# SISTEMAS OPERATIVOS

# RELATÓRIO DO 2º PROJETO

## ESTRUTURA DE MENSAGENS ENTRE CLIENTES E SERVIDORES

Iniciando o programa cliente verificamos os argumentos passados e depois sincronizamos, logo após isso ser feito o pedido do cliente é guardado em uma estrutura do tipo tlv_request().

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

No registo das mensagem para o servidor pode conter duas estruturas distintas:

R/E - id - [tamanho bytes] PID (ID_conta, senha) [atraso ms] operação lista_de_args

R/E - id - [tamanho bytes] ID_conta operação código_retorno campos_de_resposta

Logo após o processamento do pedido, o servidor envia uma mensagem de resposta para o cliente com a estrutura na função tlv_reply().

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

Na estrutura da mensagem que o servidor envia ao cliente aparece o código de retorno podendo confirmar a operação ou descrever o erro.

```
static const char *RC_STR[] = {
  [RC_OK] = "OK",
  [RC_SRV_DOWN] = "SRV_DOWN",
  [RC_SRV_TIMEOUT] = "SRV_TIMEOUT",
  [RC_USR_DOWN] = "USR_DOWN",
  [RC_LOGIN_FAIL] = "LOGIN_FAIL",
  [RC_OP_NALLOW] = "OP_NALLOW",
  [RC_ID_IN_USE] = "ID_IN_USE",
  [RC_ID_NOT_FOUND] = "ID_NOT_FOUND",
  [RC_SAME_ID] = "SAME_ID",
  [RC_NO_FUNDS] = "NO_FUNDS",
  [RC_TOO_HIGH] = "TOO_HIGH",
  [RC_OTHER] = "OTHER"};
```

## MECANISMOS DE SINCRONIZAÇÃO

Foram usados semáforos e mutexes para que os processos sincronizem as suas ações.

Foi usado apenas um semáforo para a sincronização do buffer unitário.

```
void *processCounter(void *num)
{
  logBankOfficeOpen(slog, *(int *)num, pthread_self());
  while (online)
  {
    condRequest(num);
    tlv_reply_t reply;
    tlv_request_t tlv_request;
    getRequest(&tlv_request);
    sem_getvalue(&sem1, &val1);
    logSyncMechSem(slog, *(int *)num, SYNC_OP_SEM_WAIT, SYNC_ROLE_CONSUMER, tlv_request.value.header.pid, val1);
    sem_wait(&sem1);
```

Foi usado um mutex para a sincronização de acesso ao array de contas (operações), garantido por um mutex lock no ínicio e um mutex unlock no fim para garantir apenas um acesso de cada vez.

```
void condRequest(void *num)
{
  logSyncMech(slog, *(int *)num, SYNC_OP_MUTEX_LOCK, SYNC_ROLE_CONSUMER, 0);
  pthread_mutex_lock(&mutex);
  while (!(requests > 0))
  {
    logSyncMech(slog, *(int *)num, SYNC_OP_COND_WAIT, SYNC_ROLE_CONSUMER, 0);
    pthread_cond_wait(&cond1, &mutex);
  }
  requests--;
  pthread_mutex_unlock(&mutex);
  logSyncMech(slog, *(int *)num, SYNC_OP_MUTEX_LOCK, SYNC_ROLE_CONSUMER, 0);
}
```

## ENCERRAMENTO DO SERVIDOR

Após o administrador solicitar o encerramento do servidor, todos os pedidos e operações que já estavam pendentes deverão ser processados. Após todos os pedidos e operações pendentes forem processados, o servidor se encerra e já não aceita mais nenhum pedido. O programa principal irá escrever no ficheiro slog (se sucesso) o código de retorno “OK” e será enviado como resposta o número de consumidores/threads ativos (a processar um pedido) no momento do envio.

```
void shutdown()
{
    chmod(SERVER_FIFO_PATH, S_IRUSR | S_IRGRP | S_IROTH);
}
```

Se insucesso, então será enviado os códigos de retorno OP_NALLOW ou OTHER.

```
else if(request.value.header.account_id != ADMIN_ACCOUNT_ID){
    reply.value.header.ret_code = RC_OP_NALLOW;
}
else{
    reply.value.header.ret_code = RC_OTHER;
}
```