Sistemas Operativos

Relatório do Trabalho n.º 2 - Grupo de trabalho T2G04

Estrutura das Mensagens

Cliente – Servidor / Servidor - Cliente

Após o pedido do utilizador ser processado, usa-se uma `struct tlv_request_t` para que a mensagem seja pedida, como o nome dá a entender, no formato **`TLV`**. Para tal, esta necessita de ser previamente preenchida, sendo o atributo do tipo `req_value_t` responsável por conter as informações essenciais na forma da `struct req_header_t` e, dependendo da operação pedida (criação de conta, transferência, consulta de saldo ou encerramento do servidor), uma ou nenhuma (union) de entre `req_create_account_t` ou `req_value_t`.

Posteriormente, a `struct` inicial será enviada ao servidor através de um **`FIFO`** (named pipe):

```
int writeRequest(tlv_request_t *tlv)
   {
       tlv_reply_t reply;
       int serverFIFO = open(SERVER_FIFO_PATH, O_WRONLY);
       ... // outras funcionalidades
       write(serverFIFO, tlv, sizeof(*tlv));
       close(serverFIFO);
       return 1;
   }
```

A informação passada tem um tamanho `sizeof(*tlv)`.

O mesmo se verifica na passagem de informação do servidor para o cliente, tendo como diferença a `struct` preenchida que, neste caso, será `tlv_reply_t`.

Mecanismos de Sincronização

A nossa abordagem ao problema do produtor-consumidor recorre a Mutexes e Variáveis de Condição, sendo estes os únicos tipos de mecanismos de sincronização utilizados em todo o programa do servidor (o programa do utilizador não recorre a threads, pelo que não possui mecanismos de sincronização). Assim, existem duas variáveis de condição - uma para que o produtor espere quando a fila está cheia e outra para que o consumidor espere quando a mesma estiver vazia; três mutexes relacionados com a fila de pedidos – um para o acesso ao buffer circular, outro para a variável *requests* (que conta o número de pedidos) e o terceiro para a variável *slots* (que conta o número de posições disponíveis no buffer); um mutex para a variável *activeThreads* que conta o número de threads ativos; um mutex para cada conta (guardados num array, em que a posição no array corresponde à posição da respetiva conta no array de contas) garantindo que há exclusão mútua no acesso à informação das contas.

Encerramento do Servidor

O encerramento ao servidor processa-se da seguinte forma:

1. A thread main trata o pedido de encerramento normalmente e insere-o na fila;
2. Um dos threads consumidores interpreta o pedido: altera as permissões do FIFO do servidor para apenas leitura, ativa a flag global *shutdown* e envia a resposta ao utilizador;
3. A thread main continua a ler do FIFO até ler 0 caracteres, altura em que verifica se *shutdown* está ativa, caso em que apaga o FIFO e executa *pthread_exit(),* permitindo que as restantes threads continuem a executar;
4. As threads consumidoras continuam a executar os pedidos que estejam na fila até que *requests* (o número de pedidos) seja 0, altura em que executam *pthread_cond_broadcast(),* para que qualquer thread que esteja bloqueada na variável de condição há espera de pedidos saiba que deve correr, e depois terminam;
5. As threads que estavam bloqueadas testam a flag *shutdown* e caso esteja ativa desbloqueiam o mutex associado à variável de condição para que as restantes threads bloqueadas possam também encerrar.

Os passos 4 e 5 são executados nas seguintes linhas de código (função *consumeRequest*), das quais foram omitidas as funções de *log* para facilitar a leitura:

```
while (1){
   pthread_mutex_lock(&requestsLock);
   if (shutdown && requests <= 0){
         pthread_cond_broadcast(&requestsCond);
         pthread_mutex_unlock(&requestsLock);
         break;
   }
   while (requests <= 0) {
         pthread_cond_wait(&requestsCond, &requestsLock);
         if (shutdown) {
               pthread_mutex_unlock(&requestsLock);
               pthread_exit(NULL);
         }
   }
   ... // restante código do consumidor
}
pthread_exit(NULL);
```