Relatório de Entrega de Trabalho
Disciplina de Programação Paralela e Distribuída (PPD) – Prof. César De Rose

Alunos: Luiz Fernando Soares e Pedro Henrique Leite **Usuários:** luizbalczarek e pedroleite21
Exercício: Trabalho 1 de RPC (Modelo Cliente/Servidor) **Entrega:** 09/04/2019

1) Modelagem

A modelagem do trabalho se deu com a definição de três estruturas para a implementação do exercício. Essas estruturas eram: a estrutura com as informações de uma conta, outra com as informações sobre um transação, no caso eram depósito e saque, além de uma estrutura para a geração de tokens, os quais verificam cada operação feita pela conta durante a execução.

Com a modelagem pronta dessas estruturas, foram definidas quais seriam as funções necessárias para o funcionamento dos processos do exercício.

2) Funções

Foram criadas oito funções pelo RPCGEN para a implementação do exercício. Essas funções consistem em operações bancárias, sendo elas buscar o saldo de uma conta, operação de saque e depósito, além de abertura e fechamento de contas. Para a verificação dos tokens foram criadas duas funções, uma para gerar token e outra para verificá-lo.

No lado do servidor, a maioria das funções RPC usavam diretamente o array global das com a estrutura das contas cadastradas no banco. Além disso, foi criada uma função auxiliar para verificar se a conta já existia dentro desse array.

No lado do cliente, as chamadas das funções do RPC foram encapsuladas a fim de melhor organização e visualização do código.

3) Implementação

O exercício consistia em implementar três processos relacionados com operações bancárias. O processo Administrativo foi implementado no servidor, e os processos Caixa Eletrônico e Agência foram implementados em um só cliente.

Para implementarmos estes dois processos dentro de um só cliente, foi criado um switch no início para ser identificado qual processo o cliente vai executar.

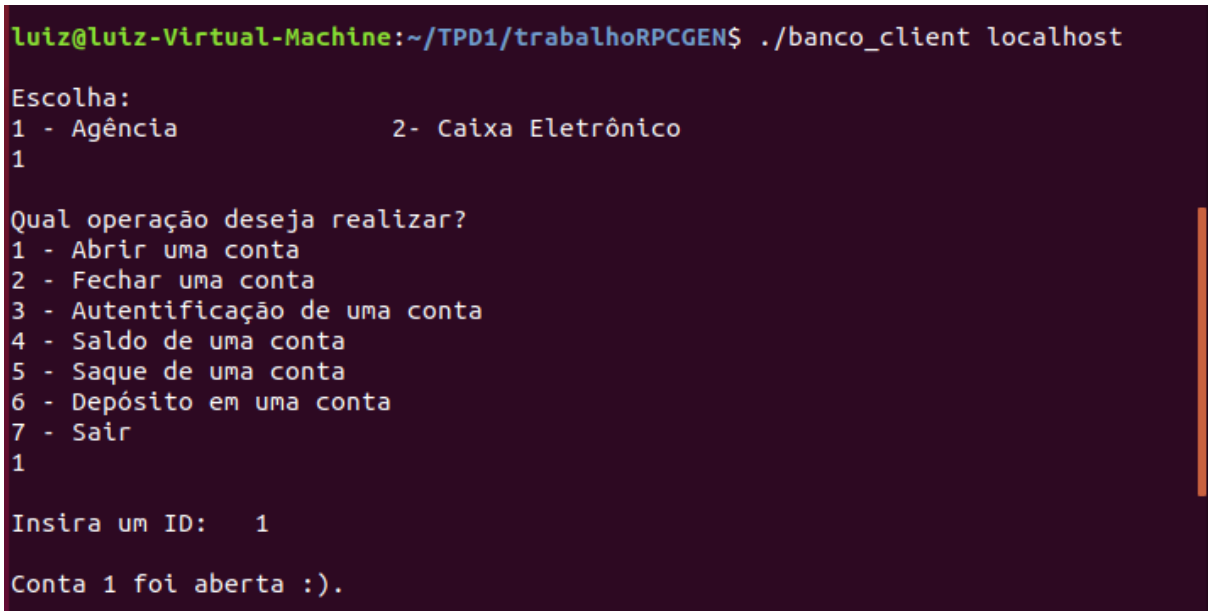
O lado cliente do programa consiste muito na interação com o usuário, então a implementação foi muito baseada na experiência dentro do console.

Para a implementação da não-idempotência entre o cliente e servidor, escolhemos como estratégia do servidor o envio do ACK depois de realizar a operação, e como estratégia do cliente, nunca realizar o pedido, já que o console não se abre para novas requisições até chegar uma resposta do servidor. Se qualquer resposta do servidor for *-1*, significa erro fatal, ou seja, essa operação não pode ser completada. A opção “5 - Saque de uma conta (com falha)” para forçar essa falha. Em um saque normal o servidor retorna 0 (OK) ou -1 (NOK) para o cliente, no saque com falha o servidor retorna 2. O server faz o saque na conta e guarda as informações da transação e com retorno 2 o *client* identifica que ocorreu timeout. Caso o usuário tente refazer a mesma transação, o server identifica que ela já foi feita e retorna 3, então o *client* mostra para o usuário que o saque já havia sido feito anteriormente.

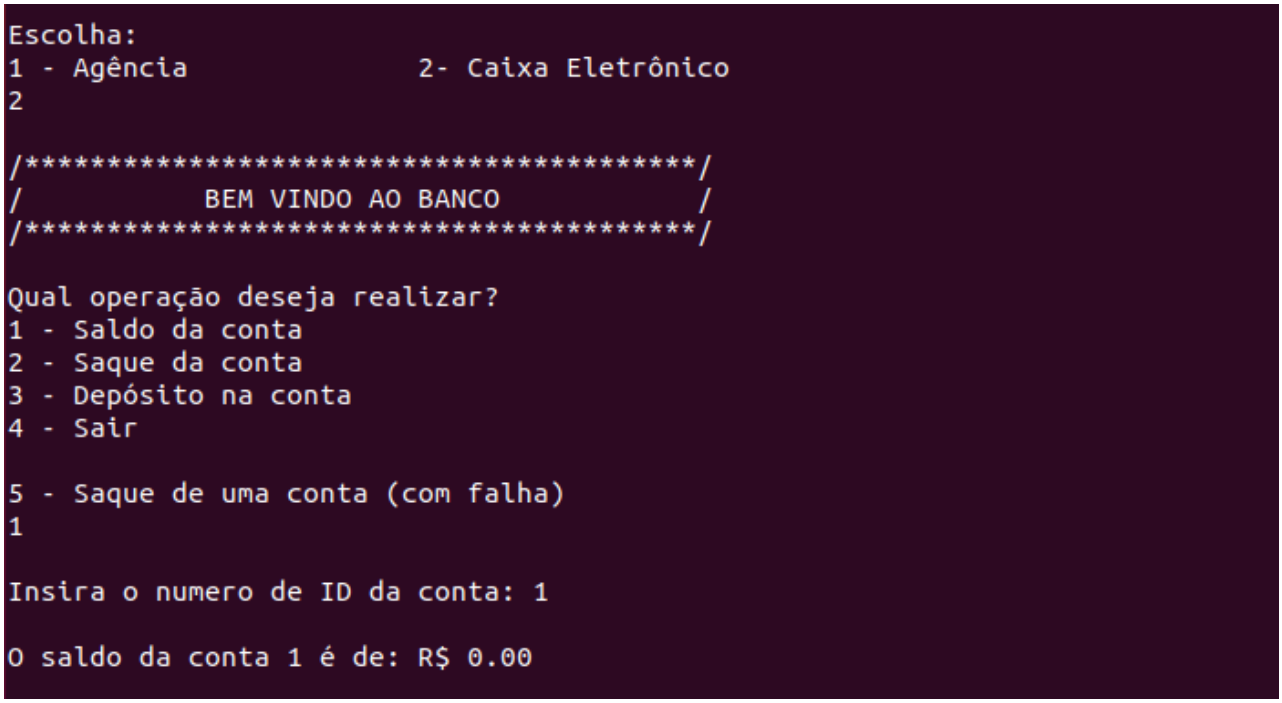
4) Dificuldades Encontradas

As dificuldades encontradas durante o exercício foram a utilização do RPCGEN, pois era uma ferramenta nova, a qual não tínhamos familiaridade.

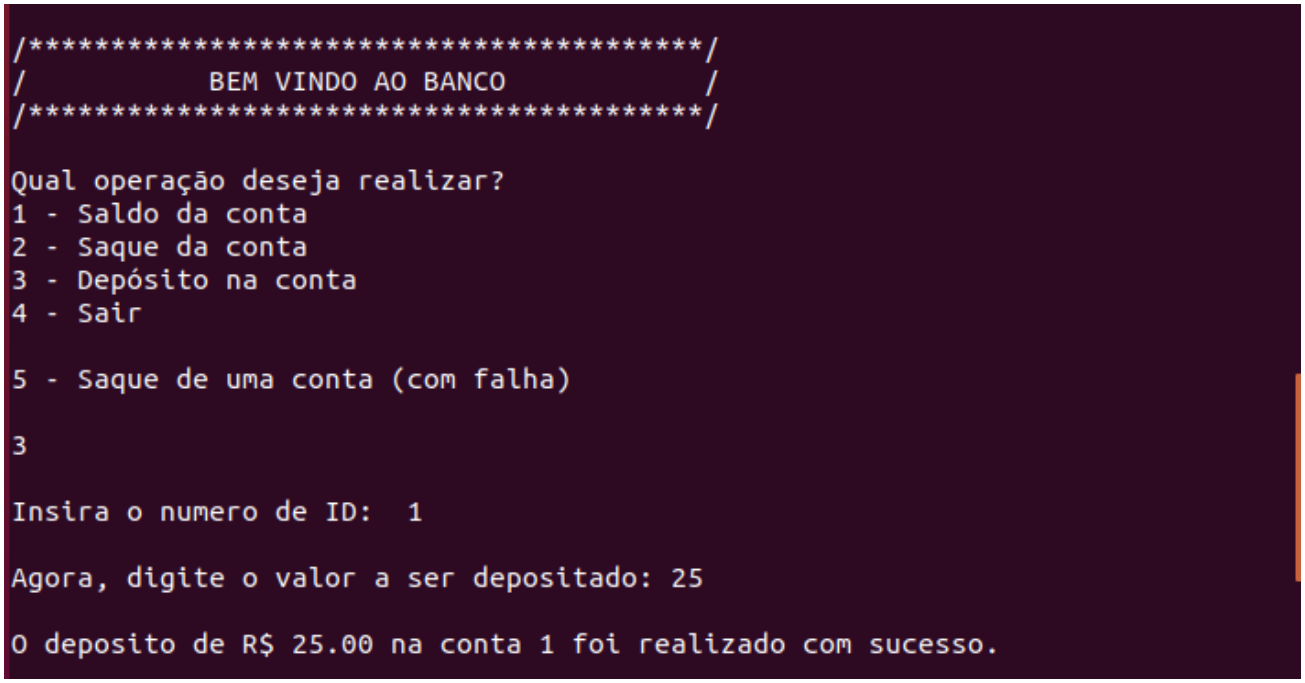
Outra dificuldade encontrada foi em como injetar falhas em alguma das opções para testar a não-idempotência.



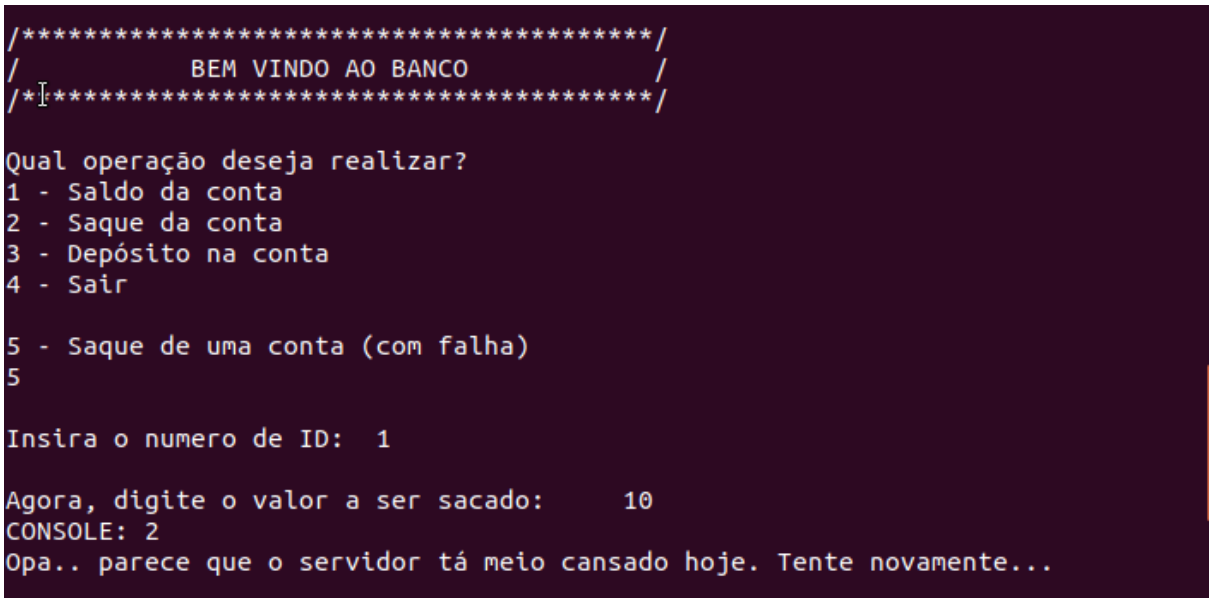
Essa imagem demonstra o switch entre os processos Agência e Caixa Eletrônico, além das operações disponíveis para o processo Agência. Na imagem foi realizado a abertura de uma conta.



Já nessa imagem, quando já tinha sido aberta uma conta no processo Agência, foi utilizado o processo Caixa Eletrônico, utilizando a função de retornar saldo.



Depósito feito pelo Caixa Automático.



Saque com erro forçado (retorno 2), para testar a não-idempotência.

```
/****************************************/
/           BEM VINDO AO BANCO           /
/****************************************/

Qual operação deseja realizar?
1 - Saldo da conta
2 - Saque da conta
3 - Depósito na conta
4 - Sair

5 - Saque de uma conta (com falha)
5

Insira o numero de ID:  1

Agora, digite o valor a ser sacado:     10
CONSOLE: 3
Essa operação parece já ter sido feita.
```

Usuário tenta realizar o mesmo saque novamente, é verificado no server que o usuário está tentando realizar a mesma operação novamente e apenas informa que o saque foi realizado anteriormente.

```
/****************************************/
/           BEM VINDO AO BANCO           /
/****************************************/

Qual operação deseja realizar?
1 - Saldo da conta
2 - Saque da conta
3 - Depósito na conta
4 - Sair

5 - Saque de uma conta (com falha)

2

Insira o numero de ID:  1

Agora, digite o valor a ser sacado:     3

O saque de R$ 3.00 da conta 1 foi realizado com sucesso.
```

Opção de saque do Caixa Automático sem erros.

```
Qual operação deseja realizar?
1 - Abrir uma conta
2 - Fechar uma conta
3 - Autentificação de uma conta
4 - Saldo de uma conta
5 - Saque de uma conta
6 - Depósito em uma conta
7 - Sair
2

Insira um ID:   1

A conta 1 foi fechada com sucesso.

Qual operação deseja realizar?
1 - Abrir uma conta
2 - Fechar uma conta
3 - Autentificação de uma conta
4 - Saldo de uma conta
5 - Saque de uma conta
6 - Depósito em uma conta
7 - Sair
4

Insira o numero de ID da conta: 1

A conta 1 nao existe.
```

Fechamento de conta pela Agência e verificação para tentar se a conta realmente foi encerrada.

```c
/*
        BANCO_CLIENT.C
         * These are only templates and you can use them
         * as a guideline for developing your own functions.
         */


        #include "banco.h"
        #include <stdio.h>
        #include <stdlib.h>
        #include <time.h>


        #define NUM_CONTA 100


        CLIENT *clnt;
        int *result_1;
        int abreconta_1_arg;
        int *result_2;
        int fechaconta_1_arg;
        int *result_3;
        int authconta_1_arg;
        int *result_4;
        transacao deposito_1_arg;
        int *result_5;
        transacao saque_1_arg;
        float *result_6;
        int retornasaldo_1_arg;
        int *result_7;
        token checksenha_1_arg;
        int *result_8;
        token gerasenha_1_arg;


        token tokenContas[NUM_CONTA];


        typedef struct {
                int transacao;
                float valor;
                int token;
                int conta;
        } HISTORICO;
```

```
HISTORICO transacoes[NUM_CONTA];



int checarSenha(int conta)
{
        tokenContas[conta].conta_id = conta;


        checksenha_1_arg = tokenContas[conta];


        result_7 = checksenha_1(&checksenha_1_arg, clnt);
        if (result_7 == (int *)NULL)
        {
                clnt_perror(clnt, "call failed");
        }


        return *result_7;
}


void gerarNovaSenha(int conta)
{
        gerasenha_1_arg = tokenContas[conta];


        result_8 = gerasenha_1(&gerasenha_1_arg, clnt);
        if (result_8 == (int *)NULL)
        {
                clnt_perror(clnt, "call failed");
        }
        else
        {
                if (*result_8 != -1)
                {
                        tokenContas[conta].token = *result_8;
                }
                else
                {
                        exit(1);
                }
        }
}
```

```
void abrirConta()
{
        printf("\nInsira um ID:\t");
        scanf("%d", &abreconta_1_arg);


        result_1 = abreconta_1(&abreconta_1_arg, clnt);
        if (result_1 == (int *)NULL)
        {
                clnt_perror(clnt, "call failed");
        }
        else
        {
                if (*result_1 == -1)
                {
                        printf("\nConta %d não foi aberta, já existe alguém com esse
número :(.\n", abreconta_1_arg);
                }
                else
                {
                        printf("\nConta %d foi aberta :).\n", abreconta_1_arg);
                }
        }
}


void fecharConta()
{
        printf("\nInsira um ID:\t");
        scanf("%d", &fechaconta_1_arg);


        result_2 = fechaconta_1(&fechaconta_1_arg, clnt);
        if (result_2 == (int *)NULL)
        {
                clnt_perror(clnt, "call failed");
        }
        else
        {
                if (*result_2 == -1)
                {
                        printf("\nNao existe a conta %d, portanto nao pode ser
fechada. \n", fechaconta_1_arg);
                }
```

```
                else
                {
                        printf("\nA conta %d foi fechada com sucesso. \n",
fechaconta_1_arg);
                }
        }
}


void depositoConta()
{
        int id;
        float valor;


        printf("\nInsira o numero de ID:\t");
        scanf("%d", &id);


        if (checarSenha(id) == 0)
        {
                printf("\nAgora, digite o valor a ser depositado:\t");
                scanf("%f", &valor);


                deposito_1_arg.id = id;
                deposito_1_arg.saldo = valor;


                result_4 = deposito_1(&deposito_1_arg, clnt);
                if (result_4 == (int *)NULL)
                {
                        clnt_perror(clnt, "call failed");
                }
                else
                {
                        if (*result_4 == -1)
                        {
                                printf("\nO deposito na conta %d nao pôde ser
finalizado.\n", id);
                        }
                        else
                        {
                                printf("\nO deposito de R$ %.2f na conta %d foi
realizado com sucesso.\n", valor, id);
                                gerarNovaSenha(id);
```

```
                        }

                }
        }
        else
        {
                printf("\n Senha da conta %d expirada \n", id);
        }
}


void saqueConta()
{
        int id;
        float valor;


        printf("\nInsira o numero de ID:\t");
        scanf("%d", &id);


        if (checarSenha(id) == 0)
        {
                printf("\nAgora, digite o valor a ser sacado:\t");
                scanf("%f", &valor);


                saque_1_arg.id = id;
                saque_1_arg.saldo = valor;
                saque_1_arg.erro = 0;


                result_5 = saque_1(&saque_1_arg, clnt);
                if (result_5 == (int *)NULL)
                {
                        clnt_perror(clnt, "call failed");
                }
                else
                {
                        if (*result_5 == -1)
                        {
                                printf("\nO saque da conta %d nao pôde ser
finalizado.\n", id);
                        }
                        else
                        {
```

```
                        printf("\nO saque de R$ %.2f da conta %d foi realizado
com sucesso.\n", valor, id);
                    }
                    gerarNovaSenha(id);
                }
        }
        else
        {
                printf("\n Senha da conta %d expirada \n", id);
        }
}


void getSaldo()
{
        printf("\nInsira o numero de ID da conta:\t");
        scanf("%d", &retornasaldo_1_arg);


        if (checarSenha(retornasaldo_1_arg) == 0)
        {
                result_6 = retornasaldo_1(&retornasaldo_1_arg, clnt);
                if (result_6 == (float *)NULL)
                {
                        clnt_perror(clnt, "call failed");
                }
                else
                {
                        if (*result_6 == -1.0)
                        {
                                printf("\nA conta %d nao existe. \n",
retornasaldo_1_arg);
                        }
                        else
                        {
                                printf("\nO saldo da conta %d é de: R$ %.2f \n",
retornasaldo_1_arg, *result_6);
                                gerarNovaSenha(retornasaldo_1_arg);
                        }
                }
        }
        else
        {
                printf("\n Senha da conta %d expirada \n", retornasaldo_1_arg);
        }
}
```

```
void checarConta()
{
        printf("\nInsira o numero de ID:\t");
        scanf("%d", &authconta_1_arg);


        if (checarSenha(authconta_1_arg) == 0)
        {
                result_3 = authconta_1(&authconta_1_arg, clnt);
                if (result_3 == (int *)NULL)
                {
                        clnt_perror(clnt, "call failed");
                }
                else
                {
                        if (*result_3 == -1)
                        {
                                printf("A conta %d nao existe. :(\n",
authconta_1_arg);
                        }
                        else
                        {
                                printf("A conta %d existe, e pronta pra ser usada
:)\n", authconta_1_arg);
                                gerarNovaSenha(authconta_1_arg);
                        }

                }
        }
        else
        {
                printf("\n Senha da conta %d expirada \n", authconta_1_arg);
        }
}


void saqueComErro() {
        int id;
        float valor;


        printf("\nInsira o numero de ID:\t");
        scanf("%d", &id);
```

```
        if (checarSenha(id) == 0)
        {
                printf("\nAgora, digite o valor a ser sacado:\t");
                scanf("%f", &valor);


                saque_1_arg.id = id;
                saque_1_arg.saldo = valor;
                saque_1_arg.erro = 1;


                result_5 = saque_1(&saque_1_arg, clnt);
                if (result_5 == (int *)NULL)
                {
                        clnt_perror(clnt, "call failed");
                }
                else
                {
                        printf("CONSOLE: %d", *result_5);
                        if (*result_5 == -1)
                        {
                                printf("\nO saque da conta %d nao pôde ser
finalizado.\n", id);
                        } else if (*result_5 == 2)
                        {
                                printf("\nOpa.. parece que o servidor tá meio cansado
hoje. Tente novamente...\n");
                        } else if(*result_5 == 3)
                        {
                                printf("\nEssa operação parece já ter sido feita.\n");
                        }
                        else
                        {
                                printf("\nO saque de R$ %.2f da conta %d foi realizado
com sucesso.\n", valor, id);
                        }
                }
        }
        else
        {
                printf("\n Senha da conta %d expirada \n", id);
        }
}
```

```c
void banco_prog_1(char *host, int opcao)
{
	int operacao = 0;


	checksenha_1_arg.token = 0;


#ifndef DEBUG
	clnt = clnt_create(host, BANCO_PROG, BANCO_VERS, "udp");
	if (clnt == NULL)
	{
		clnt_pcreateerror(host);
		exit(1);
	}
#endif /* DEBUG */


	while (1)
	{
		switch (opcao)
		{
		case 1:
			//Agencia
			printf("\nQual operação deseja realizar?\n");
			printf("1 - Abrir uma conta\n");
			printf("2 - Fechar uma conta\n");
			printf("3 - Autentificação de uma conta\n");
			printf("4 - Saldo de uma conta\n");
			printf("5 - Saque de uma conta\n");
			printf("6 - Depósito em uma conta\n");
			printf("7 - Sair\n");
			scanf("%d", &operacao);


			switch (operacao)
			{
			case 1:
				abrirConta();
				break;
			case 2:
				fecharConta();
				break;
			case 3:
				checarConta();
				break;
```

```
        case 4:
                getSaldo();
                break;
        case 5:
                saqueConta();
                break;
        case 6:
                depositoConta();
                break;
        case 7:
                exit(1);
        default:
                printf("\nArgumento inválido!!\n");
                break;
        }


        break;
case 2:
{
        //Caixa Eletronico
        printf("\n/*************************************/\n");
        printf("/           BEM VINDO AO BANCO            /\n");
        printf("/*************************************/\n");


        printf("\nQual operação deseja realizar?\n");
        printf("1 - Saldo da conta\n");
        printf("2 - Saque da conta\n");
        printf("3 - Depósito na conta\n");
        printf("4 - Sair\n\n");


        printf("5 - Saque de uma conta (com falha)\n");
        scanf("%d", &operacao);


        switch (operacao)
        {
        case 1:
                getSaldo();
                break;
        case 2:
                saqueConta();
                break;
        case 3:
```

```
                        depositoConta();
                        break;
                case 4:
                        exit(1);
                case 5:
                        saqueComErro();
                        break;
                default:
                        printf("\nArgumento inválido!!\n");
                        break;
                }


                        break;
        default:
                printf("\nArgumento inválido!!\n");
                break;
        }
        }
}


#ifndef DEBUG
        clnt_destroy(clnt);
#endif /* DEBUG */
}


int main(int argc, char *argv[])
{
        char *host;
        int opcao = 0;


        if (argc < 2)
        {
                printf("usage: %s server_host\n", argv[0]);
                exit(1);
        }
        host = argv[1];


        for (int i = 0; i < NUM_CONTA; i++)
        {
                tokenContas[i].token = 0;
        }
```

```
        printf("\nEscolha:\n1 - Agência\t\t 2- Caixa Eletrônico\n");
        fflush(stdin);
        scanf("%d", &opcao);


        banco_prog_1(host, opcao);
        exit(0);
}
```

```c
/*
        BANCO_SERVER.C
         * These are only templates and you can use them
         * as a guideline for developing your own functions.
         */


        #include "banco.h"
        #include <stdio.h>


        #define NUM_CONTA 100


        conta contas[NUM_CONTA];
        token tokenContas[NUM_CONTA];


        int erroglobal = 0;
        int valorglobal = 0;
        int idglobal = 0;


        int verificaContaBD(int _id)
        {
                for (int i = 0; i < NUM_CONTA; i++)
                {
                        if (contas[i].id == _id)
                        {
                                return _id;
                        }
                }


                return -1;
        }


        int *abreconta_1_svc(int *argp, struct svc_req *rqstp)
        {
                static int result;
                int id = *argp;


                if (verificaContaBD(id) == -1)
                {
```

```
                contas[id].id = id;
                contas[id].saldo = 0.0;
                result = id;
        }
        else
        {

                result = -1;
        }

        return &result;
}

int *fechaconta_1_svc(int *argp, struct svc_req *rqstp)
{
        static int result;
        int id = *argp;

        if (verificaContaBD(id) != -1)
        {
                contas[id].id = 999;
                contas[id].saldo = 999;
                result = 0;
        }
        else
        {
                result = -1;
        }

        return &result;
}

int *authconta_1_svc(int *argp, struct svc_req *rqstp)
{
        static int result;
        int id = *argp;

        result = verificaContaBD(id);
```

```
        return &result;
}


int *deposito_1_svc(transacao *argp, struct svc_req *rqstp)
{
        static int result;
        int id = argp->id;
        float valor = argp->saldo;


        result = -1;
        if (verificaContaBD(id) != -1)
        {
                contas[id].saldo = contas[id].saldo + valor;
                result = 0;
        }


        return &result;
}


int *saque_1_svc(transacao *argp, struct svc_req *rqstp)
{
        static int result;
        int id = argp->id;
        int valor = argp->saldo;
        int erro = argp->erro;


        result = -1;
        if (verificaContaBD(id) != -1)
        {
                if(erro == 1 && erroglobal == 1 && valorglobal != valor) erroglobal =
0;


                if(erro == 1 && erroglobal == 0)
                {
                        contas[id].saldo = contas[id].saldo - valor;
                        result = 2;
                        erroglobal = 1;
                        idglobal = id;
                        valorglobal = valor;
```

```
                }
                else if(erro == 1 && erroglobal == 1 && idglobal == id && valorglobal
== valor)
                {
                        result = 3;
                        erroglobal = 0;
                }
                else
                {
                        contas[id].saldo = contas[id].saldo - valor;
                        result = 0;
                }


        }


        return &result;
}


float *
retornasaldo_1_svc(int *argp, struct svc_req *rqstp)
{
        static float result;
        int id = *argp;


        result = -1.0;
        if (verificaContaBD(id) != -1)
        {
                result = contas[id].saldo;
        }


        return &result;
}


int *checksenha_1_svc(token *argp, struct svc_req *rqstp)
{
        static int result;
        int id = argp->conta_id;
        int token = argp->token;
```

```
        if (token == 0) //primeira senha
        {
                tokenContas[id].conta_id = id;
                tokenContas[id].token = 0;
                result = 0;
        }
        else if (tokenContas[id].token == token) //senha correta
        {
                result = 0;
        }
        else //senha sem atualizacoes
        {
                result = -1;
        }


        return &result;
}


int *gerasenha_1_svc(token *argp, struct svc_req *rqstp)
{
        static int result;
        int contaId = argp->conta_id;


        if (verificaContaBD(contaId) != -1)
        {
                tokenContas[contaId].token = tokenContas[contaId].token + 2;
                result = tokenContas[contaId].token;
        }
        else
        {
                result = -1;
        }
        return &result;
}
```