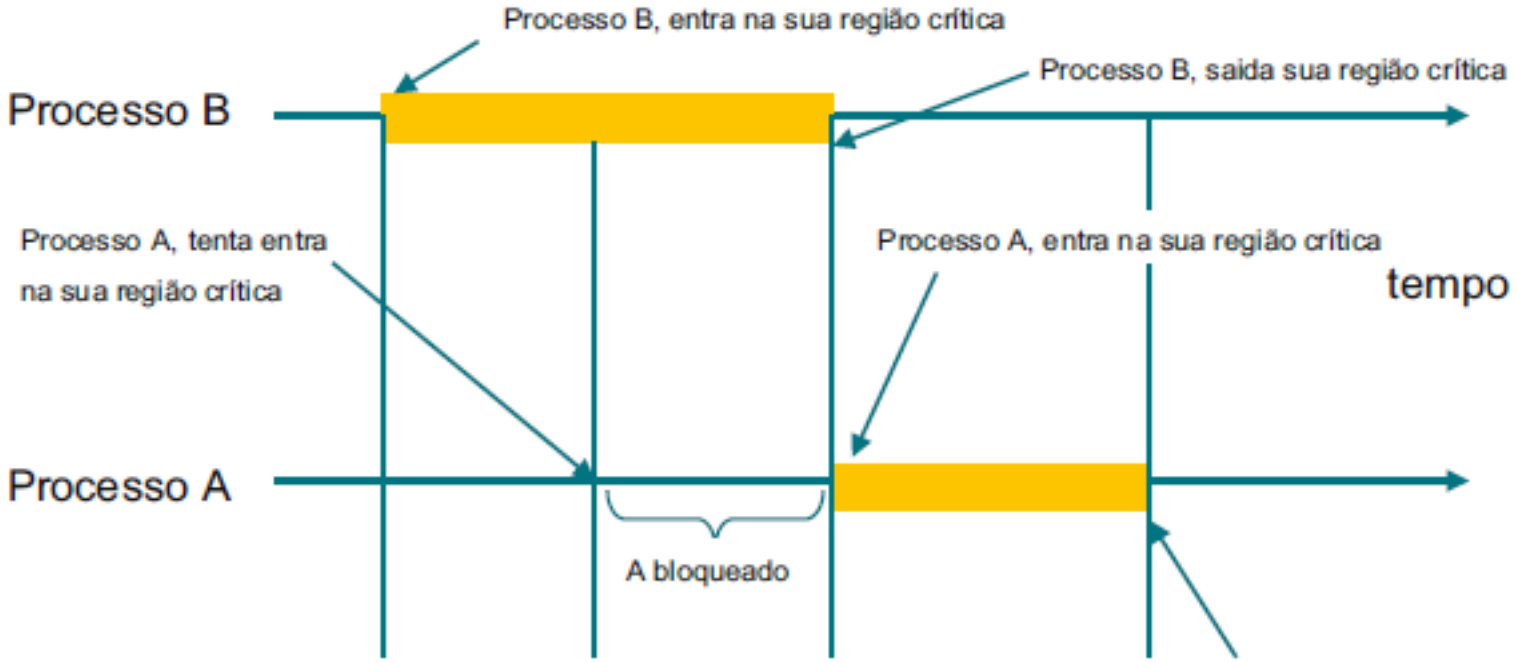


Suponha que a figura Condição de corrida seja um *buffer* de impressão. Nesse *buffer*, constam arquivos na fila de impressão na posições 4, 5, 6, e a posição 7 está vazia. Na situação ilustrada, o arquivo a ser impresso será o texto.txt, na posição 4, mas, nesse momento, dois novos **processos (A e B) decidem enviar novos arquivos para a fila de impressão ao mesmo tempo.** Nesse momento, ambos os processos (A e B) disputam para que seus arquivos sejam alocados na posição 7 do *buffer* e, consequentemente, entrem na fila de impressão.

Regiões críticas

PUC Minas

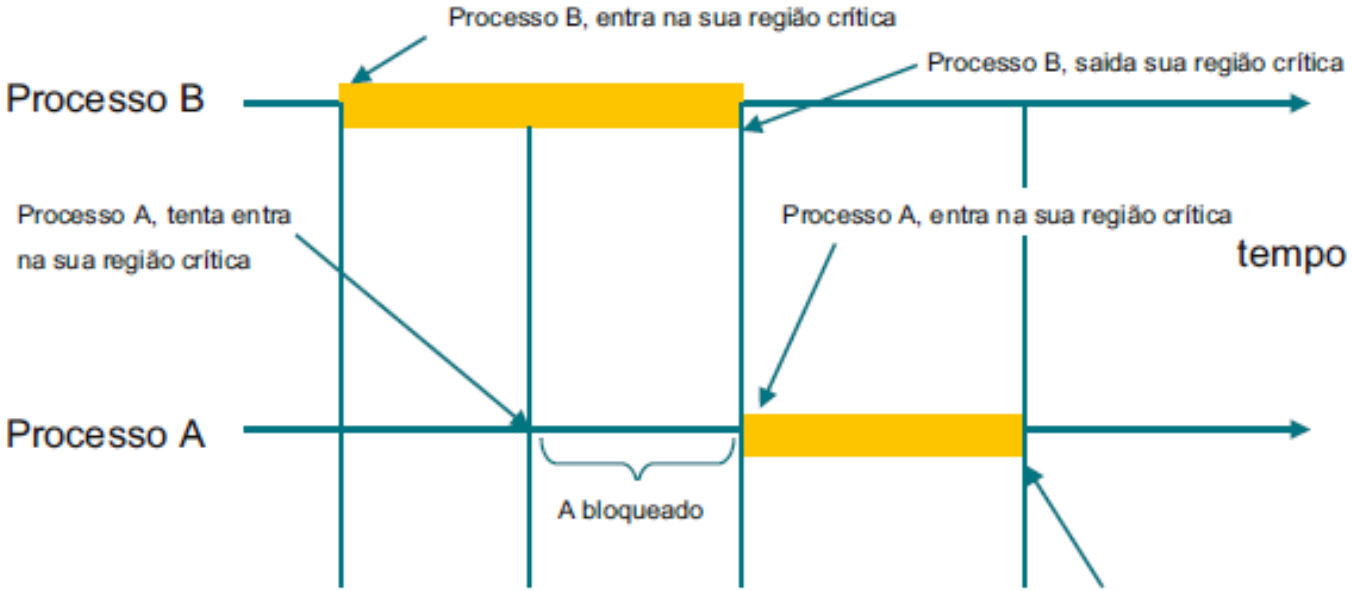
Região ou seção crítica é a parte de um programa que acessa um recurso compartilhado que não pode ser acessado por outro processo concorrentemente (TANENBAUM, 2016).



- Imagine que um determinado processo **esteja sendo executado na CPU com o objetivo de atualização de uma certa estrutura de dados,** e num determinado **instante, outro processo recebe permissões para ser executado antes que o primeiro processo esteja com a tarefa concluída.**
- **Essa condição pode gerar inconsistência de resultados** (TANENBAUM, 2016).

Fonte: Lacerda, 2019 – p 54

Regiões críticas



Na figura, **parte do código do processo B acessa o recurso compartilhado e inicia sua região crítica;** porém, num determinado momento, o processo **A precisa também alocar esse recurso.** O processo A, nesse momento, fica bloqueado até o término da região crítica de B, ou seja, não conseguirá alocar o recurso e ficará aguardando (bloqueado). Esse procedimento, exclusão mútua, garante a B permissão para terminar sua tarefa.

Ao término da região crítica de B, parte do código de A entra na sua região crítica para garantir a exclusão mútuae, consequentemente, executar a tarefa.

Exclusão mútua

PUC Minas

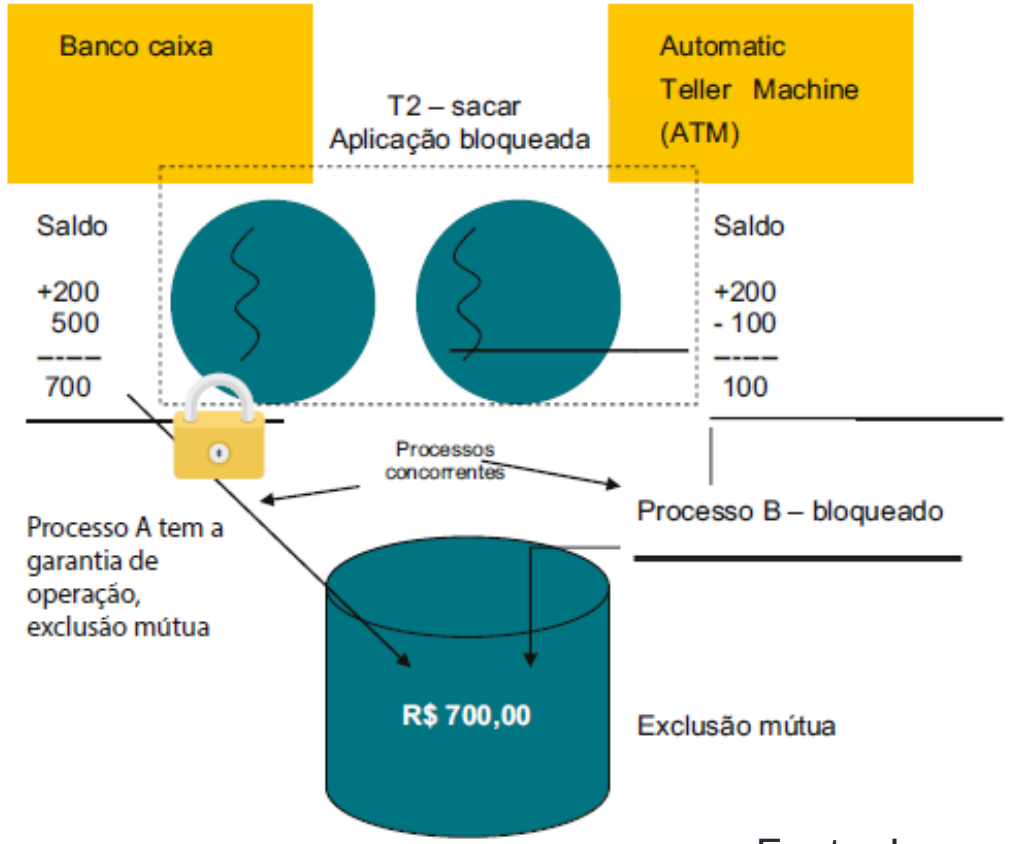
A garantia que um recurso compartilhado não será acessado por dois processos ao mesmo tempo é chamada de **exclusão mútua (*mutex*) .**

Embora processos paralelos possam cooperar entre si de forma correta e efetiva, condições são impostas para que aconteça uma boa solução de sincronismo:

- **O número de processadores e o tempo de execução dos processos devem ser irrelevantes.**
- **Nenhum processo que executa fora de sua região crítica pode bloquear outro processo.**
- **Nenhum processo deve esperar eternamente para entrar na sua região crítica.**

Exclusão mútua

A garantia que um recurso compartilhado não será acessado por dois processos ao mesmo tempo é chamada de **exclusão mútua (*mutex*) .**



No exemplo do caixa eletrônico, dois processos concorrentes chegam para serem executados na CPU.
O processo A com a finalidade de deposito no caixa, e o processo B com a finalidade de debitar um valor do
saldo.
A exclusão mútua garante exclusividade de operação para o processo A, sem interferência do processo B. Internamente, por meio de programação, o processo B fica bloqueado até o término da seção crítica em que o processo A faz o depósito.

Fonte: Lacerda, 2019 – p 55

Exclusão Mútua com Espera Ociosa

PUC Minas

Há diversas técnicas para garantir a exclusão mútua, ou seja, enquanto um processo estiver na sua região crítica, ou enquanto um processo for impedido de acessar a sua própria região crítica para acesso ao recurso compartilhado. São elas:

a) Desabilitando interrupções:

- esse mecanismo consiste em **que cada processo desabilite todas as interrupções ao entrar na sua região crítica, e ao sair, as torna ativa novamente**. Interrupções são sinais de algum dispositivo que resulta na troca de contexto, ou seja, mudança de processos na CPU. **Quando essas interrupções são desabilitadas, desabilita também qualquer interrupção feita pelo relógio.** O relógio determina um tempo de **processamento a cada processo e faz com que a CPU alterne seus processos a cada interrupção**. Com as interrupções em estado de *off* (desligadas), o processo ativo na CPU não será interrompido por qualquer outro processo e garante a exclusão mútua (TANENBAUM, 2016).

Exclusão Mútua com Espera Ociosa

PUC Minas

O problema com essa técnica refere-se a processos de usuário.

- Um processo de usuário pode ***desligar as interrupções para assegurar exclusão mútua em uma determinada tarefa e demora um tempo excessivo para habilitá-las novamente***. Esse procedimento pode causar consequências de lentidão e inanição no tempo de execução de um próximo processo na CPU.
- Imagine que o processo (processo A) que retirou as interrupções, seja um *enter* de teclado, e o próximo processo (processo B) seja o *click* do mouse. O usuário ao pressionar o botão do *mouse*, sentirá uma sensação de lentidão, pois não receberá uma resposta imediata do sistema operacional ao *click*.

Exclusão Mútua com Espera Ociosa

PUC Minas

b) Variáveis de impedimento:

- **numa solução de *software*, cria-se uma variável com valor 0.** Quando um processo deseja acessar a região crítica, ele verifica se a variável está com valor 0, caso sim, altera o valor para 1 e entra na sua região crítica. Caso contrário, espera até que o valor seja novamente 0 para entrar em sua região crítica, pois nesse instante outro processo está acessando a região compartilhada de um determinado recurso. Pode existir o problema de dois processos em preempção alterarem a variável para 1 ao mesmo tempo, causando inconsistência.

c) Alternância obrigatória:

- quando uma **região deve ser entregue a um dado processo por vez é chamado de alternância obrigatória**.

Exclusão Mútua com Espera Ociosa

PUC Minas

Nesse contexto, são criadas soluções de ***software* para que a exclusão mútua seja garantida**.

Uma variável (*turn*) recebe o valor zero, e controla a vez de quem entra na região crítica e verifica ou atualiza a memória compartilhada. **Um laço é criado para verificar a oportunidade do processo entrar na região crítica. Esse teste contínuo de verificar a variável para ascender a região crítica é chamado de espera ocupada (*busy waiting*).**

```
while(true){
        while(turn != 1)
        critical_region();
        turn = 0;
        nocritical_region();
}
```

```
while(true){
        while(turn != 0)
        critical_region();
        turn = 1;
        nocritical_region();
}
```

Uma espera ocupada deve ser evitada, pois gera custo para a CPU. Já a variável de impedimento que usa a espera ocupada é chamada de *spin lock*.

Exclusão Mútua com Espera Ociosa

PUC Minas

d) Solução de Peterson:

- criado por Gary L. Peterson, esse algoritmo de programação é uma solução de software para resolver problemas de exclusão mútua. Nesse algoritmo há apenas uma variável compartilhada (variável de travamento), cujo o valor inicial é 0. Quando algum processo deseja entrar em sua região crítica, ele deve primeiro testar essa variável. Caso ela seja 0, o processo a muda para 1 e entra na região critica. Se o valor da variável de travamento já for 1, o processo vai esperar que o valor volte para 0, antes de entrar na região crítica.

```
#define FALSE 0

#define TRUE 1

#define N 2

int turn
int interested[N]

void enter_region(int process)
{

 int other;
 other =1 - process;
 interested[process] = TRUE;
 turn = process;
 while (turn == process && interested[ohter] == true )

}

void leave_region (int process){
       interested[process] = FALSE;
}
```

Técnica para garantir a exclusão mutua

PUC Minas

a) Desabilitando interrupções: o processo desabilita todas as interrupções ao entrar na sua região crítica, e habilita as interrupções novamente ao sair.

b) Variáveis de impedimento: no software é criada uma variável com valor “0”. Quando um processo acessar a região crítica, altera esta variável para o valor “1”. Ocorre que dois processos em preempção podem alterar a variável ao mesmo tempo, causando inconsistência.

c) Alternância obrigatória: soluções de software na qual uma região crítica seja entregue a um determinado processo por vez.

d) Solução de Peterson: trata-se de um algoritmo de programação, que é uma solução de *software* para resolver problemas de exclusão.

e) Instrução TSL (test and set lock): Solução de hardware para o problema da exclusão mútua em ambiente com vários processadores.

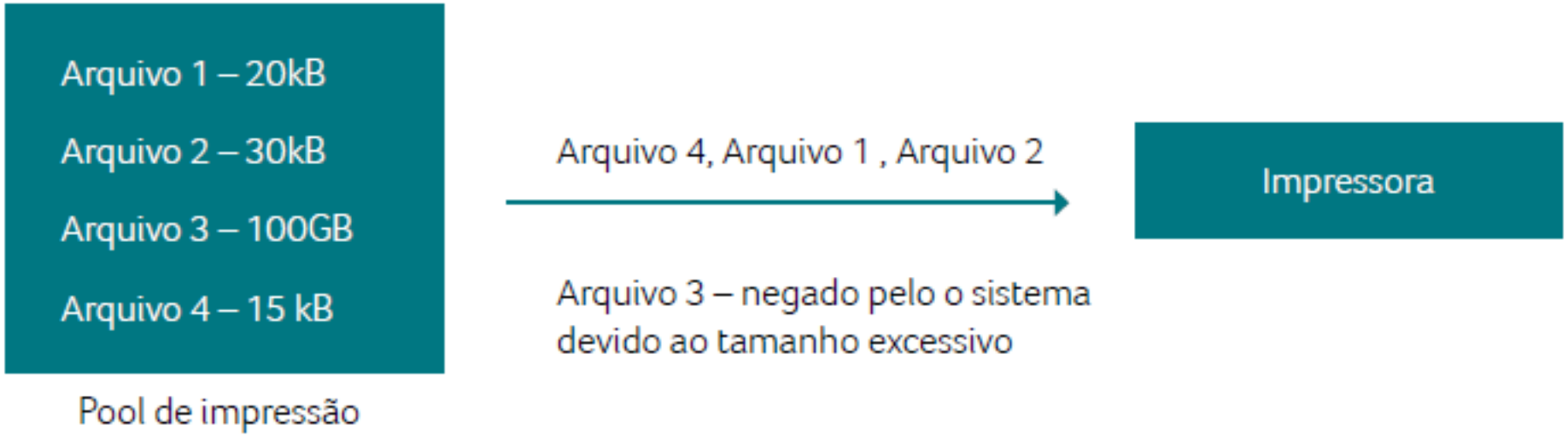
Problemas de sincronismo

PUC Minas

É preciso haver sincronismo correto entre as operações realizadas por um sistema operacional, afim de evitar efeitos colaterais de suas ações.

Velocidade de execução: dois processos com tempos de execução diferentes o processo a velocidade do processo mais veloz fica limitada a velocidade do processo menos veloz.

***Stavation* (inanição):** Um processo nunca pode ser executado devido a um impedimento provocado por outro processo com maior prioridade de execução.



Soluções de sincronismo

PUC Minas

Via Hardware

- Desabilitar as interrupções.
- Instrução Test-and-set

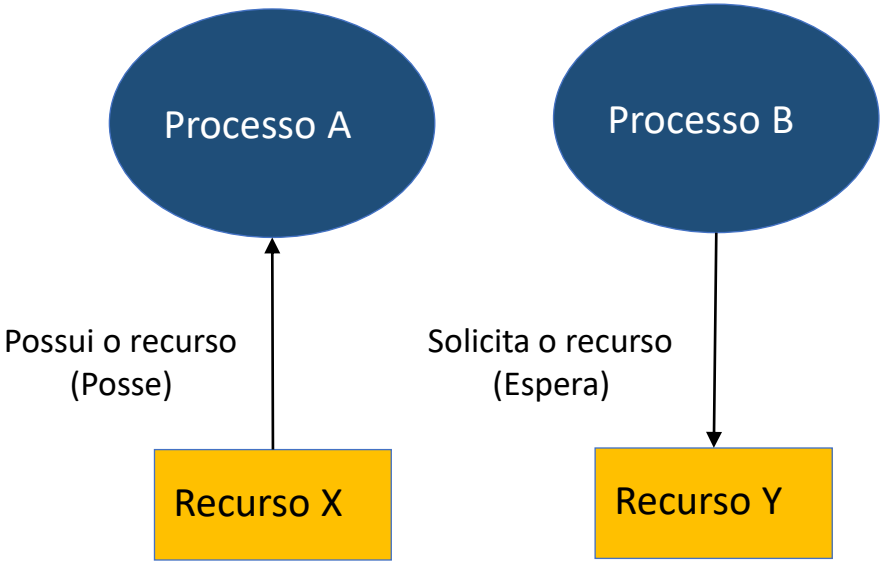
Via Software

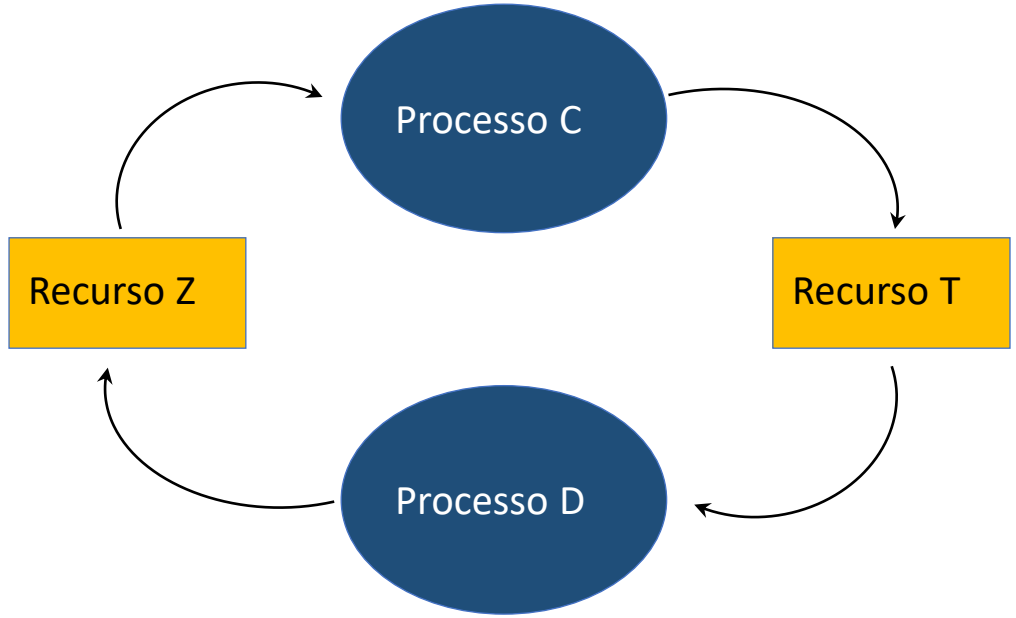
- Semáforos.
- Semáforo binário.
- Semáforo de contagem.
- Monitores.
- Troca de mensagens.

Deadlock (Impasse)

PUC Minas

Um ***deadlock*** ocorre quando **dois ou mais processos estão bloqueados aguardando por recursos que estão com outros processos, também bloqueados.** Trata-se de um impasse, pois o processo B fica esperando pelo processo C e o processo C fica esperando pelo processo B. Com isso, os processos esperam por um recurso e não são executados na CPU. Na abaixo, o processo C possui o recurso Z e precisa do recurso T para continuar a execução. Já o processo D possui o recurso T e precisa do recurso Z para continuar a sua execução. Como os processos B e C estão aguardando os recursos que estão com o outro processo e estes recursos não serão liberados, isso gera um *deadlock*.





Deadlock

PUC Minas

Em geral, há quatro condições para que ocorra um *deadlock* (MACHADO; MAIA, 2013):

• Condição de exclusão mútua: cada recurso só pode estar alocado a um único processo em um determinado instante.

• Condição de posse de espera: um processo além do processo já alocado, pode esperar por outro recurso.

• Condição de não preempção: um recurso pode não ser liberado de um processo somente porque outros processos desejam alocar o mesmo recurso.

• Condição de espera circular: um processo pode esperar por um recurso alocado a outro processo e vice-versa.

Observação importante: Deadlocks ocorrem tanto com recursos de software como de hardware.

O que fazer quando ocorrer um *Deadlock* (Impasse) ?

PUC Minas

1) Ignorar o *deadlock*.
   - ➢ Existe um custo para tratar o deadlock e é bastante alto (Algoritmo do Avestruz).

2) Detectar e Recuperar.
   - ➢ Permite que os deadlocks ocorram, tenta detectar as causas e solucionar a situação.

Algoritmos:

- Detecção com um recurso de cada tipo.
- Detecção com vários recursos de cada tipo.
- Recuperação por meio de preempção.
- Recuperação por meio de *rollback* (volta ao passado)
- Recuperação por meio de eliminação de processos.

3) Evitação Dinâmica – Alocação cuidadosa do recurso.

4) Prevenção - Negação de uma das quatro condições necessárias.