Italo Henrique Lins Carneiro – 22350066

João Caio Dutra Gontijo - 22352975

UNICEUB

CIÊNCIA DA COMPUTAÇÃO

# Projeto Prático
# Roller Coster

Brasília - DF

2025

**SUMÁRIO**

## 1. Introdução:

O problema da montanha-russa representa um desafio clássico de sincronização e concorrência em sistemas operacionais, onde múltiplos passageiros competem por assentos limitados em carros que só iniciam seu percurso quando completamente cheios. Este projeto tem como objetivo desenvolver uma simulação que gerencie de forma eficiente a interação entre passageiros e carros, garantindo que todas as operações ocorram sem condições de corrida, deadlocks ou starvation, utilizando semáforos como mecanismo principal de coordenação, o principal objetivo é criar um sistema que simule esse cenário de maneira realista, utilizando semáforos para controlar o embarque, desembarque e movimentação dos carros, enquanto coleta informações como tempo médio de espera e eficiência, este projeto não apenas mostra conceitos fundamentais de sistemas operacionais, como concorrência e exclusão mútua, mas também oferece visões para problemas do mundo real que envolvem alocação de recursos e coordenação de processos. A implementação com sucesso demonstra a importância de técnicas de sincronização na construção de sistemas grandes e eficientes, onde múltiplas entidades compartilham recursos limitados de forma ordenada e segura.

## 2. Definição do Problema:

O problema da montanha-russa consiste em simular a concorrência e sincronização de processos em um parque de diversões, onde passageiros chegam aleatoriamente para embarcar em carros de uma montanha-russa. Os carros possuem capacidade fixa e só iniciam o passeio quando estão completamente ocupados. O desafio ao todo é gerenciar a fila de passageiros, o embarque/desembarque e o movimento dos carros, evitando condições de corrida, deadlocks e starvation, utilizando técnicas de sincronização como semáforos. Além disso, a solução deve ser escalável, funcionando corretamente tanto para um único carro quanto para múltiplos carros, sempre respeitando as restrições do problema.

Parâmetros do Problema:

- n: Número total de passageiros.
- m: Número total de carros disponíveis.
- C: Capacidade de cada carro.
- T m: Tempo de passeio.
- T e: Tempo de embarque/desembarque.
- T p: Intervalo de chegada dos passageiros (aleatório entre 10 e 20 segundos).

## 3. Algoritmos Propostos:

Para resolver o problema, foi desenvolvido um programa em Python que utiliza semáforos para gerenciar a sincronização entre passageiros e carros. A solução aborda os três cenários propostos: carro único, dois carros e N carros.

Estruturas de Sincronização:

Semáforo embarque: Controla o acesso dos passageiros ao carro durante o embarque.

Semáforo desembarque: Controla o desembarque dos passageiros após o passeio.

Semáforo mutex: Garante exclusão mútua para evitar condições de corrida ao acessar variáveis compartilhadas.

Semáforo carro_cheio: Indica quando o carro está cheio e pronto para partir.

Fluxo do Programa:

Passageiros:

Chegam aleatoriamente (intervalo de 10 a 20 segundos).

Aguardam na fila até que um carro esteja disponível para embarque.

Embarcam no carro quando há assentos livres.

Aguardam o término do passeio e desembarcam.

Carros:

Aguardam até que todos os assentos estejam ocupados.

Iniciam o passeio quando cheios.

Retornam após o tempo de passeio (T m) e iniciam o desembarque.

Liberam os passageiros e ficam disponíveis para novo embarque.

## 4. Manual de Utilização:

Este código foi projetado para ser modular. A lógica principal da simulação está encapsulada, e os cenários de teste são executados a partir de um script separado.

Estrutura dos Arquivos:
simulacao_montanha_russa.py: Contém o código principal com as funções passageiro, carro, e executar_simulacao. Este arquivo funciona como um módulo ou biblioteca.
problema1.py / problema2.py / problema3.py: Scripts executáveis que importam a função executar_simulacao e a chamam com um conjunto específico de parâmetros.

Como Executar a Simulação:
Salve o código principal em um arquivo chamado simulacao_montanha_russa.py.
Salve o código de um dos problemas (ex: "Problema 1") em um arquivo separado na mesma pasta (ex: iniciar_simulacao.py).
Abra um terminal ou prompt de comando, navegue até a pasta onde salvou os arquivos.
Execute o script com o comando: python iniciar_simulacao.py.
A simulação iniciará e imprimirá o log de eventos no console, seguido por um relatório final de estatísticas.

Parâmetros da Simulação:
Para configurar um novo cenário, modifique os parâmetros passados para a função executar_simulacao:

n (int): O número total de passageiros (N_PASSAGEIROS) a serem simulados.
m (int): O número total de carros (N_CARROS) na atração.
C (int): A capacidade de assentos de cada carro (CAPACIDADE_CARRO).
Tm (float): A duração, em segundos, do passeio em si (TEMPO_PASSEIO).
Te (float): O tempo, em segundos, necessário para o embarque e também para o desembarque (TEMPO_EMBARQUE_DESEMBARQUE).
Tp_min (float): O tempo mínimo do intervalo de chegada entre passageiros (INTERVALO_CHEGADA_MIN).
Tp_max (float): O tempo máximo do intervalo de chegada entre passageiros (INTERVALO_CHEGADA_MAX).

## 5. Resultados e Análises:

Saída do Programa:

O programa gera uma saída conforme o solicitado, registrando os tempos de cada evento e calculando métricas como tempo mínimo, máximo e médio de espera na fila, além da eficiência dos carros.

Métricas de Desempenho:

Tempo de Espera: O tempo de espera dos passageiros varia conforme a aleatoriedade da chegada e a capacidade do carro.

Eficiência: A eficiência do carro é calculada como a razão entre o tempo em movimento e o tempo total (incluindo embarque/desembarque).

Conclusão:

A solução proposta garante a sincronização entre passageiros e carros, evitando deadlocks e starvation. O uso de semáforos mostrou-se eficaz para gerenciar a concorrência e assegurar o funcionamento correto do sistema. A implementação pode ser adaptada para os três cenários propostos, demonstrando flexibilidade e robustez.