UNIVERSIDADE FEDERAL DE SANTA CATARINA

CAMPUS TRINDADE

INE-DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

INE5410 - PROGRAMAÇÃO CONCORRENTE

Aluno:

Tális Breda

22102202

Relatório do Trabalho 2

Florianópolis

2023

Sumário

1. Introdução

A simulação em questão é de uma esfera inspirada na The Sphere de Las Vegas. Ela funciona com base em uma fila única, e os eventos dentro da esfera são diferentes conforme a faixa etária dos clientes. Quando é identificada a faixa etária do primeiro da fila, deve ser iniciada uma experiência voltada para essa faixa etária, e ela deve permanecer em curso até que o próximo da fila seja de faixa etária diferente, ou que tanto a esfera quanto a fila esvaziem.

A thread principal executa um loop até que N_PESSOAS tenham entrado na esfera. Cada iteração desse loop executa outro loop, que roda até que uma das condições descritas anteriormente sejam satisfeitas. Para cada iteração desse segundo loop, vários semáforos são usados para controlar o que fazer na situação atual, quantas vagas estão disponíveis, quando algum cliente chega na fila e quando ele pode entrar na esfera.

A solução principal envolve uma thread auxiliar, chamada de thread de controle, que utiliza dois semáforos e algumas condições para determinar se a experiência deve continuar ou deve ser pausada.

Quando a experiência é pausada, a faixa etária da próxima experiência é definida como a faixa etária do primeiro da fila, a não ser que todas as N_PESSOAS tenham entrado na esfera, nesse caso a simulação termina.

Neste trabalho, foram utilizados diversos conceitos aprendidos durante a disciplina, principalmente os conhecimentos relacionados à programação concorrente em Python e a sincronização em geral.

Ao longo desse relatório, os passos definidos aqui serão explicados com mais detalhes.

2. Inicialização da simulação e das estruturas

2.1. Cliente

A classe Cliente é relativamente simples, criada para guardar os dados de um cliente (espectador) de forma simples. Os atributos dessa classe são:

- **id**: inteiro que representa o ID único deste cliente
- **faixaEtaria**: inteiro que representa a faixa etária do cliente (0 = "A", 1 = "B", 2 = "C"). Ao chamar o método **getFaixaEtaria()** do cliente, o retorno é a string.

- **thread**: thread que realiza as ações do cliente

2.2. StatusIxfera

A classe **statusIxfera** é bastante complexa, e executa grande parte das operações da simulação por possuir vários atributos necessários para a execução. Esses diversos atributos são listados a seguir:

- **filaClientes:** autoexplicativo, é a fila de clientes, que é abastecida pela thread criadora de clientes
- **tipoApresentação:** string que representa a faixaEtaria da apresentação ocorrendo atualmente
- **semaforoEntrada:** semáforo que controla a entrada de clientes na esfera, incrementada pela thread principal e decrementada pelo cliente
- **semaforoVagas:** semáforo que ajuda no controle da entrada, inicializado com N_VAGAS, decrementado quando um cliente entra na esfera, e incrementado quando algum sai
- **semaforoControle:** semáforo usado por uma thread auxiliar para controlar entradas e saídas
- **semaforoFim:** semáforo que controla o fim de uma apresentação
- **semaforoChegada:** semáforo que controla a chegada de clientes na fila, incrementado quando alguém chega e decrementado quando alguém enrta
- **semaforoRetornoControle:** outro semáforo utilizado pela thread auxiliar, decrementado pela thread principal para definir o próximo passo da simulação
- **lock:** lock usado para acesso às listas da classe, para evitar condições de corrida
- **pessoasNaApresentacao:** número de pessoas presentes na apresentação em determinado momento
- **pessoasQueEntraram:** número de pessoas que entraram na apresentação
- **pessoasQueSairam:** número de pessoas que saíram da apresentação
- **filaMensagens:** instância da classe multiprocessing.Queue enviada pelas threads dos clientes para a thread auxiliar
- **statusFim:** status definido pela thread auxiliar, e utilizado pela thread principal para definir se a apresentação é pausada ou não
- **n_pessoas, n_vagas, permanencia, max_intervalo, semente, unid_tempo:** parâmetros de entrada da simulação
- **qtdClientes:** lista de números inteiros que representa quantos clientes de cada faixa etária entraram na fila

- **temposEspera:** lista dos tempos totais de cada faixa etária, utilizada para calcular os tempos médios
- **tempoExperiencia:** tempo total que a experiência ficou rodando
- **lockEspera:** lock utilizado para acessar as listas **temposEspera** e **qtdClientes**
- **primeiraVez:** booleano utilizado pela thread auxiliar para saber se foi a primeira vez naquela apresentação que a thread foi acessada

3. Implementação das threads

Além das threads padrão dos clientes e da criadora de clientes, foi utilizada outra thread, chamada no programa de threadControle, para controlar entradas e saídas da esfera.

3.1. criaClientes

É a thread que cria as threads dos clientes. Ela possui uma lista que armazena cada uma das threads criadas para serem finalizadas posteriormente, e um loop que executa N_PESSOAS vezes, criando os clientes, associando-os a threads, adicionando-os na fila da esfera e esperando um tempo aleatório.

3.2. threadCliente

Funciona com base em diversos semáforos. Em resumo, quando inicializada, a thread incremente alguns semáforos, indicando que chegou na fila, espera uma vaga para entrar na esfera, fica PERMANENCIA na esfera, sai e grava o tempo que ficou esperando

3.3. threadControleFim

Thread auxiliar, usada para controlar entradas e saídas na esfera. A maneira como ela funciona é como uma corrida entre threads. Essa thread decrementa um semáforo que pode ser incrementado por um cliente entrando, ou por um cliente que é o último a sair da esfera, deixando-a vazia. A thread que incrementa o semáforo também envia uma mensagem através de **filaMensagens**, e a thread de controle utiliza essa mensagem para definir o **statusFim** e incrementar o **semaforoRetornoControle**, para que a thread principal possa rodar e saber o que deve ser feito.

4. Sincronização

A sincronização da simulação é feita através de diversos semáforos, cada um controlando uma parte do programa.

A thread principal executa um loop até que o próximo cliente tenha uma faixa etária diferente da atual ou que a thread de controle defina que a experiência deve ser pausada.

Ela decrementa diversos semáforos, como o **semaforoRetornoControle**, assim aguardando que a **threadControle** defina o próximo passo. A partir daí, decrementa também o **semaforoVagas**, esperando liberar uma vaga na esfera, decrementa o **semaforoChegada**, esperando um cliente chegar na fila, e incrementa o **semaforoEntrada**, avisando ao cliente que ele pode entrar na esfera.

Ao definir que aquela apresentação não deve continuar, seja porque o próximo da fila é de faixa etária diferente ou porque a esfera esvaziou e não há clientes na fila, a thread principal aguarda o **semaforoFim**, que é incrementado também pela **threadControle**.

Além dos semáforos, dois locks são usados, como definido anteriormente, para controlar o acesso e alterações a listas, que são a fila de clientes e as listas de quantidades de clientes de cada faixa etária e tempo de espera de cada faixa etária.

5. Pausas no evento

5.1. Mudança de faixa etária

A mudança de faixa etária ocorre quando o próximo cliente da fila é de uma faixa etária diferente da que está sendo aceita pela apresentação naquele momento. Quando isso é percebido, o programa pára de aceitar novos clientes e espera os que estão na esfera finalizarem. Assim que todos finalizarem, o **semaforoFim** será incrementado a apresentação atual será pausada e será possível iniciar uma nova.

Ao reinício do loop, o método **defineFaixaEtaria()** olha para o primeiro cliente da fila e define o tipo da experiência como a faixa etária do mesmo.

5.2. Esfera e fila vazias

Quando não há nenhum cliente na esfera e nenhum cliente na fila, a experiência deve ser pausada. Isso é feito através da thread do último cliente a sair da esfera e da thread de controle.

Quando o cliente é o último a sair, ele incrementa o **semaforoControle**, notificando através de um código que a experiência deve ser pausada. A thread de controle então define o **statusFim** como 2 e incremente o **semaforoRetornoControle**, o que é lido pela thread principal e o loop principal quebra, indo para o fim daquela experiência. A thread de controle também incrementa o **semaforoFim**, para que a thread principal continue o loop quando chegar a esse semáforo.

6. Conclusão

Esse trabalho se mostrou mais complexo do que parecia ser com base no enunciado. Foi um desafio realizar a comunicação entre as threads do cliente e a principal, e pausar a experiência nos momentos certos.

Ao final, foi possível cumprir os requisitos de forma robusta, utilizando dos conhecimentos relacionados a Python e a sincronização aprendidos na disciplina. O uso de semáforos foi crucial, visto a quantidade que foi usada, além dos locks para evitar condições de corrida e da Queue, que apesar de ser voltada para programas que usam vários processos, foi usada por conta da sua implementação nativa de travas e sincronização, que evitam problemas que poderiam surgir ao usar uma lista básica.