Estrutura das mensagens trocadas entre:

A comunicação entre clientes e servidor é feita através de *FIFO*s. O servidor receberá os pedidos dos clientes através de um *FIFO* comum, de nome **requests**, e cada cliente receberá a resposta do servidor através de um *FIFO* dedicado, de nome **ansXXXXX**, em que **XXXXX** representa o *PID* do cliente.

- **Clientes->servidor**

Tem um *FIFO* dedicado, de nome **ansXXXXX** do cliente através do qual receberá as respostas aos pedidos de reserva enviados ao servidor, através do *FIFO* **requests**.

O cliente envia ao servidor, através do *FIFO* **requests**, um pedido de reserva com os seguintes dados: *PID* do cliente, número de lugares a reservar e lista de lugares preferidos.

O cliente aguarda, no *FIFO* **ansXXXXX**, pela resposta do servidor que indica a resposta á reserva. Se a resposta não for recebida dentro do tempo **time_out** o cliente terminará a execução.

- **Servidor ->clientes**

Cria o *FIFO* **requests** através do qual recebe os pedidos de reserva de lugares. As respostas aos pedidos são dadas através de *FIFO*s dedicados, criados pelos clientes(**ansXXXXX**).

Caso o pedido seja inválido, a resposta equivale a um número negativo representativo do motivo pelo qual o pedido não foi executado. Caso o pedido seja válido, a resposta é a lista de lugares que foram resevados para o cliente.

Mecanismos de sincronização utilizados

Para a sincronização das threads criamos três mutexes. Uma delas serve para que as threads não escrevam ao mesmo tempo no ficheiro de texto “slog.txt”. A segunda mutex serve para que apenas uma thread de cada vez possa receber o próximo pedido da fifo **requests**, evitando que o mesmo pedido seja executado mais do que uma vez. Por último, temos uma thread para a reserva de lugares, evitando que duas threads façam reservas ao mesmo tempo, o que poderia ser problemático caso os pedidos executados incidissem nos mesmos lugares.

```
pthread_mutex_t mut=PTHREAD_MUTEX_INITIALIZER; //to receive order
pthread_mutex_t mut2=PTHREAD_MUTEX_INITIALIZER; //to make reservation
pthread_mutex_t mut3=PTHREAD_MUTEX_INITIALIZER; //to write on file
```

Exemplo da utilização de uma mutex:

```
pthread_mutex_lock(&mut2);
int invalid = isReservationValid(order1, o_size);
pthread_mutex_unlock(&mut2);
```

Encerramento do servidor

O cliente aguarda, no *FIFO* **ansXXXXX**, pela resposta do servidor, que lhe indicará se a reserva foi concretizada com sucesso ou não. Se a resposta não for recebida dentro do tempo especificado pelo cliente em **time_out,** o cliente terminará a execução.

O servidor termina a sua execução quando passa o tempo recebido como **time_out**. Quando isto acontece, existe uma flag que fica a 1, que indica que o programa tem de terminar. Quando a flag é ativada, as threads não podem ler um novo pedido, acabando a sua execução assim que terminarem com o pedido atual. Com a função pthread_join(), o programa fica à espera que as threads terminem. É chamada a função que escreve no ficheiro de texto "sbook.txt" os lugares reservados por clientes e termina o programa.

```
flag = 1;
for(int n = 0; n < num_ticket_offices; n++)
    pthread_join(tids[n], NULL);

writeBookingsFile();
close(requests);
unlink("requests");
free(seats);
return 0;
```

Trabalho realizado por:

- Eduardo Silva
- Helena Montenegro
- Juliana Marques