Universidade do Porto

Faculdade de Engenharia

FEUP

Simulação de um sistema de reserva de lugares

Sistemas Operativos

2º ano do Mestrado Integrado em Engenharia Informática e Computação

Anderson Rogério da Silva Gralha - up201710810

Arthur Johas Matta - up201609953

Fernando Oliveira - up201005231

13 de maio de 2018

Mensagens Cliente - Servidor

- Cliente - Servidor

Para a troca de mensagens entre o cliente e servidor é aberto o fifo 'requests' e enviada a mensagem relacionada a requisição. O formato enviado para o servidor é [PID NUM_WANTED_SEATS PREF_SEAT_LIST], onde PID é o pid do cliente, NUM_WANTED_SEATS é a quantidade de lugares desejados, e PREF_SEAT_LIST é a lista de lugares que o cliente aceita que sejam atribuídos a ele. Caso o número de lugares disponíveis dentre os especificados em PREF_SEAT_LIST seja menor que NUM_WANTED_SEATS, nenhum lugar é atribuído.

```
int fdRequest;
...
do {
   fdRequest = open("requests", O_WRONLY);
   if (fdRequest == -1) sleep(1);
} while (fdRequest == -1);

sprintf(message, "%d %s %s", pid, argv[2], argv[3]);
...
write(fdRequest, message, messagelen);
```

Para que seja recebida a resposta, é criado um fifo no formato ansPID, onde PID é o pid do cliente que fez a requisição. A espera por esta resposta é de no máximo TIMEOUT segundos. Este valor é passado na criação do client.

- Servidor - Cliente

Para a troca de mensagens entre servidor e cliente, é enviada uma mensagem ao fifo "ansPID", onde PID é o id do processo do cliente, contendo:

- ➢ Um código de erro em caso de fracasso:
  - -1 → a quantidade de lugares pretendidos é superior ao maximo permitido
  - -2 → o número de identificadores dos lugares pretendidos não é válido
  - -3 → pelo menos um dos identificadores dos lugares pretendidos não é válido
  - -4 → outros erros nos parâmetros
  - -5 → pelo menos um dos lugares pretendidos não está disponivel
  - -6 → sala cheia

- Uma mensagem no formato [<Número de lugares desejados> <Lista de lugares reservados>]

Sincronização

Para efeitos de sincronização de processos, foram utilizados dois mutexes, um para acessar o buffer de requisições, outro para acessar a lista de assentos da sala.

O processo principal, ao receber uma requisição, a coloca em um buffer e sinaliza os demais processos. Todos os processos auxiliares, ao iniciarem, bloqueiam o primeiro mutex, verificam se há alguma requisição e, se houver, um processo coleta a mesma, desbloqueia o mutex, e segue lidando com aquela requisição; os demais processos desbloqueiam o mutex e ficam a espera da sinalização do processo principal. Essa sinalização é feita através de variáveis de condição.

Após coletar uma requisição, o processo auxiliar que a coletou verifica se a mesma é válida e em caso afirmativo, prossegue com a reserva; em caso negativo, envia ao cliente um código indicando o erro. A reserva é feita bloqueando o mutex da lista de assentos, tentando reservar um assento, e desbloqueando o mutex. Repete-se este ciclo até se reservar o número de assentos desejados ou terminar de percorrer a lista de assentos preferidos. Se o número de assentos reservados for inferior ao número de assentos desejados, a reserva é cancelada e os assentos são liberados, com o diferencial que todos os assentos são liberados de uma vez, ao contrário do ciclo anterior.

Encerramento do servidor

Primeiramente, fecha-se e destrói-se o fifo responsável por receber as requisições. Em seguida, ordena-se que as threads criadas sejam canceladas e aguarda-se até que as mesmas sejam canceladas. Para efeito de consistência, uma thread desabilita temporariamente comandos de cancelamento enquanto está a executar uma reserva (Ver excerto de código abaixo). Caso uma thread receba um comando de cancelamento enquanto os mesmos estão desabilitados, o comando é posto em uma pilha até que cancelamentos sejam novamente habilitados. Por fim, destrói-se os mutexes utilizados para sincronização de processos, e salva-se os assentos reservados no arquivo "sbook.txt".

```
// Terminate threads
for (int i = 0; i < num_tickets_offices; i++) {
      pthread_cancel(tids[i]);
      pthread_join(tids[i], NULL);
```

```
}
```

Código: Cancelamento de cada uma das threads criadas.

```
// Disable cancellation until operation is done
if(pthread_setcancelstate(PTHREAD_CANCEL_DISABLE, NULL)){
      fprintf(stderr, "[TICKET OFFICE %d]: Error setting cancellation
state. Ignoring request\n", tnum);
      continue;
}

...

// Enable cancellation
if(pthread_setcancelstate(PTHREAD_CANCEL_ENABLE, NULL)){
       fprintf(stderr, "[TICKET OFFICE %d]: Error enabling cancellation
state. Exiting\n", tnum);
       pthread_exit(NULL);
}
```

Código: Desabilitação e habilitação de cancelamento das threads.