INSTITUTO DE INFORMÁTICA
Universidade Federal de Goiás

Aula 09 - Deadlocks

Marcus Mitra Muniz Inacio marcus.mitra@discente.ufg.br
Samuel Santos Machado 201705643
Vinicius de Freitas Castro vinicius.castro@discente.ufg.br
Matheus Pamplona Oliveira matheus.pamplona@discente.ufg.br

2023

INSTITUTO DE INFORMÁTICA UFG

UFG UNIVERSIDADE FEDERAL DE GOIÁS

PROCESSOS DE SINCRONIZAÇÃO

Race Condition

- **Definição:**
  - A condição de corrida ocorre quando vários processos acessam dados compartilhados simultaneamente, resultando em possíveis corrupções de dados.

- **Relevância:**
  - Destaca a necessidade de sincronização para garantir a integridade dos dados em ambientes de programação concorrente.

Critical Section

- **Definição:**
  - A seção crítica é uma parte do código onde processos manipulam dados compartilhados, potencialmente enfrentando condições de corrida.

- **Relevância:**
  - Identifica áreas críticas que requerem exclusão mútua para evitar conflitos e garantir consistência nos dados.

The Critical-Section Problem

- **Objetivo:**
  - Projetar um protocolo que permite que processos cooperem na manipulação de dados compartilhados, garantindo (1) exclusão mútua, (2) progresso e (3) espera limitada.

- **Desafios:**
  - Enfrenta desafios como deadlocks e inversão de prioridade.

- **Solução de Peterson:**
  - a solução de Peterson fornece uma maneira de coordenar o acesso de dois processos a uma seção crítica de código, garantindo que apenas um deles acesse essa seção por vez. Isso é particularmente relevante em sistemas operacionais e ambientes concorrentes, onde múltiplos processos estão em execução simultaneamente e podem competir por recursos compartilhados.

Mutex Locks

- **Funcionamento:**
  - Garante exclusão mútua, exigindo que um processo adquira uma trava antes de entrar em uma seção crítica e a libere ao sair.

- **Aplicações:**
  - Ampla aplicação para controlar o acesso a recursos críticos.

Semaphores

- **Definição:**
  - Semáforos possuem um valor inteiro e são utilizados para fornecer exclusão mútua e resolver diversos problemas de sincronização.

- **Versatilidade:**
  - Permitem a implementação de soluções flexíveis para diferentes cenários de concorrência.

Problemas de Vitalidade

- **Desafios Adicionais:**
  - Além dos desafios básicos da seção crítica, soluções podem enfrentar problemas de vitalidade, como deadlocks e inversão de prioridade.

- **Considerações Finais:**
  - Destaca a importância de abordagens robustas na sincronização de processos para evitar situações indesejadas.

EXEMPLO DE SEMAFORO EM CÓDIGO

Code of P_1

```
S1;
signal(mutex);
```

Code of P_2

```
wait(mutex);
S2;
```

Deadlocks

Deadlocks

- **Definição:**
  - Um impasse (deadlock) é uma situação em que um conjunto de processos concorrentes é incapaz de concluir suas tarefas devido a bloqueios mútuos, onde cada processo aguarda recursos que estão sendo retidos por outros.

- **Características:**
  - Os processos envolvidos estão em um estado de espera indefinida, impedindo a conclusão bem-sucedida de suas operações.

Prevenção de Deadlocks

- **Exclusão Mútua:**
  - Garantir que apenas um processo tenha acesso exclusivo a um recurso em um determinado momento, reduzindo as chances de bloqueios.

- **Posse e Espera (Hold and Wait):**
  - Um processo deve solicitar todos os recursos necessários de uma só vez e só pode começar a execução após obter todos os recursos necessários.

- **Não Preempção:**
  - Recursos não podem ser retirados à força de um processo. Isso ajuda a evitar a interrupção de processos em execução.

Prevenção de Deadlocks

- **Espera Circular:**
  - Impor uma ordem total de recursos e garantir que os processos solicitem recursos seguindo essa ordem, eliminando assim a possibilidade de espera circular.

- **Detecção e Recuperação:**
  - Implementar algoritmos de detecção de deadlock para identificar situações de impasse. Após a detecção, a recuperação pode envolver a liberação de recursos ou até mesmo a interrupção de alguns processos.

- **Evitar Deadlocks por Construção (Banker's Algorithm):**
  - Algoritmo que avalia a segurança de alocação de recursos antes de conceder solicitações, prevenindo deadlock por meio de uma abordagem proativa.

Gráfico de alocação de recursos

Os Deadlocks podem ser mais compreendido em um gráfico direcionado chamado gráfico de alocação de recursos do sistema. Neste gráfico consiste em um conjunto de vértices V e um conjunto de arestas E.

Aonde V pode ter dois tipos de vértices::

P **= {P1, P2,…, Pn}, o conjunto que consiste todos os processos ativos no sistema.**

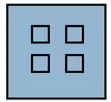
R = {R1, R2,…, Rm}, o conjunto que consiste todos os tipos de recursos no sistema.

- Aresta de solicitação – uma aresta de um processo para um recurso Pi→Rj
- Aresta de atribuição – um vértice de um recurso para um processo Rj → Pi

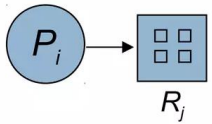
Gráfico de alocação de recursos

- Processo

- Tipo de recurso com 4 instâncias



- Pi requisita uma instância de



- Pi está em posse de uma instância

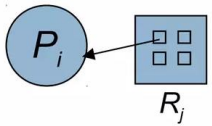
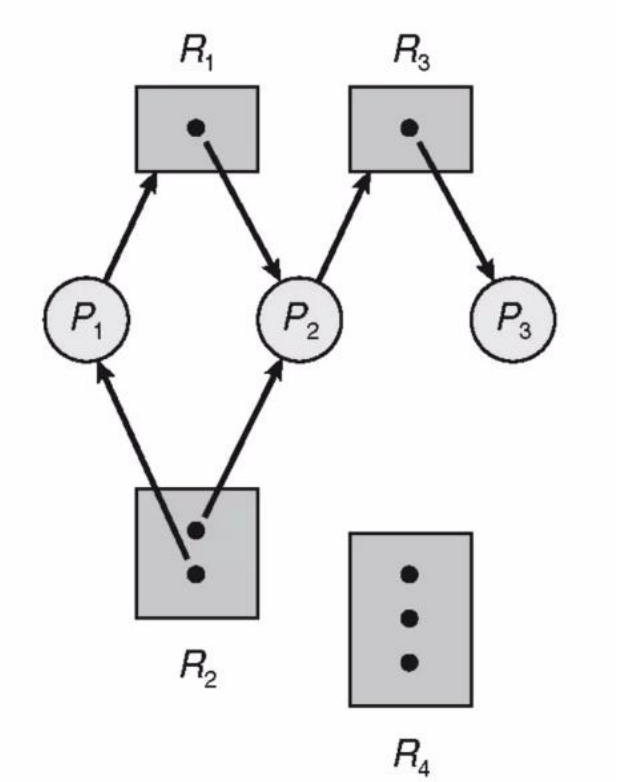


Gráfico de alocação de recursos



Neste Cenário:

Os conjuntos P, R, e E:
- P = {P1 , P2 , P3 }
- R = {R1 , R2 , R3 , R4 }
- E = {P1 → R1 , P2 → R3 , R1 → P2 , R2 → P2 , R2 → P1 , R3 → P3}

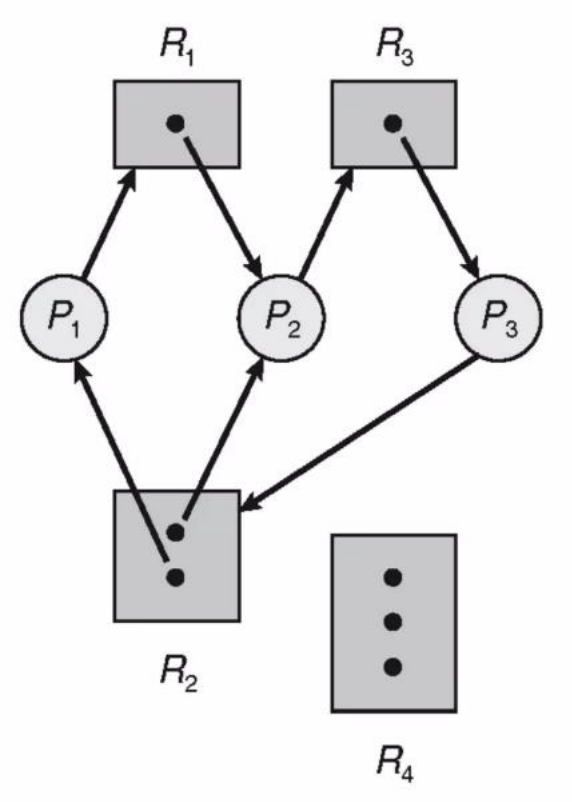
Instâncias de recursos:
- Uma instância do tipo de recurso R1
- Duas instâncias do tipo de recurso R2
- Uma instância do tipo de recurso R3
- Três instâncias do tipo de recurso R4

Estados dos processos:
- Processo P1: está mantendo uma instância do tipo de recurso R2 e está aguardando uma instância do tipo de recurso R1.

- Processo P2: está mantendo uma instância de R1 e um instância de R2 e está aguardando uma instância de R3.

- Processo P3: está segurando uma instância de R3

Gráfico de alocação de recursos



Neste Cenário:

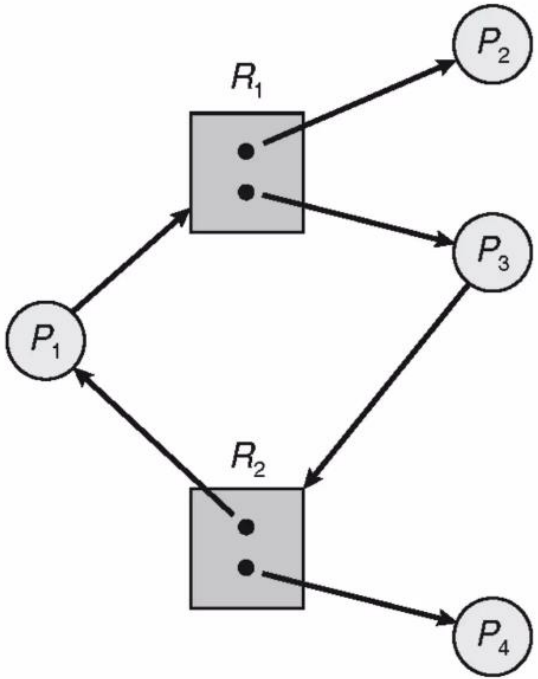
• Nessa figura o processo P3 solicita uma instância do tipo de recurso R2.
• Como nenhuma instância de recurso está disponível, ele adiciona uma aresta de solicitação P3 para R2.

Existindo dois ciclos mínimos na figura aonde:
1. P1 → R1 → P2 → R3 → P3 → R2 → P1
2. P2 → R3 → P3 → R2 → P2

Os processos P1,P2 ,P3 estão em um impasse(deadlock), aonde:

• O processo P2 está aguardando o recurso R3, que está esperando pelo recurso P3.
• O processo P3 está aguardando o processo P1 ou o P2 para liberar o recurso do R2.
• E o processo P1 está aguardando o processo P2 liberar recurso do R1.

Gráfico de alocação de recursos (Sem Deadlock)



Neste Cenário:

- Temos um ciclo, porém não há impasse(deadlock).
- O Processo P4 pode liberar sua instância do tipo de recurso do R2. Esse recurso pode então ser alocado para P3, quebrando o ciclo.

Entao, podemos concluir que se o gráfico não contém ciclos então ele não possui deadlock.

E se o gráfico ter um ciclo e houver apenas uma instância por tipo de recurso, haverá conflito(deadlock) e caso haja várias instâncias por tipo de recurso, possibilidade de impasse(deadlock).

Metodologias para lidar com um deadlock

- Não lidar com o deadlock e esperar o limite de tempo de execução do processo
  Mais barato e mais comum , usado por sistemas como linux , UNIX e windows
- Usar um protocolo de preempção para evitar que as condições para um deadlock ocorre ou ordenar os recursos necessarios e aloca-los para os diferentes processos
  Requer gastos , mas e eficiente
- Usar um algoritmo de detecção e recuperação de deadlock
  Dificil implementaçao , visto em casos especificos , como databases

Obrigado


INSTITUTO DE
INFORMÁTICA
UFG


UFG
UNIVERSIDADE
FEDERAL DE GOIÁS