U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Mestrado Integrado em Engenharia Informática e Computação

Sistemas Operativos

Simulação de um sistema de reserva de lugares

Implementação de uma arquitetura cliente/servidor baseada em FIFOs

Grupo 3, turma 3

Ângelo Miguel Tenreiro Teixeira, up201606516@fe.up.pt
Henrique Melo Lima, up201606525@fe.up.pt
Guilherme José Ferreira do Couto Fonseca da Silva, up201603647@fe.up.pt

13 de Maio, 2018

Os lugares (Seats)

Para guardar os lugares, usamos um array de Seats (ints), que caso estejam livres, terão o valor 0, caso não estejam terão o valor igual ao PID do processo cliente que os reservou. Desta forma sabemos facilmente se um determinado Seat está reservado ou não, e, caso esteja, a quem pertence.

Sincronização entre Threads (bilheteiras) na leitura de pedidos

Para impedir que duas thread processem o mesmo request, usamos um mutex (readRequestsMutex). Na função executada pelas threads (initTicketOffice()), é chamada uma função processClientMsg(), responsável por ir buscar o request do cliente, caso esteja disponível. Nesta função é usado o mutex para que apenas uma thread leia o FIFO dos pedidos de cada vez, não havendo problemas de sincronização ou informação perdida. Assim que um request é lido, o mutex é desbloqueado, para que outra thread possa aceder ao FIFO.

Sincronização entre Threads (bilheteiras) no fecho

Quando termina o tempo de execução do servidor, o processo contador envia um sinal (SIGUSR1) ao processo que está efetivamente a realizar o trabalho de servidor, e que contém as threads. Como não é possível saber qual thread recebe o sinal, usamos uma variável boleana global (g_tickets_are_open) que é iniciada a 1. Assim que o sinal é recebido por qualquer thread, o seu valor é colocado a 0. Para que haja sincronização na leitura do seu valor, é usado um mutex (ticketOfficeMutex) que impede as threads de acederem ao seu valor enquanto uma outra o poderá modificar, fazendo com que nenhum pedido seja processado após o fecho das bilheteiras.

Na função principal executada por cada thread, é executado um ciclo que depende da variável (while(g_tickets_are_open)) no qual é chamada a função de processamento dos requests. Nesta função, é novamente verificado o valor da variável, para garantir que não foi alterado entretanto.

Sincronização entre Threads (bilheteiras) na alocação de lugares

Uma vez que o programa server (responsável por gerir os lugares) vai ter múltiplas threads em execução ao mesmo tempo, é necessário um mecanismo de sincronização entre threads, de forma a que a informação esteja atualizada em cada uma delas. Para isto, cada Seat terá um semáforo com nome (do tipo “/semx”, em que x representa o número do lugar), que é apenas criado da primeira vez que é necessário aceder à informação daquele lugar. Desta forma, só no caso de tentativa de reserva para todos os lugares é que são criados tantos semáforos quantos lugares existentes.

De cada vez que uma thread quiser aceder à informação de um Seat, caso o semáforo respetivo já exista, é aberto e, se alguma outra thread estiver a modificar esse seat, a primeira espera por esta. Caso não exista ainda semáforo associado, este é criado e a thread executa logo (é usada a macro O_CREAT, para que o semáforo seja criado se não existir).

Sincronização entre cliente e servidor

Ao desenvolver a aplicação, reparámos que, ao dar um timeout reduzido ao cliente, o servidor por vezes conseguia abrir o FIFO de resposta, mas já não conseguia escrever, porque entretanto o cliente o fechava, por não obter resposta até àquele instante. Para resolver este problema, usamos também semáforos com nomes do tipo "/semAnswerx", em que x representa o PID do cliente a responder. Desta forma, antes do servidor abrir o FIFO de resposta, invoca sem_wait() e só o liberta após a escrita, fazendo com que o cliente não feche o FIFO antes do servidor escrever nele, caso já tenha a resposta pronta, pois também aqui é invocada sem_wait().

Graceful Shutdown

Quando o tempo (timeout) do cliente expira, o "pai", que conta o tempo, envia um sinal para o filho terminar. Contudo, uma vez que este poderia estar a meio de processar uma resposta do servidor, é necesário usar um semáforo com nome (é usado o mesmo que anteriormente - /semAnswerx) sendo que, quando o cliente "filho" vai ler uma resposta, invoca sem_wait(), sendo que o "pai" faz sem_wait() antes de matar o filho quando acaba o tempo. Desta forma, o filho não é terminado a meio da leitura de uma resposta no fim do tempo de execução.