Sistemas Operativos
Relatório 2º Trabalho

Joaquim Rodrigues, up201704844
José Gomes, up201707054

Estrutura das mensagens trocadas

User-Server:

O user utiliza uma struct para comunicar com o server. Essa struct, denominada *tlv_request* envia, como o próprio nome indica, um pedido para o servidor. Esta struct possui como parâmetros um *op_type* (operação a realizar no server), um *uint32_t* (o tamanho da própria struct a ser enviada para o server) e uma outra struct que pode englobar outras (*req_value*, que contém uma struct com informação necessária, *req_header*, e pode conter uma das duas structs: *req_create_account* e *req_transfer*, se alguma destas for a operação a realizar). O envio desta struct para o user é feito através de um FIFO e cada request é guardado numa queue (de modo a que sejam tratados de forma sequencial e ordenada).

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

Server-User:

O server, após processar o pedido envia uma outra struct, *tlv_reply*, para o user, através de um FIFO cuja parte do nome é o pid enviado no request. Esta struct possui um *op_type* com o tipo de operação realizada, o tamanho da própria struct e, de forma semelhante ao request, possui uma struct *value*, que neste caso possui também outras structs indicando o básico da operação feita.

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

Mecanismos de sincronização

Sincronização da queue de requests:

De forma a que apenas um pedido seja acedido de cada vez e que não existam diversos threads a tratar o request nesse pedido, torna-se necessária a sincronização da queue. Para isto, foi implementado um mutex (*queue_mutex, no ficheiro server.c*). Cada thread tenta bloquear este mutex numa tentativa de aceder à queue para retirar um pedido, dando unlock após retirar o pedido. Se a queue estiver vazia, um thread dará lock do mutex e espera que algo entre na queue.

Sincronização do acesso às contas:

Para que cada conta não seja acedida simultaneamente por 2 threads, o que poderia trazer complicações a nível da realização de operações na mesma, cada conta terá associada a si um mutex. Cada thread a realizar uma operação na conta é responsável por dar lock nesse mesmo mutex para realizar os procedimentos e unlock no final. Para obter este comportamente foi criada a struct **account_mut** que é composta por um mutex e uma **bank_account**.

Encerramento do servidor

Aquando da chamada do administrador para o encerramento do servidor, dá-se a verificação das threads que estão ativas (consegue fazer-se porque cada thread tem associado a si um booleano) e, caso algum thread ainda esteja a processar um pedido, espera-se, até que todos os threads estejam livres, fechando assim todos os threads e libertando os recursos utilizados.