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

#define TIMEOUT3   //segundos
#define BUFFERSIZE4096
#define MAX_TAM_GRUPO 10
#define LIMITE_ESPERA 30 //segundos
#define MSG_BOAS_VINDAS "Servidor conversa em grupo\r\nAguarde, cliente "
#define MSG_ARRANQUE_CONVERSA "Pode iniciar conversa...\r\n"

struct GrupoSockets{
int tamGrupo;
SOCKET sockets[MAX_TAM_GRUPO];
};

void fechaSockets(SOCKET *grupo, int tam);
void AtendeCliente(LPVOID param);
int difunde(SOCKET s1, SOCKET *grupo, int tam, int sock_number);
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;
struct GrupoSockets *parametrosThreadAtendeCliente;
int ocorrenciaTimeout;
fd_set fd_accept;
struct timeval tempoEspera;
char cliente_id_msg[4];

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


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

```
// Questão 1 -  crie um socket TCP
if((sock= socket(PF_INET, SOCK_STREAM, IPPROTO_TCP))  == INVALID_SOCKET)
Abort("Impossibilidade de abrir socket", sock);



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*/

// Questão 2 - utilize o porto passado na linha de comandos
serv_addr.sin_port=htons(atoi(argv[1]));  /*Escuta no porto Well-Known*/

// Questão 3 - registe o servidor para escuta no porto definido
if(bind(sock, (struct sockaddr*)&serv_addr,sizeof(serv_addr)) == SOCKET_ERROR)
Abort("Impossibilidade de registar-se para escuta", sock);

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]);

cliaddr_len=sizeof(cli_addr);
contador = 0;
while(1){
        ocorrenciaTimeout=0; //false
```

```
// Questão 4 - defina o número mínimo de clientes, antes de activar o timeout

if(contador >= 4){


// Questão 5 - defina o timeout usando a função select e as variáveis fd_accept e

        // tempoEspera

FD_ZERO(&fd_accept);

FD_SET(sock, &fd_accept);

switch(select(32, &fd_accept, NULL, NULL, &tempoEspera )){

case -1:

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){


// Questão 6 - Atende pedidos de ligação

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));

// Questão 7 - Envie a mensagem de boas vindas ao cliente, definida na constante

                    // MSG_BOAS_VINDAS

send(newSock, MSG_BOAS_VINDAS, strlen(MSG_BOAS_VINDAS), 0);

// Questão 8 - Construa a mensagem com identificador do número do cliente e

                    // envie-a ao cliente

sprintf(cliente_id_msg, "cliente %d", contador+1);

send(newSock, cliente_id_msg, strlen(cliente_id_msg), 0);

grupoSockets[contador++] = newSock;

}

if(contador == MAX_TAM_GRUPO || (ocorrenciaTimeout && contador>=4)){

tam = contador;

contador = 0;

parametrosThreadAtendeCliente = (struct GrupoSockets *)malloc(sizeof(struct GrupoSockets));

if(parametrosThreadAtendeCliente == NULL){

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

}else{

parametrosThreadAtendeCliente->tamGrupo = tam;

for(i=0; i<tam; i++)

parametrosThreadAtendeCliente->sockets[i] = grupoSockets[i];
```

```
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, contador);

}


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



}

}

}

}


/*_____ fechaSockets _____

_____*/

void fechaSockets(SOCKET *grupo, int tam)

{

int i;


for(i=0; i<tam; i++)

closesocket(grupo[i]);

}
```

```
/*_____ AtendeCliente _____

Atende cliente.

_____*/


void AtendeCliente(LPVOID param){

fd_set fdread, fdtemp;

struct GrupoSockets *grupo;

int i;

//struct timeval timeout = {TIMEOUT, 0};


grupo = (struct GrupoSockets *)param;


// Questão 9 - Envie a todos os cliente ligados a mensagem de arranque de conversa (definida

      //  na constante MSG_ARRANQUE_CONVERSA

for(i=0; i<grupo->tamGrupo; i++){

send(grupo->sockets[i], MSG_ARRANQUE_CONVERSA,
strlen(MSG_ARRANQUE_CONVERSA), 0);

}



// Questão 10 - Inicie fdread de modo a poder testar os diversos sockets para efeitos de leitura

FD_ZERO(&fdread);

for(i=0; i<grupo->tamGrupo; i++){

FD_SET(grupo->sockets[i], &fdread);

}
```

```
while(1){

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

fdtemp=fdread;

// Questão 11 - Invoque a função select, passando-lhe fdtemp, sem tempo limite de espera


switch(select(32, &fd_temp, 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->sockets, grupo->tamGrupo);

free(grupo);

return;


case  0:printf(".");

break;


default:

// Questão 12 - teste os diversos sockets para efeitos de leitura

for(i=0; i<grupo->tamGrupo; i++){

if(FD_ISSET(grupo->sockets[i], &fdtemp)){

if(difunde(grupo->sockets[i], grupo->sockets, grupo->tamGrupo,i) <= 0){

fechaSockets(grupo->sockets, grupo->tamGrupo);
```

```
free(grupo);
return;
}
}

}

break;
} //switch
} //while
}

/*_____ difunde _____
Recebe um caractere em s1 e reenvia-o para o grupo.
Não reenvia para s1.
Quando ocorre um problema com um dos elementos do grupo,
o grupo e' eliminado.

Devolve:
SOCKET_ERROR : se houve erro
  0 : EOF
      >= 0 : se leu algum byte
_____*/
int difunde(SOCKET origem, SOCKET *grupo, int tam, int sock_number)
{
int result, i;
```

```
char c;

static int n_msg[MAX_TAM_GRUPO];


// Questão 13 - Inicialize a variável contadora de mensagens

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

if (grupo[i] != INVALID_SOCKET && tam != i)

n_msg[i]=sock_number;

// Questão 14 - Receba bytes em origem e

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




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

   if(origem!=grupo[i]){

í// Questão 15 - teste se o número da mensagem a enviar é ímpar e envie os bytes recebidos

if( (n_msg[i]%2) != 0 ){

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


}

//Questão 16 - teste se o número da mensagem a enviar é par e converta vogais minúsculas

       // em maiúsculas e  vogais maiúsculas em minúsculas, conforme pedido

if((tam%2) == 0 ){

    if (c == 'A' || c == 'E' || c == 'I' || c == 'O' || c == 'U') //verifique se o caracter é uma vogal
maiúscula

    c = c + ('a' - 'A'); // converta-o para minúscula

                                          se o caracter for uma vogal minúscula

                                             c = c - ('a' - 'A'); // converta-a para maiúscula

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

```
}

   }

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

break;



}

      // Questão 17 - Caso tenha recebido uma mensagem completa, incremente o contador de mensagens para esse cliente

if(result == sizeof(c)){

n_msg[ i ]+=1;



}



}



if(result==0)

fprintf(stderr, "<SER_%d> Connection closed by foreign host\n", GetCurrentThreadId());



if(result == SOCKET_ERROR){

fprintf(stderr, "<SER_%d> Erro no acesso para I/O a um dos sockets (%d)\n", GetCurrentThreadId(), WSAGetLastError());

}



return result;

}
```

```
/*_____ 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);

}
```