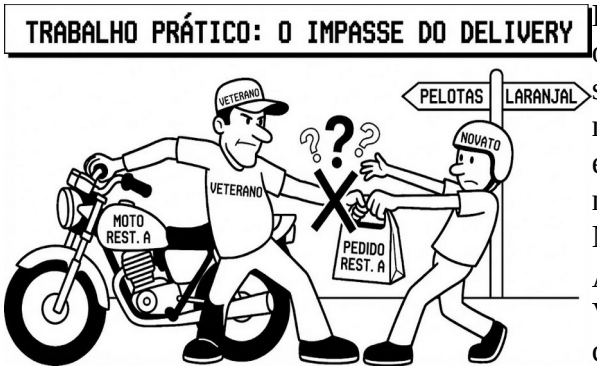
Grupos de 1 até 2 participantes: Entregar Etapa 1 (Impasse do Delivery) e Etapa 2 (Simulador SO)
Grupos de 1 até 3 participantes: Entregar Etapa 1, Etapa 2 e Etapa Extra.
O que entregar?

- Etapa 1 – Código e relatório (o código precisa ser apresentado para pontuar)
- Etapa 2 – Vídeo com a participação de todos os componentes do grupo, com duração máxima de 10 minutos, gravado em velocidade normal (1x), se preferir pode ser artigo no formato SBC de 4 até 8 páginas.
- Etapa Extra – Código e relatório.

OBSERVAÇÕES IMPORTANTES:

- Trabalho em grupo deve ser realizado por todos participantes.
- É importante ter em mente que ao participar de um grupo, está assinando por todo o trabalho. Logo, poderá ser questionado por qualquer atividade relacionada (Não será aceita a resposta: "eu não fiz essa parte, foi o fulano").
- É um trabalho em grupo e todos devem participar. Em casos de problemas no grupo, me enviem um e-mail e esse grupo poderá ter apresentações individualizadas.
- Problemas com cópia (inclui cópia de IA)/compartilhamento de trabalho (mesmo que parcial), a nota será zerada.

ETAPA 1 – Simulação - Impasse do Delivery



Em Pelotas, o aplicativo de entregas "Laranjal Foods" opera com uma regra rígida: cada restaurante possui sua própria moto personalizada (adesivada com a marca da lanchonete). Portanto, para realizar uma entrega do Restaurante X, o entregador precisa necessariamente pegar a Comida do Restaurante X e a Moto do Restaurante X.

A frota de entregadores é mista, composta por Veteranos (muito experientes) e Novatos (recém-chegados).

O sistema de despacho do aplicativo tem um bug: ele ocasionalmente aloca dois entregadores simultaneamente para o mesmo restaurante quando a demanda está alta (force esse bug e após ocorrer, faça algo para resolver “durante a execução do simulador”).

O problema é que os entregadores possuem "rituais" diferentes de trabalho, o que gera travamentos no sistema:

Comportamento das Threads

Tipo 1: O Entregador Veterano

Ele prioriza a logística. Ele não quer pegar o peso da comida sem garantir que tem transporte.

1. Escolhe (ou recebe) um restaurante alvo `i`.
2. Bloqueia a Moto `i`.
3. Simula o tempo de caminhada até o balcão (`sleep`).
4. Tenta pegar o Pedido `i`.
5. Se conseguir ambos, faz a entrega, libera os recursos e volta para a fila.

Tipo 2: O Entregador Novato

Ele é ansioso para garantir a comissão. Ele quer pegar o pacote logo para marcar no app que "coletou".

1. Escolhe (ou recebe) um restaurante alvo `i`.
2. Bloqueia o Pedido `i`.
3. Simula o tempo de caminhada até o estacionamento (`sleep`).
4. Tenta pegar a Moto `i`.
5. Se conseguir ambos, faz a entrega, libera os recursos e volta para a fila.

A Situação de Deadlock

O deadlock ocorrerá especificamente quando um Veterano e um Novato decidirem atender o mesmo restaurante quase ao mesmo tempo.

1. O Veterano pega a chave da moto do McDonald's e vai buscar o lanche.
2. O Novato pega o lanche do McDonald's e vai buscar a moto.
3. **Resultado:**
    1. O Veterano chega no balcão: "Cadê o lanche? O Novato pegou. Vou esperar ele soltar." (Mas o novato não vai soltar, pois ele precisa da moto para entregar).
    2. O Novato chega na moto: "Cadê a chave? O Veterano pegou. Vou esperar ele soltar."
    3. Ambos ficam travados eternamente segurando recursos vitais um do outro.

Especificações do Trabalho

Desenvolva um programa em C utilizando a biblioteca `pthread` que simule este ambiente. Pode utilizar Java, apenas se utilizar semáforos (não inclui solução baseada no uso de syncronized).

Requisitos Técnicos:

1. **Definições:**
    - Defina um número de restaurantes (ex: 5 ou 10).
    - Crie um array de mutexes para os pedidos.
    - Crie um array de mutexes para as motos.
2. **Threads:**
    - Crie threads de Entregadores Veteranos.
    - Crie threads de Entregadores Novatos.
    - Dica: Para forçar o deadlock, faça com que as threads escolham o restaurante aleatoriamente, mas garanta que o número de threads seja maior que o número de restaurantes.
3. **Logging (Saída):** O programa deve narrar o que está acontecendo para que o deadlock seja visível.
    - [Veterano 1]: Peguei a chave da moto do Restaurante 0.
    - [Novato 2]: Peguei o lanche do Restaurante 0.
    - [Veterano 1]: Aguardando lanche do Restaurante 0...
    - [Novato 2]: Aguardando moto do Restaurante 0...
    - (Após isso, nenhuma mensagem sobre o Restaurante 0 aparece mais -> Deadlock confirmado)

Obs.: O objetivo deste trabalho é desenvolver o raciocínio sobre o uso de concorrência e deadlocks. Ferramentas de IA podem ser utilizadas como "Monitores Virtuais" para esclarecer dúvidas de sintaxe ou conceitos teóricos. Entretanto, a lógica e a implementação devem ser autorais. Pedir a solução pronta impede que você desenvolva a habilidade de diagnosticar problemas complexos, que é o foco da avaliação. A implementação deve ser fruto do seu esforço intelectual. Dúvidas conceituais podem ser sanadas com IA; *copy-paste* de código pronto, não.

ETAPA 2 – Estudo Comparativo de Simuladores de Sistemas Operacionais: SOSim vs. Alternativas de Mercado

Objetivo: Expandir a compreensão dos conceitos de Sistemas Operacionais através da análise crítica e comparativa. Pesquise por um simulador de SO alternativo disponível na internet, aprenda suas funcionalidades e compare diretamente com o simulador clássico SOsim[1], utilizando um roteiro de experimentos pré-definido a seguir.

Instruções Gerais:

1. **O Simulador Base:** Utilize o **SOsim** (disponível para Windows/Linux via Wine) como referência padrão.
2. **O Simulador Desafiante:** Pesquise e escolha livremente outro simulador de SO disponível na web (ex: *CPU-OS Simulator, GitHub projects, Web-based simulators, etc.*).
    - *Requisito:* O simulador escolhido deve suportar, minimamente, visualização de Gerência de Processos e Gerência de Memória.
3. **O Produto Final:** Crie um vídeo apresentando a execução simultânea (ou sequencial editada) dos experimentos abaixo em ambos os simuladores, destacando as diferenças, limitações e vantagens de cada um. Vídeo de, no máximo, 10 minutos, gravado em velocidade normal (1x).

Roteiro de Comparação (Itens Obrigatórios no Vídeo):

a) Visão Geral e UX (Experiência do Usuário)

- Apresente o simulador escolhido (Nome, onde encontrar, plataforma).
- Compare a interface principal das duas ferramentas: Qual oferece melhor visualização dos componentes (CPU, Memória, Disco)?
- Qual deles apresenta uma curva de aprendizado mais amigável para um estudante iniciante?

b) Gerência de Processos (Execução Comparativa) *Tente replicar o seguinte cenário em ambas as ferramentas e compare os resultados:*

- Criação de Processos: Crie processos CPU-bound e I/O-bound em ambos.
    - Análise: O simulador novo permite distinguir esses tipos claramente como o SOSim?
- Estados do Processo: Mostre o diagrama de transição de estados na prática nas duas ferramentas.
    - Análise: Os estados (Pronto, Executando, Bloqueado) são visíveis da mesma forma?
- Escalonamento e Clock: Configure o Escalonamento Circular (Round Robin).
    - Análise: Explique como cada simulador lida com a fatia de tempo (quantum) e o clock.
- Experimento de Prioridade:
    - No SOSim, configure:
        - 2 processos Prioridade 3 (I/O);
        - 2 processos Prioridade 2 (Misto);
        - 2 processos Prioridade 1 (CPU).
    - Tente replicar essa configuração no simulador novo.

1 http://www.training.com.br/sosim/

- *Discussão:* Se invertermos as prioridades, o comportamento visualizado no simulador novo é consistente com o visto no SOSim? Se o simulador novo não suportar prioridades, explique essa limitação.

c) Gerência de Memória

- Políticas de Busca: O SOSim permite alternar entre Paginação Antecipada e Por Demanda. Verifique se o novo simulador possui essa distinção.
- Page Faults: Gere carga na memória e tente visualizar/contar os page faults (falhas de página) em ambos os simuladores.
    - Análise: Qual ferramenta oferece melhores logs ou gráficos sobre o desempenho da memória?
- Estruturas de Controle: Compare a visualização da Tabela de Páginas (PCB) e dos Frames de memória física nas duas ferramentas.

d) Conclusão

- Análise Crítica: O simulador que você encontrou é melhor, pior ou complementar ao SOSim?
- Aderência Teórica: Qual dos dois representou mais fielmente os conceitos vistos nas aulas teóricas?
- Recomendação: Se você tivesse que ensinar SO para um colega, qual dos dois utilizaria e por quê?

ETAPA EXTRA (Somente para grupo com até 3 participantes) – Desenvolvimento de um escalonador

Crie um programa para simular o escalonamento de um conjunto de tarefas conhecidas (é de conhecimento o tempo de execução de cada tarefa).

O programa deverá receber como parâmetro na linha de execução: nome do arquivo que possui informações sobre as tarefas que serão escalonadas (cada linha apresenta o nome da tarefa e um número inteiro que representa o tempo de execução) e um número que indicará a quantidade de processadores que se deseja utilizar na simulação.

Implemente dois algoritmos de escalonamento: o primeiro é o clássico SJF (*Shortest Job First*) que irá executar as menores tarefas primeiro. O segundo algoritmo é o oposto do SJF, executando as maiores tarefas primeiro.

Como saída espera-se dois arquivos (um para cada escalonamento). O arquivo deve conter o id do processador e o nome de cada tarefa com o instante inicial e final (em segundos). Respeite EXATAMENTE o formato do exemplo abaixo, pois essa saída será entrada de outro programa para validação do resultado.

Por exemplo, forma de executar o programa: ./trabalho_escalonador tarefas.txt 2

Conteúdo do arquivo de entrada: “tarefas.txt” (nome da tarefa e tempo total de execução separados pelo caractere espaço).

```
a1      5
a2      1
a3      10
b1      10
b2      3
b3      7
b4      8
c1      8
c2      2
```

SAÍDA1 (menor_primeiro.txt):

```
Processador_1
a2;0;1
b2;1;4
b3;4;11
c1;11;19
b1;19;29

Processador_2
c2;0;2
a1;2;7
b4;7;15
a3;15;25
```

SAÍDA 2 (maior_primeiro.txt)

```
Processador_1
a3;0;10
b4;10;18
b3;18;25;
c2;25;27

Processador_2
b1;0;10
c1;10;18
a1;18;23
b2;23;26
a2;26;27
```