Nome: Barbara Letícia da Silva.
CJ3029921

Atividade de Pesquisa - Sistemas Operacionais

Descreva como dois processos podem se comunicar, usando memória compartilhada.

A memória compartilhada é um dos métodos mais rápidos e eficientes para a comunicação entre processos (Interprocess Communication – IPC), visto que possibilita que dois ou mais processos tenham acesso direto a uma área comum de memória.

A comunicação entre dois processos por memória compartilhada ocorre da seguinte forma: primeiro, um processo (normalmente o servidor) cria um segmento de memória compartilhada operando uma chamada do sistema, como shmget(), que retorna um identificador para essa área.

Em seguida, tanto esse processo quanto o outro interessado (cliente) mapeiam esse segmento em seus correspondentes espaços de memória por meio da função shmat(), adquirindo um ponteiro para acessar diretamente os dados.

Com a memória devidamente mapeada, os processos podem trocar informações lendo e escrevendo diretamente nesse espaço compartilhado, o que torna a comunicação rápida e eficiente.

Sendo assim, para garantir que não haja conflitos durante o acesso simultâneo, é fundamental usufruir de mecanismos de sincronização, como semáforos, mutexes ou flags, que controlam a ordem e o momento de leitura e escrita.

Por fim, após a troca de dados, os processos devem desanexar o segmento com shmdt() e, quando ele não for mais necessário, o criador deve removê-lo com shmctl(), permitindo assim os recursos do sistema.

O que é um condição de corrida?

Uma condição de corrida (race condition) é um problema que acontece em sistemas concorrentes quando dois ou mais processos ou threads acessam um recurso compartilhado como uma variável ou uma área de memória ao mesmo tempo sem a devida sincronização, sendo que pelo menos um deles altera esse recurso. O fator “corrida” vem da ocorrência onde o resultado da execução depende de quem chega primeiro para acessar ou modificar o recurso, ou seja, da ordem exata de execução das instruções.

Esse tipo de ação gera uma conduta imprevisível e incorreta, pois diferentes execuções do mesmo programa podem produzir resultados diferentes, conforme o momento em que cada processo executa sua parte. Esses erros costumam ser difíceis de detectar e reproduzir, pois nem sempre ocorrem da mesma forma.

Para evitar condições de corrida, é crucial utilizar mecanismos de sincronização, como semáforos, mutexes ou regiões críticas, que garantem que apenas um processo ou thread possa acessar o recurso compartilhado de cada vez.

Assim, o acesso concorrente é controlado e o comportamento do programa se torna previsível e correto.

Descreva uma situação em que pode haver uma condição de corrida.

Um programa que atualiza o saldo de uma conta bancária. Se duas threads tentam debitar o mesmo valor ao mesmo tempo, e o sistema não tem mecanismos para garantir que uma thread finalize a operação antes da outra, o saldo final pode ser incorreto. Ex. R$2.000,00 em conta e tenta realizar a compra de um celular (R$ 1500) e uma geladeira (R$ 1700) exatamente ao mesmo tempo. O saldo final será de R$ -1.200,00, já que ambos acessaram o mesmo recurso ao mesmo tempo, antes de ocorrer o débito da outra compra.

O que é uma região crítica de um processo? Por que ela precisa ser protegida de execução simultânea?

Uma região crítica de um processo, também chamada de seção crítica, é um código que precisa ser tratado de forma indivisível para garantir a consistência de dados compartilhados. Apenas um processo ou thread executa de cada vez. A região crítica é protegida por mecanismos de sincronização, como semáforos, mutexes ou locks, para garantir a exclusão mútua.

A região crítica precisa ser protegida contra execução simultânea para garantir a integridade dos dados e a correta execução de programas concorrentes. A execução simultânea pode levar a conflitos, como condições de corrida, devido a vários processos ou threads tentando acessar e modificar recursos compartilhados ao mesmo tempo. Proteger ela garante que apenas um processo por vez acesse o recurso compartilhado fazendo com que os dados permaneçam corretos.

O que é espera ocupada?

Espera ocupada (busy waiting) é uma técnica em que um processo ou thread fica em um laço contínuo, verificando repetidamente uma condição, sem liberar o processador.

Descreva as seguintes estratégias de comunicação/sincronização entre processos.

- **Monitores**

Criamos um monitor que encapsula o buffer e suas operações (inserir/retirar). O monitor garante que apenas uma operação seja executada de cada vez, evitando condições de corrida.

- **Semáforos**

Utilizamos dois semáforos, um para controlar o acesso ao buffer pelo produto (cheio/vazio) e outra para controlar o acesso pelo consumidor (cheio/vazio).

- **Mutex**

Mutex(Mutual Exclusion - Exclusão Mútua).

Um Mutex é um mecanismo de sincronização que garante que apenas uma thread/processo por vez acesse um recurso compartilhado, evitando Race Condition.

- **Test Set and Lock (variável de lock)**

Test and Set Lock (TSL), ou "Teste e Defina Bloqueio", é uma operação atômica de hardware que serve para implementar mecanismos de exclusão mútua em sistemas concorrentes, prevenindo condições de corrida e garantindo que apenas um processo ou thread tenha acesso a um recurso compartilhado por vez.