UNIVERSITA' DEGLI STUDI DI NAPOLI FEDERICO II

PROGETTO DI LABORATORIO DI SISTEMI OPERATIVI

Relazione

Meo Raffaele N86001891

blank page

Indice

Compilazione ed esecuzione

Per la compilazione del Server e del Client è necessario eseguire lo script ***compile.sh*** dando, da terminale, il seguente comando:

"./compile.sh"

I due script contengono i seguenti codici utili alla compilazione:

"gcc server.c -pthread -o server.out"

"gcc client.c -lpthread -o client.out"

Per quanto concerne invece l'esecuzione, è necessario eseguire:

./server.out

./client.out

NB: *Prima della compilazione ed esecuzione degli script, è consigliato attivare i permessi d'esecuzione attraverso il seguente comando da inviare mediante riga di comando:*

"chmod +x <nome file>"

Comunicazione Client - Server (Socket)

Per la comunicazione client server avviene tramite socket.

- **Client.c**

Creazione del socket:

```
if((c_fd=socket(AF_INET,SOCK_STREAM,0))<0){
        perror("Errore nella creazione del socket");
        return -1;
  }
```

Cerca di connettersi con il server:

```
if(connect(c_fd,(struct sockaddr *)&s_address,sizeof(s_address))<0){
        perror("Errore nella connessione del client");
        return -1;
   }
```

- **Server.c**

Creazione del socket:

```
if((s_fd = socket(AF_INET, SOCK_STREAM, 0))<0){
        perror("Errore nella creazione della socket!");
        closeServerLog();
        return -1;
    }
```

Con la funzione bind(), assegna un indirizzo locare al socket, cioè assegna l'indirizzo s_address al socket s_fd:

```
 if(bind(s_fd,(struct sockaddr*)&s_address, sizeof(s_address))<0){
        perror("bind fallito!");
        closeServerLog();
        return -1;
    }
```

Si mette in ascolto, cioè in attesa di connessioni da parte dei client, il secondo membro indica quante connessioni possono essere accettate:

```
listen(s_fd,10);
```

Se accetta la connessione da parte di un client, il secondo e terzo membro si riferiscono ad un client:

```
c_fd = accept(s_fd, (struct sockaddr*) &c_address, &c_len);
```

Scambio dei messaggi

Lo scambio dei messaggi avviene tramite due funzioni:

- con la write() invio il messaggio (buffer) tramite socket:

```
int write(int socket, char buffer, int dimensione_buffer)
```

- con la read() leggo il messaggio, lo salvo nella variabile buffer:

```
int read(int socket, char buffer, int dimensione_buffer)
```

La write e la read restituiscono un numero di bytes maggiore di 0, in caso il numero di bytes sia minore o uguale a zero, vuol dire che si è verificato un errore, rispettivamente, nella lettura o scrittura del messaggio inviato tramite socket.

Thread e mutex

• Thread

Ad ogni nuova connessione viene creato un thread tramite questa funzione:

```
pthread_create(&tid, NULL, gestisci, (void*)&c_fd);
pthread_detach(tid);
```

Di tipo detach perché non ci interessa il tipo di ritorno.

Quindi, ogni client che si collega, il server fa eseguire la funzione gestisci, passando come parametro (void*)&c_fd.

• Mutex

Ho usato un mutex per evitare la race condition (interferenza tra processi) su risorse condivise quando bisogna scrivere sul file log.txt. (lato server)

Creazione del mutex (globale):

```
pthread_mutex_t mutex;
```

Inizializzata nel main del server in questo modo:

```
if(pthread_mutex_init(&mutex,NULL)!=0){
        printf("Errore nella creazione del semaforo!\n");
        closeServerLog();
        return -1;
    }
```

Per bloccare il mutex finché il semaforo si libera, per evitare che due o più client scrivano in contemporanea sul file.

- Se era sbloccato, il thread chiamante ne prende possesso bloccandolo immediatamente e la funzione ritorna subiti.

- Se, invece, era bloccato da un altro thread, il thread chiamante viene sospeso sino a quando il possessore non lo rilascia:

```
pthread_mutex_lock(&mutex);
```

Per sbloccare il mutex:

```
pthread_mutex_unlock(&mutex);
```

Gestione segnali

• Client.c

```
signal(SIGINT,handler);
signal(SIGPIPE,handler);
signal(SIGHUP,handler);
signal(SIGQUIT,handler);
signal(SIGTSTP,handler);
```

```
void handler(int signo){
   pid_t pid=getpid();

   switch(signo){
      case SIGINT:{
         printf("\n(SIGINT) Disconnessione...\n");
         sleep(1);
         close(soc);
         exit(1);
         break;
      }
      case SIGPIPE:{
         printf("\n(SIGPIPE) Disconnessione...\n");
         sleep(1);
         close(soc);
         exit(1);
         break;
      }
      case SIGHUP:{
         printf("\n(SIGHUP) Disconnessione...\n");
         sleep(1);
         close(soc);
         exit(1);
         break;
      }
      case SIGQUIT:{
         printf("\n(SIGQUIT) Disconnessione...\n");
         sleep(1);
         close(soc);
         exit(1);
      }
```

```
        case SIGTSTP:{
            printf("\n(SIGTSTP) Disconnessione...\n");
            sleep(1);
            close(soc);
            exit(1);
        }
    }
}
```

Vengono gestiti tutto nello stesso modo, cioè vene chiudo il programma e chiusa la connessione con il server (close(c_fd)).

- **Server.c**

```
signal(SIGINT,handler);
signal(SIGPIPE,handler);
signal(SIGHUP,handler);
signal(SIGQUIT,handler);
signal(SIGTSTP,handler);
```

```
void handler(int signo){
    pid_t pid=getpid();
    switch(signo){
        case SIGINT:{
            closeServerLog();
            kill(pid,SIGTERM);
            break;
        }
        case SIGPIPE:{
            pthread_exit(NULL);
            break;
        }
        case SIGHUP:{
            closeServerLog();
            kill(pid,SIGTERM);
            break;
        }
        case SIGQUIT:{
            closeServerLog();
            kill(pid,SIGTERM);
        }
        case SIGTSTP:{
            closeServerLog();
            kill(pid,SIGTERM);
            break;
        }
    }
}
```

Se viene chiuso il terminale da parte di in client, viene gestito dal serve in questo modo:

```
if(read(c_fd,buffer2,N)<=0){
                printf("Errore nella lettura sulla socket!(1)..Utente %d disconnesso!\n", c_fd-4);
                disconnessioneLog(c_fd);
                close(c_fd);
                pthread_exit(0);
        }
```

Se il client chiude il terminale, il server, quando fa la read(…), riceve un numero di bytes minore o uguale a 0 e sa che non ha ricevuto nulla da parte del clients e termina la connessione, elimina il thread e scrive che il client si è disconnesso sul file log.txt.

Gestione file log.txt

Nelle funzioni riguardante la scritta sul file log.txt si trova il seguente codice:

```
FILE *pf;
int ora, minuto, secondi,
   giorno, anno;
char mese[9];
time_t nowtime;
struct tm* dati;

time (&nowtime);
dati=localtime(&nowtime);
ora=dati->tm_hour;
minuto=dati->tm_min;
secondi=dati->tm_sec;
giorno=dati->tm_mday;
getMonth(dati->tm_mon,mese);
anno=dati->tm_year + 1900;
```

Oltre a scrivere quando il server è stato aperto/chiuso e quando un client si è connesso/ disconnesso ho inserito anche la data, e ho utilizzato la libreria <time.h>, in particolare ho utilizzato le seguenti variabili:

- time_t nowtime la quale memorizza l'ora e il giorno corrente del calendario;
- struct tm* dati la quale indica la seguente struttura dati:

```
struct tm {
  int tm_sec;       //secondi da 0 a 59
  int tm_min;       //minuti da 0 a 59
  int tm_hour;      //ore da 0 1 23
  int tm_mday;      //giorni da 1 a 31
  int tm_mon;       //mesi da 0 a 11
  int tm_year;      //il numero degli anni dal 1900
  int tm_wday;      //giorno della settimana da 0 a 6
  int tm_yday;      //giorno dell'anno da 0 a 365
  int tm_isdst;     //ora legale
};
```

- *time*(&nowtime) la quale calcola l'ora corrente con giorno odierno e lo codifica nel formato di nowtime;

- dati=localtime(&nowtime) il cui il valore di nowtime viene espresso sotto forma della struttura dati prima descritta;
- getMonth(dati->tm_mon, mese) è una funzione che trasforma l'intero dati->tm_mon, che indica il mese espresso in intero (da 0 a 11) in stringa nel mese corrispondente.

E poi assegno ogni valore alle variabili: ora, minuto, secondi, giorno, anno (che sono interi) e mese (stringa).

Codice sorgente

- **Client.c**

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <netinet/in.h>
#include <sys/signal.h>
#include <pthread.h>
#include <string.h>

#define NAME "mySocket"
#define N 256
#define PORT 3000

#define clear() printf("\e[H\e[J\e[3J")

int soc;


void handler(int signo){
    pid_t pid=getpid();

    switch(signo){
        case SIGINT:{
            printf("\n(SIGINT) Disconnessione...\n");
            sleep(1);
            close(soc);
            exit(1);
            break;
        }
        case SIGPIPE:{
            printf("\n(SIGPIPE) Disconnessione...\n");
            sleep(1);
            close(soc);
            exit(1);
            break;
        }
        case SIGHUP:{
```

```
            printf("\n(SIGHUP) Disconnessione...\n");
            sleep(1);
            close(soc);
            exit(1);
            break;
        }
        case SIGQUIT:{
            printf("\n(SIGQUIT) Disconnessione...\n");
            sleep(1);
            close(soc);
            exit(1);
        }
        case SIGTSTP:{
            printf("\n(SIGTSTP) Disconnessione...\n");
            sleep(1);
            close(soc);
            exit(1);
     }
    }
}

char scegliOperazione(){
     char choice='\0';
     do{
          printf("Operazione scelta: ");
          scanf("%s",&choice);
          if(choice!='a' && choice!='s' && choice!='m' && choice!='d'){
               printf("Errore, puoi digitare a, s, m oppure d. Riprova!
\n");
               }
     }while(choice!='a' && choice!='s' && choice!='m' && choice!='d');

     return choice;
}

char sceltaFinale(){
     char choice='\0';
     do{
          scanf("%s",&choice);
               if(choice!='c' && choice!='e'){
               printf("Errore, puoi digitare c oppure e. Riprova!\n");
               printf("Clicca (c) per continuare oppure (e) per
disconnetterti: ");
                }
     }while(choice!='c' && choice!='e');

     return choice;

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


     signal(SIGINT,handler);
     signal(SIGPIPE,handler);
     signal(SIGHUP,handler);
     signal(SIGQUIT,handler);
```

```
     signal(SIGTSTP,handler);


     int c_fd,
         ret;
     struct sockaddr_in s_address;
     char buffer[N];

     if((c_fd=socket(AF_INET,SOCK_STREAM,0))<0){
          perror("Errore nella creazione del socket");
          return -1;
     }
     s_address.sin_family=AF_INET;
     s_address.sin_addr.s_addr=htonl(INADDR_ANY);
     s_address.sin_port=htons(PORT);

     if(connect(c_fd,(struct sockaddr *)&s_address,sizeof(s_address))<0)
{
          perror("Errore nella connessione del client");
          exit(1);
     }

     soc=c_fd;

     while(1){
          clear();
          float res=0, num1=0,num2=0;
          char choice='\0',fine='\0';

          if(read(c_fd,buffer,N)<=0){
               printf("Errore nella lettura sulla socket!(Primo valore)
\n");
               sleep(1);
               close(c_fd);
               exit(1);
          }
          printf("%s",buffer);
          scanf("%f", &num1);

          write(c_fd,&num1,sizeof(float));

          bzero(buffer,N);

          if(read(c_fd,buffer,N)<=0){
               printf("Errore nella lettura sulla socket!(Secondo
valore)\nDisconnessione...\n");
               sleep(1);
               close(c_fd);
               exit(1);
          }
          printf("%s",buffer);
          scanf("%f", &num2);

          write(c_fd,&num2,sizeof(float));

          bzero(buffer,N);
```

```
          if(read(c_fd,buffer,N)<=0){
               printf("Errore nella lettura sulla socket!(Scelta
operazione)\nDisconnessione...\n");
               sleep(1);
               close(c_fd);
               exit(1);
          }
          printf("%s",buffer);

          choice=scegliOperazione();
          write(c_fd,&choice,1);

          bzero(buffer,N);

          if(read(c_fd,&res,sizeof(float))<=0){
               printf("Errore nella lettura sulla socket!(Risultato)
\nDisconnessione...\n");
               sleep(1);
               close(c_fd);
               exit(1);
          }
          printf("\n\nIl risultato è: %.3f\n\n",res);


          write(c_fd,"1",1);
          bzero(buffer,N);
          if(read(c_fd,buffer,N)<=0){
               printf("Errore nella lettura sulla socket!
(1)\nDisconnessione...\n");
               sleep(1);
               close(c_fd);
               exit(1);
          }
          printf("%s",buffer);

          fine=sceltaFinale();
          write(c_fd,&fine,1);

          bzero(buffer,N);
          switch(fine){
               case'e':{
                    printf("Disconnessione...\n");
                    sleep(1);
                    close(c_fd);
                    exit(1);
               }
               case 'c':{
                    bzero(buffer,N);
                    break;
               }
          }
     }
     return 0;
}
```

- **Server.c**

```c
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <netinet/in.h>
#include <sys/signal.h>
#include <pthread.h>
#include <time.h>
#include <string.h>

#define NAME "mySocket"
#define N 256
#define PORT 3000


#define clear() printf("\e[H\e[J\e[3J")

pthread_mutex_t mutex;

void getMonth(int numMese, char* mese){
    switch(numMese){
        case 0:{
            strcpy(mese,"Gennaio");
            break;
        }
        case 1:{
            strcpy(mese,"Febbraio");
            break;
        }
        case 2:{
            strcpy(mese,"Marzo");
            break;
        }
        case 3:{
            strcpy(mese,"Aprile");
            break;
        }
        case 4:{
            strcpy(mese,"Maggio");
            break;
        }
        case 5:{
            strcpy(mese,"Giugno");
            break;
        }
        case 6:{
            strcpy(mese,"Luglio");
            break;
        }
        case 7:{
            strcpy(mese,"Agosto");
            break;
```

```
        }
        case 8:{
            strcpy(mese,"Settembre");
            break;
        }
        case 9:{
            strcpy(mese,"Ottobre");
            break;
        }
        case 10:{
            strcpy(mese,"Novembre");
            break;
        }
        case 11:{
            strcpy(mese,"Dicembre");

        }
    }
}

void openServerLog(){

    FILE *pf;
    int ora, minuto, secondi,
        giorno, anno;
    char mese[9];
    time_t nowtime;
    struct tm* dati;

    time (&nowtime);
    dati=localtime(&nowtime);
    ora=dati->tm_hour;
    minuto=dati->tm_min;
    secondi=dati->tm_sec;

    giorno=dati->tm_mday;
    getMonth(dati->tm_mon,mese);
    anno=dati->tm_year + 1900;
    pthread_mutex_lock(&mutex);
    pf=fopen("log.txt","a+");
    fprintf(pf,"\n\nSERVER APERTO ------------------------- %d : %d :
%d ----- %d - %s - %d\n",ora,minuto,secondi,giorno,mese,anno);
    fclose(pf);
    pthread_mutex_unlock(&mutex);
}

void closeServerLog(){

    FILE *pf;
    int ora, minuto, secondi,
        giorno, anno;
    char mese[9];
    time_t nowtime;
    struct tm* dati;

    time (&nowtime);
```

```
    dati=localtime(&nowtime);
    ora=dati->tm_hour;
    minuto=dati->tm_min;
    secondi=dati->tm_sec;

    giorno=dati->tm_mday;
    getMonth(dati->tm_mon,mese);
    anno=dati->tm_year + 1900;
    pthread_mutex_lock(&mutex);
    pf=fopen("log.txt","a+");
    fprintf(pf,"SERVER CHIUSO ------------------------- %d : %d : %d
----- %d - %s - %d\n\n\n",ora,minuto,secondi,giorno,mese,anno);
    fclose(pf);
    pthread_mutex_unlock(&mutex);

}

void connectionLog(int c_fd){


    FILE *pf;
    int ora, minuto, secondi,
        giorno, anno;
    char mese[9];
    time_t nowtime;
    struct tm* dati;

    time (&nowtime);
    dati=localtime(&nowtime);
    ora=dati->tm_hour;
    minuto=dati->tm_min;
    secondi=dati->tm_sec;

    giorno=dati->tm_mday;
    getMonth(dati->tm_mon,mese);
    anno=dati->tm_year + 1900;
    pthread_mutex_lock(&mutex);
    pf=fopen("log.txt","a+");
    fprintf(pf,"Utente %d connesso\t\t%d : %d : %d\t\t%d - %s -
%d\n",c_fd-4,ora,minuto,secondi,giorno,mese,anno);
    fclose(pf);
    pthread_mutex_unlock(&mutex);

}

void disconnessioneLog(int c_fd){

    FILE *pf;
    int ora, minuto, secondi,
        giorno, anno;
    char mese[9];
    time_t nowtime;
    struct tm* dati;

    time (&nowtime);
    dati=localtime(&nowtime);
```

```
    ora=dati->tm_hour;
    minuto=dati->tm_min;
    secondi=dati->tm_sec;

    giorno=dati->tm_mday;
    getMonth(dati->tm_mon,mese);
    anno=dati->tm_year + 1900;
    pthread_mutex_lock(&mutex);
    pf=fopen("log.txt","a+");
    fprintf(pf,"Utente %d disconnesso\t\t%d : %d : %d\t\t%d - %s -
%d\n",c_fd-4,ora,minuto,secondi,giorno,mese,anno);
    fclose(pf);
    pthread_mutex_unlock(&mutex);
}

void getOperation(char choice, char* oper){
     switch(choice){
          case 'a':{
              strcpy(oper,"Addizione");
              break;
          }
          case 's':{
              strcpy(oper,"Sottrazione");
              break;
          }
          case 'm':{
              strcpy(oper,"Moltiplicazione");
              break;
          }
          case 'd':{
              strcpy(oper,"Divisione");
              break;
          }
     }
}

void operazioneLog(int c_fd,float num1,float num2,char choice,float res)
{
    FILE *pf;
    int ora, minuto, secondi,
        giorno, anno;
    char mese[9],oper[12];

    time_t nowtime;
    struct tm* dati;

    time (&nowtime);
    dati=localtime(&nowtime);
    ora=dati->tm_hour;
    minuto=dati->tm_min;
    secondi=dati->tm_sec;

    giorno=dati->tm_mday;
    getMonth(dati->tm_mon,mese);
    anno=dati->tm_year + 1900;
```

```
    getOperation(choice,oper);

     pthread_mutex_lock(&mutex);
    pf=fopen("log.txt","a+");
    fprintf(pf,"Utente %d ha scelto come elementi %.2f e %.2f e come
operazione la %s, risultato %.2f\t\t%d : %d : %d\t\t%d - %s -
%d\n",c_fd-4,num1,num2,oper,res,ora,minuto,secondi,giorno,mese,anno);

     fclose(pf);
     pthread_mutex_unlock(&mutex);

}

void handler(int signo){
    pid_t pid=getpid();

    switch(signo){
        case SIGINT:{
            closeServerLog();
         printf("Disconnessione...\n");
         sleep(1);
            kill(pid,SIGTERM);
            break;
        }
        case SIGPIPE:{
            pthread_exit(NULL);
            break;
        }
        case SIGHUP:{
            closeServerLog();
         printf("Disconnessione...\n");
         sleep(1);
            kill(pid,SIGTERM);
            break;
        }
        case SIGQUIT:{
            closeServerLog();
         printf("Disconnessione...\n");
         sleep(1);
            kill(pid,SIGTERM);
        }
     case SIGTSTP:{
         closeServerLog();
         printf("Disconnessione...\n");
         sleep(1);
            kill(pid,SIGTERM);
            break;
     }

    }
}

void* gestisci(void* arg){
     int c_fd=*(int *)arg;
```

```
     while(1){
          char fine='\0';
          float res=0, num1=0,num2=0;
          char choice='\0';
               //lettura del messaggio scritto dal client
          if(write(c_fd, "Inserisci il primo valore:
",strlen("Inserisci il primo valore: "))<=0){
               printf("Errore nella scrittura sulla socket!(Primo
valore)\n");
                    disconnessioneLog(c_fd);
                    close(c_fd);
                    pthread_exit(0);
          }
          if(read(c_fd, &num1, sizeof(float))<=0){
               printf("Errore nella lettura sulla socket!(Primo
valore)..Utente %d disconnesso!\n", c_fd-4);
               disconnessioneLog(c_fd);
               close(c_fd);
               pthread_exit(0);
          }


          if(write(c_fd, "Inserisci il secondo valore:
",strlen("Inserisci il secondo valore: "))<=0){
               printf("Errore nella scrittura sulla socket!(Secondo
Valore)\n");
               disconnessioneLog(c_fd);
               close(c_fd);
               pthread_exit(0);
          }

          if(read(c_fd, &num2, sizeof(float))<=0){
               printf("Errore nella lettura sulla socket!(Secondo
Valore)..Utente %d disconnesso!\n", c_fd-4);
               disconnessioneLog(c_fd);
               close(c_fd);
               pthread_exit(0);
          }


          if(write(c_fd,"Che operazione vuoi effettuare?\n\ta.
\tAddizione\n\ts.\tSottrazione\n\tm.\tMoltiplicazione\n\td.
\tDivisione\n",strlen("Che operazione vuoi effettuare?\n\ta.
\tAddizione\n\ts.\tSottrazione\n\tm.
\tMoltiplicazione\n\td\t.Divisione\n"))<=0){
               printf("Errore nella scrittura del socket!(Scelta
operazione)\n");
               disconnessioneLog(c_fd);
               close(c_fd);
               pthread_exit(0);
          }

          if(read(c_fd,&choice,N)<=0){
               printf("Errore nella lettura sulla socket!(Scelta
operazione)..Utente %d disconnesso!\n", c_fd-4);
               disconnessioneLog(c_fd);
```

```
                close(c_fd);
                pthread_exit(0);
          }


          switch(choice){
                case 'a':{
                      res=num1+num2;
                      break;
                }
                case 's':{
                      res=num1-num2;
                      break;
                }
                case 'm':{
                      res=num1*num2;
                      break;
                }
                case 'd':{
                      res=num1/num2;
                      break;
                }
          }
          if(write(c_fd,&res,sizeof(float))<=0){
                printf("Errore nella scrittura del socket!(Risultato)
\n");
                disconnessioneLog(c_fd);
                close(c_fd);
                pthread_exit(0);
          }
          operazioneLog(c_fd,num1,num2,choice,res);

          char buffer2[N];
          if(read(c_fd,buffer2,N)<=0){
                printf("Errore nella lettura sulla socket!(1)..Utente %d
disconnesso!\n", c_fd-4);
                disconnessioneLog(c_fd);
                close(c_fd);
                pthread_exit(0);
          }


          if(write(c_fd,"Clicca (c) per continuare oppure (e) per
disconnetterti: ",strlen("Clicca (c) per continuare oppure (e) per
disconnetterti: "))<=0){
                printf("Errore nella scrittura del socket!(Fine)\n");
                disconnessioneLog(c_fd);
                close(c_fd);
                pthread_exit(0);
          }

          if(read(c_fd, &fine, 1)<=0){
                printf("Errore nella lettura sulla socket!(Fine)..Utente
%d disconnesso!\n", c_fd-4);
                disconnessioneLog(c_fd);
                close(c_fd);
```

```
                pthread_exit(0);
            }

            switch(fine){
                case'e':{
                    printf("Utente %d disconnesso\n",c_fd-4);
                    disconnessioneLog(c_fd);
                    close(c_fd);
                    pthread_exit(0);

                    break;
                }
                case 'c':
                    break;
            }

    }
}


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

    signal(SIGINT,handler);
    signal(SIGPIPE,handler);
    signal(SIGHUP,handler);
    signal(SIGQUIT,handler);
    signal(SIGTSTP,handler);

    int s_fd, c_fd, c_len, s_len;
    pthread_t tid;

       struct sockaddr_in s_address, c_address;
    //ASSEGNAZIONE DI UN INDIRIZZO ALLA SOCKET

       s_address.sin_family= AF_INET;
       s_address.sin_addr.s_addr = htonl(INADDR_ANY);
       s_address.sin_port = htons(PORT);

    s_fd = socket(AF_INET, SOCK_STREAM, 0);

    if((s_fd = socket(AF_INET, SOCK_STREAM, 0))<0){
        perror("Errore nella creazione della socket!");
        closeServerLog();
        return -1;
    }


     if(bind(s_fd,(struct sockaddr*)&s_address, sizeof(s_address))<0){
        perror("bind fallito!");
        closeServerLog();
        return -1;
    }

    if(pthread_mutex_init(&mutex,NULL)!=0){
                printf("Errore nella creazione del semaforo!\n");
                closeServerLog();
```

```
                return -1;
        }
    clear();
    printf("Server in esecuziuone\n");

    openServerLog();

    //SETTAGGIO SOCKET IN ASCOLTO
    listen(s_fd,10);
    c_len=sizeof(struct sockaddr_in);
    while(1){
        c_fd = accept(s_fd, (struct sockaddr*) &c_address, &c_len);
        connectionLog(c_fd);
        printf("Utente %d connesso\n",c_fd-4);


        pthread_create(&tid, NULL, gestisci, (void*)&c_fd);
        pthread_detach(tid);


    }
    return 0;
}
```