```
/*_____________________________________conversaGrupoVariavelTCP.c_____________
_____________________*/
/*============================ Servidor concorrente TCP
========================
Este servidor destina-se a colocar em contacto grupos de clientes.
Forma grupos de forma sucessiva.
Um grupo fica formado quando o limite MAX_TAM_GRUPO e' atingido ou
se agurada mais de LIMIT_ESPERA segundo.
Em cada grupo, sempre que um byte e' recebido num dos sockets,
reencaminha-o para os restantes.
Ao contrario de conversaGrupoTCP.C, este codigo suporta baixas na
constituicao dos grupos: um grupo apenas deixa de existir quando
passam a existir menos de dois sockets operacionais.
================================================================================
==========
*/

#include <stdio.h>
#include <stdlib.h>
#include <winsock.h>

#define TIMEOUT        3   //segundos
#define BUFFERSIZE     4096
#define MAX_TAM_GRUPO 5
#define LIMITE_ESPERA 30 //segundos
#define MSG_BOAS_VINDAS "Servidor conversa em grupo\r\nAguarde...\r\n"
#define MSG_ARRANQUE_CONVERSA "Pode iniciar conversa...\r\n"

void fechaSockets(SOCKET *grupo);
void AtendeCliente(LPVOID param);
int difunde(int indexOrigem, SOCKET *grupo);
void Abort(char *msg, SOCKET s);

//_____________________________________________________________________
//
//  Saber usar gethostname e getpeername IMPORTANTE! nao esta nestes
exercicios.
//_____________________________________________________________________


/*_____________________________________ main
_____________________________________________
*/
int main(int argc,char *argv[]){

      SOCKET sock = INVALID_SOCKET, newSock = INVALID_SOCKET;
      int iResult;
      int cliaddr_len;
      struct sockaddr_in cli_addr, serv_addr;
      WSADATA wsaData;
      SECURITY_ATTRIBUTES sa;
      DWORD thread_id;
      SOCKET grupoSockets[MAX_TAM_GRUPO];
      int contador, tam, i;
```

```
	SOCKET *parametrosThreadAtendeCliente;
	int ocorrenciaTimeout;
	fd_set fd_accept;
	struct timeval tempoEspera;

	if(argc!=2){
		fprintf(stderr, "Usage: %s <porto de escuta>\n",argv[0]);
		exit(EXIT_SUCCESS);
	}

	/*=============== INICIA OS WINSOCKS ==============*/
	iResult = WSAStartup(MAKEWORD(2,2), &wsaData);
	if (iResult != 0) {
		printf("WSAStartup failed: %d\n", iResult);
		getchar();
		exit(1);
	}

	/*================== ABRE SOCKET PARA ESCUTA DE CLIENTES
================*/
	if((/*ABRE = SOCKET*/) == INVALID_SOCKET)
		Abort("Impossibilidade de abrir socket", sock);

	/*=================== PREENCHE ENDERECO DE ESCUTA
=======================*/
	memset((char*)&serv_addr, 0, sizeof(serv_addr));
	serv_addr.sin_family=AF_INET;
	serv_addr.sin_addr.s_addr=/**/;  /*Recebe de qq interface*/
	serv_addr.sin_port=/**/;  /*Escuta no porto Well-Known*/

	/*====================== REGISTA-SE PARA ESCUTA
=========================*/
	if(/*REGISTA SOCKET*/) == SOCKET_ERROR)
		Abort("Impossibilidade de registar-se para escuta", sock);

	/*============ AVISA QUE ESTA PRONTO A ACEITAR PEDIDOS
==================*/
	if(/*ESCUTA*/) == SOCKET_ERROR)
		Abort("Impossibilidade de escutar pedidos", sock);

	printf("<SER> Servidor conversa em grupo pronto no porto de escuta:
%s\n", argv[1]);


	/*========== PASSA A ATENDER CLIENTES DE FORMA CONCORRENTE
=============*/
	/*...*/; //sizeof (porque?para que?)
	contador = 0;

	for(i=0; i</**/; i++){
		/*COLOCA OS SOCKETS CRIADOS COMO INVALIDOS/VAZIOS/-1*/
	}

	while(1){
```

```
            /* Caso o grupo ja' tenha pelo menos 2 elementos,
             * aguarda LIMITE_ESPERA ate' dar a constituicao do grupo
             * por concluida. Caso contrario, aguarda um novo pedido
             * de ligacao sem timeout.
             * O timeout e' realizado 'a custa da funcao "select()".
             */
            ocorrenciaTimeout=0; //false

            if(contador >= 2){

                  FD_ZERO(/**/);
                  /**/

                  tempoEspera.tv_sec = LIMITE_ESPERA;
                  tempoEspera.tv_usec = 0;

                  switch(select(/**/)){
                        case SOCKET_ERROR:
                              fprintf(stderr,"<SERV> Erro ao invocar
\"select()\" para efeitos de timeout de ligacao (error: %d)!\n");
                              continue;
                        case 0:
                              ocorrenciaTimeout = 1;
                              break;
                        default:
                              break;
                  }

            }

            if(!ocorrenciaTimeout){
                  /*====================== ATENDE PEDIDO
========================*/
                  if(/*.........*/ == SOCKET_ERROR){

                        if(WSAGetLastError() == WSAEINTR)
                              continue;

                        fprintf(stderr,"<SERV> Impossibilidade de aceitar
cliente...\n");
                        continue;
                  }

                  printf("<SER> Novo cliente conectado: <%s:%d>.\n",
/*....*/(cli_addr.sin_addr),/**/(cli_addr.sin_port));
                  send(newSock, MSG_BOAS_VINDAS, strlen(MSG_BOAS_VINDAS),
0);
                  grupoSockets[contador++] = newSock;

            }

            if(contador ==/**/ || (ocorrenciaTimeout && contador>=2)){

                  tam = contador;
```

```
                parametrosThreadAtendeCliente = /**/;

                if(parametrosThreadAtendeCliente == NULL){

                        printf("<SER> Nao foi possivel reservar espaco
para passar parametros\n!");
                        fechaSockets(grupoSockets);

                }else{

                        for(i=0; i<tam; i++){
                              /*copia para struct*/ = /**/;
                        }

                        for (i =/**/; i</*COLOCA OS VAZIOS A INVALID (na
struct, no array ja estavam)*/; i++){
                              parametrosThreadAtendeCliente[i] =
INVALID_SOCKET;
                        }

                        sa.nLength=sizeof(sa);
                        sa.lpSecurityDescriptor=NULL;

                        if(CreateThread(&sa,0
,(LPTHREAD_START_ROUTINE)AtendeCliente,
(LPVOID)parametrosThreadAtendeCliente, (DWORD)0, &thread_id)==NULL){
                              printf("<SER> Nao foi possivel iniciar uma
nova thread (error: %d)!\n", GetLastError());
                              printf("<SER> O grupo actual nao sera'
atendido!\n");
                              fechaSockets(grupoSockets);
                        }

                        printf("<SER> Um novo grupo acaba de ser formado
com %d elementos.\n", tam);

                }

                //Prepara constituicao do proximo grupo
                contador = 0;

                for(i=0; i<MAX_TAM_GRUPO; i++){
                        grupoSockets[i] = INVALID_SOCKET;
                }
            }
      }
}

/*_______________________________ fechaSockets
_____________________________________
_____________________________________________________________________________
_____*/
void fechaSockets(SOCKET *grupo)
```

```
{
      int i;

      for(i=0; i<MAX_TAM_GRUPO; i++){
            if(grupo[i]!=INVALID_SOCKET){
                  closesocket(grupo[i]);
            }
      }
}

/*_____________________________ AtendeCliente
_____________________________________
Atende cliente.
_____________________________________________________________________________
_____*/

void AtendeCliente(LPVOID param){
      fd_set fdread;
      SOCKET *grupo;
      int i, grupoVazio;

      /**/ = (/*..de que tipo..*/)/**/;

      for(grupoVazio = 1, i=0; i<MAX_TAM_GRUPO; i++){
            if(grupo[i] != INVALID_SOCKET){
                  grupoVazio = 0; //false;
                  send(grupo[i], MSG_ARRANQUE_CONVERSA,
strlen(MSG_ARRANQUE_CONVERSA), 0);
            }
      }

      if(grupoVazio){
            free(grupo);
            return;
      }

      while(1){

            /*==================== PROCESSA PEDIDO
==========================*/

            /**/(&fdread);
            for(i=0; i<MAX_TAM_GRUPO; i++){
                  if(grupo[i] != INVALID_SOCKET){
                        /**/(grupo[i], &fdread);
                  }
            }

            switch(select(32, &fdread, NULL, NULL, NULL)){ // Sem timeout

                  case SOCKET_ERROR:
                        if(WSAGetLastError()==WSAEINTR)
                              break;
```

```
                        fprintf(stderr,"<SER_%d> Erro na rotina select
(%d) ...\n", GetCurrentThreadId(), WSAGetLastError());
                        fechaSockets(grupo);
                        free(grupo);
                        return;

                  default:
                        for(i=0; i<MAX_TAM_GRUPO; i++){

                              if(grupo[i] != INVALID_SOCKET){
                                    if (/**/(grupo[i], &fdread)){
                                          if(/**/ < 2){ //chama uma funcao
(qual?) e verifica se devolveu < 2
                                                fprintf(stderr,"<SER_%d>
Grupo encerrado ...\n", GetCurrentThreadId());
                                                fechaSockets(grupo);
                                                free(grupo);
                                                return;
                                          }
                                    }
                              }

                        }

                        break;
            } //switch
      } //while
}

/*_____________________________ difunde
_____________________________________
Recebe um caractere em grupo[indexOrigem] e reenvia-o para o grupo.
Não reenvia para grupo[indexOrigem].
Quando ocorre um problema com um dos elementos do grupo,
o elemento e' eliminado.

Devolve: numero de sockets activos / tamanho do grupo
_____________________________________________________________________
_____*/
int difunde(int indexOrigem, SOCKET *grupo)
{
      int result, i, socketsActivos;
      char c;

      if ((result = recv(/**/, &c, sizeof(char), 0)) == sizeof(char)){

            for(i=0; i<MAX_TAM_GRUPO; i++){

                  if(/**/ != INVALID_SOCKET && /**/ != i){

                        result = send(/**/, &c, sizeof(char), 0);

                        if(result == 0 || result == SOCKET_ERROR){
                              /**/(grupo[i]);
```

```
                              grupo[i] = INVALID_SOCKET;
                              fprintf(stderr,"<SER_%d> Menos um cliente
...\n", GetCurrentThreadId());
                        }

                  }

            }

      }else{
            /**/(grupo[indexOrigem]);
            grupo[indexOrigem] = INVALID_SOCKET;
            fprintf(stderr,"<SER_%d> Menos um cliente ...\n",
GetCurrentThreadId());
      }

      for(socketsActivos=0, i=0; i<MAX_TAM_GRUPO; i++){
            socketsActivos += (grupo[i] != INVALID_SOCKET);
      }

      return socketsActivos;
}

/*_____________________________________
Abort_____________________________________________
Mostra a mensagem de erro associada ao ultimo erro dos Winsock e abandona
com
"exit status" a 1
_____________________________________________________________________________
______
*/
void Abort(char *msg, SOCKET s)
{
      fprintf(stderr,"\a<SER_%d> Erro fatal: <%d>\n", WSAGetLastError(),
GetCurrentThreadId());

      if(s != INVALID_SOCKET)
            closesocket(s);

      exit(EXIT_FAILURE);
}
```