Comunicação entre processos

Thread (processador e SO)

- Quando se trata de processadores e sistemas operacionais, threads são conceitos diferentes, mas conexos
- Thread do Processador
  - Por meio de métodos como o hyper-threading, suportam a execução paralela de várias tarefas por núcleo
- Thread do Sistema Operacional
  - Unidade básica de execução dentro do sistema operacional
- O SO gerencia as threads de software, enquanto o processador executa as threads de hardware

Thread (SO)

- Em uma aplicação concorrente (execução cooperativa de processos e threads), os processos precisam se comunicar entre eles
- Solicitam o uso de recursos como memória, arquivos, dispositivos de entrada/saída e registros

Dependência

- Como um processo passa a informação para outro processo?
- Necessário garantir que 2 ou mais processos não invadam uns aos outros quando estão em regiões críticas
  - Quando um processo estiver usando uma região de memória, o outro processo deve aguardar a sua vez
- É necessário existir uma hierarquia quando houver dependências
  - Se o processo A produz dados e o processo B os imprime
    - B deve esperar até que A produza dados para serem impressos

Condições de disputa ou condições de corrida

- Acontecem quando 2 ou mais processos estão compartilhando uma região da memória (ex.: lendo ou escrevendo dados)
  - E o resultado depende das informações de quem executa e quando
- Suponha que 2 funcionários do banco atualizem o saldo do mesmo cliente simultaneamente
- O processo do primeiro funcionário lê o registro do cliente e soma ao saldo o valor sacado pelo cliente
- Porém, antes de gravar o novo saldo no arquivo, o segundo funcionário lê o registro do mesmo cliente que está sendo atualizado e lança um crédito a ser somado ao saldo

Exclusão mútua

- Quando um processo estiver lendo ou gravando dados em sua região crítica, ele deve fazer outros processos esperarem
- Ou um acessa, ou outro acessa, os 2 ao mesmo tempo não
- A parte do programa em que o processo acessa a memória compartilhada é chamada de região crítica ou seção crítica

4 itens devem ser satisfeitos

- Dois ou mais processos jamais estarão ao mesmo tempo em suas regiões críticas
- Não se pode afirmar nada sobre o número e a velocidade de CPUs
- Nenhum processo que esteja executando fora de sua região crítica pode bloquear outros processos
- Nenhum processo deve esperar sem ter uma previsão para entrar em sua região crítica

Exclusão mútua – soluções propostas

- Exclusão mútua com espera ociosa
- Dormir e acordar
- Semáforos
- Monitores e troca de mensagens

Exclusão mútua com espera ociosa

- Existem alguns métodos que impedem que um processo invada outro quando um deles está em sua região crítica
- Desabilitando interrupções
  - São desabilitadas por cada processo assim que entra em sua região crítica e ativadas novamente antes de sair dela
    - A CPU não será disponibilizada para outro processo
  - Em sistemas com múltiplos processadores, a interrupção acontece em apenas um processador e os outros acessariam normalmente a memória compartilhada, comprometendo essa solução

Variáveis de impedimento

- Uma variável chamada lock, inicializada com o valor 0
- O processo testa e verifica o valor dessa variável antes de entrar na região crítica
  - Caso o valor seja 0, o processo o altera para 1 e entra na região crítica
  - O outro processo verifica se o valor é 1, e aguarda
- Não resolve totalmente o problema de exclusão mútua e ainda mantêm a condição de disputa
  - Variável sendo alterada ao mesmo tempo pelos processos

Alternância obrigatória

- Utiliza uma variável turn compartilhada que informa qual processo poderá entrar na região crítica (ordem)
- Deve ser alterada para o processo seguinte, antes de deixar a região crítica
- Quando um processo deseja entrar em sua região crítica, ele examina se sua entrada é permitida e, caso não seja, o processo fica esperando até que consiga entrar
- Desta forma, o processo A fica impedido pelo processo B, que NÃO está na sua região crítica
- Viola a condição de que nenhum processo que esteja executando fora de sua região crítica pode bloquear outros processos

Solução de Peterson

- Dois procedimentos escritos em C, baseado na definição de duas primitivas (enter_region e leave_region) utilizadas pelos processos que desejam utilizar sua região crítica
- Antes de entrar na região crítica, todo processo chama enter_region com os valores 0 ou 1
- Este apontamento faz com que o processo aguarde até que seja seguro entrar
- Depois de finalizar a utilização da região crítica, o processo chama leave_region e permite que outro entre
- Como a solução de Alternância Obrigatória, a Solução de Peterson precisa da espera ociosa

Instrução TSL (hardware)

- TSL - *test and set lock* / teste e atualize a variável de impedimento
- A instrução `TSL RX, LOCK` faz uma cópia do valor do registrador RX para LOCK
- Um processo somente pode entrar em sua região crítica se o valor de LOCK for 0

Espera ociosa - quem usa

- Alternância Obrigatória
- Solução de Peterson
- Instrução TSL

Dormir e acordar

- As soluções apresentadas até aqui utilizam a espera ociosa
  - Os processos ficam em um laço ocioso até que possam entrar na região crítica
- Para resolver este problema, são realizadas chamadas sleep (dormir) e wakeup (acordar) ao sistema, que bloqueiam/ desbloqueiam o processo, ao invés de gastar tempo de CPU com a espera ociosa
- A chamada sleep faz com que o processo que a chamou durma até que outro processo o desperte, e a chamada wakeup acorda um processo

Problema do produtor/consumidor

- Normalmente acontece em programas concorrentes em que um processo gera informações (produtor) para uso de outro processo (consumidor)
- Imagine que dois processos compartilham um buffer (memória) de tamanho fixo
  - O problema acontece quando o produtor quer inserir um novo item, porém o buffer está cheio ou o consumidor deseja remover um item e o buffer está vazio
  - A solução é colocar o processo, impedido pela capacidade do buffer, para dormir (através da chamada sleep) até que o outro modifique o buffer e acorde o anterior (por meio da chamada wakeup)

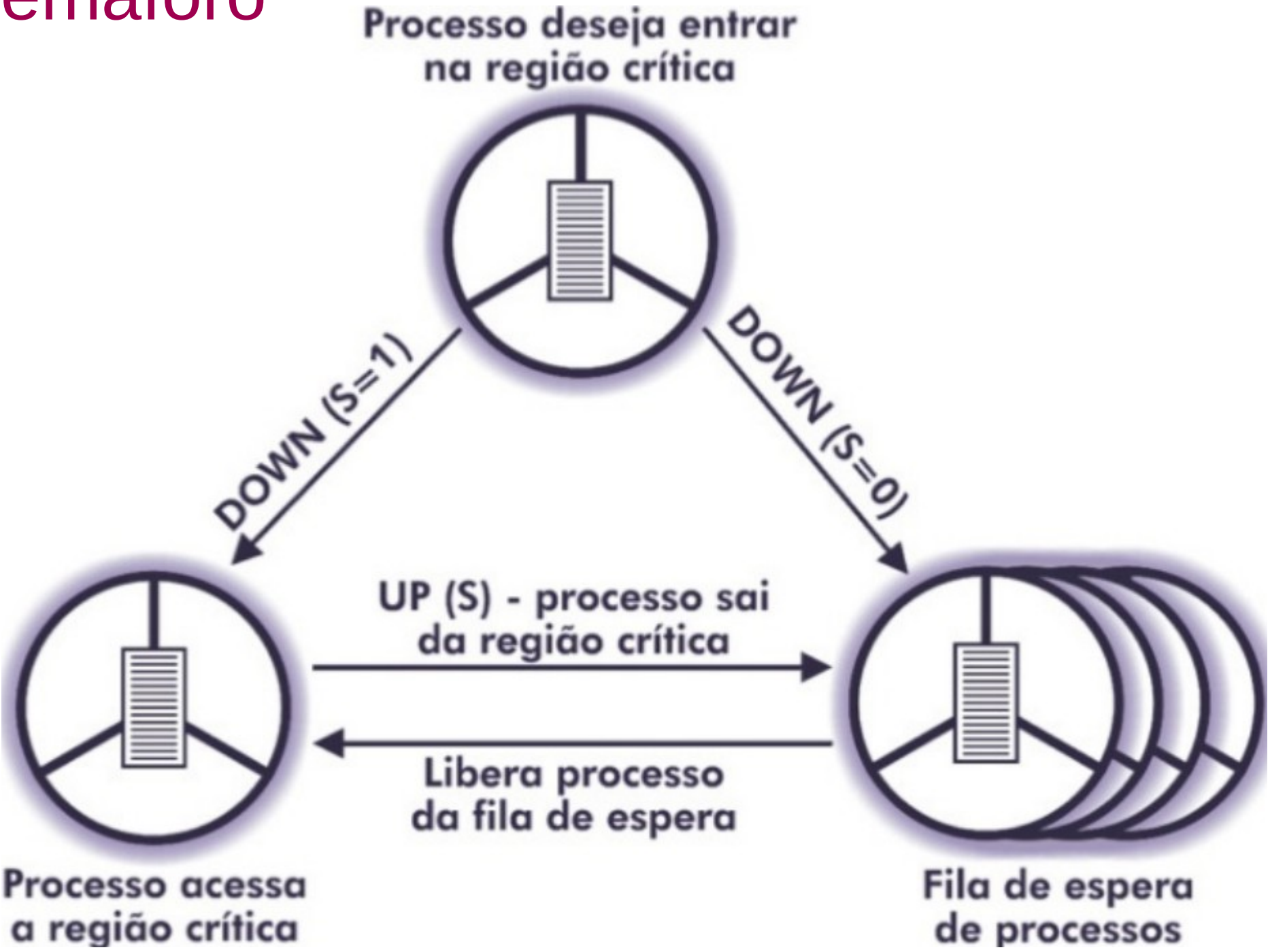
Semáforo

- É um dos mecanismos utilizados em projetos de sistemas operacionais e em aplicações concorrentes
- Grande parte das linguagens de programação disponibiliza procedimentos para semáforos
- Semáforo é uma variável inteira que realiza duas operações:
  - DOWN (decrementa uma unidade ao valor do semáforo)
  - UP (incrementa uma unidade ao valor do semáforo)
- As rotinas DOWN e UP são indivisíveis e executadas no processador

Semáforo

- Um semáforo com o valor 0 indica que nenhum sinal de acordar foi salvo e um valor maior que 0 indica que um ou mais sinais de acordar estão pendentes
- Os semáforos são classificados como:
  - Binários, também conhecidos como mutexes (mutual exclusion semaphores), que recebem os valores 0 ou 1
  - Contadores, que recebem qualquer valor inteiro positivo
- O semáforo com o valor igual a 1 significa que nenhum recurso está utilizando o processo e valor igual a 0 significa que o recurso está em uso

Semáforo


Processo deseja entrar
na região crítica
DOWN (S=1)
DOWN (S=0)
UP (S) - processo sai
da região crítica
Libera processo
da fila de espera
Processo acessa
a região crítica
Fila de espera
de processos

Semáforo

- Quando um processo deseja entrar em sua região crítica, é executada a instrução DOWN
- Caso o valor do semáforo seja igual a 1, o valor é decrementado e o processo pode entrar em sua região crítica
- Caso o valor seja 0 e a operação DOWN seja executada, o processo é impedido de entrar em sua região crítica, permanecendo em fila no estado de espera
- O processo que utiliza o recurso executa a instrução UP ao deixar a região crítica incrementa o valor do semáforo e libera o acesso ao recurso

Monitores

- É uma coleção de rotinas, estrutura de dados e variáveis que ficam juntos em um módulo ou pacote
- Também pode ser definido como uma unidade de sincronização de processos de alto nível
- Um processo, ao chamar uma rotina de um monitor, verifica se existe outro processo ativo
  - Caso esteja, o processo que chamou é bloqueado até que o outro deixe o monitor, senão, o processo que o chamou poderá entrar
- Garante a exclusão mútua, uma vez que só um processo pode estar ativo no monitor

Monitores

- Sob o ponto de vista do programa do usuário, o compilador é o responsável por definir a exclusão mútua nas entradas do monitor
- É preciso definir métodos para suspenderem os processos caso não possam prosseguir
  - Mesmo que a implementação da exclusão mútua em monitores seja fácil
- É necessário introduzir variáveis condicionais, com duas operações: wait e signal
  - Se um método do monitor verifica que não pode prosseguir, um sinal wait é emitido (bloqueando o processo), permitindo que outro processo que estava bloqueado acesse o monitor

Troca de mensagens

- Esse método utiliza duas chamadas ao sistema:
  - send (destination, &message) - envia uma mensagem para um determinado destino
  - receive (source, &message) - recebe uma mensagem de uma determinada origem
- Caso nenhuma mensagem esteja disponível, o receptor poderá ficar suspenso até chegar alguma
- Usa mecanismos para garantir o recebimento da mensagem
- Copiar mensagem é mais lento do que semáforo ou monitor