TRABALHO PRÁTICO Nº 2

Simulação de um sistema de *home banking*

Implementação de uma arquitetura cliente/servidor baseada em *FIFO*s

---

Sumário

Pretende-se desenvolver uma aplicação cliente/servidor que permita simular a realização de operações bancárias através da Internet (*home banking*). A simulação será feita através de processos a correr num único computador, sendo a comunicação entre clientes e servidor feita através de *pipes* com nome (FIFOs).

Objetivos

Familiarizar os estudantes com a programação de sistema, em ambiente Unix/Linux, envolvendo a gestão de processos/*threads* e a utilização de mecanismos de comunicação e de sincronização entre processos/*threads*.

Descrição Geral

A aplicação a desenvolver será constituída por um programa utilizador (**user**), que simula um utilizador[1] do serviço de *home banking*, e por um programa servidor (**server**), que processa os pedidos provenientes dos utilizadores. O serviço de *home banking* é autenticado e deve permitir a criação de contas e o seu encerramento remoto (apenas *administrador*), a consulta de saldo e transferências bancárias (apenas *clientes*); só devem ser autorizadas operações que possam ser integralmente satisfeitas e cujo valor final do(s) saldo(s) se situe no intervalo permitido. Os requisitos de qualidade de serviço prestado aos clientes impõem que seja assegurado um elevado nível de concorrência. A definição dos mecanismos de sincronização necessários para garantir o correto processamento das operações é deixada ao critério do implementador, sendo apenas requerido que o atendimento dos pedidos seja feito de acordo com o problema do produtor-consumidor.

A comunicação entre utilizadores e servidor será feita através de FIFOs. O servidor receberá os pedidos autenticados (incluem ID da conta e senha) por parte dos utilizadores através de um FIFO comum, de nome **/tmp/secure_srv**, e cada utilizador receberá a respetiva resposta através de um FIFO dedicado, de nome **/tmp/secure_XXXXX**, em que XXXXX representa o PID do processo **user** (ver figura). Todos os pedidos despoletam o envio de uma mensagem de resposta com um código de retorno que indicará o sucesso da operação ou a razão do seu insucesso (por exemplo, a autenticação falhou). No sentido de otimizar a quantidade de dados trocados, as mensagens devem seguir o formato TLV[2] e o seu tamanho deve ser ajustado consoante o tipo de pedido. O formato destas mensagens será especificado adiante (ver protocolo de comunicação).

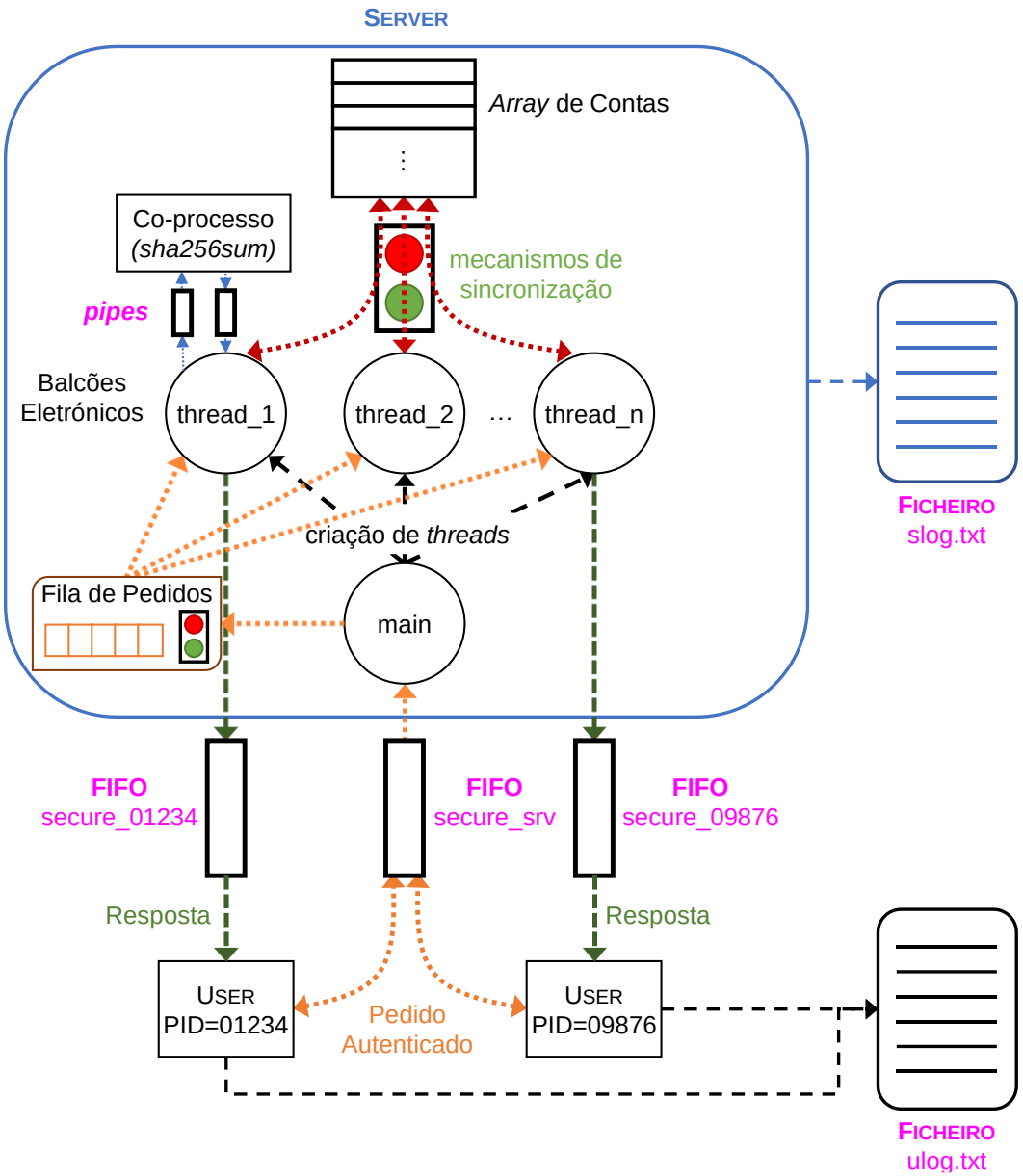
O programa **user** deve permitir a realização de operações autenticadas no serviço de *home banking*, sejam elas provenientes de clientes ou do administrador. A cada execução corresponderá o envio de um único pedido, cujos argumentos são recebidos através da linha de comandos, e a receção da resposta respetiva através do FIFO **/tmp/secure_XXXXX**; os argumentos devem ser validados e não

---

[1] Um utilizador do serviço é alguém que envia pedidos ao servidor: um cliente ou o administrador do sistema.

[2] TLV, sigla inglesa para indicar "tipo, tamanho e valor" (*Type-Length-Value*).

devem ser enviados pedidos inválidos. Uma vez enviado o pedido, o programa deve terminar após ter sido recebida uma resposta ou terem passado FIFO_TIMEOUT_SECS (=30) segundos. No final, o FIFO deve ser removido.



O programa **server** será constituído por um conjunto de "balcões eletrónicos", simulados por *threads*, que processam, por ordem de chegada, os pedidos dos utilizadores. Todos os pedidos são autenticados e, após receção (*thread main*), devem ser encaminhados para um balcão eletrónico livre para serem processados; se não houver balcões eletrónicos disponíveis, os pedidos ficam em fila de espera. A autenticação deve ser feita mediante a criação de um coprocesso, que invocará o utilitário de linha de comandos **sha256sum**, e será bem-sucedida se o sumário criptográfico[3] resultante da concatenação da senha com o "sal"[4] corresponder ao constante nos dados da conta[5]. Após receber a mensagem de encerramento, o programa servidor deve impedir o envio de novos pedidos através do FIFO de nome **/tmp/secure_srv** e processar aqueles que ainda estiverem pendentes, incluindo os que constarem no *buffer* desse FIFO. É sugerido que as permissões de acesso ao FIFO sejam colocadas

[3] Um sumário criptográfico, em inglês *[cryptographic] hash*, é um identificador de tamanho fixo que resulta do mapeamento não invertível (num só sentido) de um conjunto de dados de tamanho arbitrário.

[4] O "sal", em inglês *salt*, é usado para que a senhas iguais correspondam sumários criptográficos diferentes.

[5] Por questões de segurança, apenas o sumário criptográfico e o "sal" devem ser guardados no servidor.

em modo de "apenas leitura"[6] (exemplo: *r--r--r--*) para impossibilitar o envio de novos pedidos (deixa de ser possível abrir o FIFO em modo de escrita). No final, o FIFO deve ser removido.

O *thread* principal (*main*) terá 0 (zero) como identificador e os "balcões eletrónicos" (*threads*) devem ser numerados de acordo com a ordem de criação, sendo o primeiro o número 1 (um). Todas as operações realizadas pelo servidor e pelos utilizadores, incluindo as relativas aos mecanismos de sincronização usados, devem ser registadas em dois ficheiros de texto, **slog.txt** e **ulog.txt**, respetivamente. O formato destes ficheiros será especificado ao longo do documento.

Protocolo de Comunicação

O protocolo de comunicação a ser usado pela aplicação de *home banking* foi concebido para permitir o uso de mensagens de tamanho variável e assim otimizar a quantidade de dados trocados entre utilizadores e servidor; nesse sentido, todas as mensagens são definidas no formato TLV (tipo, tamanho e valor). Para efeitos de teste e validação é definido um parâmetro adicional em cada pedido, o atraso de operação[7] (em milissegundos), com o objetivo de conseguir reproduzir situações de concorrência e competição; a sua implementação deve ser feita recorrendo à função **usleep()**. É deixado ao critério do implementador definir quais os campos que devem ser incluídos nas mensagens, a sua ordem e o seu formato específico, existindo apenas os seguintes requisitos:

1. Os pedidos devem incluir o identificador do processo **user** (PID XXXXX) para onde enviar a resposta (FIFO **/tmp/secure_XXXXX**), ID da conta e senha associada para efeitos de autenticação, e o atraso de operação;
2. As respostas aos pedidos devem incluir o ID da conta e o código de retorno;
3. O tamanho da mensagem (pedido ou resposta) deve ser ajustado para serem trocados apenas os dados necessários[8].

No sentido de agilizar a realização deste trabalho, é sugerida uma possível implementação das estruturas de dados necessárias para a troca de mensagens no formato TLV. A definição destas estruturas recorre a uma extensão do compilador [*__attribute__((packed))*] para impedir a utilização de *padding*[9], facilitando a leitura das mensagens e prevenindo o envio de *bytes* desnecessários. São ainda disponibilizadas funções para a escrita de mensagens de registo (ver descrição das operações) recorrendo a estas estruturas; é dada liberdade ao implementador para optar pela sua adoção, adaptação ou mesmo redefinição.

Os códigos de retorno que devem ser usados nas respostas aos pedidos, por ordem (ver notas abaixo), são os seguintes:

- OK – operação bem-sucedida;
- SRV_DOWN – não é possível abrir o FIFO **/tmp/secure_srv** (servidor indisponível);
- SRV_TIMEOUT – o pedido expirou (a resposta não foi recebida dentro do tempo limite);
- USR_DOWN – não é possível abrir o FIFO **/tmp/secure_XXXXX** (utilizador indisponível);
- LOGIN_FAIL – a autenticação falhou porque o par (ID da conta, senha) não é válido;
- OP_NALLOW – a operação não é permitida ao utilizador (*cliente* ou *administrador*);
- ID_IN_USE – o ID da conta já está em uso (corresponde a uma conta já criada);
- ID_NOT_FOUND – a conta com o ID especificado não existe;
- SAME_ID – as contas de origem e de destino são a mesma;
- NO_FUNDS – a operação não pode ser realizada porque o saldo corrente é insuficiente;
- TOO_HIGH – a operação não pode ser realizada porque o saldo final é demasiado elevado;
- OTHER – qualquer outro erro que não tenha sido especificado.

---

[6] A função ***int fchmod(int** fd**, mode_t** mode**)*** permite alterar as permissões do FIFO usando um descritor válido.

[7] O atraso deve ser introduzido imediatamente após ter sido obtido o acesso exclusivo a uma conta ou antes de encerrar o serviço; no caso de transferências, o atraso pode ser introduzido até duas vezes (uma por conta).

[8] Por questões de simplicidade, devem ser enviados os MAX_PASSWORD_LEN (=20) + 1 carateres da senha.

[9] https://en.wikipedia.org/wiki/Data_structure_alignment#Data_structure_padding.

Nota 1: Se ocorrerem múltiplos erros, deve ser reportado apenas o que aparecer primeiro na lista.
Nota 2: Os códigos de retorno SRV_DOWN, SRV_TIMEOUT, USR_DOWN e LOGIN_FAIL são transversais a todas as operações pelo que não serão referidos explicitamente aquando da descrição das operações.

Os códigos de retorno SRV_DOWN e USR_DOWN indicam a impossibilidade de envio de uma mensagem (não é possível abrir o FIFO usado na comunicação); o código de retorno SRV_TIMEOUT indica a impossibilidade de receber uma resposta em tempo útil (em FIFO_TIMEOUT_SECS (=30) segundos). Para facilitar a análise de execução dos programas, devem ser registadas todas as tentativas falhadas de envio de mensagens e, no caso do programa utilizador, as falhas de resposta do servidor, para mostrar a razão do insucesso do pedido; a mensagem de resposta deve ter o seu tipo, tamanho, ID da conta e código de retorno definidos corretamente (ver exemplo de execução). A operação deve ser realizada independentemente de ser possível enviar uma resposta ao utilizador.

O texto associado a cada código de retorno é o que constará no registo das operações (ficheiros **slog.txt** e **ulog.txt**). Na definição das estruturas das mensagens é usada uma enumeração (*enum ret_code*) para atribuir um valor numérico a cada código de retorno; se assim o entender, é dada liberdade ao implementador para definir um formato alternativo.

Descrição das Operações

Nesta secção são descritas as operações que devem ser suportadas pela aplicação e os respetivos códigos de retorno. Para cada uma das operações são indicados quais os argumentos necessários à sua execução, a gama de valores permitida para cada um deles e o formato das mensagens de registo, tanto para o servidor (**slog.txt**) como para os utilizadores (**ulog.txt**). O administrador é o utilizador do sistema cujo ID de conta é ADMIN_ACCOUNT_ID (=0); a conta que lhe está associada deve ter sempre um saldo de 0€.

O formato genérico das mensagens de registo relativas às mensagens trocadas entre utilizadores e servidor deve ser o seguinte:

- ***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***

A letra **R** é usada nas mensagens que são recebidas e a letra **E** é usada nas mensagens que são enviadas. O **id** será o ID do *thread* para o servidor ou o PID para o utilizador. O **tamanho** representa o valor do campo tamanho da mensagem TLV, em *bytes*. O **atraso** representa o atraso de operação introduzido, em milissegundos. A **operação** será o texto definido para cada operação; idem para o **código de retorno**. A **lista de argumentos** e os **campos de resposta**, caso existam, seguem o formato específico definido em cada operação. A senha deve ser representada entre aspas e não deve conter espaços. Para facilitar a leitura das mensagens de registo, os campos **id**, **tamanho**, **PID**, **ID da conta**, **senha**, **atraso**, **operação** e **código de retorno** devem ocupar, respetivamente, WIDTH_ID (=5), WIDTH_TLV_LEN (=3), WIDTH_ID, WIDTH_ACCOUNT (=4), MAX_PASSWORD_LEN (=20), WIDTH_DELAY (=5), WIDTH_OP (=8) e WIDTH_RC (=12) carateres.

No sentido de agilizar o processo de desenvolvimento, são disponibilizadas as funções *logRequest()*[10] e *logReply()*[11] para imprimir, respetivamente, os pedidos (*request*) e as respostas (*reply*) recorrendo às estruturas de dados sugeridas. Se o implementador optar por definir estruturas de dados próprias, deverá definir as suas próprias funções e garantir a correta impressão das mensagens de registo.

Para facilitar a análise da execução, o servidor deve imprimir a mensagem relativa à receção dos pedidos tanto quando estes são colocados na fila de espera (produtor) como quando são retirados para processamento (consumidor).

---

[10] ***int logRequest(int** fd**, int** id**, const tlv_request_t ***request**)*.

[11] ***int logReply(int** fd**, int** id**, const tlv_reply_t ***reply**)*.

Criação de Conta

A criação de conta apenas pode ser solicitada pelo administrador do sistema e tem como argumentos o ID da conta a ser criada, o saldo inicial e a senha associada. O formato da lista de argumentos, cujos campos **ID da conta** e **saldo** devem ocupar, respetivamente, WIDTH_ACCOUNT (=4) e WIDTH_BALANCE (=10) carateres, é o seguinte:

- ***ID_da_conta saldo€ "senha"***

Por questões de segurança, a senha não deve ser guardada no servidor, mas sim o sumário criptográfico (*hash*), e deve ter um "sal" (*salt*) associado. Assim, aquando da criação de uma nova conta, a aplicação deve gerar um "sal" pseudoaleatório composto por SALT_LEN (=64) carateres hexadecimais (0-9a-f). O sumário criptográfico, que será usado para verificar a senha durante o processo de autenticação, é calculado recorrendo ao utilitário **sha256sum**: recebe como entrada o resultado da concatenação da senha com o "sal" e coloca na saída HASH_LEN (=64) carateres hexadecimais. As restrições impostas a cada um destes valores são as seguintes:

- 1 ≤ ID da conta < MAX_BANK_ACCOUNTS (=4096)
- MIN_BALANCE (=1) ≤ saldo ≤ MAX_BALANCE (=1000000000)
- MIN_PASSWORD_LEN (=8) ≤ comprimento da senha ≤ MAX_PASSWORD_LEN (=20)

A mensagem de resposta ao pedido não contém quaisquer campos adicionais. Caso o pedido não seja bem-sucedido, pode ser devolvido um dos seguintes códigos de retorno: OP_NALLOW (pedido realizado por um cliente), ID_IN_USE (conta existente) ou OTHER (erro não especificado). Como referido anteriormente, e aplicável às restantes operações, o programa **user** deve fazer uma validação prévia dos valores e apenas enviar pedidos válidos.

Se o pedido for bem-sucedido, e apenas para o servidor, deve ser impressa uma mensagem de registo adicional (informativa) com o "sal" e o sumário criptográfico para permitir a sua validação[12]:

- ***I - id_do_balcão*** - ***ID_da_conta sal ...final_do_sumário_criptográfico***

Para facilitar a leitura destas mensagens de registo, os campos **id do balcão**, **ID da conta**, **sal** e **final do sumário criptográfico** (últimos caracteres) devem ocupar, respetivamente, WIDTH_ID (=5), WIDTH_ACCOUNT (=4), SALT_LEN (=64) e WIDTH_HASH (=5) caracteres. É disponibilizada a função *logAccountCreation()*[13] para o efeito, desde que seja usada a estrutura *bank_account_t* fornecida.

Consulta de Saldo

A consulta do saldo de conta não requer qualquer argumento. Caso a autenticação seja bem-sucedida, será enviado o valor do saldo como resposta e cuja impressão na mensagem de registo (campos de resposta) deve ocupar WIDTH_BALANCE (=10) carateres: ***saldo€***. Caso contrário, deve ser ignorado o valor do saldo (embora impresso) e devolvido um dos seguintes códigos de retorno: OP_NALLOW (pedido realizado pelo administrador) ou OTHER (erro não especificado).

Transferência

As transferências têm como destinatário um outro cliente do banco (internas) e, assim, envolvem a atualização do saldo nas contas de origem e de destino. Os argumentos desta operação são o ID da conta de destino e o montante em causa. As restrições impostas a cada um destes valores são as seguintes:

- 1 ≤ **ID da conta destino** < MAX_BANK_ACCOUNTS (=4096)
- 1 ≤ **montante** ≤ MAX_BALANCE (=1000000000)

---

[12] O comando **echo -n "<senha><sal>" | sha256sum** pode ser usado para validar o correto funcionamento do coprocesso, em que **<senha>** representa a senha definida e **<sal>** o sal gerado (sem os carateres **<** e **>**).

[13] ***int logAccountCreation(int** fd**, int** id**, const bank_account_t ***account**)*.

O formato da lista de argumentos, cujos campos **ID da conta destino** e **montante** devem ocupar, respetivamente, WIDTH_ACCOUNT (=4) e WIDTH_BALANCE (=10) caracteres, é o seguinte:

- ***id_da_conta_destino montante€***

Caso a operação seja bem-sucedida, para além do código de retorno (OK), será enviado como resposta o saldo corrente após a operação; a impressão dos campos de resposta é a definida para a consulta do saldo. Caso contrário, o saldo corrente é devolvido inalterado e o código de retorno será um dos seguintes: OP_NALLOW (pedido realizado pelo administrador), ID_NOT_FOUND (a conta de destino não existe), SAME_ID (as contas de origem e destino são a mesma), NO_FUNDS (saldo insuficiente), TOO_HIGH (saldo final passaria a ser demasiado elevado) ou OTHER (erro não especificado).

Encerramento do Servidor

O encerramento do serviço apenas pode ser solicitado pelo administrador do sistema e não requer qualquer argumento. Após esta operação, deve ser impossibilitado o envio de novos pedidos através do FIFO de nome **/tmp/secure_srv** (é sugerido que as permissões de acesso ao FIFO sejam colocadas em modo de "apenas leitura") e devem ser processados aqueles que ainda estiverem pendentes, incluindo os que constarem no *buffer* desse FIFO. O programa só deve terminar depois de todos os pedidos e operações pendentes terem sido processados. Caso a operação seja bem-sucedida, para além do código de retorno (OK), será enviado como resposta o número de consumidores/*threads* ativos (a processar um pedido) no momento do envio. Caso contrário, deve ser enviado um dos seguintes códigos de retorno: OP_NALLOW (pedido realizado por um cliente) ou OTHER (erro não especificado).

Mecanismos de Sincronização

Nesta secção são especificadas as mensagens de registo que devem ser impressas sempre que for invocada uma função relativa a mecanismos de sincronização considerada relevante. Para facilitar a análise de execução, as mensagens de registo especificam três categorias: produtor, consumidor e *array* de contas (operações). Como já referido, o tratamento dos pedidos deve aplicar os mecanismos de sincronização de acordo com o problema produtor-consumidor; os mecanismos de sincronização associados à realização das operações bancárias são deixados ao critério do implementador.

As funções consideradas relevantes, por tipo de mecanismo de sincronização, são as seguintes:

- **mutex** - *pthread_mutex_lock()*, *pthread_mutex_trylock()* e *pthread_mutex_unlock()*;
- **semáforo** - *sem_init()*, *sem_post()* e *sem_wait()*;
- **variável de condição** - *pthread_cond_signal()*, *pthread_cond_wait()*.

Atendendo a que há interesse em saber o valor associado a um semáforo num dado momento, as mensagens de registo envolvendo semáforos apresentam adicionalmente, no final da mensagem de registo, o valor retornado pela função *sem_getvalue()*[14] no formato "[val=***valor***]" (sem aspas). A mensagem de registo deve ser impressa imediatamente antes da invocação das funções *sem_init()* e *sem_wait()*, mas imediatamente depois da invocação da função *sem_post()*. O formato da mensagem de registo relativa aos mecanismos de sincronização é o seguinte:

- ***S - id - <categoria>sid função***

O **id** será o ID do *thread* e a **categoria** será a letra P (produtor), C (consumidor) ou A (*array* de contas) consoante a categoria. O **sid** representa o PID associado ao pedido[15] nas categorias produtor e consumidor e o ID da conta[16] na categoria *array* de contas. A **função** será o texto definido para identificar cada uma das funções consideradas relevantes. Para facilitar a leitura destas mensagens de registo, os campos **id** e **sid** devem ocupar, cada um, WIDTH_ID (=5) carateres. São disponibilizadas

[14] O valor retornado pode não corresponder ao valor atual do semáforo (ver documentação da função).

[15] Enquanto o PID associado ao pedido não for conhecido pelo *thread*, deve ser usado temporariamente 0 (zero) como ID.

[16] Se o mecanismo de sincronização se aplicar a todas as contas, deve ser usado o ID 9999.

as funções *logSyncMech()*[17] e *logSyncMechSem()*[18] para o efeito.

Atraso de Operação

A especificação de um valor de atraso a introduzir na realização de uma operação tem como objetivo reproduzir situações de concorrência e competição; a sua implementação deve ser feita recorrendo à função **usleep()**. No caso das operações bancárias, o atraso deve ser introduzido (e registado) imediatamente após ter sido obtido acesso exclusivo à conta (dentro da secção crítica); aquando do encerramento, o atraso deve ser introduzido (e registado) imediatamente antes de ser impossibilitado o envio de novos pedidos. No caso das transferências bancárias, dado que estão envolvidas duas contas, o atraso deve ser introduzido sempre que for garantido o acesso exclusivo a cada uma das contas; se o mecanismo de sincronização garantir o acesso exclusivo a ambas contas em simultâneo, o atraso deve ser introduzido apenas uma vez.

A mensagem de registo relativa à introdução do atraso de operação deve seguir o seguinte formato, sendo que os campos **categoria** e **sid** devem ser omitidos aquando do encerramento:

- ***A - id - <categoria>sid*** [***atraso*** ms]

Os campos **id**, **categoria** e **sid** são os usados nas mensagens de registo relativas aos mecanismos de sincronização. O **atraso** é o tempo de atraso, em milissegundos, que será introduzido e deve ocupar WIDTH_DELAY (=5) carateres. São disponibilizadas as funções *logDelay()*[19] e *logSyncDelay()*[20] para o efeito.

Especificação dos Programas

Os programas **server** e **user** devem trocar entre si mensagens que permitam a criação de contas, consulta do saldo, realização de transferências e o encerramento do servidor; os códigos associados a estas operações são, respetivamente, 0, 1, 2 e 3. Ambos programas devem continuar a executar normalmente no caso em que uma das extremidades de comunicação seja fechada abruptamente. Para simplificar a implementação, as senhas não devem conter espaços.

Para agilizar o desenvolvimento destes programas, são disponibilizados os ficheiros **constants.h** (conjunto de macros com as constantes definidas ao longo do documento), **types.h** (define a estrutura *bank_account_t*, as mensagens no formato TLV e um conjunto de enumerações úteis), **sope.h** (define as funções de registo e inclui os ficheiros *constants.h* e *types.h*) e **log.c** (implementação das funções de registo).

Programa server

O programa servidor (**server**) recebe, através de argumentos da linha de comandos, o número de "balcões eletrónicos" (*threads*) a ser criados, que serão no máximo MAX_BANK_OFFICES (=99), e a senha associada à conta de administrador.

```
shell$ ./server 10 "senha_insegura"
```

Antes de começar a atender pedidos, o programa servidor deve proceder à criação da conta de administrador [ADMIN_ACCOUNT_ID (=0)], com saldo de 0€ (ver criação de conta), e criar os "balcões eletrónicos"; devem ser impressas mensagens de registo referentes à abertura e encerramento dos balcões no seguinte formato:

- ***I - id - ação tid***

---

[17] ***int logSyncMech(int** fd**, int** id**, sync_mech_op_t** smo**, sync_role_t** role**, int** sid**)**.*

[18] ***int logSyncMechSem(int** fd**, int** id**, sync_mech_op_t** smo**, sync_role_t** role**, int** sid**, int** val**)**.*

[19] ***int logDelay(int** fd**, int** id**, uint32_t** delay_ms**)**.*

[20] ***int logSyncDelay(int** fd**, int** id**, int** sid**, uint32_t** delay_ms**)**.*

O **id** será o ID do *thread*, a **acção** será OPEN (abertura) ou CLOSE (encerramento) e o **tid** é o identificador POSIX do *thread*. Para facilitar a leitura destas mensagens de registo, os campos **id** e **ação** devem ocupar, respetivamente, WIDTH_ID (=5) e WIDTH_STARTEND (=5) carateres. São disponibilizadas as funções *logBankOfficeOpen()*[21] e *logBankOfficeClose()*[22] para o efeito.

Uma vez criada a conta de administrador, o servidor criará o FIFO de nome **/tmp/secure_srv** e ficará à escuta de pedidos autenticados; a execução das operações bancárias deve introduzir o atraso de operação especificado (ver atraso de operação). Numa fase inicial, uma vez que apenas a conta de administrador foi criada, o servidor só aceitará pedidos autenticados por parte do administrador: criação de contas e encerramento do serviço de *home banking* (termina o processo servidor). Após a criação de uma ou mais contas, os novos clientes passam a poder enviar pedidos: consulta de saldo e transferências. O programa só deve terminar depois de todos os pedidos e operações pendentes terem sido processados. No final, o FIFO de nome **/tmp/secure_srv** deve ser removido.

Programa user

O programa utilizador (**user**) recebe, através de argumentos da linha de comandos, o ID da conta, a senha associada, o atraso de operação que deve ser solicitado (em milissegundos), a operação a realizar (código) e a lista de argumentos que devem ser usados no envio do pedido (entre aspas - argumento único); se não existirem argumentos, deve ser passada uma "*string*" vazia.

```
shell$ ./user 1 "senha_fraca" 5000 1 ""
```

As operações de consulta de saldo (=1) e encerramento (=3) não recebem quaisquer argumentos; as operações de criação de conta (=0) e transferência (=2) devem receber os argumentos, respetivamente, pela seguinte ordem: "*ID_nova_conta saldo_inicial senha*" e "*ID_conta_destino montante*".

Como já referido, todos os pedidos incluem o ID da conta e senha associada para efeitos de autenticação, o atraso de operação (para efeitos de teste e de avaliação) e a operação pretendida. O programa **user** deve criar o FIFO de nome **/tmp/secure_XXXXX** antes de enviar o pedido, em que XXXXX é o PID do processo, para poder receber a resposta por parte do servidor. O programa termina após ter recebido uma resposta ao pedido enviado ou terem passado FIFO_TIMEOUT_SECS (=30) segundos. No final, o FIFO de nome **/tmp/secure_XXXXX** deve ser removido.

Exemplo de Execução

Nesta secção é fornecido um exemplo de execução do servidor e, sempre que necessário, são apresentadas as mensagens de registo geradas. As diversas execuções do programa **user** pretendem demonstrar a realização bem-sucedida de cada uma das operações e algumas situações de erro. Apesar das mensagens de registo serem impressas cada uma na sua linha, por questões de espaço, algumas mensagens podem ser apresentadas em mais do que uma linha.

No arranque do servidor é feita a inicialização dos mecanismos de sincronização, criada a conta de administrador e são abertos os balcões eletrónicos, os quais ficam à espera dos pedidos.

```
$ ./server 3 "ultra_top_secret"
```

```
S - 00000 - P00000 SEM_INIT [val=3]
S - 00000 - P00000 SEM_INIT [val=0]
I - 00001 - OPEN  140177458939648
S - 00001 - C00000 SEM_WAIT [val=0]
I - 00003 - OPEN  140177442154240
S - 00003 - C00000 SEM_WAIT [val=0]
S - 00000 - A00000 MUTEX_LOCK
```

[21] ***int logBankOfficeOpen(int** fd**, int** id**, pthread_t** tid**).***

[22] ***int logBankOfficeClose(int** fd**, int** id**, pthread_t** tid**).***

```
A - 00000 - A00000 [   0 ms]
I - 00002 - OPEN  140177450546944
S - 00002 - C00000 SEM_WAIT [val=0]
I - 00000 - 0000 be57454973a3ac4719242ba4d3bd59b0d9cb39d730e794f3aa613765ee3dc1ac ...9218c
S - 00000 - A00000 MUTEX_UNLOCK
```

A validação do sumário criptográfico gerado para uma conta pode ser feita através do seguinte comando (a criação da variável serve apenas efeitos de apresentação):

```
$ salt="ultra_top_secretbe57454973a3ac4719242ba4d3bd59b0d9cb39d730e794f3aa613765ee3dc1ac";
  echo -n $salt | sha256sum
```

```
c5759537dec2014e8471e1738147cefa4698d44271092c47764581f9cd49218c  -
```

Se o administrador tentar criar uma conta de cliente (ID=1, saldo=25€, senha="top_secret") com credenciais inválidas, será devolvido o código de retorno LOGIN_FAIL.

```
$ ./user 0 "wrong_password" 10 0 "1 25 top_secret"
```

```
E - 12472 - [ 62 bytes] 12472 (0000, "wrong_password")      [   10 ms] CREATE   0001
       25€ "top_secret"
R - 12472 - [  8 bytes] 0000   CREATE   LOGIN_FAIL
```

(registo do servidor)

```
R - 00000 - [ 62 bytes] 12472 (0000, "wrong_password")      [   10 ms] CREATE   0001
       25€ "top_secret"
S - 00000 - P12472 SEM_WAIT [val=3]
S - 00000 - P12472 MUTEX_LOCK
S - 00000 - P12472 MUTEX_UNLOCK
S - 00000 - P12472 SEM_POST [val=1]
S - 00001 - C00000 MUTEX_LOCK
S - 00001 - C12472 MUTEX_UNLOCK
S - 00001 - C12472 SEM_POST [val=3]
R - 00001 - [ 62 bytes] 12472 (0000, "wrong_password")      [   10 ms] CREATE   0001
       25€ "top_secret"
E - 00001 - [  8 bytes] 0000   CREATE   LOGIN_FAIL
S - 00001 - C00000 SEM_WAIT [val=0]
```

Se o administrador tentar criar uma conta de cliente (ID=1, saldo=25€, senha="top_secret") com credenciais válidas, será devolvido o código de retorno OK (conta criada).

```
$ ./user 0 "ultra_top_secret" 10 0 "1 25 top_secret"
```

```
E - 12524 - [ 62 bytes] 12524 (0000, "ultra_top_secret")    [   10 ms] CREATE   0001
       25€ "top_secret"
R - 12524 - [  8 bytes] 0000   CREATE        OK
```

A partir deste momento, o cliente da conta 1 pode consultar o seu saldo e realizar transferências; a transferência não será bem-sucedida porque ainda não foi criada a conta 2 (ID_NOT_FOUND).

```
$ ./user 1 "top_secret" 1234 1 ""
```

```
E - 12565 - [ 33 bytes] 12565 (0001, "top_secret")          [ 1234 ms] BALANCE
R - 12565 - [ 12 bytes] 0001  BALANCE        OK      25€
```

```
$ ./user 1 "top_secret" 567 2 "2 10"
```

```
E - 12592 - [ 41 bytes] 12592 (0001, "top_secret")          [  567 ms] TRANSFER 0002
       10€
R - 12592 - [ 12 bytes] 0001 TRANSFER ID_NOT_FOUND          10€
```

Uma vez criada a conta 2, a transferência será bem-sucedida. A consulta do saldo da conta 2 permite

verificar que foi efetuada corretamente em ambas as contas.

```
$ ./user 0 "ultra_top_secret" 10 0 "2 3 a_secret"
```

```
E - 12611 - [ 62 bytes] 12611 (0000, "ultra_top_secret")     [   10 ms] CREATE   0002
       3€ "a_secret"
R - 12611 - [  8 bytes] 0000   CREATE        OK
```

```
$ ./user 1 "top_secret" 567 2 "2 10"; ./user 2 "a_secret" 890 1 ""
```

```
E - 12636 - [ 41 bytes] 12636 (0001, "top_secret")          [  567 ms] TRANSFER 0002
      10€
R - 12636 - [ 12 bytes] 0001 TRANSFER        OK      15€
E - 12660 - [ 33 bytes] 12660 (0002, "a_secret")            [  890 ms] BALANCE
R - 12660 - [ 12 bytes] 0002  BALANCE        OK      13€
```

O encerramento não é permitido a clientes (OP_NALLOW), apenas ao administrador do sistema. Convém notar que a resposta tem 8 bytes em caso de erro (apenas ID da conta e código de retorno), mas tem 12 bytes em caso de sucesso (inclui o valor retornado).

```
$ ./user 1 "top_secret" 5000 3 ""
```

```
E - 12717 - [ 33 bytes] 12717 (0001, "top_secret")          [ 5000 ms] SHUTDOWN
R - 12717 - [  8 bytes] 0001 SHUTDOWN OP_NALLOW 0
```

```
$ ./user 0 "ultra_top_secret" 5000 3 ""
```

```
E - 12725 - [ 33 bytes] 12725 (0000, "ultra_top_secret")     [ 5000 ms] SHUTDOWN
R - 12725 - [ 12 bytes] 0000 SHUTDOWN        OK 0
```

Se houver um cliente que tente enviar um pedido depois do servidor encerrar, deve ser apresentada a mensagem SRV_DOWN para indicar esse erro.

```
$ ./user 1 "top_secret" 5000 1 ""
```

```
E - 12746 - [ 33 bytes] 12746 (0001, "top_secret")          [ 5000 ms] BALANCE
R - 12746 - [  8 bytes] 0001  BALANCE        SRV_DOWN       0€
```

Notas sobre o desenvolvimento

- Os nomes dos programas, a ordem dos argumentos da linha de comandos, os nomes indicados para algumas funções e constantes, os nomes e a estrutura dos ficheiros contendo resultados devem ser respeitados escrupulosamente. **O incumprimento pode conduzir à não avaliação do trabalho**, com as inerentes consequências;
- Todos os parâmetros internos devem ser facilmente modificáveis (com recurso a diretivas **#define** ou constantes com nome);
- Todas as estruturas de comunicação e sincronização devem ser destruídas no final da execução dos programas.
- Tudo o que não estiver especificado e que não venha a ser especificado numa nova versão do trabalho (a publicar, se necessário) poderá ser especificado pelos elementos do grupo de trabalho, devendo as especificações adicionais, contidas num ficheiro com o nome **addspecs.pdf**, ser enviadas juntamente com o código.

Validação da solução

- Para validar a solução desenvolvida serão realizados testes automáticos com o auxílio de um programa validador e ficheiros de testes.

Realização e entrega do trabalho

- Deverá ser elaborado um pequeno relatório (2 páginas) onde se descreva a estrutura das mensagens trocadas entre clientes e servidor (e vice-versa), os mecanismos de sincronização utilizados e a forma como é feito o encerramento do servidor (focar apenas estes aspetos). Sempre que necessário, a descrição deve ser acompanhada por excertos de código. O relatório, com o nome **report.pdf**, deve ser enviado juntamente com o código;
- A **data limite** para a entrega do trabalho é **17/05/2019, às 23:55**;
- Oportunamente serão dadas outras informações sobre a realização e entrega do trabalho, nomeadamente, quanto à estrutura de diretórios a utilizar.