
Universidade Federal de Lavras
PPGCC
PCC508 – Sistemas Operacionais

# Tópico 7 Lista Avaliativa

Douglas Aquino T. Mendes
14 de janeiro de 2025

# Sumário

**1 Introdução** **2**

**2 Questões** **2**

2.1 Questão 1 . . . . . 2

2.2 Questão 2 . . . . . 4

2.3 Questão 3 . . . . . 4

2.4 Questão 4 . . . . . 4

# 1 Introdução

Este documento tem como objetivo apresentar o desenvolvimento das atividades avaliativas para o tópico 7 da disciplina de Sistemas Operacionais, focando na implementação de códigos em linguagem C. Serão apresentadas as questões, a resolução, os códigos desenvolvidos, seguido da apresentação dos resultados da execução do código.

# 2 Questões

## 2.1 Questão 1

**Pergunta:** 1) Considere um algoritmo de detecção de deadlock quando existe mais de um recurso. Encontramos a seguinte situação:
Recursos existentes E = (4 2 3 1)
Recursos Disponíveis A = (2 1 0 0)

Matriz de alocação:

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{bmatrix}$$

Matriz de requisição:

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 2 & 1 & 0 & 0 \end{bmatrix}$$

**Resposta:** Vamos aplicar o algoritmo de detecção de deadlocks com múltiplos recursos de cada tipo para verificar se o sistema se encontra em deadlock. Esse algoritmo se baseia na comparação de vetores para encontrar um processo que possa ser executado até o fim [1, p. 309]. Os dados iniciais estão visíveis acima, com isso, o primeiro passo é verificar processos que podem ser atendidos:

Processo $P_1$ :

- Requisição R[1] = (2,0,0,1)
- Recursos Disponíveis A = (2,1,0,0)
- Verificação: $R[1] \leq A$
- $(2, 0, 0, 1) \leq (2, 1, 0, 0)$
- Falso $(R[1][3] = 1 > A[3] = 0$
- **P processo $P_1$ não pode ser atendido**

Processo $P_2$ :

- Requisição R[2] = (0,0,1,1)
- Recursos Disponíveis A = (2,1,0,0)
- Verificação: $R[2] \leq A$
- $(0, 0, 1, 1) \leq (2, 1, 0, 0)$
- Falso $(R[2][2] = 1 > A[2] = 0$

- **P processo $P_2$ não pode ser atendido**

Processo $P_3$ :

- Requisição R[3] = (2,1,0,0)
- Recursos Disponíveis A = (2,1,0,0)
- Verificação: $R[2] \leq A$
- $(2, 1, 0, 0) \leq (2, 1, 0, 0)$
- Verdadeiro
- **P processo $P_3$ pode ser atendido**
- Atualiza os recursos disponíveis: A = A + C[3]
- A = (2,1,0,0) + (0,1,2,0) = (2,2,2,0)
- Marcamos $P_3$ como concluído: Finish = (false,false,true)

**Passo 2:** Verificar novamente os processos pendentes

Processo $P_1$ :

- Requisição R[1] = (2,0,0,1)
- Recursos Disponíveis A = (2,2,2,0)
- Verificação: $R[1] \leq A$
- $(2, 0, 0, 1) \leq (2, 2, 2, 0)$
- Falso $(R[1][3] = 1 > A[3] = 0$
- **P processo $P_1$ não pode ser atendido**

Processo $P_2$ :

- Requisição R[2] = (0,0,1,1)
- Recursos Disponíveis A = (2,2,2,0)
- Verificação: $R[2] \leq A$
- $(0, 0, 1, 1) \leq (2, 2, 2, 0)$
- Falso $(R[2][3] = 1 > A[3] = 0$
- **P processo $P_2$ não pode ser atendido**

Após analisar todos os processos, os processos $P_1 e P_2$ não podem ser atendidos, mesmo após $P_3$ ser concluído. Isso indica que os processos $P_1 e P_2$ estão em deadlock.

## 2.2 Questão 2

**Pergunta:** 2) Descreva quais são as maneiras de se recuperar de um deadlock, e se elas são viáveis na prática.

**Resposta:** Existem três maneiras principais de recuperar um sistema de um deadlock (impasse).

**Recuperação por preempção:** Um recurso pode ser retirado do seu atual proprietário e realocado para outro processo. Essa abordagem pode ser eficaz para recursos como impressoras, mas pode ser difícil ou impossível para recursos como arquivos ou registros de banco de dados [1, p. 310].

**Recuperação por reversão:** Um processo que possui um recurso necessário para romper o impasse pode ser revertido para um estado anterior, antes de ter adquirido o recurso. O estado do processo pode ser salvo periodicamente em arquivos de *checkpoint*, e o processo pode ser reiniciado a partir de um desses *checkpoints*. Essa abordagem pode ser eficaz, mas envolve a perda de trabalho realizado pelo processo desde o último checkpoint [1, p. 310].

**Recuperação por eliminação de processos:** Um ou mais processos envolvidos no impasse podem ser eliminados. Essa é a abordagem mais simples, mas também a mais drástica, pois pode resultar na perda de dados ou no término prematuro de processos importantes [1, p. 310].

Na prática, a viabilidade dessas abordagens depende do sistema e da natureza do impasse. Em alguns casos, uma combinação dessas técnicas pode ser usada. Por exemplo, um sistema pode tentar primeiro recuperar por preempção de recursos não críticos. Se isso não funcionar, pode tentar reverter processos para checkpoints. Se tudo mais falhar, pode ser necessário eliminar processos. É importante observar que nenhuma dessas abordagens é ideal e todas elas envolvem compensações. A melhor abordagem para lidar com impasses é evitá-los em primeiro lugar. Isso pode ser feito por meio de técnicas como ordenação de recursos, alocação cuidadosa de recursos ou o uso de algoritmos como o algoritmo do banqueiro [1, p. 316]. No entanto, mesmo com essas técnicas, os impasses ainda podem ocorrer e é importante ter um plano para lidar com eles quando eles surgirem.

## 2.3 Questão 3

**Pergunta:** Suponha que existam 3 processos e um tipo de recurso somente. No sistema, existem 4 instâncias do recurso. Cada processo precisa no máximo de 2 instâncias do mesmo. Existe alguma circunstância que poderia gerar deadlock, neste caso? Explique.

**Resposta:** Como cada processo precisa no máximo de 2 instâncias, e existem 4 instâncias disponíveis, sempre haverá a possibilidade de um processo liberar recursos suficientes para que outro processo prossiga. Portanto, nunca haverá uma espera circular, e um deadlock não poderá ocorrer.

## 2.4 Questão 4

**Pergunta:** 4) Explique brevemente o algoritmo do banqueiro para um tipo de recurso.

**Resposta:** O algoritmo do banqueiro é uma técnica para evitar deadlocks, modelando a alocação de recursos como um banco que concede crédito a um grupo de clientes (processos). O "banco"(sistema operacional) possui um número limitado de unidades de um recurso. Cada cliente (processo) deve declarar antecipadamente o número máximo de unidades do recurso que ele precisará [1, p. 313]. O algoritmo funciona da seguinte forma:

- Quando um cliente solicita unidades do recurso, o banqueiro verifica se a concessão da solicitação levaria a um estado inseguro. Um estado é considerado seguro se existir uma sequência de alocações e liberações de recursos que permita a todos os clientes obterem o número máximo de unidades que eles solicitaram. Um estado é inseguro se não existir tal sequência, o que pode levar a um impasse [1, p. 312].
- Se conceder a solicitação levar a um estado seguro, o banqueiro a concede. Caso contrário, o banqueiro adia a solicitação.

O objetivo do algoritmo é garantir que o sistema sempre permaneça em um estado seguro, evitando a ocorrência de deadlocks.

# Referências

[1] A. S. Tanenbaum e H. Bos, *SISTEMAS OPERACIONAIS MODERNOS*, 4ª ed. Boston: Pearson, 2021.