Universidade Federal de Santa Catarina

Centro de Engenharias da Mobilidade
Curso de Engenharia Mecatrônica

Impasses

Disciplina: Sistemas Operacionais

Aluno: Maykon Hopka

Joinville – 2025

1 Exercícios

Exercício 1

A alteração em qual fator aumentou a quantidade de travamentos no programa

Solução 1.1: Exercício 1

Aumentar o numero de threads e transferências

Exercício 2

Faça a modelagem do grafo de alocação de recursos para explicar a razão do programa travar

Solução 1.2: Exercício 2

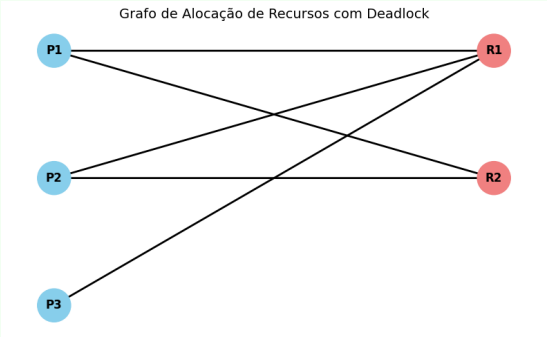

Figura 1: Grafo de alocação de recursos mostrando deadlock

- No sistema, temos três processos (P1, P2 e P3) e dois recursos compartilhados (R1 e R2).
- O processo **P1** está de posse do recurso **R1** e solicita **R2**.
- O processo **P2** está de posse do recurso **R2** e solicita **R1**.
- O processo **P3** solicita **R1**, mas não consegue obtê-lo, pois ele está alocado para P1.
- Neste cenário, há um ciclo de espera entre P1 e P2:
  - P1 → R2 (espera por R2, que está com P2)
  - P2 → R1 (espera por R1, que está com P1)
- Esse ciclo caracteriza um **deadlock**, pois os processos ficam esperando indefinidamente pela liberação de recursos uns dos outros.
- O processo P3 está bloqueado, mas não causa o deadlock diretamente — ele apenas aguarda a liberação de um recurso envolvido no ciclo.

Resumo: O programa pode travar quando múltiplas threads tentam adquirir os mesmos semáforos em ordens diferentes. Se a thread A segura o semáforo da conta 1 e tenta acessar a conta 2, enquanto a thread B segura o semáforo da conta 2 e tenta acessar a conta 1, nenhuma das duas prossegue — esse é o cenário modelado pelo grafo de alocação de recursos com deadlock.

Exercício 3

Pergunte à sua IA favorita (GPT, Gemini, etc) para encontrar uma solução e teste se a solução proposta realmente funciona.

Solução 1.3: Exercício 3

Para evitar esse problema, a IA sugeriu u: **sempre adquirir os semáforos na mesma ordem**, independentemente de qual conta é origem ou destino. A ordem foi definida com base no número da conta (ID). Assim, elimina-se a possibilidade de deadlock.

```
// Transferência com ordenação de semáforos para evitar deadlock
void transferir (conta_t *contaDeb, conta_t *contaCred, float valor){
    conta_t *primeira, *segunda;

    // Ordena os locks para sempre adquirir na mesma ordem
    if (contaDeb->numero < contaCred->numero) {
        primeira = contaDeb;
        segunda = contaCred;
    } else {
        primeira = contaCred;
        segunda = contaDeb;
    }

    sem_wait(&(primeira->trava));
    sem_wait(&(segunda->trava));

    if(contaDeb->saldo >= valor){
        contaDeb->saldo -= valor;
        contaCred->saldo += valor;
    }

    sem_post(&(segunda->trava));
    sem_post(&(primeira->trava));
}

// Correção na função da thread
void *auxThread(void *arg){
    int origem,destino;
    long n_trans=(long)arg;
    float valor;
    for (int i=0;i<n_trans;i++){
        origem = random() % n_contas;
        destino = random() % n_contas;
        while(destino == origem){
            destino = random() % n_contas;
        }
        valor = 1.0 * (random() % 10);
        transferir(&contas[origem], &contas[destino], valor);
    }
    pthread_exit(NULL); // Encerra a thread corretamente
}
```

Resultado: Com essas modificações, o programa do Gepeto(ChatGPT) pareceu funcionar nos meus testes.