SISTEMAS OPERACIONAIS

AULA 8 – CONCORRÊNCIA E SINCRONIZAÇÃO DE PROCESSOS, PARTE I

Prof.ª Sandra Cossul, Ma.

REVISANDO ALGUNS CONCEITOS...

- **MULTITAREFA / MULTIPROGRAMAÇÃO:** O processador executa múltiplas tarefas alternando entre as mesmas a cada período de tempo. Essa troca é muito rápida e frequente, não se tornando perceptível a nível de usuário.

- **MULTIPROCESSAMENTO:** sistema que tem dois ou mais processadores, com o intuito de aumentar o poder de processamento do sistema, fazendo com que os processos sejam executados simultaneamente.

REVISANDO ALGUNS CONCEITOS...

- **MULTICORE:** processador com mais de um núcleo.
- O CPU tem unidades de processamento separadas, que são chamadas de núcleos (cores). Maioria dos sistemas atuais tem 4 ou 8 cores (no mesmo chip)
- Cada núcleo pode executar instruções de programas
- O CPU pode rodar várias instruções no mesmo tempo, melhorando o tempo de execução da aplicação.
- Tem suporte a multithreading!

REVISANDO ALGUNS CONCEITOS...

- **MULTITHREADING:** técnica de execução de aplicações que permite que um processo tenha múltiplos segmentos de códigos (threads).
- Múltiplas threads do mesmo processo são executadas ao mesmo tempo.
- Processador alterna entre as múltiplas threads para "parecer" que estão rodando simultaneamente.
- Threads são leves para executar e compartilham o mesmo espaço de endereços.

INTERAÇÃO ENTRE PROCESSOS

- Nem sempre um programa sequencial é a melhor solução para um determinado problema

- Muitas vezes programas estruturados na forma de várias tarefas interdependentes que cooperam entre si para atingir os objetivos de uma aplicação são mais eficientes

JUSTIFICATIVA DE SISTEMAS COM TAREFAS COOPERANTES

- **Atender vários usuários simultâneos** (servidor de banco de dados ou de email)
- **Uso de computadores multiprocessador** (tarefas podem ser "quebradas" e escalonadas simultaneamente nos CPUs disponíveis)
- **Modularidade** (dividir um sistema grande e complexo em módulos com responsabilidades próprias que cooperam entre si)
- **Construção de aplicações interativas** (navegadores Web, editores de texto e jogos são exemplos de aplicações com alta interatividade)

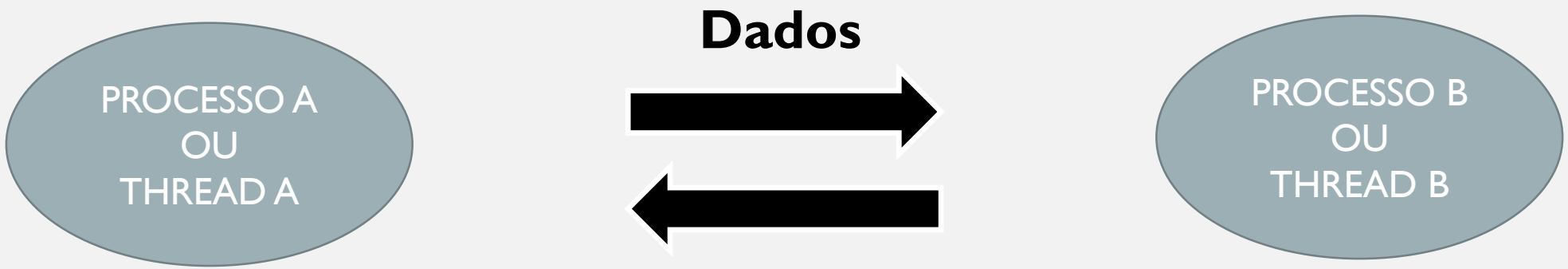
SINCRONIZAÇÃO DE PROCESSOS

- **Simultaneidade/Concorrência** – aspecto fundamental da estrutura de um SO
  - **IPC** – Interprocess Communication (comunicação entre processos)
  - Compartilhamento e competição por recursos (memória, arquivos e E/S)
  - Sincronização das atividades de vários processos
  - Alocação de tempo do processador para os processos

- **Obs.:** Os mesmos problemas e soluções citados se aplicam tanto a processos como a threads.

COMUNICAÇÃO ENTRE PROCESSOS

- **Processos necessitam se comunicar com outros processos:**
  - Como um processo passa informação para um outro?
  - Como garantir que dois processos não entrem em conflito?
  - Como executar dois processos com dados dependentes?



PROCESSOS CONCORRENTES

- **Condição de corrida (disputa):** ocorre quando múltiplos processos ou threads escrevem e lêem dados de forma que o resultado final depende da ordem de execução das instruções dos processos.

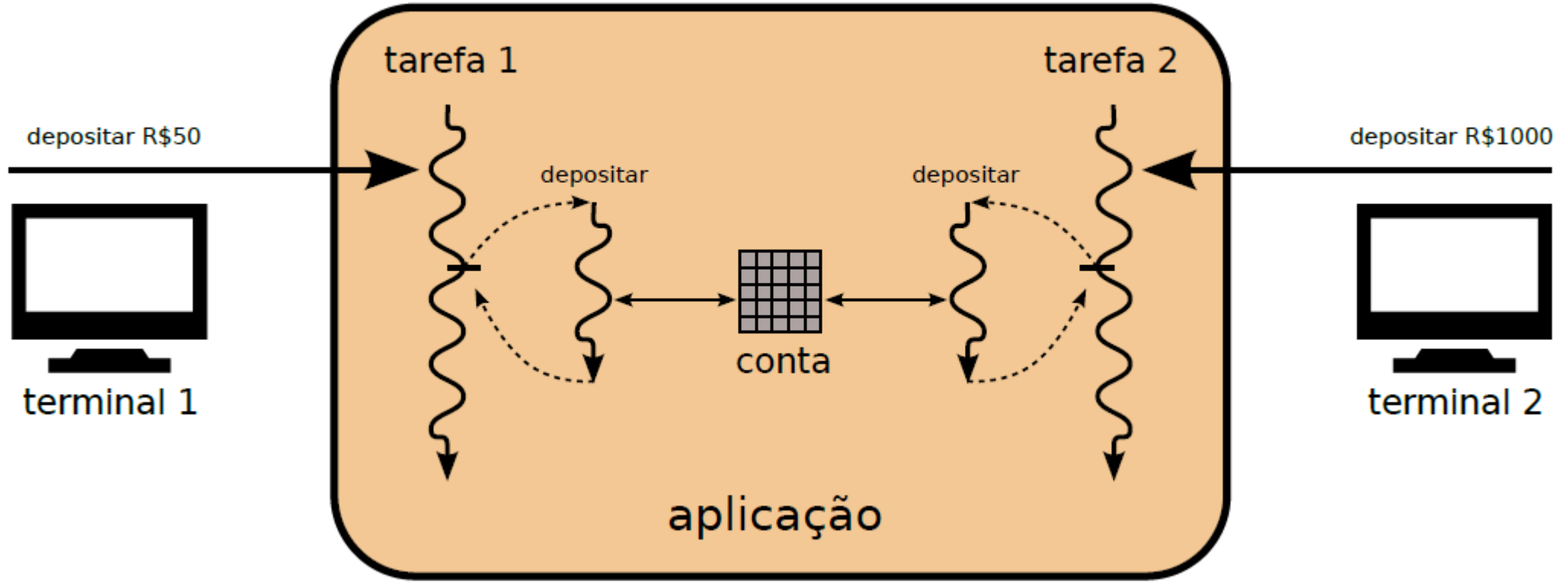
- **Exemplo 1:**
  - Processo P1 e P2 compartilham uma mesma variável global a
  - P1 atualiza variável a (a = 1)
  - P2 atualiza variável a (a = 2)
  - Então os dois processos estão em uma "corrida" para escrever na variável a
  - O processo que atualiza por último (perde a corrida) determina o valor final de a

PROCESSOS CONCORRENTES

- **Exemplo 2:**
  - Processo P3 e P4 compartilham as variáveis globais "b" e "c"
  - Valores iniciais: b = 1 c = 2
  - Durante a execução, P3 faz: **b = b + c**
  - Durante a execução, P4 faz: **c = b + c**
  - Os dois processos atualizam variáveis diferentes
  - No entanto, o valor final das variáveis depende da ordem de execução

PROCESSOS CONCORRENTES

- **Exemplo 3: Depósito de valor em um saldo de conta bancária**
- Acesso concorrente a variáveis compartilhadas



PROCESSOS CONCORRENTES

- **Exemplo 3: Depósito de valor em um saldo de conta bancária**
- Caso o depósito da tarefa t1 execute integralmente **antes** ou **depois** do depósito efetuado por t2, em ambas as execuções o saldo inicial da conta passa de R$ 0,00 para R$ 1050,00, conforme o esperado.

- No entanto, caso as operações de depósito t1 e t2 se entrelacem, podem ocorrer interferências entre ambas, levando a resultados incorretos.
- Por exemplo, um dos depósitos se perder, em que só é concretizado o depósito da tarefa que realizou a escrita do resultado na memória por último.

SINCRONIZAÇÃO DE PROCESSOS

- **Sincronizar processos** envolve o uso de técnicas que fazem o controle de acesso aos dados compartilhados, de forma a evitar condições de corrida.

- Essas técnicas devem ser "bem" implementadas, pois podem resultar em perda de eficiência do sistema e outras condições como **deadlock** e **starvation**.

DEADLOCK

- **Deadlock**
  - Situação em que 2 ou mais processos não conseguem continuar a execução porque um está esperando o(s) outro(s) fazer(em) alguma ação.

- **Exemplo:**
  - *2 processos*: P1 e P2 e *2 recursos:* R1 e R2
  - Cada processo precisa acesso ao dois recursos para conseguir executar
  - O SO aloca R1 para P2 e R2 para P1
  - Cada processo fica aguardando indefinidamente pelo outro recurso (deadlock!)

STARVATION

- **Starvation** (passar fome)
  - Situação em que um processo no estado pronto nunca é escalonado para executar; nunca é escolhido!

- **Exemplo:**
  - *3 processos*: P1, P2 e P3 que periodicamente requerem acesso ao recurso R
  - *Situação 1*: P1 tem o controle do recurso R. P2 e P3 ficam atrasados, esperando pelo recurso. Quando o recurso é liberado por P1, tanto o P2 como o P3 podem utilizar o recurso.
  - Assuma que o SO garante o recurso para P3 e logo em seguida aloca novamente para P1. O processo P2 pode ter o acesso negado ao recurso indefinidamente, mesmo que não haja deadlock.

ASPECTOS DE GERENCIAMENTO DO SO

- **Acompanhamento dos processos**
  - Isto é feito pelos blocos de controle de processos, comentado em aula anterior
- **Alocação e desalocação de recursos para os processos ativos**
  - Muitas vezes, os processos requerem acesso aos mesmos recursos: *Tempo do processador, Memória, Arquivos* e *Dispositivos de E/S*.
- **Proteção dos dados e recursos físicos**
  - Garantir que um processo não seja interferido por outros processos
- **O funcionamento de um processo e a saída que ele produz devem ser independentes da velocidade com que sua execução é realizada em relação a velocidade de execução de outros processos simultâneos.**

INTERAÇÃO ENTRE PROCESSOS

- Podemos classificar as formas de interação entre processos baseado no grau de percepção da existência de outros processos:

- **Sem conhecimento:**
  - processos independentes
  - *Relação*: competição por recursos

INTERAÇÃO ENTRE PROCESSOS

- **Com conhecimento indireto**
  - não sabe o ID do processo, mas compartilham acesso ao mesmo objeto (ex.: buffer de E/S)
  - *Relação*: cooperação por compartilhamento

- **Com conhecimento direto**
  - conseguem se comunicar pelo IDs do processos
  - são criados para trabalhar em conjunto em alguma atividade
  - *Relação*: cooperação por comunicação

PROCESSOS SEM CONHECIMENTO
COMPETIÇÃO ENTRE PROCESSOS POR RECURSOS

COMPETIÇÃO ENTRE PROCESSOS POR RECURSOS

- **Competição entre processos:**
  - Dois processos precisam acessar o mesmo recurso durante a execução
  - Um processo não sabe da existência do outro
  - No entanto, um é afetado pela execução do outro
  - Não há troca de informações entre os processos competidores

- **Problemas de controle:**
  - *Necessidade de exclusão mútua*
  - *Deadlock*
  - *Starvation*

Controle da competição envolve o SO pois é este que aloca os recursos!

COMPETIÇÃO ENTRE PROCESSOS POR RECURSOS

- **Exclusão mútua**
  - Requerimento de que quando um processo está em uma região crítica (que faz o acesso a um recurso compartilhado), nenhum outro processo pode estar na região crítica solicitando acesso ao recurso ao mesmo tempo.

Recurso crítico
→ recurso sendo requisitado

Região crítica
→ parte do código que faz o acesso ao recurso crítico

Os processos devem ser capazes de por si só requerer exclusão mútua.

REGIÕES CRÍTICAS

- **O que não representa um risco?**
  - Trechos de código para realizar cálculos ou outras atividades

- **O que representa um risco?**
  - Certas situações em que se desvia para o código que trata dos recursos compartilhados

- **O que precisamos garantir?**
  - Que dois ou mais processos não entrem juntos na região crítica

PROCESSOS COM CONHECIMENTO INDIRETO

COMPETIÇÃO ENTRE PROCESSOS POR COMPARTILHAMENTO

COOPERAÇÃO ENTRE PROCESSOS POR COMPARTILHAMENTO

- **Cooperação entre processos por compartilhamento:**
  - processos que interagem entre si sem ter conhecimento explícito da existência dos outros
  - **Exemplo:**
    - múltiplos processos tem acesso a variáveis, arquivos ou dados compartilhados.
    - processos utilizam e atualizam o objeto compartilhado sem fazer referência aos outros processos, mas sabem que outros processos podem ter acesso aos mesmos objetos.
    - Por isso, os processam precisam **cooperar** para garantir o gerenciamento dos dados compartilhados e a **integridade** dos mesmos pelos mecanismos de controle.

COOPERAÇÃO ENTRE PROCESSOS POR COMPARTILHAMENTO

- **Os dados ficam alocados em recursos** (dispositivos, memória), então os mesmos problemas de **exclusão mútua**, **deadlock** e **starvation** acontecem.

- **Diferença**: dados são acessados em dois modos: escrita e leitura e apenas operações de *escrita* precisam ser **mutuamente exclusivas.**

- **Requerimento:** coerência dos dados !!!
  - Uso de regiões críticas.

PROCESSOS COM CONHECIMENTO DIRETO
COMPETIÇÃO ENTRE PROCESSOS POR COMUNICAÇÃO

COOPERAÇÃO ENTRE PROCESSOS POR COMUNICAÇÃO

- **Cooperação entre processos por comunicação:**
  - processos interagem para realizar uma atividade
  - Comunicação: sincronização e coordenação

  - Exclusão mútua não é um requerimento pois nada é compartilhado
  - No entanto, tem-se os problemas de *deadlock* e *starvation* (devido a comunicação entre os processos)

COOPERAÇÃO ENTRE PROCESSOS POR COMUNICAÇÃO

- **Deadlock:**
  - dois processos bloqueados, cada um esperando pela comunicação com o outro.

- **Starvation**
  - 3 processos P1, P2 e P3
  - P1 tenta se comunicar com P2 ou P3
  - P2 e P3 tentem se comunicar com P1
  - Situação: P1 e P2 começam a trocar informação repetidamente, enquanto o P3 fica bloqueado esperando pela comunicação com P1 (P3 is starving!)

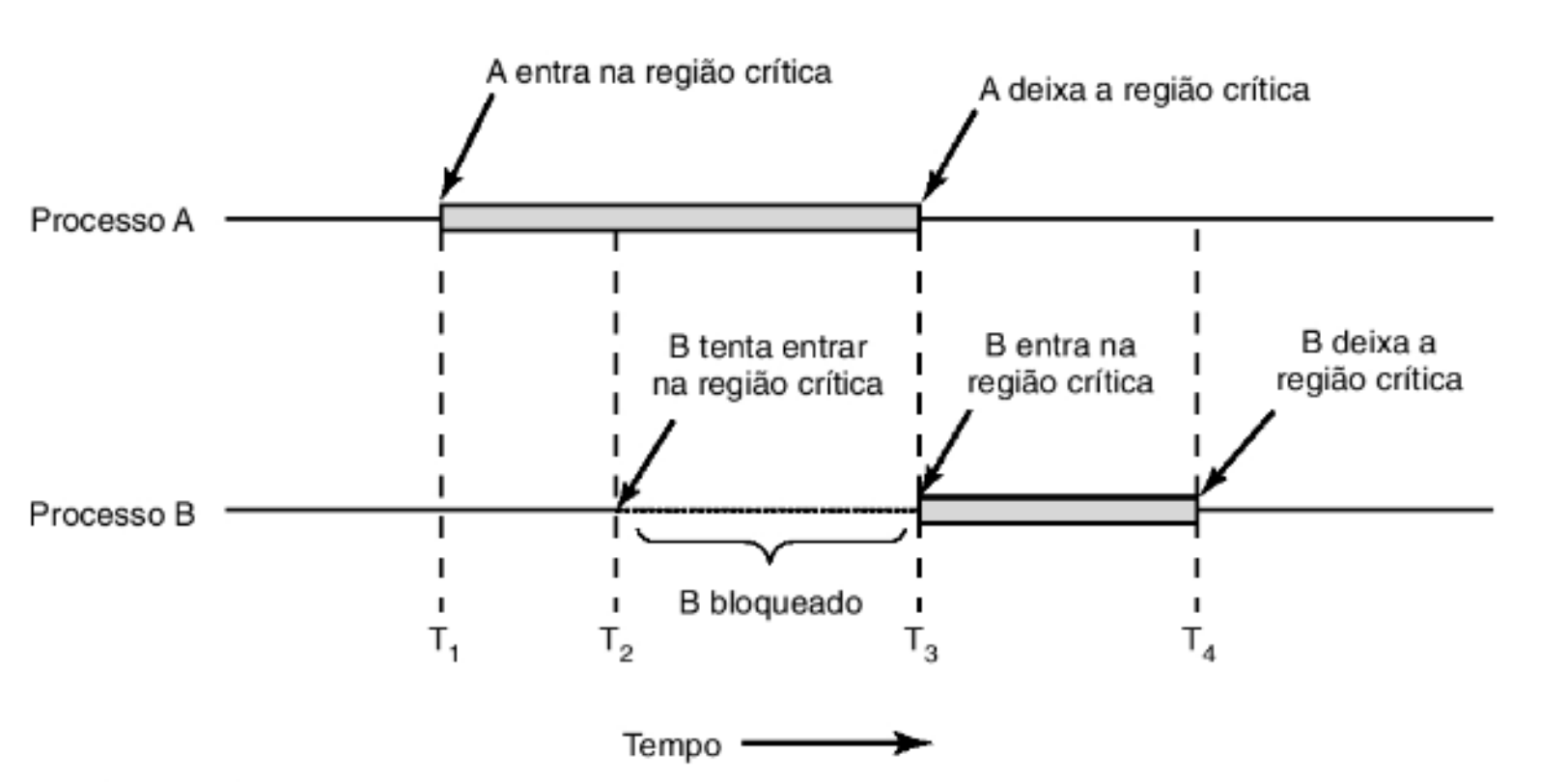
REQUERIMENTOS PARA EXCLUSÃO MÚTUA

REQUERIMENTOS PARA EXCLUSÃO MÚTUA

1. **Apenas um processo por vez deve ser permitido executar a região crítica,** dentre todos os processos que possuem regiões críticas para o mesmo recurso.
2. Um processo que para em sua região *não crítica* **não deve interferir** nos **outros processos**
3. Um processo que requer acesso a *região crítica* não pode ser atrasado indefinidamente**: sem deadlock ou starvation**
4. Quando nenhum processo estiver em uma seção crítica, **qualquer processo que solicite** entrada em sua região crítica **deve ter permissão**
5. **Não são feitas suposições** sobre as **velocidades** relativas do **processo** ou ao **número de processadores**
6. Um processo permanece dentro de sua ***região crítica*** por um **tempo finito**.

REQUERIMENTOS PARA EXCLUSÃO MÚTUA

Exemplo:



FORMAS DE IMPLEMENTAR EXCLUSÃO MÚTUA

- **Abordagem por Software**
  - os próprios processos se coordenam entre si para garantir exclusão mútua;
  - propensa a alto custo de processamento e bugs (erros)
  - Algoritmos de Dekker e **Petterson**

- **Abordagem por Hardware**
  - uso de instruções de máquina com propósito específico

- **Abordagem alternativa: Suporte do SO ou de uma Linguagem de Programação**
  - monitores, semáforos e transmissão de mensagens

ABORDAGEM POR SOFTWARE
ALGORITMO DE PETTERSON

- **Uso de duas variáveis:** bool **flag[2]** = {"false", "false"} e int **turn**
- Um valor verdadeiro de flag[n] indicar que o processo n quer entrar na seção crítica.
- A entrada na seção crítica é concedida para o processo P0 se P1 não quiser entrar em sua seção crítica e se P1 tiver dado prioridade a P0 definindo turn = 0.

```
P0:       flag[0] = true;
P0_gate:  turn = 1;
          while (flag[1] == true && turn == 1)
          {
              // busy wait
          }
          // critical section
          ...
          // end of critical section
          flag[0] = false;
```

```
P1:       flag[1] = true;
P1_gate:  turn = 0;
          while (flag[0] == true && turn == 0)
          {
              // busy wait
          }
          // critical section
          ...
          // end of critical section
          flag[1] = false;
```

ABORDAGEM POR HARDWARE

- **Opção 1: Desativar interrupções**
  - Um processo vai continuar executando até que requisite um serviço ao SO ou até que seja interrompido
  - Então, para garantir exclusão mútua, é suficiente evitar que um processo seja interrompido
  - A **região crítica** não pode ser interrompida, o que garante **exclusão mútua**

- **Processo desabilita interrupções**

ao entrar na região crítica e habilita

ao sair.

```
while (true) {
/* disable interrupts */;
/* critical section */;
/* enable interrupts */;
}
```

ABORDAGEM POR HARDWARE

- **Opção 1: Desativar interrupções**
  - **Ao desligar interrupções, a preempção por tempo ou por recursos deixa de funcionar** – caso a tarefa entre em um laço infinito dentro da seção crítica, o sistema inteiro será bloqueado.
  - **Enquanto as interrupções estão desativadas, os dispositivos de E/S deixam de ser atendidos pelo núcleo -** o que pode causar perda de dados ou outros problemas.

ABORDAGEM POR HARDWARE

- **Opção 1: Desativar interrupções**
  - **A tarefa que está na seção crítica não pode realizar operações de entrada/saída, pois os dispositivos não irão responder.**
  - **Não funciona em multiprocessadores –** desabilitar interrupções em um núcleo da CPU não evita que as tarefas em outros núcleos acessem a seção crítica.

ABORDAGEM POR HARDWARE

- **Opção 2: Instruções de máquina especiais**
  - acesso a uma posição de memória bloqueia o acesso na mesma localização
  - Durante a execução de certas instruções de máquina (escrita e leitura ou leitura e teste), o acesso para aquela localização de memória sendo referenciada é **bloqueado** para qualquer outra instrução que faz referência ao mesmo local.
  - **Instruções atômicas**

ABORDAGEM POR HARDWARE

- **Opção 2: Instruções de máquina especiais**
  - ***Instruções "compare-and-swap"***
    - Compara o conteúdo de um endereço de memória com um certo valor *(valor de teste)* e, **somente se eles iguais,** modifica o conteúdo daquele local de memória para um novo valor.
    - **Comparação –** valor na memória e um valor de teste
    - **Swap (troca) –** atualiza o valor se são iguais
    - O valor "antigo" de memória é retornado, então a memória foi atualizada se o valor de retorno é o mesmo do valor de teste.
    - Executada no modo atômico, isto é, não pode ser interrompida

ABORDAGEM POR HARDWARE

- **Opção 2: Instruções de máquina especiais**
  - **Vantagens:** aplicável a qualquer número de processos (e processadores); simples e fácil de verificar e tem suporte para várias regiões críticas (cada região crítica com sua variável).
  - **Desvantagens:**
    - busy waiting – enquanto um processo está esperando para entrar na seção crítica, consome tempo de CPU
    - possibilidade de deadlock e starvation.

ABORDAGEM POR HARDWARE

- **Opção 2: Instruções de máquina especiais**

- Os mecanismos de exclusão mútua usando instruções atômicas são amplamente usados no interior do sistema operacional, para controlar o acesso a seções críticas dentro do núcleo
  - buffers de arquivos, conexões de rede, etc.

PROBLEMAS ACESSO CONCORRENTE

- O acessso concorrente de diversos processos aos mesmos recursos pode provocar problemas de consistência → **condições de disputa**
- Uma forma de eliminar esses problemas é forçar o acesso a esses recursos em **exclusão mútua**, ou seja, um processo (ou thread) por vez
- Vimos algumas soluções para garantir exclusão mútua, porém essas soluções tem alguns problemas que impedem o uso em larga escala nas aplicações de usuário:

- **Ineficiência -** as tarefas que aguardam o acesso a uma seção crítica ficam testando continuamente uma condição, consumindo tempo de processador sem necessidade *(busy waiting)*.

PROBLEMAS ACESSO CONCORRENTE

- **Injustiça -** não há garantia de ordem no acesso à seção crítica; dependendo da duração de quantum e da política de escalonamento, uma tarefa pode entrar e sair da seção crítica várias vezes, antes que outras tarefas consigam acessá-la.

- **Dependência -** tarefas desejando acessar a seção crítica podem ser impedidas de fazê-lo por tarefas que não têm interesse na seção crítica naquele momento.

ABORDAGEM ALTERNATIVA: SO E LINGUAGENS DE PROGRAMAÇÃO

- **Assunto da próxima aula!**

- Vamos falar de estruturas de controle e sincronização mais sofisticados que resolvem os problemas citados anteriormente.

CONCEITOS CHAVES - CONCORRÊNCIA

1. **Operação atômica**
2. **Seção crítica**
3. **Deadlock**
4. **Exclusão mútua**
5. **Condição de corrida**
6. **Starvation**

EXERCÍCIO

- (3) A situação que ocorre quando dois ou mais processos ficam impossibilitados de executar porque um fica esperando pelo outro realizar alguma ação.
- (1) Ação implementada como uma sequência de uma ou mais instruções que são "indivisíveis", ou seja, nenhum outro processo pode interromper a execução. Garante isolamento dos outros processos concorrentes.
- (6) A situação que ocorre quando um processo pronto para executar é ignorado pelo escalonador e nunca é escolhido para executar
- (2) Uma seção de código dentro de um processo que requer acesso a recursos compartilhados e que não deve ser executado enquanto outro processo estiver em uma seção de código correspondente.
- (5) Uma situação em que vários *threads* ou processos leem e gravam um item de dados compartilhado, e o resultado final depende do tempo relativo de sua execução.
- (4) A exigência de que quando um processo está em uma seção crítica que acessa recursos, nenhum outro processo pode estar em uma seção crítica que acessa qualquer um desses recursos compartilhados.

BIBLIOGRAFIA

- Tanenbaum, A. S. **Sistemas Operacionais Modernos.** Pearson Prentice Hall. 3rd Ed., 2009.
- Silberschatz, A; Galvin, P. B.; Gagne G.; **Fundamentos de Sistemas Operacionais.** LTC. 9th Ed., 2015.
- Stallings, W.; **Operating Systems: Internals and Design Principles**. Prentice Hall. 5th Ed., 2005.
- Maziero, C. A.; **Sistemas Operacionais: Conceitos e Mecanismos.** DINF – UFPR, 2019.
- *baseado nos slides da Prof.ª Roberta Gomes (UFES) e do Prof. Felippe Fernandes da Silva