Departamento de Informática
Licenciatura em Engenharia Informática

Redes de Computadores

Implementação de um protocolo de comunicação
Versão 20

Daniel Gonçalo Jesus Ramos, 29423

Évora 2013

Conteúdo

ÍNDICE DE QUADROS

ÍNDICE DE FIGURAS

INTRODUÇÃO

Neste trabalho pretende-se implementar o protocolo de comunicação definido no primeiro trabalho que foi realizado anteriormente.

O intuito deste trabalho é simular um sistema de HomeBanking através de uma consola, onde será possível simular as várias operações permitidas por este tipo de sistema. Entre essas operações, estão por exemplo operações como consulta de saldo, transferência de fundos, e pagamento de serviços. Será também possível administrar o sistema através de algumas simples operações disponíveis.

O protocolo de transporte escolhido foi o protocolo TCP/IP em que é conhecido pela sua completa gama de funcionalidades, como por exemplo, controlo de fluxo, roteamento, estado de ligação entre dois pontos e integridade de dados.

Neste protocolo que foi implementado as ligações cliente-server não são mantidas por tempo indefinido, logo uma operação efetuada pelo cliente é feito numa única ligação ao servidor, sendo necessário reestabelecer a ligação quando se quiser efetuar uma nova operação. Na parte do desenvolvimento é explicado como se deve lidar com este tipo de situação, em que é usado um sistema de sessão para manter o utilizador loggedin.

DESENVOLVIMENTO

Para implementar a abordagem cliente-servidor, o trabalho foi dividido em duas partes. A primeira parte inclui a estrutura e código do cliente a segunda parte inclui a estrutura e código do servidor. De seguida será explicado quais foram as estruturas implementadas para cada programa, bem como as estruturas que são partilhadas em comum.

Implementação do servidor

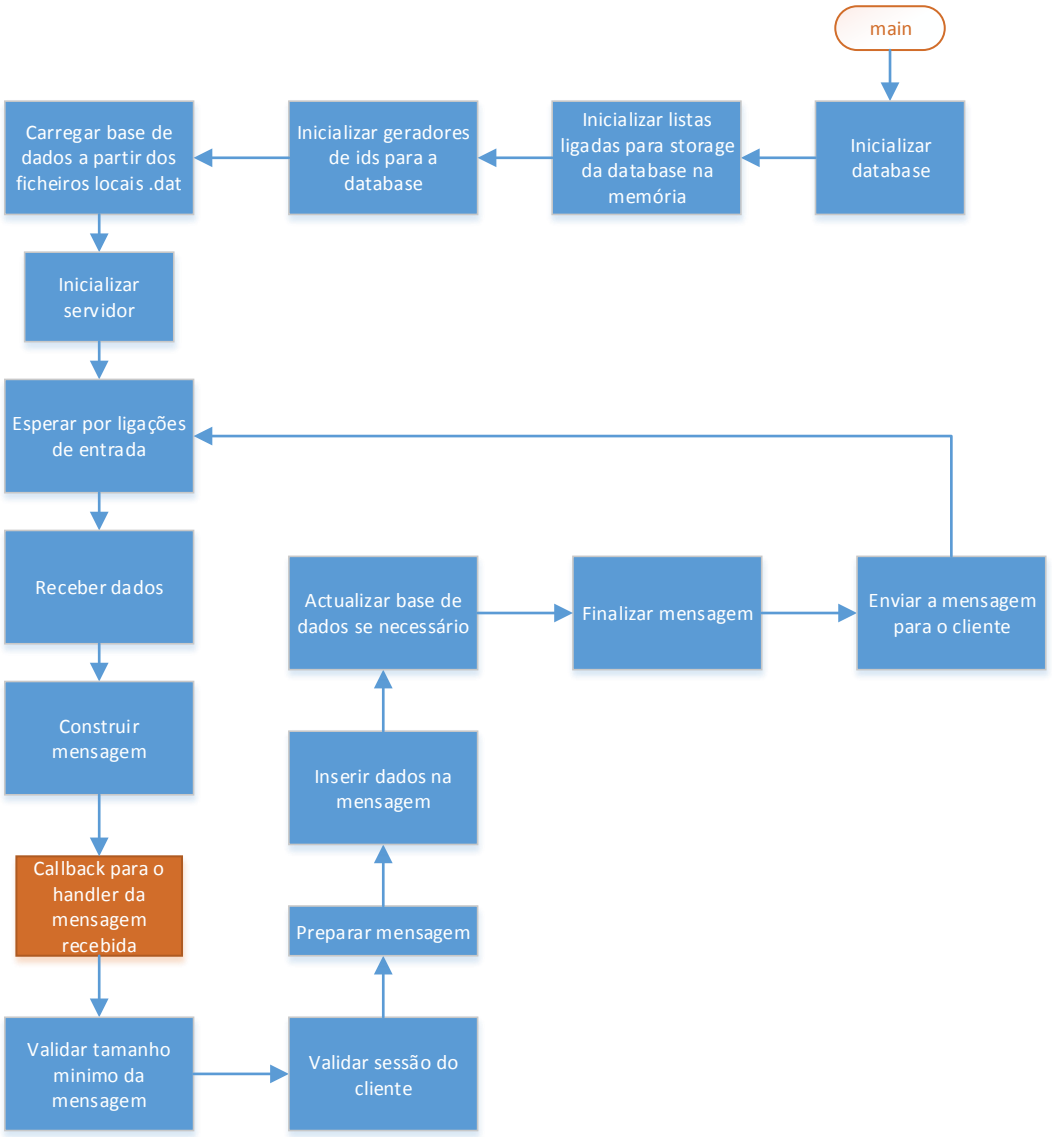

Figura 1: Fluxo simples do servidor:

Base de dados

O servidor ao arrancar inicia a base de dados em que é carregado vários ficheiros que pertencem aos utilizadores, contas, movimentos e serviços. Os dados são mantidos em memória através da implementação de uma lista simplesmente ligada que contém como elemento uma das várias estruturas que representam os mesmos.

```
typedef struct Utilizador
{
        int id;
        int flags;
        char username[32];
        char password[32];
        bool congelada;
}Utilizador;
```

```
typedef struct Conta
{
        int contaID;
        int clientID;
        int saldo;
}Conta;
```

```
typedef struct Movimento
{
        int id;
        int tipo;
        int owner;
        int contaO;
        int ContaD;
        int Montante;
}Movimento;
```

```
typedef struct Servico
{
        int id;
        char
nome[256];
        int nib;
}Servico;
```

As estruturas acima referidas são escritas e lidas diretamente do ficheiro apropriado. Quando é necessário adicionar dados à base de dados, a estrutura adequada é escrita no fim do ficheiro. Por sua vez se algum dado precisar de ser eliminado, o conteúdo do ficheiro apropriado tem que ser reescrito de novo, mantendo os dados que não desejáveis de serem eliminados.

Incluído na base de dados, está também incluído a geração de ids para cada tipo de estrutura. Cada vez que é carregada a base de dados, o id máximo inicial é incrementado de acordo com o maior id alocado na última sessão.

Processamento de mensagens

O servidor aguarda que uma ligação seja estabelecida por um cliente válido. Quando uma ligação é estabelecida, o servidor processa a mensagem ao ler inicialmente o cabeçalho (Quadro 1) da mensagem.

Quadro 1 - Cabeçalho das mensagens

Cabeçalho das mensagens		
CÓDIGO	DATA SIZE	DATA

Caso o cabeçalho não cumpra os requisitos mínimos, a ligação é terminada. Caso o cabeçalho seja válido, é então extraído o código da mensagem. A estrutura auxiliar usada para definir uma mensagem válida para processamento é dada pela estrutura seguinte:

```
typedef struct mensagem_s
{
        char buffer[65536];
        int size;
        int rpos;
}mensagem_s;
```

Os dados são então lidos para a estrutura acima referida, formando assim a mensagem completa.

Para evocar a função apropriada para cada mensagem, é usado um callback que é evocado através de uma tabela de mensagens que contém os endereços das funções associadas a cada mensagem, sendo que a mensagem e o socket são enviados como argumentos.

A estrutura que mantem as referências de cada função e os respetivos ids das mensagens é definida de acordo com a seguinte estrutura.

```
typedef struct opcode_handler
{
        int msgId;
        func *callback;
}opcode_handler;
```

Tendo sido evocada a função apropriada para a mensagem lida, é então processado a mensagem, retornando então ao cliente uma resposta.

Durante o processamento da mensagem é verificado alguns dados para que seja minimamente seguro processar a mensagem sem que o servidor vá abaixo, essa verificação é feita em relação ao tamanho da mensagem contra o tamanho mínimo possível que a mensagem poderá ter, para que seja possível processar a mesma.

Caso ocorram erros de dados durante o processamento, o resultado da mensagem é acompanhado por um código de erro que poderá ser consultado na tabela de erros.

Criação das mensagens e manipulação das mesmas

Para que seja possível enviar mensagens entre o cliente e o servidor é necessário que de alguma forma seja possível formar uma mensagem e manipular o seu conteúdo. De tal forma que se definiu o seguinte procedimento básico para manipular o conteúdo da mesma.

Para inicializar uma mensagem usando a estrutura **mensagem_s**, é necessário "inicializar" a mensagem recorrendo à função **preparePacket** que prepara o cabeçalho da mensagem.

Para inserir dados na mensagem, pode ser utilizada uma das funções contidas no ficheiro msg_streamer.c que permite manipular diretamente o buffer de dados da mensagem. É possível inserir dados que vão desde os 8 aos 32 bits, incluindo strings completas. As funções que têm a denominação **msg_put** permitem inserir dados na mensagem. As funções que têm a denominação **msg_recv** permitem obter dados da mensagem.

Para finalizar uma mensagem e prepara-la para envio, é necessário por fim evocar a função **finalizePacket** que modifica o header da mensagem, atualizando o tamanho da mensagem.

Criação de sessões de cliente

Para que o servidor saiba se um cliente está ou não autentificado, é requerido que o servidor crie um token de sessão que ficará guardado em memória até este expirar. Cada sessão dura 10 minutos a partir do momento que o utilizador faz login.

Implementação do cliente

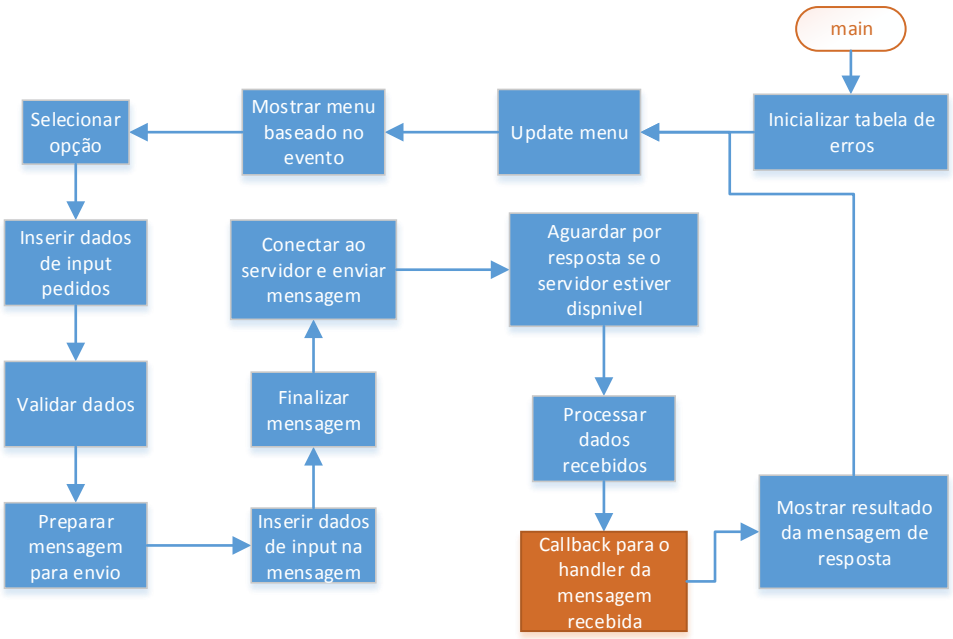

Figura 2: Fluxograma simples do cliente

O cliente ao inicializar começa por iniciar a tabela de erros suportada por este protocolo, processando de seguida eventos da consola, consoante os pedidos do utilizador.

Processamento de mensagens e criação das mensagens e manipulação das mesmas

O processamento e manipulação de mensagens é feito da mesma maneira que no servidor, sendo usado callbacks para as funções associadas a cada mensagem recebida a partir do servidor e usando as funções do ficheiro msg_streamer.c para manipular o seu conteúdo.

Processamento de pedidos e eventos de consola

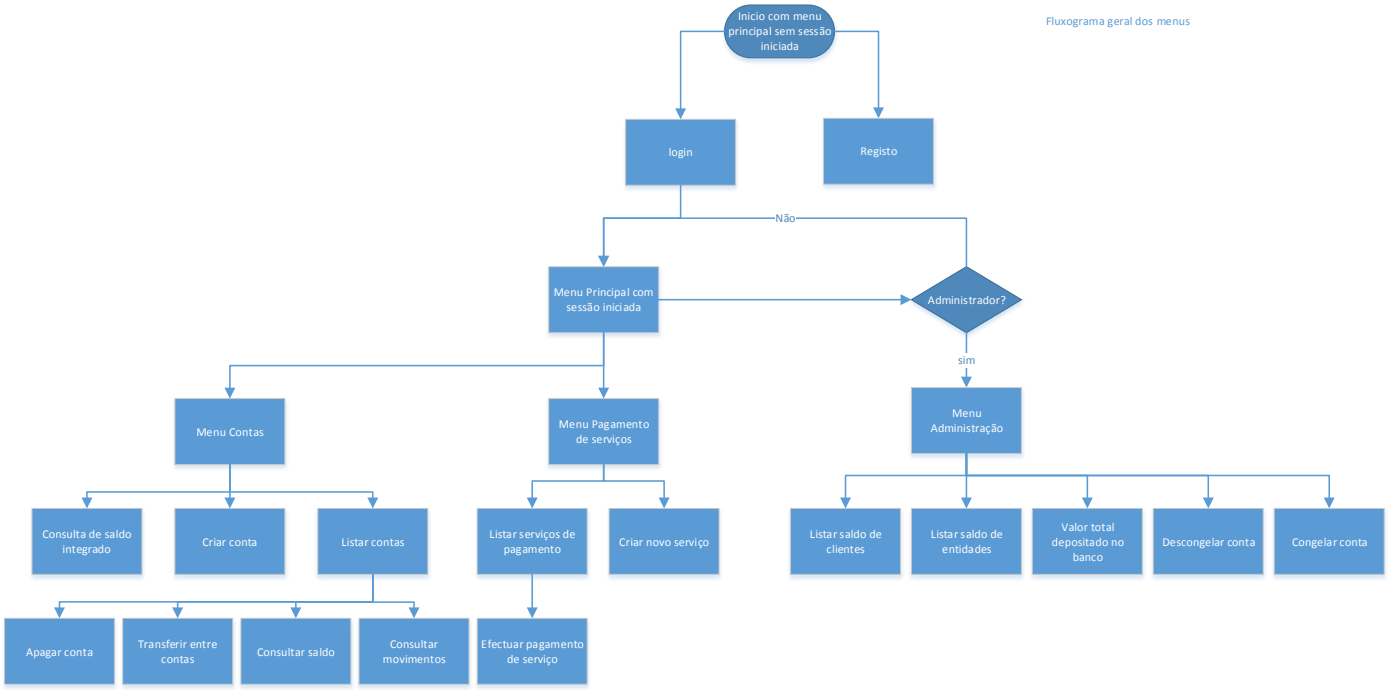

Figura 3: Fluxograma dos menus

Para processar os pedidos a partir da consola, é usado uma pequena máquina de estados em que ao ser lançado um evento, é mostrado o conteúdo adequado ao utilizador a partir da função **updateMenu**.

Quando é processado um evento de consola, o utilizador tem a opção de escolher uma das opções do menu que foi mostrada. Ao escolher uma opção do menu, irá ser pedido vários tipos de inputs, que caso seja válidos serão processados pelo cliente, formando uma mensagem válida, sendo de seguida enviada para o servidor para validação.

Dependendo também do tipo de utilizador (Cliente, Entidade, Admin) algumas opções poderão não estar disponíveis.

Quando uma mensagem é recebida pelo cliente, existe a possibilidade de ocorrer um erro. Esse erro é mostrado na consola consoante a tabela de erros em memória. Caso o resultado da resposta seja positiva, serão ou não mostrados conteúdos extra na consola, por exemplo, a listagem de contas ou serviços.

Mensagens suportadas por este protocolo

De seguida é apresentada uma lista de todas as mensagens suportadas por este protocolo, cada mensagem contem uma descrição do seu significado, a sua estrutura e os meta-dados da sua estrutura.

As mensagens que contem o formato CMSG são mensagens que são enviadas exclusivamente pelo cliente, enquanto as que contém o formato SMSG, são mensagens que são enviadas exclusivamente pelo servidor.

O nome da mensagem descrita por extenso não contem qualquer significado em termos práticos e apenas deverá ser usado para criar um enumerator para facilitar o uso de cada tipo de mensagem.

Desde que foi realizado o primeiro trabalho que foram feitas algumas alterações na estrutura de algumas mensagens, logo deve-se notar alguma redução no tamanho da mensagem.

Mensagens de cliente

Mensagem: CMSG_LOGIN

Definição: Permite enviar um pedido de login ao servidor

Código da mensagem: 1

Parâmetros a enviar: (Código, Tamanho da mensagem, Username, Password)

Meta-dados dos parâmetros: (char, short, string, string)

Mensagem: CMSG_REGISTO

Definição: Permite registar uma nova conta de utilizador

Código da mensagem: 2

Parâmetros a enviar: (Código, Tamanho da mensagem, Username, Password, Tipo de conta)

Meta-dados dos parâmetros: (char, short, string, string, int)

Mensagem: CMSG_CRIAR_CONTA

Definição: Permite criar uma conta

Código da mensagem: 3

Parâmetros a enviar: (Código, Tamanho da mensagem, Tamanho do token, Token)

Meta-dados dos parâmetros: (char, short, char, string)

Mensagem: CMSG_LISTAR_CONTAS_PROPRIAS

Definição: Permite obter uma lista de contas próprias

Código da mensagem: 4

Parâmetros a enviar: (Código, Tamanho da mensagem, Token)

Meta-dados dos parâmetros: (char, short, int)

Mensagem: CMSG_CONSULTAR_SALDO

Definição: Permite saber o saldo de uma conta

Código da mensagem: 5

Parâmetros a enviar: (Código, Tamanho da mensagem, Token, AccountID)

Meta-dados dos parâmetros: (char, short, char, int, int)

Mensagem: CMSG_CONSULTAR_MOVIMENTOS

Definição: Permite consultar os movimentos feitos por uma conta

Código da mensagem: 6

Parâmetros a enviar: (Código, Tamanho da mensagem, Token, AccountID)

Meta-dados dos parâmetros: (char, short, int, int)

Mensagem: CMSG_CONSULTA_SALDO_INTEGRADA

Definição: Permite consultar o saldo total da conta de cliente/entidade

Código da mensagem: 7

Parâmetros a enviar: (Código, Tamanho da mensagem, Token)

Meta-dados dos parâmetros: (char, short, int)

Mensagem: CMSG_MOVIMENTO_ENTRE_CONTAS

Definição: Permite movimentar dinheiro de uma conta para outra conta própria

Código da mensagem: 8

Parâmetros a enviar: (Código, Tamanho da mensagem, Token, Montante, Conta de origem, Conta de destino)

Meta-dados dos parâmetros: (char, short, int, int, int, int)

Mensagem: CMSG_APAGAR_CONTA

Definição: Permite apagar uma conta própria. O dinheiro é transferido para outra conta.

Código da mensagem: 9

Parâmetros a enviar: (Código, Tamanho da mensagem, Token, Conta de origem, Conta de destino)

Meta-dados dos parâmetros: (char, short, int, int, int)

Mensagem: CMSG_LISTAR_SERVICOS_PAGAMENTO

Definição: Permite obter uma lista de serviços de pagamento

Código da mensagem: 10

Parâmetros a enviar: (Código, Tamanho da mensagem, Token)

Meta-dados dos parâmetros: (char, short, int)

Mensagem: CMSG_EFECTUAR_PAGAMENTO_SERVICO

Definição: Permite efetuar o pagamento de um serviço

Código da mensagem: 11

Parâmetros a enviar: (Código, Tamanho da mensagem, Token, ID Serviço, Conta de origem, Montante)

Meta-dados dos parâmetros: (char, short, int, int, int, int)

Mensagem: CMSG_CRIAR_NOVO_SERVICO

Definição: Permite criar um novo serviço pela entidade

Código da mensagem: 12

Parâmetros a enviar: (Código, Tamanho da mensagem, Token, Nome do serviço, NIB)

Meta-dados dos parâmetros: (char, short, int, string, int)

Mensagem: CMSG_LISTAR_CLIENTES

Definição: Permite obter uma lista de clientes pelo administrador

Código da mensagem: 13

Parâmetros a enviar: (Código, Tamanho da mensagem, Token)

Meta-dados dos parâmetros: (char, short, int)

Mensagem: CMSG_LISTAR_ENTIDADES

Definição: Permite obter uma lista de entidades pelo administrador

Código da mensagem: 14

Parâmetros a enviar: (Código, Tamanho da mensagem, Tamanho do token, Token)

Meta-dados dos parâmetros: (char, short, char, string)

Mensagem: CMSG_VALOR_DEPOSITADO_BANCO

Definição: Permite obter o total de dinheiro depositado no banco

Código da mensagem: 15

Parâmetros a enviar: (Código, Tamanho da mensagem, Token)

Meta-dados dos parâmetros: (char, short, int)

Mensagem: CMSG_CONGELAR_CONTA

Definição: Permite congelar uma conta de cliente/entidade pelo administrador

Código da mensagem: 16

Parâmetros a enviar: (Código, Tamanho da mensagem, Token, Utilizador a ser congelado)

Meta-dados dos parâmetros: (char, short, int, int)

Mensagem: CMSG_DESCONGELAR_CONTA

Definição: Permite descongelar uma conta de cliente/entidade pelo administrador

Código da mensagem: 17

Parâmetros a enviar: (Código, Tamanho da mensagem, Token, Utilizador a ser descongelado)

Meta-dados dos parâmetros: (char, short, int, string)

Mensagens de servidor

Mensagem: SMSG_LOGIN_RESPONSE

Definição: Devolve o resultado de uma tentativa de login

Código da mensagem: 18

Parâmetros a enviar: (Código, Tamanho da mensagem, Resultado, Token, Tipo de conta)

Meta-dados dos parâmetros: (char, short, short, int, int)

Mensagem: SMSG_REGISTO_RESPONSE

Definição: Devolve o resultado de uma tentativa de registo

Código da mensagem: 19

Parâmetros a enviar: (Código, Tamanho da mensagem, Resultado)

Meta-dados dos parâmetros: (char, short, short)

Mensagem: SMSG_LISTAR_CONTAS_PROPRIAS_RESPONSE

Definição: Devolve uma lista de contas próprias

Código da mensagem: 20

Parâmetros a enviar: (Código, Tamanho da mensagem, Resultado, Total de contas, [ID Conta 1, …, ID Conta N])

Meta-dados dos parâmetros: (char, short, short, int, [int])

Mensagem: SMSG_CRIAR_CONTA_RESPONSE

Definição: Devolve o resultado da criação de uma conta própria

Código da mensagem: 21

Parâmetros a enviar: (Código, Tamanho da mensagem, Resultado)

Meta-dados dos parâmetros: (char, short, short)

Mensagem: SMSG_CONSULTAR_SALDO_RESPONSE

Definição: Devolve o saldo de uma conta própria

Código da mensagem: 22

Parâmetros a enviar: (Código, Tamanho da mensagem, Resultado, Saldo)

Meta-dados dos parâmetros: (char, short, short, int)

Mensagem: SMSG_CONSULTAR_MOVIMENTOS_RESPONSE

Definição: Devolve uma lista de movimentos efetuados em uma conta. O tipo de operação mencionada pode ser do tipo débito ou crédito

Código da mensagem: 23

Parâmetros a enviar: (Código, Tamanho da mensagem, Resultado, Total movimentos, [Tipo de operação 1, Quantia 1, Conta 1, …, Tipo de operação N, Quantia N, Conta N])

Meta-dados dos parâmetros: (char, short, short, short, [char, int, int])

Mensagem: SMSG_CONSULTA_SALDO_INTEGRADA_RESPONSE

Definição: Devolve o total do saldo de todas as contas

Código da mensagem: 24

Parâmetros a enviar: (Código, Tamanho da mensagem, Resultado, Saldo total)

Meta-dados dos parâmetros: (char, short, short, int)

Mensagem: SMSG_MOVIMENTO_ENTRE_CONTAS_RESPONSE

Definição: Devolve o resultado de mover dinheiro entre contas próprias

Código da mensagem: 25

Parâmetros a enviar: (Código, Tamanho da mensagem, Resultado)

Meta-dados dos parâmetros: (char, short, short)

Mensagem: SMSG_APAGAR_CONTA_RESPONSE

Definição: Devolve o resultado de apagar uma conta própria

Código da mensagem: 26

Parâmetros a enviar: (Código, Tamanho da mensagem, Resultado)

Meta-dados dos parâmetros: (char, short, short)

Mensagem: SMSG_LISTAR_SERVICOS_PAGAMENTO_RESPONSE

Definição: Devolve uma lista de serviços de pagamento

Código da mensagem: 27

Parâmetros a enviar: (Código, Tamanho da mensagem, Resultado, Total serviços, [ID Serviço 1, Nome do serviço 1, …, ID Serviço N, Nome do serviço N])

Meta-dados dos parâmetros: (char, short, short, int, [int, string])

Mensagem: SMSG_CRIAR_NOVO_SERVICO_RESPONSE

Definição: Devolve o resultado da criação de um novo serviço

Código da mensagem: 28

Parâmetros a enviar: (Código, Tamanho da mensagem, Resultado)

Meta-dados dos parâmetros: (char, short, short)

Mensagem: SMSG_EFECTUAR_PAGAMENTO_SERVICO_RESPONSE

Definição: Devolve o resultado da transação de um pagamento de serviço

Código da mensagem: 29

Parâmetros a enviar: (Código, Tamanho da mensagem, Resultado)

Meta-dados dos parâmetros: (char, short, short)

Mensagem: SMSG_LISTAR_CLIENTES_RESPONSE

Definição: Devolve uma lista de contas de cliente e o respetivo saldo

Código da mensagem: 30

Parâmetros a enviar: (Código, Tamanho da mensagem, Resultado, Total de clientes, [Username 1, Saldo 1, …, Username N, Saldo N])

Meta-dados dos parâmetros: (char, short, short, int, [string, int])

Mensagem: SMSG_LISTAR_ENTIDADES_RESPONSE

Definição: Devolve uma lista de entidades de cliente e o respetivo saldo

Código da mensagem: 31

Parâmetros a enviar: (Código, Tamanho da mensagem, Resultado, Total de clientes, [Username 1, Saldo 1, …, Username N, Saldo N])

Meta-dados dos parâmetros: (char, short, short, int, [string, int])

Mensagem: SMSG_VALOR_DEPOSITADO_RESPONSE

Definição: Devolve o total depositado no banco

Código da mensagem: 32

Parâmetros a enviar: (Código, Tamanho da mensagem, Resultado, Total depositado)

Meta-dados dos parâmetros: (char, short, short, int)

Mensagem: SMSG_VALOR_CONGELAR_CONTA_RESPONSE

Definição: Devolve o resultado da tentativa de congelação de uma conta

Código da mensagem: 33

Parâmetros a enviar: (Código, Tamanho da mensagem, Resultado)

Meta-dados dos parâmetros: (char, short, short)

Mensagem: SMSG_VALOR_DESCONGELAR_CONTA_RESPONSE

Definição: Devolve o resultado da tentativa de descongelação de uma conta

Código da mensagem: 34

Parâmetros a enviar: (Código, Tamanho da mensagem, Resultado)

Meta-dados dos parâmetros: (char, short, short)

Tabela de códigos de erro

As respostas dadas pelo servidor contém sempre algum tipo de resultado. Se a operação for executada com sucesso o erro por defeito adicionado às mensagens será o erro com o código 0.

Diferente das mensagens enviadas por este protocolo, o código de erros é codificado baseado num tipo de grupo de erros. Esse grupo é identificado pelo primeiro dígito do código seguido do erro nos seguintes dígitos.

1XX – Contem erros relacionados com a autentificação, registo, ou contas de cliente/entidade.

2XX – Contem erros relacionados com as operações efetuadas sob as contas próprias.

3XX – Contem erros relacionados com as operações efetuadas sob os serviços.

4XX – Contem erros relacionados com as operações efetuadas pelos administradores.

5XX – Contem erros gerais

Tabela de erros	
Código de erro	Mensagem
0	Operação feita com sucesso
502	Operação inválida
501	Sessão expirou
201	A conta não existe
202	Saldo Insuficiente
301	Nome de serviço inválido
302	NIB Inválido
303	O serviço não existe
401	Impossível congelar/descongelar contas administrativas
101	Nome de utilizador ou palavra-chave inválidos
102	Tipo de conta inválida
103	Nome de conta cliente/entidade inválida
104	Conta congelada

Estruturas e funções implementadas

De seguida segue um resumo de todas as funções implementadas neste trabalho prático. As funções são separadas em três grupos, cliente, servidor e shared (Funções partilhadas pelo cliente e servidor)

Cliente

Ficheiro: sconnection.c e sconnection.h

Contem funções que permite estabelecer ligações com o servidor.

Função: callback (func Callback, int arg1, mensagem_s *arg2) **Descrição:** Evocação da função com o handler da mensagem.
Função: InitConnection **Descrição:** Inicia a ligação ao servidor.
Função: InitConnection **Descrição:** Inicia a ligação ao servidor.
Função: SendData(int sock, char *mensagem, int size)

Descrição: Envia a mensagem codificada para o servidor.
Função: RecvData(int socket) **Descrição:** Permite receber dados do servidor.
Função: ConnectToServer(int sock) **Descrição:** Estabelece uma ligação ao servidor.
Função: ConnectAndSendRequest(mensagem_s * s) **Descrição:** Estabelece uma ligação ao servidor e envia a mensagem.

Ficheiro: mensagens.c e mensagens.h

Contém as funções que tratam de processar um opcode específico.

Função: initializeErrorTable() **Descrição:** Inicializa a tabela de erros.
Função: msg_no_action_handler() **Descrição:** Processamento genérico.
Função: msg_login_response_handler(int socket, mensagem_s *msg) **Descrição:** Processamento do login.
Função: msg_registo_response_handler(int socket, mensagem_s * msg) **Descrição:** Processamento do registo.
Função: msg_listar_contas_proprias_response_handler(int socket, mensagem_s * msg) **Descrição:** Processamento da listagem de contas próprias.
Função: msg_criar_conta_response_handler(int socket, mensagem_s * msg) **Descrição:** Processamento da criação de uma conta
Função: msg_consultar_saldo_response_handler(int socket, mensagem_s * msg) **Descrição:** Processamento da consulta do saldo de uma conta.
Função: msg_consultar_movimentos_response_handler(int socket, mensagem_s * msg) **Descrição:** Processamento da consulta de movimentos.
Função: msg_consulta_saldo_integrada_handler(int socket, mensagem_s * msg) **Descrição:** Processamento da consulta de saldo integrada.
Função: msg_movimento_entre_contas_handler(int socket, mensagem_s * msg) **Descrição:** Processamento do movimento entre contas.
Função: msg_apagar_conta_response_handler(int socket, mensagem_s * msg) **Descrição:** Processamento do apagar conta.
Função: msg_listar_servicos_pagamento_handler(int socket, mensagem_s * msg) **Descrição:** Processamento da listagem de serviços.
Função: msg_criar_novo_servico_handler(int socket, mensagem_s * msg) **Descrição:** Processamento da criação de um novo serviço.
Função: msg_efectuar_pagamento_servico_handler(int socket, mensagem_s * msg) **Descrição:** Processamento do efetuar pagamento de serviço.
Função: msg_listar_clientes_handler(int socket, mensagem_s * msg) **Descrição:** Processamento da listagem de clientes.
Função: msg_listar_entidades_handler(int socket, mensagem_s * msg) **Descrição:** Processamento da listagem de entidades.
Função: msg_valor_depositado_handler(int socket, mensagem_s * msg) **Descrição:** Processamento do valor depositado no banco.
Função: msg_congelar_conta_handler(int socket, mensagem_s * msg) **Descrição:** Processamento da congelação de uma conta.
Função: msg_descongelar_conta_handler(int socket, mensagem_s * msg) **Descrição:** Processamento da descongelação de uma conta.

Ficheiro: consola.c e consola.h

Contém as funções que tratam dos eventos acionados através da consola e o envio de mensagens para o servidor.

Função: mostrarMenuAuth() **Descrição:** Mostra o menu sem início de sessão.
Função: mostrarMenuPrincipal() **Descrição:** Mostra o menu com início de sessão.
Função: mostrarMenuContas() **Descrição:** Mostra o menu das contas.
Função: mostrarMenuServico() **Descrição:** Mostra o menu do serviço.
Função: mostrarMenuServicos() **Descrição:** Mostra o menu dos serviços.
Função: mostrarMenuConta() **Descrição:** Mostra o menu das operações sobre uma conta.
Função: mostrarMenuAdmin() **Descrição:** Mostra o menu administrativo.
Função: updateMenu() **Descrição:** Handling eventos do menu.

Ficheiro: cliente.c e cliente.h

Contem as funções principais do programa.

Função: main() **Descrição:** Função principal.

Servidor

Ficheiro: servidor.c e servidor.h

Contem funções que permite escutar por ligações e processar as mensagens para as funções apropriadas.

Função: callback(func Callback, int arg1, mensagem_s *arg2) **Descrição:** Evocação da função com o handler da mensagem.
Função: inicializarServidor() **Descrição:** Inicializa o servidor.
Função: disconnect(int socket) **Descrição:** Desconecta o cliente.
Função: recvData(int socket) **Descrição:** Recebe dados do socket.
Função: sendData(int socket, mensagem_s *s) **Descrição:** Envia dados para o cliente.
Função: waitForClient(int socket) **Descrição:** Espera por uma ligação do cliente.
Função: main() **Descrição:** Função principal.

Ficheiro: mensagens.c e mensagens.h

Contem as funções que permite processar cada mensagem individualmente.

Função: msg_no_action_handler(int socket, mensagem_s *msg) **Descrição:** Processamento para mensagens sem um handler específico.
Função: msg_registo_handler(int socket, mensagem_s *msg) **Descrição:** Processamento do registo de novo utilizador.
Função: msg_login_handler(int socket, mensagem_s *msg) **Descrição:** Processamento do login.
Função: msg_criar_conta_handler(int socket, mensagem_s *msg) **Descrição:** Processamento da criação de uma conta.
Função: msg_listar_contas_proprias_handler(int socket, mensagem_s *msg) **Descrição:** Processamento da listagem de contas próprias.
Função: msg_consultar_saldo_handler(int socket, mensagem_s *msg) **Descrição:** Processamento da listagem do saldo.
Função: msg_consultar_movimentos_handler(int socket, mensagem_s *msg) **Descrição:** Processamento da consulta de movimentos de uma conta.
Função: msg_consulta_saldo_integrada_handler(int socket, mensagem_s *msg) **Descrição:** Processamento da consulta de saldo integrada.
Função: msg_movimento_entre_contas_handler(int socket, mensagem_s *msg) **Descrição:** Processamento do movimento entre contas.
Função: msg_apagar_conta_handler(int socket, mensagem_s *msg) **Descrição:** Processamento do apagar conta.
Função: msg_listar_servicos_handler(int socket, mensagem_s *msg) **Descrição:** Processamento da listagem dos serviços.
Função: msg_efectuar_pagamento_servico_handler(int socket, mensagem_s * msg) **Descrição:** Processamento do pagamento de um serviço.
Função: msg_criar_novo_servico_handler(int socket, mensagem_s *msg) **Descrição:** Processamento da criação de um novo serviço.
Função: msg_listar_clientes_handler(int socket, mensagem_s *msg) **Descrição:** Processamento da listagem de clientes.
Função: msg_listar_entidades_handler(int socket, mensagem_s *msg) **Descrição:** Processamento da listagem de entidades.
Função: msg_valor_depositado_banco_handler(int socket, mensagem_s *msg) **Descrição:** Processamento do valor depositado no banco.
Função: msg_congelar_conta_handler(int socket, mensagem_s *msg) **Descrição:** Processamento da congelação de uma conta.
Função: msg_descongelar_conta_handler(int socket, mensagem_s *msg) **Descrição:** Processamento da descongelação de uma conta.

Ficheiro: linkedlist.c e linkedlist.h

Contem uma implementação de uma lista simplesmente ligada.

Função: LinkedList *list_new() **Descrição:** Cria uma nova lista.
Função: list_empty(LinkedList *list) **Descrição:** Verifica se a lista esta vazia.
Função: list_insert(void *new_element, LinkedList *list) **Descrição:** insere um element na lista.
Função: list_remove(void *value, LinkedList *list) **Descrição:** Remove um elemento da lista.

Função: list_destroy(LinkedList *list, bool destroyElements) **Descrição:** Apaga todo o conteúdo da lista.
Função: Begin(LinkedList *list) **Descrição:** Início da iteração dos elementos da lista.
Função: Next(LinkedListNode *node) **Descrição:** Obtem o seguinte elemento da lista na iteração.

Ficheiro: database.c e database.h

Contem funções que permite ler e gravar dados numa database.

Função: genMovimentoID() **Descrição:** Gera um id para os movimentos.
Função: genServicoID() **Descrição:** Gera um id para os serviços.
Função: genToken() **Descrição:** Gera um id para um token criado.
Função: genUtilizadorID() **Descrição:** Gera um id para os utilizadores.
Função: genAccountID() **Descrição:** Gera um id para as contas.
Função: inicializarDatabase() **Descrição:** Inicializa a database.
Função: destroyDatabase() **Descrição:** Liberta os recursos ocupados pela database.
Função: ContasAdd(Conta *c) **Descrição:** Adiciona uma conta à database.
Função: UtilizadorAdd(Utilizador *c) **Descrição:** Adiciona um utilizador à database.
Função: SessionAdd(Session *s) **Descrição:** Adiciona sessão à database.
Função: MovimentoAdd(Movimento *m) **Descrição:** Adiciona um movimento à database.
Função: ServicoAdd(Servico *s) **Descrição:** Adiciona um serviço à database.
Função: GetCliente(char *username) **Descrição:** Obtém um utilizador da database.
Função: GetSession(int token) **Descrição:** Obtém uma sessão da database.
Função: GetConta(int id) **Descrição:** Obtém uma conta da database.
Função: GetMovimento(int id) **Descrição:** Obtém um movimento da database.
Função: GetServico(int id) **Descrição:** Obtém um serviço da database.
Função: list_get_conta(int id, LinkedList *list) **Descrição:** Obtém uma conta da database.
Função: list_get_client(char *username, LinkedList *list) **Descrição:** Obtém um utilizador da database.
Função: list_get_session(int token, LinkedList *list) **Descrição:** Obtém uma sessão da database.
Função: list_get_movimento(int id, LinkedList *list)

Descrição: obtém um movimento da database.
Função: list_get_servico(int id, LinkedList *list) **Descrição:** Obtém um serviço da database.
Função: DBInsertUtilizador(Utilizador *utilizador) **Descrição:** Insere um utilizador na database.
Função: DBLoadUtilizadores() **Descrição:** Carrega os utilizadores da database.
Função: DBWriteUtilizadores() **Descrição:** Escreve todos os utilizadores em memória na database.
Função: DBInsertConta(Conta *conta) **Descrição:** Insere uma conta na database.
Função: DBLoadContas() **Descrição:** Carrega as contas da database.
Função: DBWriteContas() **Descrição:** Escreve todas as contas em memória na database.
Função: DBInsertMovimento(Movimento *movimento) **Descrição:** Insere um movimento na database.
Função: DBLoadMovimentos() **Descrição:** Carrega os movimentos da database.
Função: DBWriteMovimentos() **Descrição:** Escreve todos os movimentos em memória na database.
Função: DBInsertServico(Servico *servico) **Descrição:** insere um serviço na database.
Função: DBLoadServicos() **Descrição:** Carrega os serviços da database.
Função: DBWriteServicos() **Descrição:** Escreve todos os serviços em memória na database.
Função: createNewDatabaseUtilizadores() **Descrição:** Cria uma database vazia para os utilizadores.
Função: createNewDatabaseContas() **Descrição:** Cria uma database vazia para as contas.
Função: createNewDatabaseMovimentos() **Descrição:** Cria uma database vazia para os movimentos.
Função: createNewDatabaseServicos() **Descrição:** Cria uma database vazia para os serviços.

Shared

Ficheiro: servidor.c e servidor.h

Contem funções que permitem manipular os dados de um buffer. Normalmente este buffer é um array de chars.

Função: preparePacket(char msgId) **Descrição:** Prepara a mensagem, adicionando o header.
Função: finalizePacket(mensagem_s *s) **Descrição:** Finaliza a construção da mensagem, corrigindo o tamanho total dos dados.
Função: msg_put_byte(char value, mensagem_s *msg) **Descrição:** Insere um byte na mensagem.
Função: msg_put_byte_at(int pos, char value, mensagem_s * msg) **Descrição:** Insere um byte na posição designada.
Função: msg_put_short(short value, mensagem_s *msg)

Descrição: Insere um valor de 16 bits na mensagem.
Função: msg_put_int(int value, mensagem_s *msg) **Descrição:** Insere um valor de 32 bits na mensagem.
Função: msg_put_int_at(int pos, int value, mensagem_s * msg) **Descrição:** Insere um valor de 32 bits na posição designada.
Função: msg_put_int64(__int64 value, mensagem_s *msg) **Descrição:** Insere um valor de 64 bits na mensagem.
Função: msg_put_string(char *value, mensagem_s *msg) **Descrição:** Insere uma string na mensagem.
Função: msg_recv_byte(mensagem_s *msg) **Descrição:** Obtém um byte da mensagem.
Função: msg_recv_short(mensagem_s *msg) **Descrição:** Obtém um valor de 16 bits da mensagem.
Função: msg_recv_int(mensagem_s *msg) **Descrição:** Obtém um valor de 32 bits da mensagem.
Função: msg_recv_int64(mensagem_s *msg) **Descrição:** Obtém um valor de 64 bits da mensagem.
Função: msg_recv_string(mensagem_s *msg) **Descrição:** Obtém uma string da mensagem.
Função: StringValida(char *buffer, int maxlen) **Descrição:** Valida uma string dado um tamanho máximo.

Estruturas implementadas

typedef struct opcode_handler **{** **int msgId;** **func *callback;** **}opcode_handler;**	**typedef struct LinkedListNode** **{** **struct LinkedListNode *next;** **int index;** **void *element;** **}LinkedListNode;**	**typedef struct LinkedList** **{** **LinkedListNode *head;** **LinkedListNode *current;** **int size;** **}LinkedList;**
Contém as referências de cada função para cada mensagem.	Representa um nó de uma linkedlist.	Representa uma linkedlist.
typedef struct Utilizador **{** **int id;** **int flags;** **char username[32];** **char password[32];** **bool congelada;** **}Utilizador;**	**typedef struct Session** **{** **int token;** **time_t sessionExpire;** **Utilizador *cliente;** **}Session;**	**typedef struct Conta** **{** **int contaID;** **int clientID;** **int saldo;** **}Conta;**
Contém os dados de um utilizador.	Contém os dados de uma sessão.	Contém os dados de uma conta.
typedef struct Movimento **{** **int id;** **int tipo;** **int owner;** **int contaO;** **int ContaD;** **int Montante;** **}Movimento;**	**typedef struct Servico** **{** **int id;** **char nome[256];** **int nib;** **}Servico;**	**typedef struct mensagem_s** **{** **char buffer[65536];** **int size;** **int rpos;** **}mensagem_s;**
Contém os dados de um movimento.	Contém os dados de um serviço.	Contém os dados de uma mensagem.

CONCLUSÃO

Com este trabalho, pretendeu-se adquirir alguns conhecimentos em como implementar uma abordagem cliente-servidor de forma simplificada. Penso que os objetivos foram atingidos embora haja algumas partes que pudessem ser melhoradas, como o suporte para envio de múltiplas mensagens ao realizar uma operação única. No entanto penso que a qualidade do trabalho em si está bastante boa.

Embora este trabalho não parecesse difícil ao inicio, notou-se que acabou por ser um trabalho bastante demoroso na sua implementação, logo algumas coisas podem não ter ficado tão boas. Penso de qualquer forma que os objetivos foram todos alcançados.