

## 1º Trabalho Prático de Sistemas Operacionais I - 2023

Desenvolva uma aplicação concorrente em *C/Threads* no Linux que simule o comportamento de passageiros que utilizam ônibus para irem de um ponto de origem até um ponto de destino. A sincronização deve ser feita com semáforos e variáveis de condição.

No seu algoritmo suponha que:

Há *S* pontos de ônibus.

Há *C* carros que representam os ônibus.

Há *P* passageiros que desejam andar de ônibus.

Há *A* assentos em cada ônibus.

Garanta que  $P \gg A \gg C$ .

Forneça argumentos de entrada determinando *S*, *C*, *P* e *A*, nesta ordem, ao iniciar a aplicação.

Há um processo chamado *onibus* que irá receber tais argumentos de entrada (representa o início da aplicação).

O processo *onibus* irá gerar uma thread para cada um dos *P* passageiros, cada um dos *C* ônibus e cada um dos *S* pontos de ônibus especificados.

Os pontos, carros e passageiros são numerados sequencialmente.

Os ônibus podem partir de pontos distintos, escolhidos aleatoriamente por eles próprios.

Apenas um ônibus pode estar em um ponto por vez (desde o início da aplicação).

Se um ponto estiver vazio (sem passageiros esperando), o ônibus segue viagem para o próximo ponto, em ordem crescente do número do ponto.

Quando um ônibus chegar em um ponto já ocupado por outro ônibus, este que está chegando não espera e parte para o próximo ponto, sempre considerando a ordem crescente do número do ponto.

O tempo de viagem entre um ponto de ônibus e outro é aleatório e não fixo para todos os ônibus.

Um ônibus pode “ultrapassar” outro ônibus durante a viagem e chegar antes ao próximo ponto.

Enquanto houver passageiro na fila do ponto e o ônibus tiver assento livre, o ônibus permite a entrada de passageiros.

Enquanto houver passageiro para viajar, os ônibus continuam circulando entre os pontos, seguindo a ordem crescente do número do ponto de ônibus. Quando um ônibus chega ao último ponto, ele vai para o primeiro ponto (uma fila circular).

Quando não houver mais passageiros, todos os ônibus param de circular e a aplicação acaba.

Os passageiros indicam, com argumentos de entrada fornecidos aos passageiros, os pontos de origem e destino, respectivamente.

Os pontos de origem e destino para cada passageiro são determinados aleatoriamente, dentro da faixa de *S* (de 0 a *S*-1) pontos de ônibus existentes, passados como argumentos de entrada para a thread passageiro pelo processo *onibus*. O ponto de destino pode ser menor que o ponto de origem. Lembre-se que os ônibus viajam seguindo uma fila circular da ordem crescente do número do ponto. Os passageiros, ao chegarem para embarcar em um ponto de ônibus, respeitam a fila (critério FIFO). A fila é controlada pelo ponto de ônibus.

Os passageiros determinam se eles próprios chegaram ao ponto de destino e então descem do ônibus.

Ao chegar no destino, cada passageiro resolve o que precisa naquele local (tempo aleatório) e então finaliza.

Cada passageiro gera um arquivo de rastro no formato texto chamado *passageiroXX.trace* – onde *XX* é o número do passageiro e contendo a *string* <horário que chegou no ponto de origem X, horário que entrou no ônibus Y, horário que desceu do ônibus Y, ponto de destino W que desceu >.

Desenvolva uma aplicação concorrente *onibus* que garanta os requisitos acima, seguindo as diretrizes passadas aqui e em aula. As questões omissas e/ou ambíguas devem ser fixadas pelo professor. Qualquer dúvida, entre em contato com o professor o mais rápido possível para orientação.

O seu código deverá ser capaz de executar corretamente a simulação pedida. A saída do programa deve ser: (1) uma animação na tela dos passageiros, pontos e ônibus em funcionamento e (2) os arquivos de rastro dos passageiros gravados em disco. O trabalho será apresentado pelo grupo todo ao professor.

Apenas uma versão por grupo do trabalho deve ser entregue, via *edisciplinas*, até a data máxima fixada em sala de aula. Destaque como comentário no início do código o número do grupo e os nomes dos integrantes do grupo que de fato participaram do desenvolvimento do trabalho. Os nomes que não aparecerem nessa relação ficarão com nota "zero". A correção considerará a funcionalidade e a qualidade do programa desenvolvido. Como qualidade entende-se (sem estar limitado a): legibilidade, comentários, estrutura/modularidade, emprego adequado das estruturas de programação e comunicação/sincronização de processos/threads.