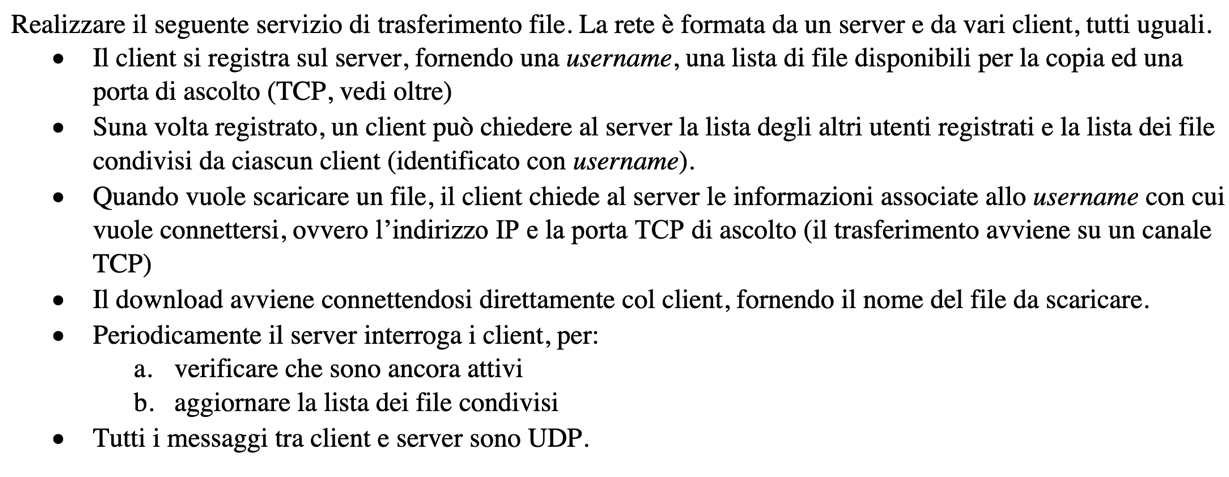
1 Esercizio, FTP con UDP (anche TCP)



Client.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <errno.h>

#include <sys/wait.h>

#define MAX\_LINE 1000

#define DAEMON\_PORT 60327

int main(int argc, char\*\*argv){

int sockfd, n;

int sockfd2;

extern int errno;

struct sockaddr\_in local\_addr, remote\_addr, daemon\_addr;

socklen\_t len = sizeof(struct sockaddr\_in );

char mesg[MAX\_LINE];

char notify[MAX\_LINE];

char nome[MAX\_LINE];

char portaContat[MAX\_LINE];

char ip[MAX\_LINE];

char listaFile[MAX\_LINE];

char file[MAX\_LINE];

char ipA[MAX\_LINE];

char receiveline[MAX\_LINE];

char indirizzoContattare[MAX\_LINE];

char ignore[MAX\_LINE];

char TCPport[MAX\_LINE];

char alert[MAX\_LINE];

FILE\* log;

log = fopen("log.txt", "w");

fclose(log);

FILE\* addFile;

addFile = fopen("addFile.txt", "w"); *//file per aggiungere i nuovi file alla lista*

fclose(addFile);

if(argc < 5){

printf("Use: Destination\_IP Destination\_Port Listening\_PORT, TCP\_port");

return 0;

}

*//demone per ACK*

if(!fork()){

char ACK[MAX\_LINE];

if((sockfd2=socket(PF\_INET,SOCK\_DGRAM,0)) <0){

printf("\nError in socket opening ... exit!");

return -1;

}

memset(&daemon\_addr,0,sizeof(daemon\_addr));

daemon\_addr.sin\_family = AF\_INET;

daemon\_addr.sin\_port=htons(DAEMON\_PORT);

if(bind(sockfd2, (struct sockaddr \*) &daemon\_addr, len)<0){

printf("\nBinding error! Error code n.%d \n",errno);

return -1;

}

for(;;){

n = recvfrom(sockfd2,ACK,MAX\_LINE-1,0,(struct sockaddr \*) &daemon\_addr,&len);

ACK[n] = 0;

*//printf("ho ricevuto un messaggio da ip: %s e porta %d",inet\_ntoa(daemon\_addr.sin\_addr),ntohs(daemon\_addr.sin\_port));*

addFile = fopen("addFile.txt", "r");

char buff[MAX\_LINE];

fgets(buff, MAX\_LINE, addFile);

buff[strcspn(buff,"\n")]=0;

*//printf("aggiorno il file con la stringa: %s\n", buff);*

sendto(sockfd2,buff,strlen(buff),0,(struct sockaddr \*)&daemon\_addr,len);

memset(buff,0, strlen(buff));

}

}

*//serverTCP*

if(!fork()){

int sockfd,newsockfd,n;

struct sockaddr\_in local\_addr,remote\_addr;

socklen\_t len;

char mesg[1000];

char buffer[MAX\_LINE];

FILE\* f;

if((sockfd=socket(AF\_INET,SOCK\_STREAM,0)) <0){

printf("\nErrore nell'apertura del socket");

return -1;

}

memset((char \*) &local\_addr,0,sizeof(local\_addr));

local\_addr.sin\_family = AF\_INET;

local\_addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

local\_addr.sin\_port = htons(atoi(argv[4]));

if(bind(sockfd, (struct sockaddr \*) &local\_addr, sizeof(local\_addr))<0){

printf("\nErrore nel binding. Errore %d \n",errno);

return -1;

}

listen(sockfd,5);

for(;;){

len = sizeof(remote\_addr);

newsockfd = accept(sockfd,(struct sockaddr \*)&remote\_addr, &len);

if(fork() == 0){ *//se sono il figlio chiudo la socket vecchia e lascio aperta quella generata da accept e procedo con la comunicazione su quella*

close(sockfd);

n = recv(newsockfd,mesg,999,0); *//riceverò il nome del file da*

mesg[n] = 0;

printf("ho ricevuto una richiesta per il file: %s\n", mesg);

f = fopen(mesg, "r");

fgets(buffer, MAX\_LINE, f); *//presuppongo file da una sola riga*

strcat(buffer, " ");

send(newsockfd,buffer,strlen(buffer),0); *//invierà il file appena bufferizzato*

printf("file %s inviato con successo\n", mesg);

return 0;

}else close(newsockfd); *//il padre chiude la nuova socket e mantenendo aperta quella vecchia si rimette in ascolto*

}

}

*//PARTE UDP*

if(!fork()){

if((sockfd=socket(PF\_INET,SOCK\_DGRAM,0)) <0){

printf("\nError in socket opening ... exit!");

return -1;

}

memset(&remote\_addr,0,len);

remote\_addr.sin\_family = AF\_INET;

inet\_pton(AF\_INET, argv[1], &(remote\_addr.sin\_addr));

remote\_addr.sin\_port=htons(atoi(argv[2]));

strcpy(notify, "N");

sendto(sockfd,notify,strlen(notify), 0, (struct sockaddr \*)&remote\_addr, len);

printf("inserisci nome utente\n");

fgets(mesg, MAX\_LINE, stdin);

sendto(sockfd,mesg,strlen(mesg), 0, (struct sockaddr \*)&remote\_addr, len); *//invio nome utente*

strcpy(mesg,argv[3]);

sendto(sockfd,mesg,strlen(mesg), 0, (struct sockaddr \*)&remote\_addr, len); *//invio porta*

strcpy(mesg,argv[4]);

sendto(sockfd,mesg,strlen(mesg), 0, (struct sockaddr \*)&remote\_addr, len); *//invio portaTCP*

printf("inserisci lista dei file \n");

fgets(file, MAX\_LINE, stdin);

addFile = fopen("addFile.txt", "w");

file[strcspn(file, "\n")]=0;

fputs(file, addFile);

fclose(addFile);

sendto(sockfd,file,strlen(file), 0, (struct sockaddr \*)&remote\_addr, len); *//invio lista file*

printf("Inserisci who o modifica la lista dei file condivisi manipolando il file addFile\n");

fgets(mesg, MAX\_LINE,stdin);

sendto(sockfd,mesg,strlen(mesg), 0, (struct sockaddr \*)&remote\_addr, len); *//invio messaggio*

close(sockfd);

*// receiver*

if((sockfd=socket(PF\_INET,SOCK\_DGRAM,0)) <0){

printf("\nError in socket opening ... exit!");

return -1;

}

memset(&local\_addr,0,sizeof(local\_addr));

local\_addr.sin\_family = AF\_INET;

local\_addr.sin\_port=htons(atoi(argv[3]));

if(bind(sockfd, (struct sockaddr \*) &local\_addr, len)<0){

printf("\nBinding error! Error code n.%d \n",errno);

return -1;

}

for(;;){

n = recvfrom(sockfd,alert,MAX\_LINE-1,0,(struct sockaddr \*) &remote\_addr,&len); *//ricevo X o W*

alert[n] = 0;

*// printf("alert: %s\n", alert);*

while(strcmp(nome,"X") != 0 || strcmp(nome,"X\n") != 0)

{

n = recvfrom(sockfd,nome,MAX\_LINE-1,0,(struct sockaddr \*) &remote\_addr,&len); *//ricevo il nome dell'utente*

nome[n] = 0;

if(strcmp(nome,"X") == 0 || strcmp(nome,"X\n") == 0) break;

n = recvfrom(sockfd,ip,MAX\_LINE-1,0,(struct sockaddr \*) &remote\_addr,&len);

ip[n] = 0; *//ricevo l'indirizzo*

n = recvfrom(sockfd,TCPport,MAX\_LINE-1,0,(struct sockaddr \*) &remote\_addr,&len);

TCPport[n] = 0; *//ricevo la portaTCP*

n = recvfrom(sockfd,file,MAX\_LINE-1,0,(struct sockaddr \*) &remote\_addr,&len);

file[n] = 0; *//ricevo elenco file*

*//scrivo su file le credenziali degli utenti*

log = fopen("log.txt", "a");

fputs(nome, log);

fflush(log);

fputs("\n",log);

fflush(log);

fputs(ip, log);

fflush(log);

fputs("\n", log);

fflush(log);

fputs(TCPport, log);

fflush(log);

fputs("\n", log);

fflush(log);

fputs(file, log);

fflush(log);

fputs("\n", log);

fflush(log);

fclose(log);

printf("\n");

printf("utente: %s, ip: %s, PORTA TCP: %s, \nFILE: %s\n", nome, ip, TCPport, file);

printf("\n");

}

char indice[MAX\_LINE];

printf("inserire nome del client da contattare\n");

fgets(indice, MAX\_LINE, stdin);

indice[strcspn(indice,"\n")]=0;

log = fopen("log.txt","r");

while(fgets(nome,MAX\_LINE,log) != NULL && fgets(ipA,MAX\_LINE,log) != NULL && fgets(portaContat,MAX\_LINE,log) != NULL && fgets(file,MAX\_LINE,log) != NULL)

{

nome[strcspn(nome, "\n")]=0;

ipA[strcspn(ipA, "\n")]=0;

portaContat[strcspn(portaContat, "\n")]=0;

file[strcspn(file, "\n")]=0;

if(strcmp(nome,indice) == 0){

break;

}

}

if(!fork()){

int sockfd,n;

struct sockaddr\_in local\_addr, dest\_addr;

char sendline[1000];

char recvline[1000];

sockfd=socket(AF\_INET,SOCK\_STREAM,0);

memset( &dest\_addr, 0, sizeof(dest\_addr));

dest\_addr.sin\_family = AF\_INET;

dest\_addr.sin\_addr.s\_addr = inet\_addr(ipA);

dest\_addr.sin\_port = htons(atoi(portaContat));

printf("instauro una connessione TCP con indirizzo: %s e porta: %s\n",ipA, portaContat);

connect(sockfd, (struct sockaddr \*) &dest\_addr, sizeof(dest\_addr));

printf("inserire il nome del file da scaricare\n");

fgets(mesg, MAX\_LINE,stdin);

mesg[strcspn(mesg, "\n")] = 0;

send(sockfd,mesg,strlen(mesg),0); *//invio il nome del file che voglio*

n=recv(sockfd,recvline,999,0); *//ricevo il file*

recvline[n]=0;

printf("ho ricevuto il file: %s con contenuto: %s\n", mesg, recvline);

FILE\* new;

new = fopen(mesg, "w");

fputs(recvline, new);

fflush(new);

fclose(new);

exit(0);

}

}

return 0;

}

wait(NULL);

}

Server.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <errno.h>

#include <sys/wait.h>

#include <time.h>

#include <sys/types.h>

#include <sys/socket.h>

#define MAX\_LINE 1000

#define DAEMON\_PORT 63001

int main(int argc, char\*\*argv){

int sockfd, n;

extern int errno;

struct sockaddr\_in local\_addr, remote\_addr;

socklen\_t len = sizeof(struct sockaddr\_in );

char mesg[MAX\_LINE];

char ip[MAX\_LINE];

char nome[MAX\_LINE];

char nome\_utente[MAX\_LINE];

char porta[MAX\_LINE];

char who[MAX\_LINE];

char ignore[MAX\_LINE];

char ipAppoggio[MAX\_LINE];

char file[MAX\_LINE];

char portaTCP[MAX\_LINE];

char mesgS[MAX\_LINE];

char ack[MAX\_LINE];

char recv[MAX\_LINE];

int portaAppoggio;

FILE\* login;

FILE\* messaggi;

login = fopen("login.txt", "w");

messaggi = fopen("messaggi.txt", "w");

fclose(login);

fclose(messaggi);

*//DEMONE PER ACK*

if(!fork()){

int sockfd2;

struct timeval read\_timeout;

struct sockaddr\_in daemon\_addr;

read\_timeout.tv\_sec = 0;

read\_timeout.tv\_usec = 10;

socklen\_t len = sizeof(struct sockaddr\_in );

FILE\* log;

if((sockfd2=socket(PF\_INET,SOCK\_DGRAM,0)) <0){

printf("\nError in socket opening ... exit!");

return -1;

}

memset(&daemon\_addr,0,sizeof(daemon\_addr));

daemon\_addr.sin\_family = AF\_INET;

daemon\_addr.sin\_port=htons(DAEMON\_PORT);

if(bind(sockfd2, (struct sockaddr \*) &daemon\_addr, len)<0){

printf("\nBinding error! Error code n.%d \n",errno);

return -1;

}

setsockopt(sockfd2, SOL\_SOCKET, SO\_RCVTIMEO, &read\_timeout, sizeof read\_timeout); *//rende sockfd2 non bloccante*

for(;;){

sleep(30);

log = fopen("login.txt", "r");

char app[MAX\_LINE];

while(fgets(ignore, MAX\_LINE, log) != NULL && fgets(ip, MAX\_LINE, log) != NULL && fgets(ignore, MAX\_LINE, log) != NULL && fgets(ignore, MAX\_LINE, log) != NULL && fgets(ignore, MAX\_LINE, log) != NULL)

{

strcpy(ack, "V");

ip[strcspn(ip, "\n")]=0;

inet\_pton(AF\_INET, ip, &(daemon\_addr.sin\_addr));

sendto(sockfd2,ack,strlen(ack), 0, (struct sockaddr \*)&daemon\_addr, len);

sleep(5);

*//ping pong tra stringhe per evitare race condition*

char app[MAX\_LINE];

int x=1;

if(x == 1){

n = recvfrom(sockfd2,recv,999,0,(struct sockaddr \*) &daemon\_addr,&len);

x --;

recv[strcspn(recv, "\n")]=0;

strcpy(app, recv);

memset(recv, 0, strlen(recv));

}else{

n = recvfrom(sockfd2,ack,999,0,(struct sockaddr \*) &daemon\_addr,&len);

x++;

ack[strcspn(ack, "\n")]=0;

strcpy(app, ack);

memset(ack, 0, strlen(ack));

}

sleep(5);

printf("n: %d, rec: %s\n", n, app);

if(n <= 0){

printf("ack non ricevuto\n");

FILE\* logout;

logout = fopen("logout.txt", "w");

login = fopen("login.txt", "r");

while(fgets(nome, MAX\_LINE, login)!=NULL && fgets(ip, MAX\_LINE, login)!=NULL && fgets(porta, MAX\_LINE, login)!=NULL && fgets(portaTCP, MAX\_LINE, login)!=NULL && fgets(file, MAX\_LINE, login)!=NULL)

{

nome[strcspn(nome,"\n")]=0;

ip[strcspn(ip,"\n")]=0;

porta[strcspn(porta,"\n")]=0;

portaTCP[strcspn(portaTCP,"\n")]=0;

file[strcspn(file,"\n")]=0;

if(strcmp(inet\_ntoa(daemon\_addr.sin\_addr), ip)!=0) {

fputs(nome, logout); *//inserisco nome*

fflush(logout);

fputs("\n", logout);

fflush(logout);

fputs(ip, logout); *//inserisco ip*

fflush(logout);

fputs("\n", logout);

fflush(logout);

fputs(porta, logout); *//inserisco porta*

fflush(logout);

fputs("\n", logout);

fflush(logout);

fputs(portaTCP, logout); *//inserisco portaTCP*

fflush(logout);

fputs("\n", logout);

fflush(logout);

fputs(file, logout); *//inserisco file*

fflush(logout);

fputs("\n", logout);

fflush(logout);

}

}

fclose(logout);

fclose(login);

logout = fopen("logout.txt", "r");

login = fopen("login.txt", "w");

while(fgets(nome, MAX\_LINE, logout)!=NULL && fgets(ip, MAX\_LINE, logout)!=NULL && fgets(porta, MAX\_LINE, logout)!=NULL && fgets(portaTCP, MAX\_LINE, logout)!=NULL && fgets(file, MAX\_LINE, logout)!=NULL)

{

fputs(nome, login); *//inserisco nome*

fflush(login);

*//fputs("\n", login);*

*//fflush(login);*

fputs(ip, login); *//inserisco ip*

fflush(login);

*//fputs("\n", login);*

*//fflush(login);*

fputs(porta, login); *//inserisco porta*

fflush(login);

*//fputs("\n", login);*

*//fflush(login);*

fputs(portaTCP, login); *//inserisco portaTCP*

fflush(login);

*//fputs("\n", login);*

*//fflush(login);*

fputs(file, login); *//inserisco file*

fflush(login);

fputs("\n", login);

fflush(login);

}

fclose(logout);

fclose(login);

}else{

*//dentro ack ho la nuova stringa*

printf("Daemon: ack arrivato da indirizzo: %s e porta %d\n",inet\_ntoa(daemon\_addr.sin\_addr), ntohs(daemon\_addr.sin\_port));

login = fopen("login.txt", "r");

FILE\* temp;

temp = fopen("temp.txt", "w");

login = fopen("login.txt", "r");

while(fgets(nome, MAX\_LINE, login)!=NULL && fgets(ip, MAX\_LINE, login)!=NULL && fgets(porta, MAX\_LINE, login)!=NULL && fgets(portaTCP, MAX\_LINE, login)!=NULL && fgets(file, MAX\_LINE, login)!=NULL)

{

nome[strcspn(nome,"\n")]=0;

ip[strcspn(ip,"\n")]=0;

porta[strcspn(porta,"\n")]=0;

portaTCP[strcspn(portaTCP,"\n")]=0;

file[strcspn(file,"\n")]=0;

fputs(nome, temp); *//inserisco nome*

fflush(temp);

fputs("\n", temp);

fflush(temp);

fputs(ip, temp); *//inserisco ip*

fflush(temp);

fputs("\n", temp);

fflush(temp);

fputs(porta, temp); *//inserisco porta*

fflush(temp);

fputs("\n", temp);

fflush(temp);

fputs(portaTCP, temp); *//inserisco portaTCP*

fflush(temp);

fputs("\n", temp);

fflush(temp);

if(strcmp(inet\_ntoa(daemon\_addr.sin\_addr), ip)==0) fputs(app, temp);

else fputs(file, temp);

fflush(temp);

fputs("\n", temp);

fflush(temp);

}

fclose(temp);

fclose(login);

temp = fopen("temp.txt", "r");

login = fopen("login.txt", "w");

while(fgets(nome, MAX\_LINE, temp)!=NULL && fgets(ip, MAX\_LINE, temp)!=NULL && fgets(porta, MAX\_LINE, temp)!=NULL && fgets(portaTCP, MAX\_LINE, temp)!=NULL && fgets(file, MAX\_LINE, temp)!=NULL)

{

fputs(nome, login); *//inserisco nome*

fflush(login);

*//fputs("\n", login);*

*//fflush(login);*

fputs(ip, login); *//inserisco ip*

fflush(login);

*//fputs("\n", login);*

*//fflush(login);*

fputs(porta, login); *//inserisco porta*

fflush(login);

*//fputs("\n", login);*

*//fflush(login);*

fputs(portaTCP, login); *//inserisco portaTCP*

fflush(login);

*//fputs("\n", login);*

*//fflush(login);*

fputs(file, login); *//inserisco file*

fflush(login);

*//fputs("\n", login);*

*//fflush(login);*

}

fclose(temp);

fclose(login);

}

}

fclose(log);

}

}

*//PARTE UDP*

if(!fork()){

*// receiver*

if((sockfd=socket(PF\_INET,SOCK\_DGRAM,0)) <0){

printf("\nError in socket opening ... exit!");

return -1;

}

memset(&local\_addr,0,sizeof(local\_addr));

local\_addr.sin\_family = AF\_INET;

local\_addr.sin\_port=htons(atoi(argv[1]));

if(bind(sockfd, (struct sockaddr \*) &local\_addr, len)<0){

printf("\nBinding error! Error code n.%d \n",errno);

return -1;

}

for(;;){

*//questa recv, o riceve N e porta alla fase di registrazione, oppure who*

n = recvfrom(sockfd,mesg,999,0,(struct sockaddr \*) &remote\_addr,&len);

mesg[n] = 0;

if(strcmp(mesg, "N") == 0 || strcmp(mesg, "N\n") == 0){

*//da eseguire soltanto quando un nuovo client si vuole registrare*

n = recvfrom(sockfd,nome\_utente,999,0,(struct sockaddr \*) &remote\_addr,&len); *//riceve il nome utente*

nome\_utente[n] = 0;

printf("sto registrando l'utente %s\n", nome\_utente);

n = recvfrom(sockfd,porta,999,0,(struct sockaddr \*) &remote\_addr,&len); *//riceve il numero di porta*

porta[n] = 0;

n = recvfrom(sockfd,portaTCP,999,0,(struct sockaddr \*) &remote\_addr,&len); *//riceve il numero di porta*

portaTCP[n] = 0;

n = recvfrom(sockfd,file,999,0,(struct sockaddr \*) &remote\_addr,&len);

file[n]=0;

*//scrittura sul file login*

login = fopen("login.txt", "a");

fputs(nome\_utente,login); *//nome utente*

fflush(login);

fputs(inet\_ntoa(remote\_addr.sin\_addr), login); *//ip*

fflush(login);

fputs("\n", login);

fflush(login);

fputs(porta, login); *//porta*

fflush(login);

fputs("\n", login);

fflush(login);

fputs(portaTCP, login); *//porta TCP*

fflush(login);

fputs("\n", login);

fflush(login);

fputs(file,login); *//lista dei file*

fflush(login);

fputs("\n", login);

fflush(login);

fclose(login);

}else if(strcmp(mesg, "who") == 0 || strcmp(mesg, "who\n") == 0) *//WHO*

{

strcpy(ipAppoggio, inet\_ntoa(remote\_addr.sin\_addr));

login = fopen("login.txt", "r");

*//cerco la porta per spedire*

while(fgets(ignore, MAX\_LINE, login) != NULL && fgets(ip, MAX\_LINE, login) != NULL && fgets(porta, MAX\_LINE, login) != NULL && fgets(ignore, MAX\_LINE, login) != NULL && fgets(ignore, MAX\_LINE, login) != NULL)

{

ip[strcspn(ip, "\n")]=0;

porta[strcspn(porta,"\n")]=0;

if(strcmp(ip, ipAppoggio) == 0){

porta[strcspn(porta, "\n")] = 0;

portaAppoggio = atoi(porta);

break;

}

}

fclose(login);

login = fopen("login.txt", "r");

strcpy(mesg, "W");

if(sockfd>0){

memset(&remote\_addr, 0, sizeof(remote\_addr));

remote\_addr.sin\_family = AF\_INET;

inet\_pton(AF\_INET, ipAppoggio, &(remote\_addr.sin\_addr));

remote\_addr.sin\_port = htons(portaAppoggio);

sendto(sockfd,mesg,strlen(nome), 0, (struct sockaddr \*)&remote\_addr, len); *//invio del nome utente*

}

while(fgets(nome, MAX\_LINE, login)!= NULL && fgets(ip, MAX\_LINE, login)!= NULL && fgets(ignore, MAX\_LINE, login)!= NULL && fgets(portaTCP, MAX\_LINE, login)!= NULL && fgets(file, MAX\_LINE, login)!= NULL)

{

nome[strcspn(nome, "\n")]=0;

ip[strcspn(ip, "\n")]=0;

portaTCP[strcspn(portaTCP, "\n")]=0;

file[strcspn(file, "\n")]=0;

printf("invio a indirizzo %s e porta %d \n", ipAppoggio, portaAppoggio);

if(sockfd>0){

memset(&remote\_addr, 0, sizeof(remote\_addr));

remote\_addr.sin\_family = AF\_INET;

inet\_pton(AF\_INET, ipAppoggio, &(remote\_addr.sin\_addr));

remote\_addr.sin\_port = htons(portaAppoggio);

sendto(sockfd,nome,strlen(nome), 0, (struct sockaddr \*)&remote\_addr, len); *//invio del nome utente*

sendto(sockfd,ip,strlen(ip), 0, (struct sockaddr \*)&remote\_addr, len); *//invio dell'ip*

sendto(sockfd,portaTCP,strlen(portaTCP), 0, (struct sockaddr \*)&remote\_addr, len); *//invio della portaTCP*

sendto(sockfd,file,strlen(file), 0, (struct sockaddr \*)&remote\_addr, len); *//invio della lista file*

}

}

strcpy(mesgS,"X");

sendto(sockfd,mesgS,strlen(file), 0, (struct sockaddr \*)&remote\_addr, len); *//invio carattere di terminazione*

fclose(login);

}

}

return 0;

}

else{

wait(NULL);

for(;;){}

}

}

**2 Esercizio, chat TCP Multi client**

Immagine che contiene testo

Descrizione generata automaticamente

Client.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <errno.h>

#define DIM\_LINE 1000

int main(int argc, char\* argv[]) {

int sockfd, newsockfd, n;

struct sockaddr\_in local\_addr, remote\_addr;

socklen\_t len;

char sendline[DIM\_LINE];

char receiveline[DIM\_LINE];

if (argc!=4) {

fprintf(stderr, "Paramatri non validi\n");

exit(1);

}

if (fork()==0) {

if ((sockfd=socket(PF\_INET, SOCK\_STREAM, 0))<0) {

fprintf(stderr, "Errore socket\n");

exit(1);

}

memset(&local\_addr, 0, sizeof(local\_addr));

local\_addr.sin\_family = AF\_INET;

local\_addr.sin\_port = htons(atoi(argv[3]));

if (bind(sockfd, (struct sock\_addr\*) &local\_addr, sizeof(local\_addr))<0) {

fprintf(stderr, "Errore bind\n");

exit(1);

}

*//connessione tcp*

listen(sockfd, 1);

for (;;) {

len = sizeof(remote\_addr);

newsockfd = accept(sockfd, (struct sock\_addr\*) &remote\_addr, &len);

if (fork()==0) {

close(sockfd);

n = recv(newsockfd, receiveline, DIM\_LINE-1, 0);

receiveline[n]='\0';

printf("%s", receiveline);

exit(0);

}

else {

close(newsockfd);

}

}

}

else {

if ((sockfd=socket(PF\_INET, SOCK\_STREAM, 0))<0) {

fprintf(stderr, "Errore socket\n");

exit(1);

}

memset(&remote\_addr, 0, sizeof(local\_addr));

remote\_addr.sin\_family = AF\_INET;

remote\_addr.sin\_addr.s\_addr = inet\_addr(argv[1]);

remote\_addr.sin\_port = htons(atoi(argv[2]));

connect(sockfd, (struct sock\_adrr\*) &remote\_addr, sizeof(remote\_addr));

printf("Inserire nome utente: ");

fgets(sendline, DIM\_LINE, stdin);

send(sockfd, sendline, strlen(sendline), 0); *//nome utente mandato - contiene \n*

strncpy(sendline, argv[3], DIM\_LINE);

send(sockfd, sendline, strlen(sendline), 0); *//porta utente mandata*

while (fgets(sendline, DIM\_LINE, stdin)!=NULL && strstr(sendline, "logout")==NULL) {

send(sockfd, sendline, strlen(sendline), 0);

if (strstr(sendline, "who")!=NULL) {

while ((n=recv(sockfd, receiveline, DIM\_LINE-1, 0))) {

receiveline[n]='\0';

if (strstr(receiveline, "FINE DEGLI UTENTI CONNESSI")!=NULL) {

break;

}

printf("%s", receiveline);

}

}

}

send(sockfd, sendline, strlen(sendline), 0);

exit(0);

}

}

Server.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <errno.h>

#define DIM\_LINE 1000

int main(int argc, char\* argv[]) {

int sockfd, newsockfd, sendsockfd, n;

struct sockaddr\_in local\_addr, remote\_addr;

socklen\_t len;

char msg[DIM\_LINE];

char sendline[DIM\_LINE];

char mittente[DIM\_LINE];

char ip\_mittente[DIM\_LINE];

char porta\_mittente[DIM\_LINE];

char username[DIM\_LINE];

char porta[DIM\_LINE];

char ip[DIM\_LINE];

FILE\* login;

login = fopen("login.txt", "w");

fclose(login);

if (argc<2) {

fprintf(stderr, "Parametri non validi \n");

exit(1);

}

if ((sockfd=socket(PF\_INET, SOCK\_STREAM, 0))<0) {

fprintf(stderr, "Errore socket\n");

exit(1);

}

memset(&local\_addr, 0, sizeof(local\_addr));

local\_addr.sin\_family = AF\_INET;

local\_addr.sin\_port = htons(atoi(argv[1]));

if (bind(sockfd, (struct sock\_addr\*) &local\_addr, sizeof(local\_addr))<0) {

fprintf(stderr, "Errore bind\n");

exit(1);

}

*//connessione tcp*

listen(sockfd, 5);

for (;;) {

len = sizeof(remote\_addr);

newsockfd = accept(sockfd, (struct sock\_addr\*) &remote\_addr, &len);

*//connessione stabilita*

if (fork()==0) {

close(sockfd);

login = fopen("login.txt", "a");

n = recv(newsockfd, msg, DIM\_LINE-1, 0);

msg[n]='\0';

fputs(msg,login); *//nome utente scritto*

fflush(login);

n = recv(newsockfd, msg, DIM\_LINE-1, 0);

msg[n]='\0';

fputs(msg,login); *//porta scritta*

fflush(login);

fputs("\n", login);

fputs(inet\_ntoa(remote\_addr.sin\_addr),login);

fputs("\n", login); *//ip scritto*

fflush(login);

fclose(login);

for (;;) {

n = recv(newsockfd, msg, DIM\_LINE-1, 0);

msg[n]='\0';

if (strstr(msg, "who")!=NULL) {

login = fopen("login.txt", "r");

while (fgets(sendline, DIM\_LINE, login)!=NULL) {

send(newsockfd, sendline, strlen(sendline), 0);

fgets(sendline, DIM\_LINE, login);

fgets(sendline, DIM\_LINE, login);

}

strncpy(sendline, "FINE DEGLI UTENTI CONNESSI\n", DIM\_LINE);

send(newsockfd, sendline, strlen(sendline), 0);

fclose(login);

}

else {

if (strstr(msg, "logout")!=NULL) {

FILE\* logout;

logout = fopen("logout.txt", "w");

login = fopen("login.txt", "r");

while (fgets(username, DIM\_LINE, login)!=NULL && fgets(porta, DIM\_LINE, login)!=NULL && fgets(ip, DIM\_LINE, login)!=NULL) {

ip[strlen(ip)-1]='\0';

if (strcmp(inet\_ntoa(remote\_addr.sin\_addr), ip)!=0) {

fputs(username, logout);

fflush(logout);

fputs(porta, logout);

fflush(logout);

fputs(ip, logout);

fflush(logout);

fputs("\n", logout);

fflush(logout);

}

}

fclose(logout);

fclose(login);

logout = fopen("logout.txt", "r");

login = fopen("login.txt", "w");

while (fgets(username, DIM\_LINE, logout)!=NULL && fgets(porta, DIM\_LINE, logout)!=NULL && fgets(ip, DIM\_LINE, logout)!=NULL) {

fputs(username, login);

fflush(login);

fputs(porta, login);

fflush(login);

fputs(ip, login);

fflush(login);

}

fclose(logout);

fclose(login);

exit(0);

}

else {

login = fopen("login.txt", "r");

*//cerco mittente*

while (fgets(username, DIM\_LINE, login)!=NULL && fgets(porta, DIM\_LINE, login)!=NULL && fgets(ip, DIM\_LINE, login)!=NULL) {

ip[strcspn(ip, "\n")]=0;

if (strcmp(inet\_ntoa(remote\_addr.sin\_addr), ip)==0) {

strncpy(mittente, username, DIM\_LINE);

strncpy(ip\_mittente, ip, DIM\_LINE);

mittente[strlen(mittente)-1]='\0';

break;

}

}

fclose(login);

login = fopen ("login.txt", "r");

*//cerco altri client*

while (fgets(username, DIM\_LINE, login)!=NULL && fgets(porta, DIM\_LINE, login)!=NULL && fgets(ip, DIM\_LINE, login)!=NULL) {

ip[strcspn(ip, "\n")]=0;

porta[strcspn(porta, "\n")]=0;

if (strcmp(ip\_mittente, ip)!=0) {

strncpy(sendline, mittente, DIM\_LINE);

strncpy(sendline+strlen(sendline), ": ", DIM\_LINE-strlen(sendline));

strncpy(sendline+strlen(sendline), msg, DIM\_LINE-strlen(sendline));

sendsockfd=socket(AF\_INET, SOCK\_STREAM, 0);

if (sendsockfd>0) {

memset(&remote\_addr, 0, sizeof(remote\_addr));

remote\_addr.sin\_family = AF\_INET;

remote\_addr.sin\_addr.s\_addr = inet\_addr(ip);

remote\_addr.sin\_port = htons(atoi(porta));

connect(sendsockfd, (struct sock\_adrr\*) &remote\_addr, sizeof(remote\_addr));

send(sendsockfd, sendline, strlen(sendline), 0);

close(sendsockfd);

}

}

}

}

}

}

}

else {

close(newsockfd);

}

}

}

**3 UDP chat bidirezionale**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <errno.h>

int main(int argc, char\*\*argv)

{

int sockfd, n;

extern int errno;

struct sockaddr\_in local\_addr, remote\_addr;

socklen\_t len = sizeof(struct sockaddr\_in );

char mesg[1000];

if(argc < 4)

{

printf("Use: Destination\_IP Destination\_Port Listening\_PORT");

return 0;

}

if (!fork())

{ *// receiver*

if((sockfd=socket(PF\_INET,SOCK\_DGRAM,0)) <0)

{

printf("\nError in socket opening ... exit!");

return -1;

}

memset(&local\_addr,0,sizeof(local\_addr));

local\_addr.sin\_family = AF\_INET;

local\_addr.sin\_port=htons(atoi(argv[3]));

if( bind(sockfd, (struct sockaddr \*) &local\_addr, len)<0)

{

printf("\nBinding error! Error code n.%d \n",errno);

return -1;

}

for (;;)

{

n = recvfrom(sockfd,mesg,999,0,(struct sockaddr \*) &remote\_addr,&len);

mesg[n] = 0;

printf("From IP:%s Port:%d msg:%s \n", inet\_ntoa(remote\_addr.sin\_addr), ntohs(remote\_addr.sin\_port), mesg);

}

return 0;

}

else

{

if((sockfd=socket(PF\_INET,SOCK\_DGRAM,0)) <0)

{

printf("\nError in socket opening ... exit!");

return -1;

}

memset(&remote\_addr,0,len);

remote\_addr.sin\_family = AF\_INET;

inet\_pton(AF\_INET, argv[1], &(remote\_addr.sin\_addr));

remote\_addr.sin\_port=htons(atoi(argv[2]));

while (fgets(mesg, 1000,stdin) != NULL)

{

sendto(sockfd,mesg,strlen(mesg), 0, (struct sockaddr \*)&remote\_addr, len);

}

return 0;

}

}

**4 Esercizio, UDP chat broadcast**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <errno.h>

int main(int argc, char\*\*argv)

{

int sockfd, n;

extern int errno;

struct sockaddr\_in local\_addr, remote\_addr;

socklen\_t len = sizeof(struct sockaddr\_in );

char mesg[1000];

if(argc < 2)

{

printf("Use: Listening\_PORT \n");

return 0;

}

if (!fork())

{ *// receiver*

if((sockfd=socket(PF\_INET,SOCK\_DGRAM,0)) <0)

{

printf("\nError in socket opening ... exit!");

return -1;

}

memset(&local\_addr,0,sizeof(local\_addr));

local\_addr.sin\_family = AF\_INET;

local\_addr.sin\_port=htons(atoi(argv[1]));

if( bind(sockfd, (struct sockaddr \*) &local\_addr, len)<0)

{

printf("\nBinding error! Error code n.%d \n",errno);

return -1;

}

for (;;)

{

n = recvfrom(sockfd,mesg,999,0,(struct sockaddr \*) &remote\_addr,&len);

mesg[n] = 0;

printf("From IP:%s Port:%d msg:%s \n", inet\_ntoa(remote\_addr.sin\_addr), ntohs(remote\_addr.sin\_port), mesg);

}

return 0;

}

else

{

if((sockfd=socket(PF\_INET,SOCK\_DGRAM,0)) <0)

{

printf("\nError in socket opening ... exit!");

return -1;

}

int broadcastEnable=1;

int ret = setsockopt(sockfd, SOL\_SOCKET, SO\_BROADCAST, &broadcastEnable, sizeof(broadcastEnable));

memset(&remote\_addr,0,len);

remote\_addr.sin\_family = AF\_INET;

inet\_pton(AF\_INET, "255.255.255.255", &(remote\_addr.sin\_addr));

remote\_addr.sin\_port=htons(atoi(argv[1]));

while (fgets(mesg, 1000,stdin) != NULL)

{

sendto(sockfd,mesg,strlen(mesg), 0, (struct sockaddr \*)&remote\_addr, len);

}

return 0;

}

}

**5 Esercizio, Chat UDP a utenti selezionatiImmagine che contiene testo

Descrizione generata automaticamente**

**Client.c**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <netinet/in.h>

#define PORT 8080

#define BUFFSIZE 1024

typedef struct Suser {

char ip[256];

char port[256];

} user;

void print\_users (user \* connected\_users, int n\_users){

for (int i = 0; i < n\_users; i++){

printf("Utente connesso con ip %s e porta %s\n", connected\_users[i].ip, connected\_users[i].port);

}

}

void send\_messagge\_to\_peer (int sockfd, user usr\_to\_contact, char \* message){

struct sockaddr\_in peer\_addr;

memset(&peer\_addr, 0, sizeof(peer\_addr));

peer\_addr.sin\_family = AF\_INET;

peer\_addr.sin\_port = htons(atoi(usr\_to\_contact.port));

peer\_addr.sin\_addr.s\_addr = INADDR\_ANY;

inet\_pton(AF\_INET, usr\_to\_contact.ip, &(peer\_addr.sin\_addr));

printf("Sending message to %s %s: %s\n", usr\_to\_contact.ip, usr\_to\_contact.port, message);

sendto(sockfd, message, strlen(message) + 1, 0, (const struct sockaddr \* )&peer\_addr, sizeof(peer\_addr));

}

int main(int argc, char \*\* argv) {

if (argc < 4){

fprintf(stderr, "Usage %s porta\_local ip\_server porta\_server\n", argv[0]);

exit(1);

}

int sockfd;

if ((sockfd = socket(AF\_INET, SOCK\_DGRAM, 0)) < 0){

perror("Impossibile aprire socket");

exit(1);

}

int n, len;

len = sizeof(struct sockaddr\_in);

struct sockaddr\_in allowed\_addr;

memset(&allowed\_addr, 0, sizeof(allowed\_addr));

allowed\_addr.sin\_family = AF\_INET;

allowed\_addr.sin\_port = htons(atoi(argv[1]));

allowed\_addr.sin\_addr.s\_addr = INADDR\_ANY;

if ( bind(sockfd, (const struct sockaddr \*)&allowed\_addr, sizeof(allowed\_addr)) < 0){

perror("Impossibile bindare socket");

exit(1);

}

struct sockaddr\_in server\_addr;

memset(&server\_addr, 0, sizeof(server\_addr));

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_port = htons(atoi(argv[3]));

server\_addr.sin\_addr.s\_addr = INADDR\_ANY;

inet\_pton(AF\_INET, argv[2], &(server\_addr.sin\_addr));

sendto(sockfd, "/login", strlen("login") + 1, 0, (const struct sockaddr \* )&server\_addr, sizeof(server\_addr));

int n\_users;

user connected\_users [100];

memset(&connected\_users, 0, sizeof(connected\_users));

n\_users = 0;

char buff[BUFFSIZE];

for (;;) {

char \* ip, \* port;

n = recvfrom(sockfd, buff, BUFFSIZE - 1, 0, (struct sockaddr \* )&server\_addr, (socklen\_t \*)&len);

buff[n] = 0;

if (strncmp(buff, "done", BUFFSIZE) == 0) break;

ip = strtok(buff, " ");

port = strtok(NULL, " ");

strcpy(connected\_users[n\_users].ip, ip);

strcpy(connected\_users[n\_users].port, port);

printf("Ricevuto utente connesso con ip %s e porta %s\n", ip, port);

n\_users = n\_users+1;

fflush(stdout);

}

if (fork() == 0){

struct sockaddr\_in peer\_addr;

for (;;) {

memset(&peer\_addr, 0, sizeof(peer\_addr));

user peer\_usr;

n = recvfrom(sockfd, buff, BUFFSIZE - 1, 0, (struct sockaddr \* )&peer\_addr, (socklen\_t \*)&len);

buff[n] = 0;

inet\_ntop(AF\_INET, &(peer\_addr.sin\_addr), peer\_usr.ip, sizeof(peer\_usr.ip));

sprintf(peer\_usr.port, "%d", ntohs(peer\_addr.sin\_port));

printf("Messaggio in arrivo da %s %s: %s\n", peer\_usr.ip, peer\_usr.port, buff);

}

fflush(stdout);

}

else {

for (;;) {

int to\_contact;

print\_users(connected\_users, n\_users);

printf("Quale utente vorresti contattare?\n");

fgets(buff, BUFFSIZE, stdin);

buff[strlen(buff) - 1] = 0;

to\_contact = atoi(buff);

printf("Inserisci messaggio:\n");

fgets(buff, BUFFSIZE, stdin);

buff[strlen(buff) - 1] = 0;

send\_messagge\_to\_peer(sockfd, connected\_users[to\_contact], buff);

}

}

return 0;

}

**Server.c**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <netinet/in.h>

#define PORT 8080

#define BUFFSIZE 1024

typedef struct Suser {

char ip[256];

char port[256];

} user;

int is\_same\_user(user usr1, user usr2){

*//printf("Comparing ip: %s vs %s res %d; port %s vs %s res %d\n", usr1.ip, usr2.ip, strncmp(usr1.ip, usr2.ip, 256), usr1.port, usr2.port, strncmp(usr1.port, usr2.port, 256));*

*//printf("Comparing port %s vs %s res %d\n", usr1.port, usr2.port, strncmp(usr1.port, usr2.port, 256));*

return ((strncmp(usr1.ip, usr2.ip, 256) == 0) && (strncmp(usr1.port, usr2.port, 256) == 0));

}

int check\_presence (char \* filepath, user usr){

FILE \* f;

char buff[BUFFSIZE];

user tmp\_usr;

if ((f = fopen(filepath, "r")) == NULL){

perror("Impossibile leggere file dei presenti");

exit(1);

}

while(fgets(buff, BUFFSIZE, f)){

buff[strlen(buff) - 1] = 0;

strncpy(tmp\_usr.ip, strtok(buff, " "), 256);

strncpy(tmp\_usr.port, strtok(NULL, " "), 256);

if (is\_same\_user(usr, tmp\_usr)) return 0;

}

fclose(f);

return 1;

}

void add\_presence (char \* filepath, user usr){

FILE \* f;

if ((f = fopen(filepath, "a")) == NULL){

perror("Impossibile leggere file dei presenti");

exit(1);

}

fprintf(f, "%s %s\n", usr.ip, usr.port);

fclose(f);

}

void send\_reg\_file (char \* filepath, int sockfd, user usr, struct sockaddr\_in client\_addr){

FILE \* f;

char buff[BUFFSIZE];

user tmp\_usr;

if ((f = fopen(filepath, "r")) == NULL){

perror("Impossibile leggere file dei presenti");

exit(1);

}

*//printf("Inizio loop di sendfile\n");*

while(fgets(buff, BUFFSIZE, f)){

fflush(stdout);

buff[strlen(buff) - 1] = 0;

strncpy(tmp\_usr.ip, strtok(buff, " "), 256);

strncpy(tmp\_usr.port, strtok(NULL, " "), 256);

if (is\_same\_user(usr, tmp\_usr)) continue;

sprintf(buff, "%s %s", tmp\_usr.ip, tmp\_usr.port);

*//printf("Risultato: %d, Mandando utente: %s\n", is\_same\_user(usr, tmp\_usr), buff);*

sendto(sockfd, buff, strlen(buff) + 1, 0, (const struct sockaddr \* )&client\_addr, sizeof(client\_addr));

}

strncpy(buff, "done", BUFFSIZE);

sendto(sockfd, buff, strlen(buff) + 1, 0, (const struct sockaddr \* )&client\_addr, sizeof(client\_addr));

fclose(f);

}

int main(int argc, char \*\* argv) {

if (argc < 3){

fprintf(stderr, "Usage %s porta\_server filename\n", argv[0]);

exit(1);

}

int sockfd;

if ((sockfd = socket(AF\_INET, SOCK\_DGRAM, 0)) < 0){

perror("Impossibile aprire socket");

exit(1);

}

int n, len;

struct sockaddr\_in allowed\_addr, client\_addr;

memset(&allowed\_addr, 0, sizeof(allowed\_addr));

memset(&client\_addr, 0, sizeof(client\_addr ));

allowed\_addr.sin\_family = AF\_INET;

allowed\_addr.sin\_port = htons(atoi(argv[1]));

allowed\_addr.sin\_addr.s\_addr = INADDR\_ANY;

if ( bind(sockfd, (const struct sockaddr \*)&allowed\_addr, sizeof(allowed\_addr)) < 0){

perror("Impossibile bindare socket");

exit(1);

}

char buff[BUFFSIZE];

len = sizeof(client\_addr);

for (;;) {

memset(&client\_addr, 0, sizeof(client\_addr ));

n = recvfrom(sockfd, (char \*)buff, BUFFSIZE - 1, 0, (struct sockaddr \*)&client\_addr, (socklen\_t \*)&len);

buff[n] = 0;

*//printf("%s%lu\n", buff, strlen(buff));*

int a = 0;

if (strncmp(buff, "/login", BUFFSIZE) == 0){

user new\_user;

inet\_ntop(AF\_INET, &(client\_addr.sin\_addr), new\_user.ip, sizeof(new\_user.ip));

sprintf(new\_user.port, "%d", ntohs(client\_addr.sin\_port));

printf("Ricevuto messaggio da ip: %s, port: %s\n", new\_user.ip, new\_user.port);

if (check\_presence(argv[2], new\_user) == 1) add\_presence(argv[2], new\_user);

send\_reg\_file(argv[2], sockfd, new\_user, client\_addr);

}

fflush(stdout);

}

return 0;

}

**6 Esercizio, UDP Chat con TokenImmagine che contiene testo

Descrizione generata automaticamente**

**Input.txt**

**Gino**

**Macarena**

**8080**

**Client.c**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <netinet/in.h>

#define MAXLINE 1024

void trim\_new\_line (char \* str){

if (str[strlen(str) - 1] == '\n') str[strlen(str) - 1] = '\0';

}

int main(int argc, char \*\* argv) {

if (argc < 3){

fprintf(stderr, "Usage: %s ip port", argv[0]);

exit(1);

}

int sockfd;

if ( (sockfd = socket(AF\_INET, SOCK\_DGRAM, 0)) < 0 ) {

perror("socket creation failed");

exit(EXIT\_FAILURE);

}

char message[MAXLINE];

char username[256];

char password[256];

char port[256];

int token;

int n, len;

len = sizeof(struct sockaddr\_in);

printf("Inserire Username: ");

fgets(username, 256, stdin);

trim\_new\_line(username);

printf("Inserire Password: ");

fgets(password, 256, stdin);

trim\_new\_line(password);

printf("Inserire Porta: ");

fgets(port, 256, stdin);

trim\_new\_line(port);

sprintf(message, "/login %s %s %s", username, password, port);

struct sockaddr\_in allowed\_addr;

memset(&allowed\_addr, 0, sizeof(allowed\_addr));

allowed\_addr.sin\_family = AF\_INET;

allowed\_addr.sin\_port = htons(atoi(port));

allowed\_addr.sin\_addr.s\_addr = INADDR\_ANY;

if ( bind(sockfd, (const struct sockaddr \*)&allowed\_addr, sizeof(allowed\_addr) ) < 0 ) {

perror("bind failed");

exit(EXIT\_FAILURE);

}

struct sockaddr\_in servaddr;

memset(&servaddr, 0, sizeof(servaddr));

servaddr.sin\_family = AF\_INET;

servaddr.sin\_port = htons(atoi(argv[2]));

servaddr.sin\_addr.s\_addr = INADDR\_ANY;

inet\_pton(AF\_INET, argv[1], &(servaddr.sin\_addr));

sendto(sockfd, (const char \*)message, strlen(message), 0, (const struct sockaddr \*) &servaddr, sizeof(servaddr));

n = recvfrom(sockfd, message, MAXLINE - 1, 0, (struct sockaddr \*) &servaddr, (socklen\_t \*) &len);

message[n] = 0;

if (strcmp(message, "fail") == 0) {

fprintf(stderr, "Errore, impossibile registrarsi\n");

exit(1);

}

token = atoi(message);

char buffer[MAXLINE];

if (fork() == 0){

for (;;) {

sprintf(message, "%d ", token);

*//printf("Inserire messaggio: ");*

fgets(buffer, MAXLINE, stdin);

buffer[strlen(buffer) - 1] = 0;

strcat(message, buffer);

sendto(sockfd, (const char \*)message, strlen(message) + 1, 0, (const struct sockaddr \*) &servaddr, sizeof(servaddr));

}

}

else {

*// int receiving\_sockfd;*

*// struct sockaddr\_in allowed\_addr;*

*// if ( (receiving\_sockfd = socket(AF\_INET, SOCK\_DGRAM, 0)) < 0 ) {*

*// perror("socket creation failed");*

*// exit(EXIT\_FAILURE);*

*// }*

*// memset(&allowed\_addr, 0, sizeof(allowed\_addr));*

*// allowed\_addr.sin\_family = AF\_INET;*

*// allowed\_addr.sin\_port = htons(atoi(port));*

*// allowed\_addr.sin\_addr.s\_addr = INADDR\_ANY;*

*// if ( bind(receiving\_sockfd, (const struct sockaddr \*)&allowed\_addr, sizeof(allowed\_addr) ) < 0 ) {*

*// perror("bind failed");*

*// exit(EXIT\_FAILURE);*

*// }*

for (;;) {

n = recvfrom(sockfd, message, MAXLINE - 1, 0, (struct sockaddr \*) &servaddr, (socklen\_t \*) &len);

message[n] = 0;

printf("%s\n", message);

}

}

close(sockfd);

return 0;

}

**Server.c**

Porta diversa per ogni client, modificare 1092.168.56.255 con indirizzo broadcast della rete corretta

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <netinet/in.h>

#include <string.h>

#include <time.h>

#define MAXLINE 1024

int check\_presence(char \* filepath, char \* registration){

FILE \* f;

char buff[MAXLINE];

char \* tmp\_user, \* tmp\_passw, \* tmp\_token;

char tmp [MAXLINE];

strncpy(tmp, registration, MAXLINE);

char \* username = strtok(tmp, " ");

char \* password = strtok(NULL, " ");

srand(time(NULL));

if ((f = fopen(filepath, "r")) == NULL){

if ((f = fopen(filepath, "w")) == NULL){

perror("Impossibile creare file di presenza");

exit(1);

}

fclose(f);

if ((f = fopen(filepath, "r")) == NULL){

perror("Impossibile leggere file di presenza");

exit(1);

}

}

while (fgets(buff, MAXLINE, f)){

tmp\_user = strtok(buff, " ");

tmp\_passw = strtok(NULL, " ");

tmp\_token = strtok(NULL, " ");

tmp\_token = strtok(NULL, " ");

if (strcmp(tmp\_user, username) == 0 && strcmp(tmp\_passw, password) == 0) {fclose(f); return atoi(tmp\_token);}

else if (strcmp(tmp\_user, username) == 0) {fclose(f); return 0;}

}

fclose(f);

return rand();

}

int add\_presence (char \* filepath, char \* registration, int token){

FILE \* f;

if ((f = fopen(filepath, "a")) == NULL){

perror("Impossibile appendere file di presenza");

exit(1);

}

fprintf(f, "%s %d\n", registration, token);

fclose(f);

return token;

}

void get\_username (char \* filepath, char \* sender\_token, char \* username\_ptr) {

FILE \* f;

char buff[MAXLINE];

char username[256];

char password[256];

char port[256];

char token[32];

if ((f = fopen(filepath, "r")) == NULL){

perror("Impossibile leggere file di presenza");

exit(1);

}

while (fgets(buff, MAXLINE, f)){

sscanf(buff, "%s %s %s %s", username, password, port, token);

*//printf("Comparing: %s vs %s\n", token, sender\_token);*

if (strcmp(token, sender\_token) == 0){

strcpy(username\_ptr, username);

fclose(f);

return;

}

}

}

void send\_message\_to\_all(char \* filepath, char \* received\_message, int sockfd){

FILE \* f;

char buff[MAXLINE];

char sending\_message[MAXLINE];

char sender\_username[256];

char \* body;

char \* sender\_token;

char username[256];

char password[256];

char port[256];

char token[32];

if ((f = fopen(filepath, "r")) == NULL){

perror("Impossibile leggere file di presenza");

exit(1);

}

*//sscanf(received\_message, "%s %s", sender\_token, body);*

*// long int split\_point = (long int) (strstr(received\_message, " ") - received\_message) + 1;*

*// strncpy(sender\_token, received\_message, split\_point);*

sender\_token = strtok(received\_message, " ");

body = strtok(NULL, "");

*//il metodo mette nell'ultimo argomento il valore dello username trovato*

get\_username(filepath, sender\_token, sender\_username);

*//printf("Username ottenuto: %s", sender\_username);*

struct sockaddr\_in dst;

memset(&dst, 0, sizeof(dst));

dst.sin\_family = AF\_INET;

*//dst.sin\_port = htons(atoi(argv[2]));*

dst.sin\_addr.s\_addr = INADDR\_ANY;

int n, len;

len = sizeof(dst);

inet\_pton(AF\_INET, "192.168.56.255", &(dst.sin\_addr));

while (fgets(buff, MAXLINE, f)){

sscanf(buff, "%s %s %s %s", username, password, port, token);

if (strcmp(token, sender\_token) != 0){

sprintf(sending\_message, "[%s] %s", sender\_username, body);

printf("%s -> %s\n", sending\_message, username);

dst.sin\_port = htons(atoi(port));

sendto(sockfd, (const char \*)sending\_message, strlen(sending\_message), 0, (const struct sockaddr \*) &dst, sizeof(dst));

}

}

fclose(f);

}

*// Driver code*

int main(int argc, char \*\* argv) {

int sockfd;

char buffer[MAXLINE];

struct sockaddr\_in servaddr, cliaddr;

if (argc < 3){

fprintf(stderr, "Usage: %s port file", argv[0]);

exit(1);

}

if ( (sockfd = socket(AF\_INET, SOCK\_DGRAM, 0)) < 0 ) {

perror("socket creation failed");

exit(EXIT\_FAILURE);

}

int broadcastEnable=1;

int ret=setsockopt(sockfd, SOL\_SOCKET, SO\_BROADCAST, &broadcastEnable, sizeof(broadcastEnable));

memset(&servaddr, 0, sizeof(servaddr));

memset(&cliaddr, 0, sizeof(cliaddr));

servaddr.sin\_family = AF\_INET; *// IPv4*

servaddr.sin\_addr.s\_addr = INADDR\_ANY;

servaddr.sin\_port = htons(atoi(argv[1]));

if ( bind(sockfd, (const struct sockaddr \*)&servaddr, sizeof(servaddr) ) < 0 ) {

perror("bind failed");

exit(EXIT\_FAILURE);

}

int len, n;

len = sizeof(cliaddr); *//len is value/resuslt*

while(1) {

n = recvfrom(sockfd, (char \*)buffer, MAXLINE - 1, MSG\_WAITALL, ( struct sockaddr \*) &cliaddr, (socklen\_t \*)&len);

if (fork() == 0){

buffer[n] = '\0';

if(strncasecmp(buffer, "/login", strlen("/login")) == 0){

char \* registration;

int token;

registration = strtok(buffer, " ");

registration = strtok(NULL, "");

*//printf("Client : %s\n", registration);*

if ((token = check\_presence(argv[2], registration)) == 0){

fprintf(stderr, "Registrazione Fallita\n");

strcpy(buffer, "fail");

sendto(sockfd, (const char \*)buffer, strlen(buffer), 0, (const struct sockaddr \*) &cliaddr, sizeof(servaddr));

exit(1);

}

else {

add\_presence(argv[2], registration, token);

sprintf(buffer, "%d", token);

sendto(sockfd, (const char \*)buffer, strlen(buffer), 0, (const struct sockaddr \*) &cliaddr, sizeof(servaddr));

exit(0);

}

}

else {

send\_message\_to\_all(argv[2], buffer, sockfd);

exit(0);

}

}

}

close(sockfd);

return 0;

}

**7 Esercizio, Chat UDP con host in cerchioImmagine che contiene testo

Descrizione generata automaticamente**

**Node.c**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <netinet/in.h>

#define BUFFSIZE 1024

void send\_to\_next\_hop(int sockfd, char \* next\_ip, char \* next\_port, char \* buff){

struct sockaddr\_in next\_hop;

memset(&next\_hop, 0, sizeof(struct sockaddr\_in));

next\_hop.sin\_family = AF\_INET;

next\_hop.sin\_port = htons(atoi(next\_port)); *//argv[4]*

inet\_pton(AF\_INET, next\_ip, &(next\_hop.sin\_addr)); *//argv[3]*

*//printf("Sto mandando messaggio : %s\n", buff);*

if (sendto(sockfd, buff, strlen(buff) + 1, 0, (struct sockaddr \*)&next\_hop, sizeof(next\_hop)) < 0){

perror("Impossibile mandare messaggio");

exit(1);

}

}

int main (int argc, char \*\* argv){

if (argc < 7){

fprintf(stderr, "Usage: %s previp prevport nextip nextport localport name", argv[0]);

exit(1);

}

int sockfd;

if ((sockfd = socket(AF\_INET, SOCK\_DGRAM, 0)) < 0){

perror("Impossibile aprire socket");

exit(1);

}

struct sockaddr\_in allowed\_addr;

memset(&allowed\_addr, 0, sizeof(struct sockaddr\_in));

allowed\_addr.sin\_family = AF\_INET;

allowed\_addr.sin\_port = htons(atoi(argv[5]));

allowed\_addr.sin\_addr.s\_addr = INADDR\_ANY;

if (bind(sockfd, (struct sockaddr \*)&allowed\_addr, sizeof(struct sockaddr)) == -1){

perror("Impossibile fare la bind");

exit(1);

}

if (fork() == 0){

char buff[BUFFSIZE];

char destinatario[BUFFSIZE];

char messaggio[BUFFSIZE];

for (;;){

fflush(stdout);

printf("Inserire destinatario\n");

fgets(buff, BUFFSIZE, stdin);

if (buff[strlen(buff) -1] == '\n') buff[strlen(buff) -1] = '\0';

strncpy(destinatario, buff, BUFFSIZE);

printf("Inserire messaggio\n");

fgets(buff, BUFFSIZE, stdin);

if (buff[strlen(buff) -1] == '\n') buff[strlen(buff) -1] = '\0';

strncpy(messaggio, buff, BUFFSIZE);

sprintf(buff, "%s$%s$%s", argv[6], destinatario, messaggio);

send\_to\_next\_hop(sockfd, argv[3], argv[4], buff);

}

}

else {

char buff[BUFFSIZE];

char msg[BUFFSIZE];

struct sockaddr\_in prev\_hop;

int len = sizeof(prev\_hop);

int n;

char ip[256];

char port[256];

char \*mittente, \*destinatario, \*messaggio;

for (;;){

fflush(stdout);

memset(&prev\_hop, 0, sizeof(struct sockaddr\_in));

if ((n = recvfrom(sockfd, buff, BUFFSIZE, 0, (struct sockaddr \*)&prev\_hop, (socklen\_t \*)&len)) < 0){

perror("Impossibile ricevere messaggi");

exit(1);

}

buff[n] = 0;

*//printf("Ricevuto: %s\n", buff);*

inet\_ntop(AF\_INET, &(prev\_hop.sin\_addr), ip, BUFFSIZE);

sprintf(port, "%d", ntohs(prev\_hop.sin\_port));

if ((strcmp(argv[1], ip) != 0) || (strcmp(argv[2], port) != 0)){

fprintf(stderr, "Qualuno cheatta");

continue;

}

mittente = strtok(buff, "$");

destinatario = strtok(NULL, "$");

messaggio = strtok(NULL, "$");

if (strcmp(mittente, argv[6]) == 0){

fprintf(stderr, "Pacchetto tornato indietro senza aver trovato destinatario\n");

}

else if (strcmp(destinatario, argv[6]) != 0){

sprintf(msg, "%s$%s$%s", mittente, destinatario, messaggio);

send\_to\_next\_hop(sockfd, argv[3], argv[4], msg);

}

else {

if (strcmp(messaggio, "ACK") == 0){

printf("Ricevuto ACK da %s\n", mittente);

}

else {

printf("Ricevuto messaggio da %s con destinatario %s, messaggio: %s\n", mittente, destinatario, messaggio);

sprintf(msg, "%s$%s$ACK", destinatario, mittente);

send\_to\_next\_hop(sockfd, argv[3], argv[4], msg);

}

}

}

}

}