Nome: Túlio Augusto De Almeida Rezende Gec 7°

Equipe 1:O código está bem estruturado, modularizado e eficiente, com múltiplas classes representando as funções do restaurante. A abordagem de multithreading melhora o fluxo de trabalho, e as filas compartilhadas organizam o atendimento, há riscos de concorrência entre threads e a parada dos threads precisa ser refinada.

Equipe 2: usaram técnica que não vimos ainda no código, mas foi muito bem apresentado a ideia e código bem-organizado tendo fácil acesso e entendimento.

Equipe 3: O código faz uma excelente utilização de Worker Threads, permitindo o processamento assíncrono e eficiente. O uso de um buffer compartilhado e semáforo garante um controle adequado do pit stop, evitando conflitos entre as threads, pode haver bloqueio indefinido no semáforo caso alguma thread não libere corretamente o recurso. Além disso, o cálculo do consumo de combustível pode ser refinado para simulações mais realistas. O tempo de espera no pit stop poderia ser ajustado para valores mais dinâmicos.

Equipe 4: O código é bem estruturado e usa multithreading para otimizar a verificação de status dos sites. O uso de ThreadPoolExecutor melhora a eficiência ao gerenciar múltiplas requisições simultâneas. Além disso, a implementação de locks e filas garante um controle seguro sobre os dados. Porém, há riscos de requisições bloqueadas se muitos sites forem inacessíveis ou demorarem a responder.

Equipe 5: Poderia falar um pouco mais alto, estruturar a apresentação em slides para melhor compreensão da ideia, ideia um pouco mais complexa de ser implementada, apresentação bem-feita.

Equipe 7:Usou lock no código, semáforo acaba fazendo fila com os carros não passando dois carros, ideia muito bem apresentada e código muito bem-feito e organizado.

Equipe 8: O código está bem organizado e modularizado, facilitando a execução de diferentes simulações. A abordagem de criação dinâmica de arquivos torna os testes mais flexíveis e reproduzíveis. A escolha de algoritmos de escalonamento adiciona versatilidade ao funcionamento do simulador. No entanto, o uso de entrada do usuário direta pode gerar erros se os dados não forem validados corretamente. O gerenciamento de arquivos pode ser refinado para garantir o fechamento seguro dos recursos e evitar possíveis exceções.

Equipe 9: código poderia ser melhor edentado, 3 threads em paralelos e com o enceramento do expediente está encerrando tudo e não está terminando os pedidos que já estavam anotados, ideia bem apresentada.

Equipe 10:codigo muito bem edentado e ideia bem apresentada, só que pelo que lembro na hora da apresentação o código estava retornando valores errados não sei se enxerguei errado, mas parecia estar.

Equipe 11: O código implementa um simulador de escalonamento de CPU, comparando diferentes algoritmos como FCFS, PS e SJF. A estrutura é bem organizada, utilizando objetos para representar processos, o que facilita a manipulação dos dados. A abordagem de cálculo de métricas é bem feita. Porém, a ausência de validação de entrada pode gerar erros caso um algoritmo não suportado seja fornecido. A estrutura poderia permitir escalonamento preemptivo, expandindo a análise para mais algoritmos como RR.

Equipe 12: O código implementa uma simulação de elevadores, utilizando threads para gerenciar múltiplos elevadores e passageiros. O uso de semaforos evita conflitos de concorrência, garantindo que a lógica de embarque e desembarque funcione corretamente. A abordagem de diferentes estratégias de escalonamento (SJF e PS) torna a simulação mais dinâmica. Porém, há riscos de espera excessiva caso os passageiros se acumulem em determinados andares. A penalidade por excesso de passageiros sem semáforo pode ser melhor ajustada. O tempo de movimentação entre andares poderia ser refinado para maior realismo.

Equipe 13: O código apresenta uma simulação de escalonamento de processos, utilizando diferentes algoritmos como FCFS, SJF, PS e RR. A implementação é bem estruturada e permite a análise do tempo de espera e execução de cada processo. O uso de quantum no Round Robin adiciona flexibilidade à simulação. Porém, a ausência de verificação de entrada pode gerar erros se algum processo tiver valores inconsistentes. A simulação poderia incluir algoritmos preemptivos para um estudo mais amplo.

Equipe 14: O código implementa uma simulação hospitalar, onde diferentes algoritmos de escalonamento (FCFS, PS, SJF) organizam o atendimento dos pacientes. A abordagem de entrada interativa permite flexibilidade na escolha do hospital e do modo de sincronização. O uso de multithreading melhora a eficiência do atendimento médico. Porem, a verificação de entrada poderia ser melhor estruturada para evitar erros inesperados. A lógica de adição de pacientes pode ser refinada para permitir uma fila dinâmica. Além disso, o impacto do RedutorVida na simulação pode ser melhor detalhado.

Equipe 15:codigo muito bem comentado e explicado, mas poderia ter sido divido em partes para a melhor compressão e organização

Equipe 16: O código implementa uma simulação de saque bancário concorrente, garantindo que múltiplas threads acessem uma conta bancária de maneira segura usando semáforo binário. A estrutura de controle de saldo previne condições de corrida, mantendo a integridade das transações. Porém, há um risco de espera excessiva se muitos threads competirem pelo semáforo ao mesmo tempo. O tempo de processamento pode ser ajustado para evitar atrasos artificiais. Também seria interessante adicionar logs detalhados para monitoramento das operações.

Equipe 17: O código simula gestão de semáforos, utilizando multithreading para controlar sinais de trânsito e evitar colisões. A implementação de locks e eventos garante segurança na sincronização. A abordagem de estatísticas em tempo real adiciona profundidade à simulação. Porém, há um risco de espera excessiva caso os tempos de liberação dos carros sejam muito baixos. A lógica de detecção de colisões pode ser refinada para prevenir falsos positivos. Além disso, a probabilidade de liberação pode ser ajustada para variações mais dinâmicas.

Equipe 18 O código implementa uma simulação de protocolo de janela deslizante, utilizando threads e semáforos para controle de envio de pacotes. A estrutura bem organizada permite diferentes métodos de escalonamento (FCFS, SJF, PS) e uma interface interativa para o usuário. A abordagem de tratamento de pacotes perdidos torna o sistema mais robusto. Porém, o sistema pode ter esperas excessivas se muitos pacotes forem perdidos em sequência. A lógica de reenvio de pacotes poderia incluir um limite para evitar sobrecarga. Além disso, a interface poderia ter feedback visual para acompanhar o progresso da simulação.

Equipe 19:codigo poderia ser comentado e edentado melhor, mas boas explicações do projeto e ideia.

Equipe 20: O código apresenta um quebrador de senhas baseado em diferentes algoritmos de escalonamento (Round Robin e FCFS). A estrutura está bem organizada e permite ao usuário definir número de threads e tamanho do quantum, tornando a simulação flexível. A verificação de entrada evita inputs inválidos e garante que as senhas tenham o formato esperado. Porém, o código não possui tratamento de erros caso os módulos importados não estejam disponíveis. A lógica poderia incluir limite de tentativas para evitar sobrecarga de processamento.

Equipe 21: Código implementa uma simulação de pagamento em supermercado, garantindo acesso seguro ao caixa por meio de semáforo. A estrutura bem organizada evita condições de corrida, garantindo que cada cliente processe seu pagamento sem interferências. O uso de threads melhora a eficiência da simulação. Porém, há um potencial tempo de espera elevado, pois todas as operações dentro da região crítica são executadas em sequência. O tempo de processamento pode ser ajustado para evitar latências artificiais. A simulação também poderia incluir variações no número de caixas para testar diferentes cenários.