

```

/* _____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 LIMITE_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);

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

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

    /*===== PREENCHE ENDEREÇO DE ESCUTA =====*/
    memset((char*)&serv_addr, 0, sizeof(serv_addr));
    serv_addr.sin_family=AF_INET;
    serv_addr.sin_addr.s_addr=htonl(INADDR_ANY); /*Recebe de qq
interface*/
    serv_addr.sin_port=htons(atoi(argv[1])); /*Escuta no porto Well-
Known*/

    /*===== REGISTA-SE PARA ESCUTA =====*/
    if(bind(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) ==
SOCKET_ERROR)
        Abort("Impossibilidade de registrar-se para escuta", sock);

    /*===== AVISA QUE ESTA PRONTO A ACEITAR PEDIDOS =====*/
    if(listen(sock,5) == 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 =====*/
    cliaddr_len=sizeof(cli_addr);
    contador = 0;

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

    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(&fd_accept);
        FD_SET(sock, &fd_accept);

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

        switch(select(32, &fd_accept, NULL, NULL,
&tempoEspera)){
            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((newSock=accept(sock, (struct sockaddr
*)&cli_addr, &cliaddr_len)) == SOCKET_ERROR){

            if(WSAGetLastError() == WSAEINTR)
                continue;

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

        printf("<SER> Novo cliente conectado: <%s:%d>.\n",
inet_ntoa(cli_addr.sin_addr), ntohs(cli_addr.sin_port));
        send(newSock, MSG_BOAS_VINDAS, strlen(MSG_BOAS_VINDAS),
0);

        grupoSockets[contador++] = newSock;
    }

    if(contador == MAX_TAM_GRUPO || (ocorrenciaTimeout &&
contador>=2)){

        tam = contador;
    }

```

```

        parametrosThreadAtendeCliente = (SOCKET
*)malloc (MAX_TAM_GRUPO*sizeof (SOCKET));

        if(parametrosThreadAtendeCliente == NULL){

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

        }else{

                for(i=0; i<tam; i++){
                        parametrosThreadAtendeCliente[i] =
grupoSockets[i];
                }

                for(i=tam; i<MAX_TAM_GRUPO; 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;

    grupo = (SOCKET *)param;

    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
=====*/

        FD_ZERO(&fdread);
        for(i=0; i<MAX_TAM_GRUPO; i++){
            if(grupo[i] != INVALID_SOCKET){
                FD_SET(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(FD_ISSET(grupo[i], &fdread)){
                    if(difunde(i, grupo) < 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 é eliminado.

Devolve: numero de sockets activos / tamanho do grupo

---

```

    */
int difunde(int indexOrigem, SOCKET *grupo)
{
    int result, i, socketsActivos;
    char c;

    if((result=recv(grupo[indexOrigem], &c, sizeof(char),
0))==sizeof(char)){

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

            if(grupo[i] != INVALID_SOCKET && indexOrigem != i){

                result=send(grupo[i], &c, sizeof(char), 0);

                if(result == 0 || result == SOCKET_ERROR){

```

```

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

}

}

}else{
    closesocket(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);
}

```