13 de maio 2018

**Faculdade de Engenharia da Universidade do Porto**


# *Sistemas Operativos*

# ***Simulação de um sistema de reserva de lugares***

**SOPE 2º ano – MIEIC**
Jorge Alves da Silva
Pedro Miguel Moreira da Silva

**Turma 2MIEIC04 – Grupo: 04**
César Pinho up201604039@fe.up.pt
Nuno Martins up201605945@fe.up.pt
Rui Guedes up201603854@fe.up.pt

# Simulação de um sistema de reserva de lugares

No presente relatório encontra-se a seguir abordagem efetuada para cada um dos temas mencionados acompanhados quando necessário de excertos com código.

## → Estrutura das mensagens trocadas entre cliente e servidor

Para a realização deste trabalho foi necessário estipular o tipo de mensagens a serem trocadas entre o cliente e servidor (e vice-versa). A comunicação em ambos os casos é semelhante no entanto a informação contida nos dois tipos é diferente

### → Comunicação cliente-servidor

A comunicação nesta direção consiste na criação de uma string que por sua vez irá conter ordenadamente: *PID* do cliente, número de lugares que este pretende reservar, e a lista de preferências de lugares. Cada um destes elementos da string é separado por um espaço sendo que o servidor por sua vez quando recebe a string decompõe-na e guarda os seus devidos valores podendo então verificar a resposta a ser efetuada ao respetivo pedido.

### → Comunicação servidor-cliente

A comunicação nesta direção é semelhante à anteriormente especificada, no entanto, nesta direção o servidor cria uma string com elementos diferentes sendo eles ordenadamente: *PID* do cliente, valor indicador do estado do pedido, e em caso de pedidos efetuados com sucesso é adicionada também a lista dos lugares efetivamente reservados. Assim como o servidor o cliente possui uma função que decompõe a string e guarda os devidos valores agindo depois em conformidade com o tipo de resposta recebida.

## → Mecanismos de sincronização

Relativamente aos mecanismos de sincronização, na realização deste trabalho, foram usados os seguintes: semáforos, mutexes e variáveis de condição. Estes mecanismos foram utilizados em duas etapas distintas do programa:

→ Primeiramente são utilizados semáforos que permitem por sua vez estabelecer sincronismo entre a receção de pedidos e atribuição de pedidos a uma das bilheteiras (*threads*) disponíveis. Isto é feito recorrendo a dois semáforos – *empty* e *full* – em que a alternância entre valores (0 e 1) permite reconhecer a existência de pedidos ou não. Inicialmente o semáforo *empty* encontra-se a 1 e o semáforo *full* a 0, ficando as *threads* bloqueadas à espera de pedido. Aquando a receção de um pedido e colocação do mesmo no *buffer* unitário os valores dos semáforos invertem-se e uma das *threads* assume o pedido voltando os semáforos ao estado inicial. Este ciclo repete-se até que o tempo de abertura do servidor termine.

```
//Main thread is responsible to listen client requests
while( ((double)(clock() - begin) / CLOCKS_PER_SEC) < atoi(argv[3])) {

  int sem_value;
  sem_getvalue(&empty, &sem_value);

  if(sem_value == 1) {
    if(read(requests_fd, request, sizeof(request)) > 0) {
      printf(BOLDBLUE "Received request :: " BOLDYELLOW "%s\n" DEFAULT, request);
      sem_wait(&empty);
      sem_post(&full);
    }
  }
}
```

→ Numa fase mais avançada utilizam-se os três mecanismos de sincronização, isto é, após validação do pedido as *threads*, em caso de pedidos válidos, iniciam um *loop* no qual se irá tentar efetuar a reserva pretendida. Uma vez que múltiplas *threads* estão a efetuar este mesmo código recorreu-se às variáveis de condição em conjunto com *mutexes* para garantir, na respetiva secção critica, que o lugar que está a ser alugado não está a ser acedido por mais nenhuma *thread*, caso contrário a *thread* bloqueia e fica à espera que o lugar seja libertado por outra *thread* que lhe envia o sinal. De seguida são efetuadas as chamadas às funções *isSeatFree*, *bookSeat* em que a segunda só chamada caso a primeira retorna verdadeiro. Estas funções são responsáveis respetivamente por verificar se o lugar está livre e por reservar um lugar. Numa fase posterior caso a reserva não tenha sido efetuada com sucesso os lugares previamente reservados são libertados com recurso à função *freeSeat*. Estas três funções com recurso às mutexes possuem uma secção critica na qual executam o seu devido código sendo ainda efetuado um *delay* no final da secção com vista a garantir a uniformidade e segurança das operações de escrita e leitura do estado da sala (*array* de lugares).

```
//Loop local variables
int seat_number = request_info.pref_seat_list[pref_seat_list_pointer];
int access_status = seats[seat_number].access_status;

//Checks if the wanted seat is being accessed for other thread
pthread_mutex_lock(&access_lock);

while(access_status)
  pthread_cond_wait(&room_access_cond[seat_number], &access_lock);

seats[seat_number].access_status = 1;

pthread_mutex_unlock(&access_lock);

//Try to reserve the wanted seat
if(isSeatFree(seats, seat_number)) {
  bookSeat(seats, seat_number, request_info.client_pid);
  reserved_seats[reserved_seats_pointer] = seat_number;
  reserved_seats_pointer++;
  num_wanted_seats--;
}

//Free access to all pref_seat_list seats
seats[seat_number].access_status = 0;
pthread_cond_signal(&room_access_cond[seat_number]);
```

Todos os mecanismos de sincronização utilizados são partilhados por todas as threads incluindo obviamente a main thread tornando-se assim possível garantir a sincronização das mais diversas ações presentes no programa e anteriormente referidas.

## → Encerramento do servidor

O servidor encontra-se em funcionamento durante o tempo especificado na chamada à função *server*. Durante este tempo o servidor é responsável por ficar à espera de possíveis pedidos provenientes de clientes. Assim que este tempo é ultrapassado o servidor fecha e destrói imediatamente o fifo "*requests*", sinaliza todas as threads que devem terminar e destrói todas as estruturas de sincronização, fecha os seus devidos ficheiros e liberta espaço reservado até então. A terminação das threads não ocorre instantaneamente uma vez que podem existir pedidos a serem tratados. Se assim for o "main thread" (servidor) espera que que estas acabem de tratar os seus pedidos e só depois terminam. As threads que se encontravam à espera de pedidos essas sim terminam imediatamente após o tempo especificado ser ultrapassado. Desta forma é possível garantir então que não existam pedidos que ficam a meio do seu tratamento na medida em que cada pedido efetuado dentro do tempo de abertura das bilheteiras (threads) é analisado e concluído.

```
    //Terminates all threads after they execute their own requests
    terminateAllThreads(atoi(argv[2]));

    //Wait's for all threads
    for(int i = 1; i <= atoi(argv[2]); i++)
        pthread_join(thread_ids[i], NULL);

    printf(BOLDMAGENTA "Server :: " BOLDGREEN "Ticket offices handled all remaining requests\n" DEFAULT);

    //Prints information on sbook_file
    printServerBookings();

    //Prints last message on slog_file
    fprintf(slog_file, "SERVER CLOSED\n");

    return terminateServerProg(requests_fd);
}
```