Relatório Técnico do Trabalho II da Unidade II de S.O.

Autores: Jônatas Câmara dos Santos & Pedro Henrique Ribeiro Alves
Data: 09/08/2024

Resumo

Este artigo explora a implementação do Algoritmo do Banqueiro, uma técnica clássica para a prevenção de impasses em sistemas operacionais. Desenvolvido por Edsger Dijkstra, o Algoritmo do Banqueiro verifica sistematicamente se a concessão de recursos a um processo pode levar a um estado inseguro e, consequentemente, a um impasse. Este estudo detalha a estrutura e funcionamento do algoritmo, discutindo sua aplicação. Além disso, são apresentadas explicações sobre como os semáforos foram definidos.

Palavras-chave: Semáforo, Função, Cliente.

Introdução:

Em sistemas operacionais, a gestão eficaz de recursos é crucial para garantir que múltiplos processos possam ser executados simultaneamente sem conflitos. Um dos desafios mais significativos nesse contexto é a prevenção de Deadlocks, situações em que um conjunto de processos ficam permanentemente bloqueados, esperando por recursos que nunca se tornam disponíveis. Para resolver esse problema, Edsger Dijkstra propôs o Algoritmo do Banqueiro, uma abordagem preventiva que visa assegurar que o sistema sempre permaneça no estado seguro.

O Algoritmo do Banqueiro funciona de maneira análoga à política adotada por um banqueiro ao conceder empréstimos. Antes de alocar recursos a um processo, o algoritmo verifica se a concessão levará o sistema a um estado inseguro, onde a possibilidade de impasse é alta. Para isso, ele utiliza informações sobre os recursos disponíveis, os recursos atualmente alocados a cada processo, e os recursos máximos que cada processo pode requisitar no futuro.

Fundamentação Teórica:

Conceitos Básicos:

1. Estado Seguro e Inseguro

- **Estado Seguro:** Um sistema está em estado seguro se existe uma sequência de processos tal que cada processo pode obter seus recursos máximos necessários, executar e liberar os recursos, permitindo que outros processos façam o mesmo. Em um estado seguro, não há possibilidade de Deadlock.

- **Estado Inseguro:** Um estado é inseguro quando não há garantias de que todos os processos possam concluir sua execução, o que pode levar a um Deadlock.

2. Recursos e Alocação

O algoritmo considera três tipos de informações:

- **Recursos Disponíveis (Disp):** Número de instâncias de cada tipo de recurso que atualmente não estão alocadas a nenhum processo.
- **Recursos Máximos (Max):** Número máximo de instâncias de cada tipo de recurso que cada processo pode requisitar.
- **Recursos Alocados (Aloc):** Número de instâncias de cada tipo de recurso atualmente alocadas a cada processo.
- **Necessidade de Recursos (Ne):** Recursos adicionais que cada processo ainda pode requisitar para completar sua execução, calculado como **Ne = Max - Aloc**.

Funcionamento do Algoritmo do Banqueiro

OBS*:* Devido a problemas desconhecidos o algoritmo funciona apenas com auxílio do programa Valgrind.

1. **Main**:
    - Lê o número de clientes e recursos da linha de comando.
    - Inicializa os recursos disponíveis e os clientes.
    - Cria threads para cada cliente.
    - Aguarda a finalização das threads e libera a memória alocada.
2. **Inicia_recursos**:
    - Aloca memória para o vetor de recursos disponíveis.
    - Inicializa esses recursos com valores aleatórios.
3. **Inicia_clientes**:
    - Aloca memória para as matrizes Max, Aloc e Ne.
    - Inicializa essas matrizes com valores aleatórios.
    - Cria uma thread para cada cliente.
4. **Cliente**:
    - Função executada por cada thread de cliente.
    - Gera requisições de recursos aleatórias até que todas as necessidades sejam atendidas.
    - Envia a requisição para o banco (sistema de alocação de recursos).
    - Se a requisição é aceita, verifica se o cliente finalizou. Se sim, libera os recursos alocados.
5. **Gera_requisicao**:

- Gera uma requisição aleatória de recursos para um cliente específico.

6. **Requisicao**:

- Verifica se a requisição de um cliente pode ser atendida.
- Simula a alocação dos recursos e verifica se o sistema permanece em um estado seguro usando a função “seguranca”.
- Se a requisição for válida, atualiza os recursos alocados e disponíveis; caso contrário, reverte a simulação.

7. **Seguranca**:

- Implementa o algoritmo do banqueiro para verificar se o sistema está em um estado seguro após a simulação de uma requisição.
- Verifica se todos os clientes podem terminar com os recursos disponíveis.

8. **Finaliza_cliente**:

- Verifica se um cliente terminou suas necessidades de recursos.
- Se sim, libera os recursos alocados por esse cliente.

9. **Gera_rand**:

- Gera um número aleatório dentro de um limite especificado.

10. **Print_banco**:

- Imprime o estado atual dos recursos disponíveis, máximos, alocados e necessidades de cada cliente.

11. **Semáforo**:

- O semáforo “mutex” é usado para garantir que as operações de alocação e liberação de recursos sejam realizadas de maneira organizada, impedindo condições de corrida entre as threads.

Metodologia

Para a construção e análise do Algoritmo do Banqueiro e da aviação de seus resultados, seguimos os seguintes passos metodológicos:

1. Configuração do Ambiente de Teste

- **Ferramentas e Tecnologias:** Utilizamos a linguagem de programação C para implementar o algoritmo. Além disso, semáforos foram utilizados para sincronizar as operações de alocação e liberação de recursos.

2. Implementação do Algoritmo

- **Leitura de Dados:** Desenvolvemos um módulo para ler o número de clientes e recursos a partir da linha de comando, inicializando os recursos disponíveis e os clientes.
- **Inicialização de Recursos:** Alocamos memória para os vetores de recursos disponíveis, máximos, alocados e necessários, inicializando-os com valores aleatórios para simular diferentes cenários de carga de trabalho.
- **Criação de Threads:** Implementamos a criação de threads para cada cliente, onde cada thread gera requisições de recursos até que todas as necessidades sejam atendidas.
- **Função Cliente:** Cada thread de cliente executa uma função que gera requisições de recursos aleatórias, enviando-as ao sistema de alocação de recursos e verificando se a requisição pode ser atendida sem comprometer a segurança do sistema.

3. Simulação e Verificação

- **Geração de Requisições:** Desenvolvemos uma função para gerar requisições aleatórias de recursos para cada cliente, simulando diferentes padrões de demanda.
- **Simulação de Alocação:** Implementamos uma função que simula a alocação dos recursos e verifica, através do Algoritmo do Banqueiro, se o sistema permanece em um estado seguro após cada requisição.
- **Verificação de Segurança:** A função de verificação de segurança foi implementada para garantir que, após cada alocação simulada, todos os clientes possam terminar com os recursos disponíveis.

4. Coleta e Análise de Dados

- **Coleta de Dados:** Durante a execução das simulações, coletamos dados sobre o tempo de execução, número de requisições atendidas, número de rejeições de requisições e uso de CPU.
- **Análise de Desempenho:** Analisamos os dados coletados para avaliar a eficácia do Algoritmo do Banqueiro na prevenção de impasses e seu impacto no desempenho do sistema. Utilizamos métricas como tempo de espera médio, tempo de retorno médio e utilização do processador para comparar diferentes cenários de carga de trabalho.

5. Validação dos Resultados

- **Repetição de Experimentos:** Realizamos múltiplas execuções dos experimentos para assegurar a consistência dos resultados e minimizar o impacto de variáveis aleatórias.

Resultados

Os resultados mostram que, embora o Algoritmo do Banqueiro seja eficaz na prevenção de Deadlocks, ele pode causar sobrecarga significativa em sistemas com alta variação

de requisições de recursos. A análise conclui que a eficiência do algoritmo depende crucialmente do balanceamento entre a necessidade de segurança e o custo computacional, sugerindo que a sua implementação deve ser cuidadosamente considerada em sistemas onde o desempenho é um fator crítico.

Conclusão

O Algoritmo do Banqueiro se provou ser eficaz na prevenção de impasses em sistemas operacionais ao garantir estados seguros de alocação de recursos. A análise detalhada e a implementação prática mostraram que, embora robusto, o algoritmo pode introduzir uma sobrecarga significativa, especialmente em sistemas com alta variação de requisições de recursos.

Portanto, sua aplicabilidade prática deve ser cuidadosamente considerada em sistemas onde o desempenho é crítico. A eficácia do algoritmo depende de um equilíbrio entre a prevenção de impasses e o custo computacional, sugerindo que otimizações futuras são necessárias para aumentar sua eficiência em ambientes de alta demanda.

Referências

1. Aulas e arquivos de estudos (PDF) de autoria do Doutro João Batista Borges Neto
2. Páginas da internet como StackOverflow para maior compreensão da formação do algoritmo.