RELATÓRIO

PROJETO FINAL SOPE
14 – 05 – 2018

Sofia Cardoso Martins up201606033
Margarida Ramos Pereira Silva up201606214

ESTRUTURA DE MENSAGENS TROCADAS ENTRE CLIENTE E SERVIDOR

1. cliente → servidor

O cliente deverá informar o servidor acerca:

- do seu pid
- do número de lugares que pretende reservar
- dos seus lugares de preferência

Tendo em vista esta necessidade, foi utilizada uma struct client_request. Os campos da struct são preenchidos para cada um dos clientes, sendo esta enviada através do fifo requests para ser posteriormente recolhida pelo servidor.

```
struct client_request {
int client_pid;
int num_wanted_seats;
int preferences[MAX_CLI_SEATS];
};
```

2. servidor → cliente

O servidor poderá dar uma resposta ao cliente sempre que não ocorra um timeout do mesmo. Quando isto acontece, a resposta deverá ser constituída pelos seguintes parâmetros:

- estado da reserva: o estado pode variar entre -1 e -6 (valores negativos denunciam a ocorrência de um erro e zero indica que a reserva foi bem sucedida).
- sequência de números inteiros em que o primeiro indica quantos lugares foram reservados e os seguintes constituem os números desses lugares.

```
struct server_answer {
int state;
int seats[MAX_CLI_SEATS+1];
};
```

MECANISMOS DE SINCRONIZAÇÃO UTILIZADOS

De modo a assegurar a sincronização no acesso aos lugares (seats), são utilizados **mutexes** (um por cada lugar). Assim, sempre que uma thread precisa de obter o estado de um lugar (disponível ou indisponível), reservá-lo ou libertá-lo, terá de aguardar o desbloqueamento do mutex respetivo. Esta estratégia assegura que as múltiplas threads possam aceder a um lugar, desde que o mesmo lugar não esteja a ser acedido por uma outra. Por outro lado, há que garantir que a escrita nos ficheiros não possa ser feita for múltiplos processos em simultâneo. Como tal, utilizaram-se quatro **mutexes**, um por cada ficheiro de texto. Um processo tentará adquirir o mutex anteriormente à escrita no ficheiro e libertá-lo-á quando terminar essa operação.

Foi-nos também possível concluir que a receção de reservas pelo servidor e aquisição das mesmas pelas bilheteiras constitui um problema do tipo produtor-consumidor, no sentido em que o servidor pode alertar as bilheteiras para a existência de uma reserva ainda não processada, e só então uma das bilheteiras se poderá encarregar de processá-la. Atendendo a isto, optámos por usar dois **semáforos** (full e empty). Assim, o main thread aguarda que o semáforo empty esteja a zero para poder tentar recolher do fifo requests uma nova reserva, e assim que o fizer, incrementa o semáforo full. Cada uma das threads bilheteira tenta primeiro adquirir o **mutex** buffer_lock e só depois aguarda que o semáforo full esteja a um, momento em que poderá recolher uma reserva colocada num buffer e incrementar o semáforo empty. O mutex buffer_lock é utilizado para garantir que à chegada de uma nova reserva (semáforo full igual a um) apenas uma bilheteira livre tenha oportunidade de processá-la, evitando assim que múltiplas threads estejam a competir por uma mesma reserva.

ENCERRAMENTO DO SERVIDOR

O encerramento do servidor é feito assim que termina o tempo de funcionamento das bilheteiras. Para a contagem desse tempo foi utilizada a chamada do sistema **alarm**() e instalado um signal handler para o sinal SIGALRM. A contagem do tempo inicia-se assim que as bilheteiras são colocadas em funcionamento (pthread_create()).

No signal handler (alarmHandler) são executadas as seguintes operações:

- envio do **sinal SIGTERM** a todas as threads bilheteira (pthread_kill)
- espera (através de pthread_join) pelo término da execução de todas as threads bilheteira (nota: relembrar que as bilheteiras que estão a executar um pedido não terminam de imediato. Estas apenas cessam funções quanto acabam de executar o pedido que estava ativo aquando da receção do sinal SIGTERM).
- escrita da última linha do ficheiro slog.txt: SERVER CLOSED

Para o controlo do fecho das bilheteiras foi instalado um signal handler para as threads: termHandler. Dentro desta função é colocada a true a **flag** closeTicketOffices, e é testado se a thread estava a processar uma reserva aquando da receção do sinal SIGTERM; se tal acontecer, a thread deverá retornar do signal handler e retomar o processamento (sendo que quando o terminar deverá ser impedida de receber uma nova reserva, através da já referida flag closeTicketOffices). Após esta ocorrência, a thread deverá registar o seu fecho no ficheiro slog.txt e terminar. Estas últimas operações são também realizadas pelas threads que não estavam a processar nenhuma reserva.