Relatório projeto final

T4G07

Troca de mensagens entre clientes e servidor (e vice-versa)

Para efeitos de comunicação entre o cliente e servidor, foi implementado tudo como estava descrito no enunciado do projeto: O servidor cria um fifo no path indicado (tmp/secure_srv) com permissão de leitura e escrita para todos e, de seguida abre-o para leitura. Após o user ter aberto este fifo para escrita, o server é desbloqueado e prossegue para a leitura constante (até receber ordens em contrário) da informação do fifo. Após receber um request valido por parte de algum user, o servidor trata do request, cria um fifo destinado ao envio do reply para o user (como esta descrito no enunciado), envia o reply, e, após isso fecha o fifo.

Para a comunição no sentido contrario, o programa user abre o fifo (já anteriormente criado pelo servidor) para escrita, e depois prossegue para a escrita do request no mesmo. Após isto, o user fica à espera (no máximo 30 segundos) pelo reply do servidor. Caso obtenha a resposta, regista-a no ficheiro ulog.txt e depois termina. Caso contrário, após os 30 segundos, regista no ulog.txt SRV_DOWN e termina.

Mecanismos de sincronização utilizados

Os longo de todo o projeto, utilizamos mutexes e *condition variables* para efeitos de sincronização entre threads.

Para o tratamento de pedidos, estamos a utilizar *condition variables*, em conjunto com um mutex e uma fila (para se manter a ordem de chegada). A nossa implementação (baseada no problema produtor-consumidor) consiste em: manter os Bank offices (threads) adormecidos (em espera não ativa), para evitar o uso de recursos do CPU enquanto eles não têm nenhum request a tratar. Sempre que é adicionado um request à fila, é enviado um sinal (pthread_cond_signal) para acordar um deles para que ele trate do request. Um dos mutexes utlizados nesta fase serve para garantir que apenas um thread acede à fila de cada vez. É também usado um outro mutex para o incremento e decremento de uma variável utilizada para contar o número de threads que estão a dormir, ou seja, de bank offices que não estão a tratar de nenhum request. É importante salientar também que, na nossa implementação, um thread só é adormecido se após ter terminado a sua tarefa, não houver mais nenhuma na fila de espera, caso contrário, este trata imediatamente da tarefa que se encontra à frente da fila.

Encerramento do servidor

Esta funcionalidade, como é descrito no enunciado, só pode ser executada pelo administrador do servidor (id = 0) e, no nosso caso, este tipo de request não é enviado para a fila de requests, uma vez que é tratada logo pelo servidor. No caso do request enviado para o servidor ter sido corretamente autenticado (account id correspondente com a pass) e o id da conta ser do admin, o servidor inicia o encerramento do mesmo, deixando de ler mais requests e mudando o fifo de comunicação entre o cliente e o servidor para leitura apenas (para impedir o envio de mais requests). Após isto, o programa do servidor prossegue para uma função chamada waitForAllThreads que contem ciclo while onde espera que todos os threads acabem os requests que se encontram em fila e adormeçam (devido à ausência de pedidos que, nesta fase já foram todos tratados) para enviar um sinal para todos eles (pthread_cond_broadcast) e espera que os mesmos acabem com pthread_join.

```
void waitForAllThreads(pthread_t listTids[], size_t nBankOffices)
{
    while (nThreadsWaitingForRequests != nBankOffices) //Waits for the threads to finish their work
    {
        sleep(1);
    }

    pthread_cond_broadcast(&requestAdded);

    for (size_t i = 1; i <= nBankOffices; i++)
    {
        pthread_join(listTids[i], NULL);
    }
}
```