**Trabalho de Computação Concorrente 2017.2**

**Alunos:** Carolina Hiromi Kameyama 116022176

Elvis Nobrega de Alcantara 116011484

Thiago Conceição da Silva Costa 115031164

**Linguagem:** Java

## **Objetivo**

Criar um programa que simulasse como um sistema de reserva de assentos funciona, permitindo que diversos usuários pudessem reservar, visualizar e cancelar assentos diferentes ao mesmo tempo.

## Decisões

Criar um monitor que armazenasse todas as informações dos assentos e qualquer usuário poderia acessá-lo para reservar um assento caso estivesse livre ou cancelar uma reserva, um vetor de locks para que cada assento tenha sua tranca e não pudesse ser acessada ao mesmo tempo por múltiplos usuários, e um sistema de senhas para que todas as atualizações feitas no mapa de assentos fossem registradas no arquivo de saída na ordem em que foram executadas, além de implementação de um produtor consumidor, modelando em cima de um buffer.

Sobre as tecnologias escolhidas, usamos o Java para o sistema principal, por ter implementado locks e os monitores proporcionados pela JVM, e o script foi todo feito em python, a primeira alternativa tinha sido Perl, mas, python tinha algumas vantagens em questão de leitura do código, logo o escolhemos. O script compila todo o código e ainda roda usando as entradas dadas para ele.

## Solução

Criamos um sistema que possui como entrada a quantidade de assentos que podem ser reservados, esse sistema possui um roteiro de fácil modificação que simula as ações dos usuários (que seriam as threads).

O sistema cria um buffer que registra todas as ações dos usuários conforme a ordem que os mesmos executam em um arquivo de saída.

Cada thread usuário acessa o monitor e o escritor, e pode executar as funções verifica, ocupaVazio, ocupaAssento e cancela. Onde o monitor é o local onde são guardados as informações da quantidade de assentos e o mapa de assentos que guarda o estado e localização de cada assento, e o escritor é a classe acessada pelo usuário para adicionar em uma lista de strings as atualizações feitas no mapa de assentos que posteriormente é escrito em ordem em um arquivo de saída de acordo com a ordem de ações tomadas.

## Especificação da Solução

**Script:** (Script em python que compila e roda o programa)

Funcionamento:

Basicamente para rodar o programa basta executar ./trabalho.py arquivo\_log nr\_cadeiras, mas, se o python não for atualizado ou não estiver em /usr/python basta executar python trabalho.py com as entradas logo após ele.

Ele primeiro verifica as entradas, então compila os arquivos java, e depois roda.

Espera terminar e então espera mais um segundo e lê o arquivo de saída linha a linha. Se tudo estiver ok, ele vai apenas dizendo uma a uma, ação feita pelos usuários, em ordem, caso alguma inadequação exista no log ele envia uma mensagem de erro, explicando do que se trata.

**Classe Sistema:** (Classe principal, que executa todo o sistema)

Variáveis:

buffer – lista de strings que servirá de fila na thread produtorConsumidor para que outra thread Consumidor pegue um por um e escreva no arquivo de saída

produtorConsumidor – thread que recebe as strings geradas pelas “ações” dos usuários e os coloca em uma fila em que posteriormente uma thread consumidor retira as strings uma por uma da fila e as escreve no arquivo de saída

monitor – classe que armazena as informações dos assentos, o monitor é acessado pelos usuários para que as informações dos assentos sejam atualizados

usuários – threads que simulam as ações dos usuários

NUMERO\_DE\_THREADS – quantidade de usuarios atuando

roteiro – lista de classe Pair (uma classe criada para organizar como o roteiro iria funcionar)

Funções:

main – inicializa tudo e roda todo código

criaRoteiroDosUsuarios – baseado em um modo simples de criação, simula um roteiro de como os usuários utilizariam o sistema, a fim de facilitar a criação de novas threads usuário com comportamentos diferentes

**Classe Usuário:** (Thread de ações dos usuários)

Variáveis:

id – id da thread

monitor - classe que armazena as informações dos assentos, o monitor é acessado pelos usuários para que as informações dos assentos sejam atualizados

escritor – thread que o usuário acessa para adicionar as atualizações feitas à lista de strings que serão escritos no arquivo de saída

roteiro – roteiro que simula as ações do usuário

senha – senha que usuário pega para acessar o escritor de forma que o que é adicionado na lista seja na ordem em que as modificações feitas no monitor foram feitos

senhaAtual – a próxima senha a ser chamada para adicionar uma string na fila do escritor

lock – objeto usado para fazer o synchronized

retorno – variável criada para conseguir receber os valores retornados pela função ocupaVazio

Funções:

verifica – acessa o monitor e recebe o mapa de assento e uma senha que é utilizada para escrever na lista de atualizações quando for sua vez e depois adiciona uma string à fila de strings para serem escritos no arquivo de saída

ocupaVazio – acessa o monitor para tentar ocupar um assento aleatório, recebe como retorno o número do assento e uma senha e depois adiciona uma string à fila de strings para serem escritos no arquivo de saída

ocupaAssento – acessa o monitor para tentar ocupar um assento especificado na entrada da função, recebe como retorno uma senha e depois adiciona uma string à fila de strings para serem escritos no arquivo de saída

cancela – acessa o monitor para cancelar um assento já ocupado pelo usuário que chamou essa função, recebe como retorno uma senha e depois adiciona uma string à fila de strings para serem escritos no arquivo de saída

run – recebe a sequência de funções do roteiro e as executa em ordem, a partir de um switch()

**Classe ProdutorConsumidor:** (Thread que vai adicionando informação no arquivo de saída)

Variáveis:

buffer – Lista de strings que armazena todas as strings recebidas pelas threads usuário na ordem em que estas acessaram o monitor para atualizar ou verificar os assentos

fimPrograma – booleana que verifica se o programa terminou ou não, se o programa não tiver terminado a thread ProdutorConsumidor irá continuar esvaziando a lista enquanto ela não estiver vazia

arq – arquivo onde será escrito a saída

Funções:

escreveNoArquivo – escreve no arquivo de saída as strings presentes no buffer em ordem e as tira do buffer

escreveNoBuffer – adiciona à fila de strings (buffer) a string passada pela thread usuário

termina – modifica a boolean fimPrograma para true para que a thread possa terminar

run – enquanto o boolean termina é false, ou seja, o programa ainda não terminou, a thread vai verificando se há algo no buffer e executa a função escreveNoArquivo

**Classe Assento:** (classe que armazena as informações do assento)

Variáveis:

estado – estado do assento (false = livre / true = ocupado)

id\_usuario – número do usuário que está ocupando o assento

**Classe Mapa\_Assento:** (classe que armazena todos os assentos e as trancas das mesmas)

Variáveis:

tranca – vetor de locks, um para cada assento

assento – vetor de assentos

**Classe Monitor:** (classe em que está armazenado todas as informações dos assentos)

Variáveis:

mapa – mapa de assentos

qtdAssento – quantidade de assentos

senha – senha que será entregue ao usuário para que ele entre na fila do buffer

id\_aleatorio – variável inteira que recebe um número aleatório dentro da quantidade de cadeiras para que o usuário consiga reservar um assento aleatório

primeiro – booleana que serve para ajudar a escrever o mapa de assentos no buffer

retorno – variável que foi utilizada como vetor para que a função ocupaVazio conseguisse retornar 2 valores

Funções:

getMapa – função que retorna o mapa de assentos

verifica – função retorna uma senha

ocupaVazio – o usuário acessa essa função para ocupar um assento aleatório, gera um número aleatório dentro da quantidade de assentos, atualiza o assento e o mapa de assentos (o assento recebe o estado de ocupado e recebe o número do id do usuário) caso o assento não esteja ocupado por outro usuário, retorna uma senha e o número do assento para o usuário

ocupaAssento – o usuário acessa essa função para ocupar um assento determinado por ele, caso o assento estiver livre o usuário ocupa o assento, o assento e o mapa de assentos são atualizados (o assento recebe o estado de ocupado e recebe o número do id do usuário), caso contrário o usuário não ocupa o assento, retorna uma senha para o usuário

cancela – usuário acessa essa função para cancelar uma reserva de assento que possui o id do usuário, retorna uma senha para o usuário

Classe Pair: (classe criada para organizar a forma com que o roteiro funciona)

Variáveis:

l – representa a função que será executada pelo usuário (1-verifica/ 2-ocupaVazio/ 3-ocupaAssento/ 4-cancela)

r – representa o assento que o usuário irá reservar

## Considerações Finais

De fim, aprendemos muito sobre como o Java lida com concorrência, e quanto ele ajuda, porque enquanto fazíamos exemplos já montados não tinham alguns empecilhos como lidar com os locks ou até mesmo saber em quem atrelar o monitor.

Além disso, fizemos o trabalho em grupo, o que fez com que a modelagem passasse por vários passos de conversa e modificação. Não pudemos documentar cada modelagem, porque é uma fase difícil do período, mas, nos reunimos várias vezes para repensar em como a solução seria feita.