Trabalho Prático Nº2
Simulação de um sistema de reserva de lugares

Sistemas Operativos 2017/2018

Francisco Friande - up201508213
João Reis - up201203562
Pedro Silva – up201604470

Comunicação entre Client e Server

Como o servidor lê informação de todos os clientes através de um único fifo, de nome "requests" criado em "/tmp/" dado o seu carater temporário.

Tendo de enviar um pedido com vários elementos, passar cada um destes em escritas ao fifo diferentes, o programa ficaria vulnerável a escritas intercaladas de clientes diferentes e, por tal, corrupção de informação. Para evitar isso, optámos por criar um buffer que cada client cria para guardar toda a informação relativa ao pedido para apenas realizar uma escrita por pedido.

A estrutura deste buffer segue a orientação dos pedidos no enunciado:

"<PID_Client> <Num_Seats> <List_Of_Wanted_Seats>"

Onde:

- "PID_Client" é o PID do processo client que executa o pedido.
- "Num_Seats" é o número de lugares que o cliente deseja reservar.
- "List_Of_Wanted_Seats" é a lista com os números dos lugares das quais o cliente pretende reservar "Num_Seats lugares". Cada lugar está separado por um espaço.

Por exemplo:

"16413 2 12 13 64 100" significaria que o cliente com PID 16413 desejar reservar 2 dos lugares com identificador referido a seguir (12, 13, 64 ou 100).

```
// Request Goal: "<PID_Client> <Num_Seats> <List_Of_Wanted_Seats>"
char buffer[BUFFER_SIZE];
int n = 0;
// Create a default Request: "<PID_Client> <Num_Seats> 0"
snprintf(buffer, BUFFER_SIZE, "%d %d %n", getpid(), num_wanted_seats, &n);
// Add Seats: "<PID_Client> <Num_Seats> s_list[0] s_list[1] (...) s_list[num_pref_seat]"
for (int i = 0, c; i < num_pref_seat; ++i, n += c)
    snprintf(buffer + n, BUFFER_SIZE - n, "%d %n", s_list[i], &c);
//Request separator since the fifo itself works as a buffer to server
strcpy(&buffer[n - 1], "\n");
```

Comunicação entre Server e Client

A comunicação de resposta do servidor para o cliente é realizado através de fifos especificos de cada cliente, de nome ansXXXXX, onde XXXXX representa o PID do cliente.

Caso a reserva ocorra com sucesso, o servidor escreverá no fifo correspondente ao cliente em questão, o número de lugares que reservou e quais os seus identificadores.

"<Num_Seats> <Seat(0)> <Seat(1)> (…) <Seat(Num_Seats -1)"

Em caso de erro, o servidor escreverá nesse mesmo fifo a razão do erro:

"-1"	Quantidade de lugares pretendidos é superior ao permitido.
"-2"	Número de lugares pretendidos menor do que o número de identificadores dados.
"-3"	Identificadores dos lugadores pretendidos não são válidos
"-4"	Outros erros nos parâmetros
"-5"	Não conseguiu reservar o número de lugares pretendidos
"-6"	Sala cheia

Mecanismos de sincronização

Para evitar problemas de sincronização recorremos ao uso de mutexes.

Ao inicializar a estrutura de dados para os lugares (Array de structs Seat), é atribuiido a cada lugar um valor de um mutex previamente criado.

A rodear as secções criticas (quando o estado do array "seats" é verificado ou editado), existe uma chamada para bloquear o acesso ao Seat **"pthread_mutex_lock(&seats[seatNum].mut);"** e uma para desbloquear o lugar para poder ser usado por outras threads **"pthread_mutex_unlock(&seats[seatNum].mut);"**.

No entanto, como a reserva (bookSeat) apenas é feita depois de verificar se o lugar está vazio (isSeatFree), a função isSeatFree não desbloqueia o Seat caso este esteja livre para evitar que outra bilheteira reserve esse lugar.

```c
int isSeatFree(Seat *seats, int seatNum) {
    if (!seats)
        return -1;
    pthread_mutex_lock(&seats[seatNum].mut);
    DELAY();
    int result;
    if (seats[seatNum].free)
        result = 1;
    else
        result = 0;

    pthread_mutex_unlock(&seats[seatNum].mut);
    return result;
}
```

```c
void bookSeat(Seat *seats, int seatNum, int clientId) {
    if (!seats || !seats[seatNum].free) {
        pthread_mutex_unlock(&seats[seatNum].mut);
        return;
    }
    seats[seatNum].free = false;
    seats[seatNum].clientId = clientId;
    DELAY();
    pthread_mutex_unlock(&seats[seatNum].mut);
    return;
}
```

Encerramento do servidor

Ao encerrar o servidor o thread principal termina todas as threads em execução e guarda em sbook.txt os lugares reservados, um por linha.