Sistemas Operativos

Relatório do Projeto 2 - *Home Banking*

Comunicação cliente-servidor

A comunicação entre o cliente e o servidor é feita em dois processos diferentes, tal que o lado do server estabelece a criação de um fifo, por onde passará a informação do pedido (*request*) de operação a ser realizada - do *user* para o *server* - a qual deverá enviar uma resposta (*reply*) no sentido contrário - do *server* para o *user*.

As mensagens são mantidas e enviadas de um para o outro através das estruturas **tlv_request_t** e **tlv_reply_t**, fornecidas no código auxiliar.

```
typedef struct tlv_request {
  enum op_type type;
  uint32_t length;
  req_value_t value;
} __attribute__((packed)) tlv_request_t;
```

```
typedef struct tlv_reply {
  enum op_type type;
  uint32_t length;
  rep_value_t value;
} __attribute__((packed)) tlv_reply_t;
```

Os pedidos podem ser feitos através do programa *user* com uma chamada:

./user <id_conta> “<password>” <atraso_op_ms> <cod_op> “<args_op>”

Apenas serão válidos após a inicialização do programa *server*:

./server <nº threads/e-counters> “<pass_admin>”

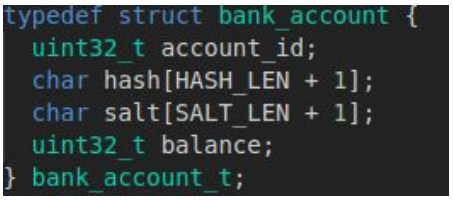
Caso contrário, a resposta indicará que o servidor está “em baixo” (SRV_DOWN). O programa server após a inicialização de todas as estruturas para o funcionamento do sistema bancário, vai ficar permanentemente à “escuta” de pedidos até ser enviado (e validado) o pedido de encerramento. Os pedidos são colocados numa fila (**request_queue_t**) criada especialmente para o contexto do problema.

```
typedef struct node node_t;

struct node{
    tlv_request_t data;
    node_t* next;
};

typedef struct {
    node_t* front;
    node_t* rear;
    sem_t request_slots;
    sem_t requests_waiting;
    int thread_number;
} request_queue_t;
```

O acesso a contas é autenticado através da geração de **hash** e **salt**. As informações relativas às contas bancárias são guardadas na estrutura dada **bank_account_t**, também fornecida no código auxiliar.



O **user** tem a sua inicialização em **user.c**, após o parsing do pedido, a estrutura **tlv_request_t** é preenchida e enviada para o server através do fifo, da mesma forma o server lidará com o *request* e retornará um *reply* para o user (o qual deverá ser interpretado e registado de seguida).

Métodos de sincronização

A concorrência (tentativas de acesso ao mesmo espaço de memória, neste caso, modificações à mesma conta por threads diferentes) foi resolvida recorrendo à associação de um mutex a cada conta no array de contas.

Foi ainda utilizado um semáforo para o processamento dos pedidos do user, lidando com o problema de acordo com o problema produtor-consumidor.

Sempre que um mecanismo de sincronização é utilizado, o seu uso é registado no log do servidor (*slog*) de acordo com as especificações do enunciado.

Encerramento do servidor

Ainda que a abertura do servidor seja feita pela chamada ao programa **server**, o seu fecho é requisitado a partir do programa **user**, através da operação de encerramento, **OP_SHUTDOWN**, sendo apenas válida quando chamada pela conta administrador.

Após a validação do pedido de fecho, o fifo é colocado em modo de leitura para impedir a emissão de novos pedidos, contudo, o programa server só terminará após o processamento de todos os pedidos existentes na *queue* e no buffer do fifo.

É impressa uma mensagem final no ficheiro *slog*, com a indicação deste shutdown.

Aos pedidos feitos após o fecho do server pelo user, é devolvido no *reply* o valor SRV_DOWN, sendo este impresso no *ulog*.