FACIMP
wyden

Processos de Aplicações Concorrentes e Condição de Corrida

Discentes:
Kauã Cantanhede dos Santos
David Marques Silva

Docente: Thalles Canela

2024

Imperatriz-Ma

Processos Concorrentes

1. Aplicações Concorrentes

Sistemas operacionais multiprogramáveis permitem a execução de múltiplos processos de forma simultânea ou pseudo-simultânea, compartilhando os recursos do sistema. Isso cria o conceito de aplicações concorrentes, onde diferentes processos podem estar em execução ao mesmo tempo, interagindo com o sistema ou entre si.

2. Compartilhamento de Recursos

Nas aplicações concorrentes, os processos frequentemente compartilham recursos do sistema (como memória, arquivos e dispositivos de E/S). Esse compartilhamento pode levar a problemas de concorrência, especialmente quando múltiplos processos tentam acessar ou modificar o mesmo recurso simultaneamente. É essencial resolver questões como:

- Troca de informações entre processos.
- Prevenção de interferência entre processos.
- Sequenciamento correto de tarefas interdependentes.

3. Mecanismos de Sincronização

Para lidar com as questões acima, os mecanismos de sincronização são cruciais. Esses mecanismos garantem que processos ou threads concorrentes possam coordenar suas execuções de forma segura, principalmente ao acessar recursos compartilhados.

Condição de Corrida

1. Definição

A condição de corrida ocorre quando dois ou mais processos acessam e manipulam dados compartilhados, e o resultado final depende da ordem de execução das operações. Isso pode levar a comportamentos imprevisíveis ou incorretos, pois a operação de um processo pode interferir na de outro.

2. Exemplo: Sistema de Verificação de Saldo Anual

Imagine que estamos simulando um sistema onde um funcionário tem um Saldo anual fixo, e a cada mês, ele faz um saque mensal para pagar suas contas. O objetivo é atualizar o saldo do salário mês a mês, aplicando o desconto e mostrando o valor final após 12 meses.

Agora, pense que cada mês é processado de maneira concorrente, ou seja, várias atualizações do salário podem ser feitas ao mesmo tempo por diferentes "processos". Se essas atualizações não forem devidamente sincronizadas, pode surgir o problema da condição de corrida.

3. **Causa**

Essa condição surge quando múltiplos processos ou threads tentam atualizar o valor do saldo ao mesmo tempo sem uma sincronização adequada. No cenário de atualização do salário com saques mensais, cada mês é processado de forma concorrente, e se duas ou mais threads tentarem aplicar o desconto simultaneamente, o valor do salário pode ser lido e modificado de maneira inconsistente.

A causa principal está na ausência de controle sobre quando e como cada thread acessa e modifica o valor compartilhado do saldo. Quando uma thread lê o valor do saldo enquanto a outra ainda está aplicando um desconto, ela pode trabalhar com um valor desatualizado, e o resultado final da operação pode ser incorreto. Sem o uso de um mecanismo de sincronização, como o lock, há uma possibilidade de que a ordem das operações de leitura e escrita varie entre execuções, levando a inconsistências no cálculo final do salário após 12 meses.

Esse problema reflete o clássico exemplo de condição de corrida, em que o recurso compartilhado (neste caso, o saldo) é acessado por processos concorrentes sem o devido controle, causando resultados imprevisíveis e com erros.

Evitando Condições de corrida: Região Crítica

1. **Exclusão Mútua**

Uma das formas principais de evitar condições de corrida é garantir a exclusão mútua, ou seja, assegurar que apenas um processo tenha acesso a um recurso compartilhado por vez. Isso pode ser feito utilizando mecanismos como semáforos ou mutexes.

2. **Região Crítica**

A região crítica é o trecho de código em que o processo acessa os recursos compartilhados. Para evitar conflitos, é necessário que apenas um processo entre na região crítica por vez, bloqueando o acesso aos outros até que o primeiro processo termine suas operações.

3. **Mecanismo de Controle**

Para controlar a entrada e saída da região crítica, usamos mecanismos de sincronização como:

- **Semáforos**: São variáveis especiais que controlam o acesso à região crítica. Eles possuem operações atômicas de incremento (up) e decremento (down) para garantir que um processo só entre na região crítica quando for seguro.
- **Mutexes**: Funcionam de forma similar aos semáforos, garantindo a exclusão mútua de processos que tentam acessar a mesma região crítica.

4. **Boas Práticas**

Além de garantir a exclusão mútua, é fundamental prevenir situações como starvation, onde um processo fica permanentemente bloqueado enquanto outros continuam a acessar os recursos. Outra prática importante é evitar interrupções dentro da região crítica, garantindo que o processo finalize suas operações sem ser interrompido.

Exemplos de Condição de Corridas em Sistemas Reais

1. **Aplicações Bancárias**

Como mencionado no exemplo anterior, em um sistema bancário, dois caixas ou terminais podem tentar acessar e atualizar o saldo de uma conta ao mesmo tempo, levando a inconsistências se não houver sincronização

2. **Sistemas de Reserva de Voos**

Em sistemas de reserva de voos, dois agentes ou usuários podem tentar reservar o mesmo assento em um voo simultaneamente. Sem mecanismos de controle de concorrência, ambos podem ser capazes de reservar o assento, resultando em um overbooking.

3. **Sistemas de Compartilhamento de Arquivos**

No compartilhamento de arquivos, múltiplos processos podem tentar acessar o mesmo arquivo para leitura e escrita ao mesmo tempo. Se o arquivo for modificado sem controle adequado, pode haver perda de dados ou corrupção de informações.

4. **Sistemas Operacionais**

Em sistemas como Linux, Windows ou iOS, diversos processos precisam acessar e modificar os recursos do sistema, como memória compartilhada ou

arquivos de sistema. Se esses processos não forem devidamente sincronizados, podem ocorrer conflitos que afetam a estabilidade e a segurança do sistema.

Semáforos e Sincronização

1. **Funcionamento dos Semáforos**

Os semáforos são um dos mecanismos mais comuns para garantir a exclusão mútua e controlar o acesso aos recursos compartilhados. Eles operam com duas funções principais:

- **Down (P):** Decrementa o valor do semáforo. Se o valor atingir zero, o processo é bloqueado até que outro processo libere o recurso.
- **Up (V):** Incrementa o valor do semáforo, liberando o recurso e permitindo que outro processo acesse a região crítica.

2. **Exclusão Mútua com Semáforos**

Os semáforos garantem que apenas um processo possa acessar um recurso compartilhado de cada vez, evitando conflitos e mantendo a consistência dos dados.

Conclusão

A concorrência em sistemas operacionais é essencial para o desempenho e a eficiência, mas também traz desafios em termos de sincronização e controle de acesso a recursos compartilhados. A condição de corrida é um problema clássico que pode causar inconsistências em sistemas bancários, de reserva de voos e outros sistemas críticos. Para resolver esses problemas, mecanismos como semáforos e mutexes garantem que os processos concorrentes possam trabalhar de forma coordenada, prevenindo conflitos e mantendo a integridade dos dados.

Referências

Communication of the ACM. *Race Conditions: A Case Study*. Communications of the ACM, Vol. 53, No. 6, 2010.

Journal of Computer Science and Engineering. *Concurrency Control and Race Conditions*. Journal of Computer Science and Engineering, 2012.

IEEE Xplore Digital Library. *Race Conditions and Deadlocks in Multi-threaded Applications*. IEEE Xplore, 2010.

ACM Transactions on Software Engineering and Methodology (TOSEM). *Improving Software Reliability through Race Condition Detection*. ACM Transactions on Software Engineering and Methodology, 2015.