UNIVERSIDADE
DE TRÁS-OS-MONTES
E ALTO DOURO

TRABALHO PRÁTICO 1

Servidor de Chaves do Euromilhões

Universidade de Trás-os-Montes e Alto-Douro

Licenciatura em Engenharia Informática

Sistemas Distribuídos

Docentes:

Hugo Paredes

Arsénio Reis

Dennis Paulino

Discentes:

Ana Dias al69691

Diana Alves al68557

Diana Ferreira al68938

1. Desenvolvimento

1.1. Protocolo

Lista de comandos:

- "CHAVE / Chave / chave + numero de chaves a criar" – Gera o número pedido de chaves do Euromilhões;
- "QUIT/ Quit / quit" – Encerra a comunicação com o servidor.

Nota: Foi implementado de forma que aceite quer letras maiúsculas quer letras minúsculas.

Mensagens/Respostas possíveis:

- "100 OK" – Conectado
- "200 Sent" – Recebido com sucesso
- "400 Bye" – Conexão terminada
- "404 NaoEncontrado" – Número de chaves inválido
- "410 Perdido" - Está em falta o número de chaves para serem criadas
- "450 MalDirecionado" – Comando desconhecido

Estados:

- A aguardar pedido
- À espera de conexões
- Conectado
- Conexão falhada
- Desconectado
- Erro no pedido

1.2. Implementação

Atendimento dos clientes:

Inicialmente foram declaradas e inicializadas as variáveis necessárias para esta implementação, como *SOCKET clientSocket* e *SOCKET* aptclientSocket*, sendo esta última um dos apoios para efetuar o atendimento simultâneo de vários clientes. Isto foi conseguido através do uso de *THREADS* e *MUTEXES* ("*one thread of execution never enters a critical section while a concurrent thread of execution is already accessing critical section*", WIKIPÉDIA, https://en.wikipedia.org/wiki/Mutual_exclusion).

```
//Criar Socket
SOCKET listening = socket(AF_INET, SOCK_STREAM, 0);
if (listening == INVALID_SOCKET) {
    fprintf(stderr, "\n A criacao do Socket falhou! Codigo Erro : %d\n", WSAGetLastError());
    return 1;
}

printf("\nSocket criado.");

//Ligar o Socket (IP Address e Porta)
struct sockaddr_in hint;
hint.sin_family = AF_INET;
hint.sin_port = htons(PORTA); //Porta
hint.sin_addr.S_un.S_addr = INADDR_ANY;

bind(listening, (struct sockaddr*)&hint, sizeof(hint));

//Configurar o socket para "ouvir": para ficar à espera que mais clientes se conectem
listen(listening, SOMAXCONN);

//Espera para se conectar
struct sockaddr_in client;
int clientSize;

//Declaração de variáveis
SOCKET clientSocket /*= accept(listening, (struct sockaddr*)&client, &clientSize)*/;
SOCKET* aptclientSocket;
HANDLE  hdlThread;
DWORD dwThread;
int conresult = 0;
```

Figura 1 - Código relativo ao atendimento dos clientes.

Comunicação com cada Cliente:

Inicialmente foram declaradas e inicializadas as variáveis necessárias para esta implementação, como *HANDLE hdlThread* e *DWORD dwThread*. Para que exista ligação entre servidor-cliente, o cliente necessita de inserir o IP do servidor que se quer conectar. Em seguida, através da função pré-definida "*hdlThread = CreateThread(NULL, 0, TratarConexao, aptclientSocket, 0, &dwThread*)" será criada uma nova thread que chama a função "*TratarConexao*" onde serão criados sockets para que cada cliente se conecte com o ID da thread (para distinguir os vários clientes) passado como parâmetro nessa mesma função. Ainda nesta função é inicializado o *dwWaiResult* para o uso de *MUTEXES*.

```
HANDLE  hdlThread;
DWORD dwThread;
```

Figura 2 - Declaração de variáveis.

```
DWORD dwCount = 0, dwWaitResult;

//Criação de Socket
SOCKET cs;
SOCKET* ptCs;

ptCs = (SOCKET*)lpParam;
cs = *ptCs;
```

Figura 3 - Atribuir ID da thread e do socket.

Garantia de sugestão de chaves únicas:

A função "`CriarChave`" irá gerar uma combinação de cinco números e duas estrelas aleatoriamente e sem repetição de números dentro da mesma chave (caso saia o número "1", na próxima posição da chave este não pode voltar a sair nos cinco números; relativamente às estrelas, estas entre si não podem ser repetidas, mas comparadas com os números pode haver repetição). De seguida, esta função chama outra, "`ComparaChaves`", que recorre a um ficheiro que contém todas as chaves geradas. Esta função compara a chave gerada com as chaves existentes no ficheiro. No caso de já existir uma chave repetida, pede ao servidor para criar outra chave através da função "`CriarChave`". Caso contrário, este recorre à função "`GuardarChaveCriada`" para guardar a nova chave no ficheiro.

```
//Função que compara a chaves geradas pelo servidor com as existentes no ficheiro
int ComparaChaves(char* chave)
{
    FILE* ficheiro;
    char c;  //Lê o caracter do ficheiro
    int a, i = 0, linha = 0;
    char stringFile[50];

    ficheiro = fopen("Chaves.txt", "r");

    //Caso o ficheiro nao exista, este é criado
    if (ficheiro == NULL) {
        ficheiro = fopen("Chaves.txt", "a");
    }

    //Lê caracter a caracter enquanto não chega ao EOF
    for(c = getc(ficheiro); c != EOF; c = getc(ficheiro))
    {
        if(c == '\n')  //Passa para a linha seguinte no caso do caracter ser "\n"
        {
            stringFile[i++] = '\0';
            linha = linha + 1;
            i = 0;
        }
        else { stringFile[i++] = c; }

        a = strcmp(chave, stringFile);
        if(a == 0)
        {
            //Existe chave igual
            fclose(ficheiro);
            return 0;
        }
    }
    //Não existe chave igual
    fclose(ficheiro);
    return 1;
}
```

Figura 4 - Código relativo à sugestão de chaves únicas.

Atendimento simultâneo de múltiplos clientes:

Para que possam ser atendidos vários clientes em simultâneo, é necessário criar *THREADS* para cada cliente que se conecte ao servidor. Sem este passo, o servidor apenas conseguiria atender um cliente de cada vez. Neste sentido, as *THREADS* permitem a subdivisão de tarefas, isto é, "*a thread of execution is the smallest sequence of programmed instructions that can be managed independently by a scheduler*", WIKIPÉDIA, https://en.wikipedia.org/wiki/Thread_(computing). Tendo em conta que cada cliente tem acesso a uma única thread, este não se apercebe se é o único ou não com ligação estabelecida ao servidor.

Em relação à utilização de *MUTEXES*, como é necessário o recurso a um ficheiro (leitura e escrita dentro do mesmo), é essencial garantir que não existe o acesso simultâneo ao mesmo. Assim, caso um cliente esteja a recorrer ao ficheiro, outro cliente será colocado em espera, e só poderá entrar nessa secção quando o primeiro cliente terminar.

```
dwWaitResult = WaitForSingleObject( ghMutex, INFINITE);
switch (dwWaitResult)
{
case WAIT_OBJECT_0:
    _try{
        //Função que repete n vezes as chaves que foram pedidas
        for (int k = 0; k < numeroChaves; k++)
        {
            CriarChave(chave);
            chaveString = intTochar(chave);
            strcat_s(msg, 8192, chaveString);
            strcat_s(msg, 8192, "\r\n");

        }
    }
    _finally{
        if (!ReleaseMutex(ghMutex))
        {
                printf("Erro");
                return 0;
        }
    }
    break;
case WAIT_ABANDONED:
    return FALSE;
}
```

Figura 5 - Acesso em simultâneo com recurso a Mutexes

Nota: Todas as Figuras foram retiradas do código que se encontra em anexo (Anexo A – Server e Anexo B – Cliente).

2. Anexo A – Server

```
#include <stdio.h>
#include <stdlib.h>
#include <winsock2.h>
#include <time.h>
#include <Windows.h>
#include <cstdint>
#include <ctype.h>
#include <iostream>

#define TRUE 1
#define PORTA 68000 //porta para a conexão ser possível
#define CHAVE 7   // 5 numeros + 2 estrelas

#pragma comment (lib, "ws2_32.lib")
#pragma warning(disable : 4996)

//THREADS: sem esta linha o código continuaria a funcionar, no entanto só se podia conectar 1
cliente de cada vez ao servidor
//Esta função serve para tratar das THREADS
DWORD WINAPI TratarConexao(LPVOID lpParam);
HANDLE ghMutex;

//Função que compara a chaves geradas pelo servidor com as existentes no ficheiro
int ComparaChaves(char* chave);
//Função que conta as linhas do ficheiro
int ContarLinhas();
//Função que cria as chaves do euromilhoes aleatóriamente
void CriarChave(int chave[]);
//Função que vai guardar a chave gerada se esta ainda não existir no ficheiro
void GuardarChaveCriada(int chave[], int numElementos);
//Função que irá ordenar os 5 números da chave
int OrdenarChave(int vetor[], int numElementos);
//Função que converte inteiros para caracteres
char* intTochar(int chave[]);

//programa principal
int main()
{
	char mensagem[2000] = "";

	strcpy_s(mensagem, 2000, "");
	srand(time(0));

	//Inicializar Winsock
	WSADATA wsData;
	WORD ver = MAKEWORD(2, 2);

	ghMutex = CreateMutex( NULL, FALSE,	NULL);

	if (ghMutex == NULL) {
		printf("\n A configuração do Mutex falhou! Codigo de Erro : %d\n", GetLastError());
		return 1;
	}

	printf("\nInicializar Winsock...");
	int wsResult = WSAStartup(ver, &wsData);
	if (wsResult != 0) {
		fprintf(stderr, "\n A configuração do Winsock falhou! Codigo de Erro : %d\n",
WSAGetLastError());
		return 1;
	}
```

```
	//Criar Socket
	SOCKET listening = socket(AF_INET, SOCK_STREAM, 0);
	if (listening == INVALID_SOCKET) {
		fprintf(stderr, "\n A criacao do Socket falhou! Codigo Erro : %d\n",
WSAGetLastError());
		return 1;
	}

	printf("\nSocket criado.");

	//Ligar o Socket (IP Address e Porta)
	struct sockaddr_in hint;
	hint.sin_family = AF_INET;
	hint.sin_port = htons(PORTA); //Porta
	hint.sin_addr.S_un.S_addr = INADDR_ANY;

	bind(listening, (struct sockaddr*)&hint, sizeof(hint));

	//Configurar o socket para "ouvir": para ficar à espera que mais clientes se conectem
	listen(listening, SOMAXCONN);

	//Espera para se conectar
	struct sockaddr_in client;
	int clientSize;

	//Declaração de variáveis
	SOCKET clientSocket /*= accept(listening, (struct sockaddr*)&client, &clientSize)*/;
	SOCKET* aptclientSocket;
	HANDLE  hdlThread;
	DWORD dwThread;

	while (TRUE)
	{
		clientSize = sizeof(client);
		clientSocket = accept(listening, (struct sockaddr*)&client, &clientSize);

		aptclientSocket = &clientSocket;

		printf("\nA tratar da nova conexao.");

		//Tratamento da comunicação com o cliente
		//Na função "CreateThread" iram ser especificados os seus atributos.
		//1ºParametro: atributo de segurança por defeito = NULL (não é utilizado nenhum nível
de segurança)
		//2ºParametro: o tamanho da stack utilizada é o defaut = 0
		//3ºParametro: nome da função que trata das conexões = TratarConexao
		//4ºParametro: apontador para o socket a ser tratada = aptclientSocket
		//5ºParametro: valor de flags por defeito = 0
		//6ºParametro: retorna o ID de cada thread criada
		hdlThread = CreateThread(NULL, 0, TratarConexao, aptclientSocket, 0, &dwThread);

		//Caso a criação da thread falhe, a execução termina e aparece uma mensagem de erro
		if (hdlThread == NULL)
		{
			printf("\nERRO: Criação da Thread falhou.");
			ExitProcess(3);
		}
	}

	//Fechar o Socket
	closesocket(clientSocket);
	//Fechar o "canal" de espera para a conexão de novos clientes
	closesocket(listening);
```

```
	//Fecha o mutex
	CloseHandle(ghMutex);
	//Limpar Winsock
	WSACleanup();

	return 1;

}

//Função que trata das conexões dos clientes ao mesmo servidor
DWORD WINAPI TratarConexao(LPVOID lpParam)
{
	//Declaração de variáveis
	const char* chaveString;
	char* pMensagem;
	int numeroChaves = 0;
	int chave[CHAVE] = { 0,0,0,0,0,0,0 };
	srand(time(0));

	char msg[8192]; //Armazena a mensagem que o servidor envia
	char rec[8192]; //Armazena a mensagem que o servidor recebe

	DWORD dwCount = 0, dwWaitResult;

	//Criação de Socket
	SOCKET cs;
	SOCKET* ptCs;

	ptCs = (SOCKET*)lpParam;
	cs = *ptCs;

	char* foo(int count);
	strcpy(msg, "100 OK");
	send(cs, msg, strlen(msg) + 1, 0);
	printf("\nResposta:%s\n", msg);

	while (TRUE) //Ciclo infinito
	{
		dwCount = 0;
		//Apaga as mensagens recebidas para o servidor e envidas do servidor, para que depois
possa receber novas mensagens
		ZeroMemory(rec, 8192);
		ZeroMemory(msg, 8192);

		//converte a string(mensagem) recebida do cliente em bytes
		int bytesRecebidos = recv(cs, rec, 8192, 0);
		switch (bytesRecebidos)
		{
		case SOCKET_ERROR:
			printf("\n Erro ao receber!\n"); //caso dê erro ao receber a mensagem do
cliente
			break;
		case 0:
			printf("\n Cliente desconectado!\n"); //caso o cliente se desconecte do
servidor
			break;
		default:
			printf("\n Cliente: %s\n", rec);  // apresenta o comando que o cliente digitou
		}

		if (strcmp(rec, "\r\n") != 0)
		{
			//Função que transforma as letras todas em minúsculas
			for (int i = 0; rec[i]; i++) {
				rec[i] = tolower(rec[i]);
```

```
		}

		if (strcmp(rec, "quit") == 0)
		{
			strcpy(msg, "400 Bye");
			send(cs, msg, strlen(msg) + 1, 0);
			printf("\nResposta:%s\n", msg);
			closesocket(cs);
			return 0;
		}

		pMensagem = strtok(rec, " ");

		if (strcmp(pMensagem, "chave") == 0)
		{
			pMensagem = strtok(NULL, " ");
			if (pMensagem != NULL) {
				numeroChaves = atoi(pMensagem);
				if (numeroChaves) {
					strcpy(msg, "200 Sent");
					send(cs, msg, strlen(msg) + 1, 0);
					ZeroMemory(msg, 8192);

					dwWaitResult = WaitForSingleObject( ghMutex, INFINITE);
					switch (dwWaitResult)
					{
					case WAIT_OBJECT_0:
						_try{
							//Função que repete n vezes as chaves que
foram pedidas
							for (int k = 0; k < numeroChaves; k++)
							{
								CriarChave(chave);
								chaveString = intTochar(chave);
								strcat_s(msg, 8192, chaveString);
								strcat_s(msg, 8192, "\r\n");

							}
						}
						_finally{
							if (!ReleaseMutex(ghMutex))
							{
									printf("Erro");
									return 0;
							}
						}
						break;
					case WAIT_ABANDONED:
						return FALSE;
					}

					//Devolve a data e hora em que foi devolvida a resposta
					time_t now = time(0);
					char* dt = ctime(&now);
					strcat_s(msg, 8192, "Enviado a ");
					strcat_s(msg, 8192, dt);

					int linhas = ContarLinhas();

					strcat_s(msg, 8192, "Criadas ");
					sprintf_s(&msg[strlen(msg)], sizeof(int), "%d", linhas);
					strcat_s(msg, 8192, " chaves no total.\n");

				}
```

```
                                        else { strcpy(msg, "404 NaoEncontrado"); }
                                }
                                else { strcpy(msg, "410 Perdido"); }
                        }
                        else { strcpy(msg, "450 MalDirecionado"); }
                        send(cs, msg, strlen(msg) + 1, 0);
                        printf("Resposta: %s\n", msg);
                }
        }
        return 1;
}


//Função que compara a chaves geradas pelo servidor com as existentes no ficheiro
int ComparaChaves(char* chave)
{
        FILE* ficheiro;
        char c;  //Lê o caracter do ficheiro
        int a, i = 0, linha = 0;
        char stringFile[50];

        ficheiro = fopen("Chaves.txt", "r");

        //Caso o ficheiro nao exista, este é criado
        if (ficheiro == NULL) {
                ficheiro = fopen("Chaves.txt", "a");
        }

        //Lê caracter a caracter enquanto não chega ao EOF
        for(c = getc(ficheiro); c != EOF; c = getc(ficheiro))
        {
                if(c == '\n')  //Passa para a linha seguinte no caso do caracter ser "\n"
                {
                        stringFile[i++] = '\0';
                        linha = linha + 1;
                        i = 0;
                }
                else { stringFile[i++] = c; }

                a = strcmp(chave, stringFile);
                if(a == 0)
                {
                        //Existe chave igual
                        fclose(ficheiro);
                        return 0;
                }
        }
        //Não existe chave igual
        fclose(ficheiro);
        return 1;
}

//Função que contas as linhas do ficheiro
int ContarLinhas() {

        FILE* ficheiro;
        char c, letra = '\n';
        int numlinhas = 0;

        ficheiro = fopen("Chaves.txt", "r");

        if(ficheiro == NULL)
        {
                printf("Erro na abertura ou na leitura do ficheiro.");
```

```
			return -1;
		}

		//Ciclo que conta as linhas do ficheiro
		for (c = getc(ficheiro); c != EOF; c = getc(ficheiro))
		{
			if(c == '\n') //Se o caracter for igual a "\n" existe mudança de linha, ou seja
incrementa o numeros de linhas
			{
				numlinhas = numlinhas + 1;
			}
		}

		fclose(ficheiro);
		return numlinhas; //Retorna o numero de linhas do ficheiro
}


//Função que cria as chaves do euromilhoes aleatóriamente
void CriarChave(int chave[])
{
		//Gera os 5 numeros aleatoriamente
		for(int a = 0; a < 5; a++)
		{
			chave[a] = rand() % 50 + 1; //rand()%(maior-menor+1) + menor; -> gera numeros
aleatorios entre 1 a 50
			for(int b = 0; b < a; b++)
			{
				if (chave[a] == chave[b]) //Não pode gerar numeros repetidos
				{
					a--;
				}
			}
		}
		//Ordena os 5 números criados anteriormente
		OrdenarChave(chave, 5);

		//Gera as 2 estrelas aleatoriamente
		int estrelas[2];
		for(int c = 0; c < 2; c++)
		{
			estrelas[c] = rand() % 12 + 1; //rand()%(maior-menor+1) + menor; -> gera numeros
aleatorios entre 1 a 12
			if(c != 0)
			{
				if(estrelas[c] == estrelas[c - 1]) //Não pode gerar numeros repetidos
				{
					c--;
				}
			}
		}
		//Faz a ordenação das estrelas por ordem crescente
		if(estrelas[0] < estrelas[1])
		{
			chave[5] = estrelas[0];
			chave[6] = estrelas[1];
		}
		else
		{
			chave[5] = estrelas[1];
			chave[6] = estrelas[0];
		}

		//Converter a chave de inteiros para caracteres
```

```
	char* chaveCompleta = intTochar(chave);
	//Vai comparar a chave (5 numeros + 2 estrelas) gerada anteriormente com as existentes no ficheiro
	int a = ComparaChaves(chaveCompleta);
	if (a == 0) //0 - existe igual
	{
		//printf("A chave gerada ja existe!\n");
		CriarChave(chave);  //A chave é única por isso se já existir uma igual deve gerar uma nova
	}
	else  //1 - não existe igual
	{
		//printf("A chave gerada ainda nao existe! Vai ser guardada no ficheiro.\n");
		GuardarChaveCriada(chave, 7);  //Como a chave ainda não existe é guardada no ficheiro
	}

}

//Função que vai guardar a chave recebida como parâmetro no ficheiro
void GuardarChaveCriada(int chave[], int numElementos)
{
	//Guarda chave gerada anteriormente (5 numeros + 2 estrelas) no ficheiro
	FILE* ficheiro = fopen("Chaves.txt", "a+");
	for(int d = 0; d < 7; d++)
	{
		if(d == 6)
		{
			fprintf(ficheiro, "%d\n", chave[d]);
		}
		else
		{
			fprintf(ficheiro, "%d ", chave[d]);
		}
	}
	fclose(ficheiro);
}

//Função que irá ordenar os 5 números da chave por ordem crescente (método insertion sort)
int OrdenarChave(int vetor[], int numElementos)
{
	int i, key, j;
	int a;
	for (i = 1; i < numElementos; i++)
	{
		key = vetor[i];
		j = i - 1;
		while (j >= 0 && vetor[j] > key)
		{
			vetor[j + 1] = vetor[j];
			j = j - 1;
		}
		vetor[j + 1] = key;
	}
	return vetor[numElementos];  //Retorna o vetor ordenado

}


//Função que converte inteiros para caracteres
char* intTochar(int chave[])
{

	char* str = (char*)malloc(2000);
	if(str == NULL)
	{
```

```
			return NULL;
		}
		strcpy_s(str, 2000, "");

		//Escrever array com a chave numa string
		for(int i = 0; i < CHAVE; i++)
		{
			if(chave[i] < 10)
			{
				sprintf_s(&str[strlen(str)], (sizeof(int) + (3 * sizeof(char))), "%d   ",
chave[i]);
			}
			else
			{
				sprintf_s(&str[strlen(str)], (sizeof(int) + (3 * sizeof(char))), "%d  ",
chave[i]);
			}
			if(i == 4)
			{
				strcat_s(str, sizeof(char) * (strlen(str) + 7), " +   ");
			}
		}
		return str;
}
```

3. Anexo B - Cliente

```
#include<stdio.h>
#include<winsock2.h>
#include<string.h>

#pragma comment(lib,"ws2_32.lib")
#pragma warning(disable : 4996)

int main(int argc, char* argv[])
{
	//Declaração de variáveis
	WSADATA wsa;
	SOCKET s;
	struct sockaddr_in server;
	char* mensagem = (char*)malloc(8192);
	char respostaServidor[8192];
	bool conectado = false;
	char* ip = (char*)malloc(20); //memória alocada para o IP
	int recv_size;
	int ws_resultado = -1; //por defeito não está conectado inicialmente

	//Inicializar Winsock
	printf("\nInicializar Socket...");
	if(WSAStartup(MAKEWORD(2, 2), &wsa) != 0)
	{
		printf("ERRO! Código de Erro: %d", WSAGetLastError());
		return 1;
	}
	printf("Socket Inicializado.\n");

	//Criar o Socket
	s = socket(AF_INET, SOCK_STREAM, 0);
	if(s == INVALID_SOCKET)
	{
		printf("Nao foi possivel criar o Socket : %d", WSAGetLastError());
	}
	printf("Socket criado.\n");

	while(conectado == false) //Enquanto não estiver conectado, tenta conectar-se
	{
		while(ws_resultado < 0) //Enquanto o IP inserido não for válido
		{
			printf("Insira o IP (servidor) para se conectar:");
			fgets(ip, 20, stdin);

			//Criar o endereço do Socket(IP address e porta)
			server.sin_addr.s_addr = inet_addr(ip);
			server.sin_family = AF_INET;
			server.sin_port = htons(68000); //porta
			//Conectar a um server remoto
			ws_resultado = connect(s, (struct sockaddr*)&server, sizeof(server));

			if(ws_resultado < 0)
			{
				printf("IP invalido! Tente novamente!!\n");
			}
		}

		//Recebe a resposta do servidor
		recv_size = recv(s, respostaServidor, 8192, 0);
		if(recv_size == SOCKET_ERROR) //Se der erro na ligação ao servidor
		{
			puts("Falha ao receber.\n");
```

```
		}
		if(strcmp(respostaServidor, "100 OK") == 0) //Se não der erro apresenta os comandos possíveis
		{
			printf("Conectado\n");
			conectado = true;
			printf("\n\tComandos Possiveis:\n");
			printf("\tPARA DETERMINAR AS CHAVES A CRIAR: CHAVE / Chave / chave + numero de chaves a criar \n");
			printf("\tPARA TERMINAR A CONEXAO: QUIT/ Quit / quit \n");
		}
	}

	while(1)//Ciclo infinito
	{
		//Apagar as mensagens vindas tanto do servidor como do cliente, para que possa receber novas mensagens
		ZeroMemory(mensagem, 8192);
		ZeroMemory(respostaServidor, 8192);

		fputs("\n\nEscreva o comando que deseja: ", stdout);
		fgets(mensagem, 8192, stdin);

		//Caso dê erro a enviar mensagens ao Servidor
		ws_resultado = send(s, mensagem, strlen(mensagem) - 1, 0);  // -1 para nao enviar "\n"
		if(ws_resultado < 0)
		{
			puts("Falha no envio.");
			return 1;
		}

		//Caso dê erro ao receber a mensagem do Servirdor
		recv_size = recv(s, respostaServidor, 8192, 0);
		if(recv_size == SOCKET_ERROR)
		{
			puts("Falha ao receber.");
		}

		//Vai buscar apenas o numero do codigo de erro à resposta do servidor
		char* cod_erro = strtok(respostaServidor, " ");
		int tent = atoi(cod_erro);
		//Compara o numero do erro com os numeros possíveis
		switch (tent)
		{
			case 200: // 200 Sent
				ZeroMemory(respostaServidor, 8192);
				recv_size = recv(s, respostaServidor, 8192, 0);
				respostaServidor[recv_size] = '\0';
				printf("\nServidor: \n%s\n", respostaServidor);
			break;
			case 450: // 450 MalDirecionado
				printf("Servidor Erro: Comando desconhecido.\n");
			break;
			case 410: // 410 Perdido
				printf("Servidor Erro: Esta em falta o numero de chave para serem criadas.\n");
			break;
			case 404: // 404 NaoEncontrado
				printf("Servidor Erro: Numero de chaves invalido.\n");
			break;
			case 400: // 400 Bye
				printf("Servidor: Conexao terminada.\n");
				system("pause");
				exit(-1);
```

```
                    break;
                }
        }

        //Fechar o socket
        closesocket(s);
        //Limpar Winsock
        WSACleanup();

        system("pause");
        return 0;
}
```