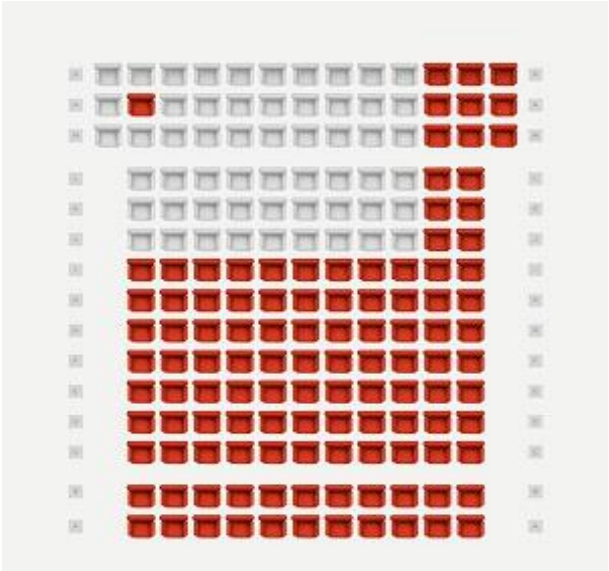
U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Projeto de SOPE

Simulação de um sistema de reserva de lugares



Projeto de SOPE 2018 -- Engenharia Informática e Computação

Regente: Jorge Alves da Silva

Turma 2

Grupo 7

- Amadeu Prazeres Pereira --- up201605646 --- up201605646@fe.up.pt
- Nuno Tiago Tavares Lopes --- up201605337 --- up201605337@fe.up.pt
- Tomás Nuno Fernandes Novo --- up201604503 --- up201604503@fe.up.pt

Supervisor: José Manuel de Magalhães Cruz

Maio, 2018

Introdução

No âmbito da unidade curricular Sistemas Operativos, foi-nos proposto a realização de um projeto com o objetivo de simular um sistema de reserva de lugares.

A concretização foi possível devido aos conhecimentos adquiridos nas aulas teóricas e práticas acerca da gestão de processos e *threads* e da utilização de mecanismos de comunicação e de sincronização entre os mesmos, tudo isto programando em ambiente Unix/Linux.

O projeto consiste na criação de dois programas: um *server* e um *client.* O segundo vai simular os pedidos de um utilizador e o primeiro vai processá-los, sendo que estes pedidos abrangem a quantidade de lugares pretendidos pelo cliente e uma lista de identificadores de lugares pretendidos. O servidor, tendo em conta as preferências do cliente, tentará satisfazer o seu pedido, dando-lhe uma resposta adequada.

Desenvolvimento do Projeto

O programa *server* é constituído por bilheteiras, que são simulados por *threads.*

Objetivando o bom funcionamento de ambos os programas, o cliente e o servidor trocam mensagens. O *client,* através de um *FIFO,* de nome *requests,* criado pelo server, envia a *struct Request* para o servidor.

```
struct Request {
  int processed;
  int clientID;
  int num_wanted_seats;
  int num_prefered_seats;
  int prefered_seats[MAX_PREFERED_SEATS];
};

write (requests, &request, sizeof(struct Request));
```

Por sua vez, o *server* envia para o cliente ou um valor inteiro negativo, no caso da ocorrência de erros ou um valor inteiro positivo, sendo que este valor se trata do número de lugares reservados seguido dos números dos lugares reservados.

```
if (ret < 0) {
    write(fd, &ret, sizeof(int));
}
else {
    write(fd, &r.num_wanted_seats, sizeof(int));
    int i;
    for (i = 0; i < r.num_wanted_seats; i++) {
           write(fd, &reserved_seats[i], sizeof(int));
    }
}
```

Os mecanismos de sincronização de utilizados foram os *mutexes*. Usufruímos duas vezes deste tipo de mecanismo no programa *server:* uma para sincronizar as *threads* com a análise dos pedidos e outra para sincronizar as *threads* com a escrita nos ficheiros.

```
pthread_mutex_t mut = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mut_files = PTHREAD_MUTEX_INITIALIZER;
```

Após ter passado o tempo definido pelo *open_time* no servidor é gerado um *alarm* que interrompe a leitura do FIFO. O servidor aguarda que threads terminem os seus processos, os ficheiros sejam escritos, e que o *FIFO requests* e os mutexes sejam destruídos. Só depois destes acontecimentos é que o servidor encerra o seu funcionamento.

```
//Waiting for the threads to finish
for(i = 0; i < num_ticket_offices; i++) {
    pthread_join(tid[i], NULL);
    slogOpenClose(i+1, 0);
}
//Destroying FIFO requests
if(destroyFIFO(FIFO_SERVER) != 0) {
    return 4;
}
//Destroying mutexes
pthread_mutex_destroy(&mut);
pthread_mutex_destroy(&mut_files);
```

Conclusão

Com este projeto foi então possível aumentar os nossos conhecimentos sobre programação em Unix/Linux, bem como melhorar o nosso código, visto que demos uso ao que aprendemos tanto nas aulas teóricas como nas práticas.

Concluímos assim que este trabalho foi benéfico para todos os elementos.