Pedro Henrique Franco Moreira Amaral e Rafael de Souza Felisberto

Nome da Instituição

Curso: Sistemas de Informação

Disciplina: Sistemas Operacionais

**SIMULAÇÃO DE CORRIDA COM COMUNICAÇÃO CARRO-EQUIPE VIA THREADS**

Poços de Caldas

2025

# SUMÁRIO

1. OBJETIVOS ................................................................................. 2

2. DESCRIÇÃO DA APLICAÇÃO ................................................................ 3

3. CÓDIGO GERADO ............................................................................ 4

4. DESCRIÇÃO DO CÓDIGO .................................................................... 5

5. RESULTADOS ................................................................................ 6

6. CONCLUSÕES ................................................................................. 7

7. BIBLIOGRAFIA ................................................................................ 8

**Simulação de Corrida com Comunicação Carro-Equipe via Threads**

# 1. OBJETIVOS

Este trabalho tem como objetivo demonstrar, de forma prática, os conceitos de comunicação entre threads e concorrência no contexto de Sistemas Operacionais, utilizando a linguagem Python. A aplicação desenvolvida simula uma corrida de carros, onde uma equipe técnica monitora e envia comandos em tempo real a um carro de corrida, utilizando threads e queues para comunicação entre os módulos.

# 2. DESCRIÇÃO DA APLICAÇÃO

A aplicação é uma simulação de corrida entre um carro e sua equipe técnica. O carro é representado por uma thread que realiza 10 voltas em uma pista. A cada volta, ele envia à equipe técnica o seu status atual: combustível, desgaste dos pneus e velocidade.  
  
A equipe técnica, também representada por uma thread, recebe esse status e analisa a situação do carro, enviando comandos apropriados, como:  
- Chamada para os boxes (reabastecimento/troca de pneus);  
- Aceleração;  
- Redução de velocidade;  
- Manutenção do ritmo.  
  
Toda a comunicação entre carro e equipe é feita por meio de filas (queue.Queue), simulando um canal de comunicação semelhante a pipes entre processos.

# 3. CÓDIGO GERADO

import threading

import queue

import time

import random

from colorama import init, Fore, Style

# Inicializa a colorama

init(autoreset=True)

# Queues de comunicação

status\_queue = queue.Queue()

command\_queue = queue.Queue()

# Função da thread do carro

def carro\_corrida():

voltas = 10

combustivel = 100

pneus = 100

velocidade = 200 # km/h média

for volta in range(1, voltas + 1):

print(Fore.CYAN + f"\n==================== VOLTA {volta} ====================")

# Simula desgaste

combustivel -= random.randint(5, 10)

pneus -= random.randint(3, 8)

# Envia status para equipe

status = {

'volta': volta,

'combustivel': combustivel,

'pneus': pneus,

'velocidade': velocidade

}

print(Fore.BLUE + f"[Carro] 🚗 Enviando status: Combustível = {combustivel}%, Pneus = {pneus}%, Velocidade = {velocidade} km/h")

status\_queue.put(status)

# Aguarda comando da equipe

comando = command\_queue.get()

if comando == 'box':

print(Fore.YELLOW + "[Carro] 🏁 Entrando nos BOXES para reabastecimento e troca de pneus...")

time.sleep(2)

combustivel = 100

pneus = 100

print(Fore.YELLOW + "[Carro] ✅ Pronto para voltar à pista!")

elif comando == 'reduzir':

print(Fore.YELLOW + "[Carro] ⚠️ Reduzindo velocidade.")

velocidade -= 20

elif comando == 'acelerar':

print(Fore.YELLOW + "[Carro] 🟢 Acelerando ao máximo!")

velocidade += 20

else:

print(Fore.YELLOW + "[Carro] ➡️ Mantendo ritmo de corrida.")

# Simula tempo da volta

time.sleep(1)

print(Fore.CYAN + "\n🏁 [Carro] Corrida finalizada! 🏆")

# Função da thread da equipe

def equipe\_tecnica():

while True:

status = status\_queue.get()

print(Fore.GREEN + f"[Equipe] 📡 Recebido: Volta {status['volta']} | Combustível = {status['combustivel']}% | Pneus = {status['pneus']}% | Velocidade = {status['velocidade']} km/h")

# Lógica de decisão

if status['combustivel'] < 50 or status['pneus'] < 45:

comando = 'box'

elif status['velocidade'] > 220:

comando = 'reduzir'

elif status['velocidade'] < 180:

comando = 'acelerar'

else:

comando = 'manter'

print(Fore.GREEN + f"[Equipe] 🗣️ Comando: {comando.upper()}")

command\_queue.put(comando)

if status['volta'] == 10:

break

# Criando e iniciando as threads

thread\_carro = threading.Thread(target=carro\_corrida)

thread\_equipe = threading.Thread(target=equipe\_tecnica)

thread\_carro.start()

thread\_equipe.start()

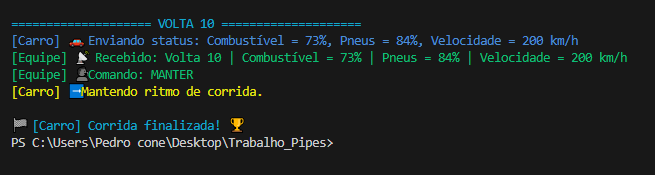
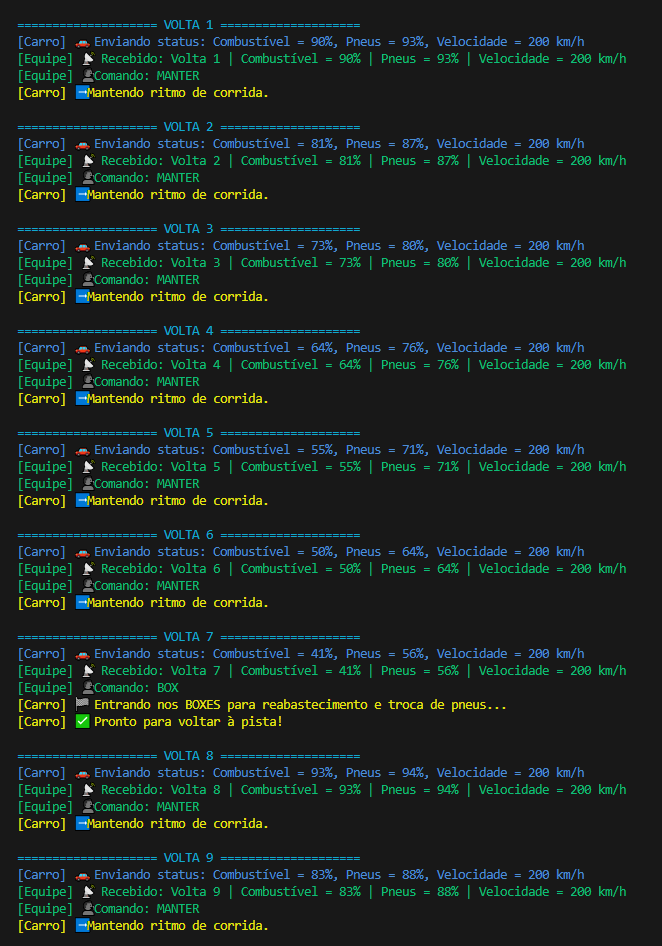
thread\_carro.join()

thread\_equipe.join()

# 4. DESCRIÇÃO DO CÓDIGO

O código é dividido em duas principais funções executadas por threads:  
  
- Thread do Carro (carro\_corrida):  
 Executa 10 voltas, simulando desgaste de combustível e pneus. Envia um dicionário com os dados de status para a status\_queue. Aguarda o comando da equipe técnica e reage conforme o comando.  
  
- Thread da Equipe Técnica (equipe\_tecnica):  
 Aguarda receber o status do carro a cada volta. Analisa os dados e decide o melhor comando, que é enviado via command\_queue.  
  
O uso da biblioteca colorama proporciona uma visualização mais clara no terminal, diferenciando mensagens por cores:  
- Azul para o carro,  
- Verde para a equipe técnica,  
- Amarelo para comandos e ações,  
- Vermelho para alertas de baixa condição.

# 5. RESULTADOS

Durante a execução do programa, é possível observar o seguinte:  
- Mensagens em tempo real entre o carro e a equipe técnica;  
- Decisões estratégicas com base em variáveis dinâmicas;  
- Comandos sendo executados na mesma thread do carro;  
- Finalização controlada da simulação ao final da 10ª volta.  
  


# 6. CONCLUSÕES

O trabalho permitiu a aplicação de conceitos fundamentais de sistemas operacionais, como concorrência, comunicação entre threads e sincronização. A simulação ilustrou como diferentes entidades (carro e equipe) podem operar paralelamente, trocando informações de forma segura e ordenada.  
  
O uso de queue.Queue se mostrou eficiente e simples para a comunicação entre threads, evitando condições de corrida. A colorização do terminal proporcionou uma leitura mais fluida dos eventos da simulação.  
  
Como melhoria futura, seria possível:  
- Adicionar múltiplos carros;  
- Simular falhas mecânicas;  
- Armazenar os dados de status em um arquivo de log para análise posterior.

# 7. BIBLIOGRAFIA

- TANENBAUM, A. S.; BOS, H. Modern Operating Systems. 4ª ed. Pearson, 2016.  
- Python Software Foundation. threading — Thread-based parallelism. Disponível em: https://docs.python.org/3/library/threading.html  
- Python Software Foundation. queue — A synchronized queue class. Disponível em: https://docs.python.org/3/library/queue.html  
- Colorama documentation. Disponível em: https://pypi.org/project/colorama/