UNIVERSITÀ DEGLI STUDI DI NAPOLI
FEDERICO II

DIPARTIMENTO DI INGEGNERIA ELETTRICA E TECNOLOGIE DELL'INFORMAZIONE

CORSO DI LAUREA IN INFORMATICA

Anno Accademico 2020/2021

Progettazione e sviluppo di un sistema Client-Server

"Gara di aritmetica"

Insegnamento

LABORATORIO DI SISTEMI OPERATIVI

Docente
Prof. FAELLA Marco

ID Gruppo: 6

Autori
Marco AGIZZA Matr. N86/2179
ma.agizza@studenti.unina.it

Luca FRANZONE Matr. N86/2158
l.franzone@studenti.unina.it

- Indice -

Indice cliccabile

1. Introduzione

1.1 Descrizione

Il sistema Client-Server commissionato prevede la possibilità per i Client di sfidarsi in una gara di aritmetica. In particolare, al termine della fase progettuale ed implementativa il prodotto software offre la possibilità al Client che ne fa uso di:

- Connettersi al Server secondo le modalità che verranno successivamente dettagliate*;*
- Utilizzare un identificativo personale espresso sottoforma di username, il quale verrà richiesto nella fase iniziale;
- Giocare con altri Client contemporaneamente una partita, rispondendo a domande di aritmetica proposte dal sistema;
- Visualizzare costantemente a schermo la classifica aggiornata;
- Scegliere, al termine della partita, di ricominciare o uscire dal gioco;

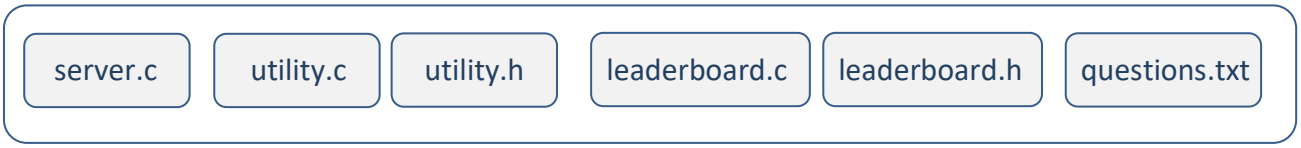
In particolare, i Client, dopo aver effettuato con successo la connessione al Server, inseriscono uno username identificativo e avviano la partita, la quale inizia con un countdown di 30 secondi (lobby di attesa) dal momento in cui vi sono almeno due Client connessi. I giocatori hanno un minuto di tempo per rispondere a ciascuna domanda proposta. Le domande riguardano semplici e banali operazioni aritmetiche (per motivi didattici le domande sono standard e riguardano banali calcoli con un utilizzo limitato alle quattro operazioni fondamentali: +, - , / , *).
Al termine dei 60 secondi previsti come tempo limite per la risposta alla domanda proposta, l'utente viene informato sull'esito, positivo o negativo, della stessa e viene mostrata la classifica aggiornata con la nuova domanda. Il posto di un giocatore nella classifica dipende dal *tempo medio* impiegato da quel giocatore per rispondere alle domande.
Nuovi giocatori (cioè nuovi Client) possono collegarsi in qualsiasi momento e unirsi alla partita, dopo aver atteso che ai Client già in gioco sia proposta una nuova domanda (così da garantire ad ogni Client lo stesso tempo utile per rispondere).
I Client giocheranno sempre e comunque in maniera sincrona, pertanto verranno loro poste le stesse domande nello stesso momento.
Il *tempo medio* è calcolato sulla base del valore del tempo totale impiegato per rispondere alle domande diviso per il numero di domande che sono state proposte al giocatore stesso. Le risposte corrette considerano il tempo effettivo impiegato per rispondere al quesito, mentre le risposte errate, o mancate, prevedono un malus temporale di 85 secondi per assicurare agli utenti che, hanno risposto in maniera corretta, uno score maggiore rispetto a coloro i quali hanno risposto in maniera errata o non hanno risposto affatto. Al termine del gioco i Client possono decidere di ricominciare a giocare o abbandonare la sessione di gioco.

La descrizione sopracitata vuole rappresentare una panoramica generale del prodotto, pertanto per i dettagli relativi all'uso, al funzionamento e all'esempio di esecuzione completo si rimanda il lettore al paragrafo successivo ***1.2 Guida all'uso****.*

1.2 Guida all'uso

1.2.1 Guida all'uso: Server

All'interno della cartella "Server" sono presenti i seguenti file:



1.2.1.1 Struttura del file questions.txt

Al fine di ottenere il corretto funzionamento dell'applicativo, viene utilizzato un file *.txt* all'interno del quale sono memorizzati tutti i quesiti che verranno proposti ai Client e, rispettivamente, tutte le risposte agli stessi. In particolare, il file dovrà necessariamente rispettare alcuni standard di formattazione, espressi graficamente di seguito:

Nome del file: *questions.txt*

```
10
Quanto fa 3 * 4 + 3 * 3 ?
1) 2
2) 20
3) 21
4) 11
~3
Quanto fa 21 + 12 - 3 ?
1) 20
2) 25
3) 29
4) 30
~4
Quanto fa 16 : 4 * 2 + 1?
1) 9
2) 8
3) 2
4) 12
~1
Quanto fa 152 - 150 + 0 * 5?
1) 7
2) 150
3) 2
4) 152
~3
Quanto 0 * 2 + 0 * 3 + 0 * 10 ?
1) 5
2) 15
3) 10
4) 0
```

[Screenshot 1] file "questions.txt"

Struttura del file:

<numero totale di domande presenti nel file>
<domanda 1>
1) <opzione 1>
2) <opzione 2>
3) <opzione 3>
4) <opzione 4>
~<numero risposta corretta alla domanda 1>
<domanda 2>
1) <opzione 1>
2) <opzione 2>
3) <opzione 3>
4) <opzione 4>
~<numero risposta corretta alla domanda 2>
...

NOTE Il file riporta in alto il numero totale di domande presenti al fine di utilizzare tale valore come indice di controllo interno per la somministrazione delle domande stesse.
La risposta corretta ad una certa domanda deve essere preceduta da una tilde (~) e fa riferimento al numero presente accanto ad una delle opzioni di risposta proposte.

1.2.1.2 Compilazione ed esecuzione del Server

La **compilazione** del Server avviene tramite il compilatore standard del C "`gcc`", e necessariamente con l'inserimento del seguente comando sulla command-line:

```
gcc -pthread -o server server.c leaderboard.c utility.c
```

A seguito della compilazione, si potrà passare alla fase di **esecuzione** del Server tramite il comando:

```
./server <Numero di porta>
```

dove il parametro `<Numero di porta>` rappresenta una porta, ossia un intero a 16 bit che, in particolare, ammette un valore compreso tra 5000 e 32768. Un esempio di esecuzione può essere il seguente:

```
./server 20000
```

in cui il Server viene eseguito e messo in attesa di connessioni da parte dei Client sulla porta 20000. Errori possibili riguardanti il comando di esecuzione del Server e resi noti all'utente possono essere i seguenti:

Non viene specificato il numero di porta.	`Usage: ./server <port>`
Errore di connessione alla porta (permesso negato), utilizzare una porta compresa tra 5000 e 32768.	`Socket binding error: Permission denied`
Errore di connessione alla porta (la porta inserita è già in utilizzo).	`Socket binding error: Address already in use`

1.2.2 Guida all'uso: Client

All'interno della cartella "Client" sono presenti i seguenti file:

1.2.1.3 Compilazione ed esecuzione del Client

La **compilazione** del Client può avvenire tramite il compilatore standard del C "`gcc`", e necessariamente con l'inserimento del seguente comando sulla command-line:

```
gcc -o client client.c utility.c
```

A seguito della compilazione, si potrà passare alla fase di **esecuzione** del Client tramite il comando:

```
./client 127.0.0.1 <Numero di porta>
```

dove il primo parametro, ossia "`127.0.0.1`", rappresenta, nelle reti TCP/IP IPv4, l'indirizzo IP di localhost e il secondo parametro `<Numero di porta>` rappresenta la porta sulla quale il Server è stato messo in attesa.
Un esempio di esecuzione può essere il seguente:

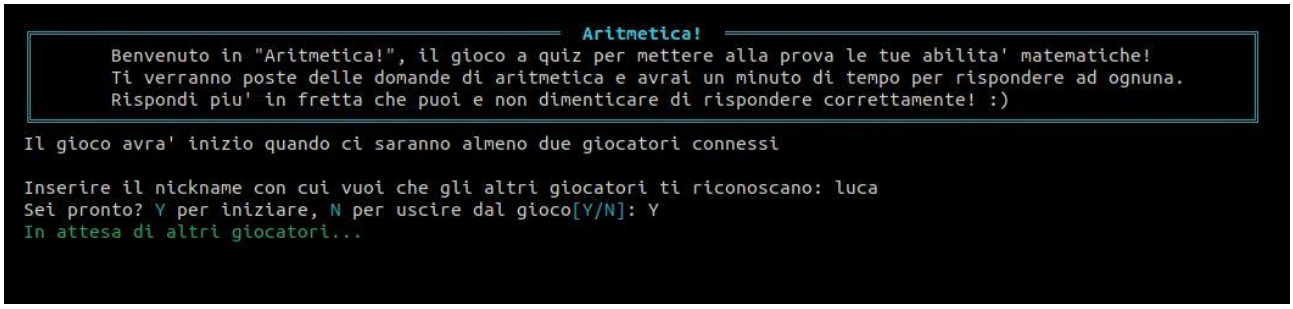
```
./client 127.0.0.1 20000
```

in cui il Client si connette al Server tramite la porta 20000 e l'indirizzo IP di localhost.
Errori possibili riguardanti il comando di esecuzione del Client è resi noti all'utente possono essere i seguenti:

Non viene specificato il numero di porta o l'indirizzo ip.	`error, usage: ./client <ip_addr> <port>`
Errore di connessione alla porta. Utilizzare una porta valida, ossia la porta sulla quale il Server si è messo in ascolto.	**`error`**`, socket connection error: : Connection refused`
Errore di connessione. La rete non è raggiungibile, controllare che i parametri immessi siano corretti e validi.	**`error`**`, socket connection error: : Network is unreachable`

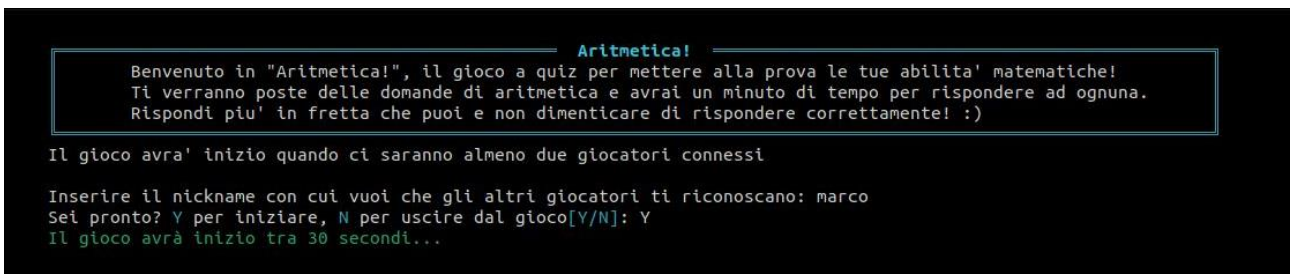
1.2.3 Guida all'uso: Esempio di esecuzione

Dopo aver opportunamente compilato ed eseguito il Server e il/i Client, la schermata iniziale mostrata lato Client è la seguente:



[Screenshot 2] Menù principale – Inserimento nickname e inizio partita (Client)

Al centro, in alto, viene mostrato un messaggio iniziale di benvenuto e di introduzione. In basso un messaggio che avvisa l'utente sulla condizione necessaria per avviare il gioco, ossia la presenza di almeno due giocatori connessi. Viene quindi chiesto al Client di inserire un nickname identificativo per il gioco e viene chiesta, infine, la conferma per iniziare a giocare. Il Client, qualora fosse il solo a giocare in quel momento, resterà in attesa di un secondo giocatore.

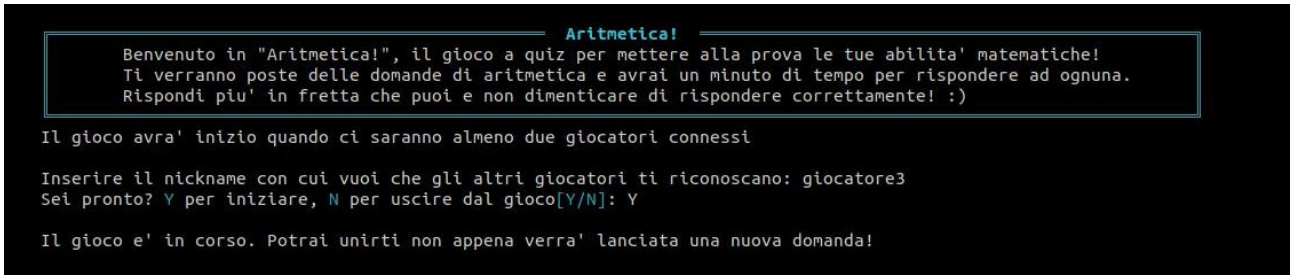


[Screenshot 3] Countdown di attesa per inizio gioco (secondo Client connesso)

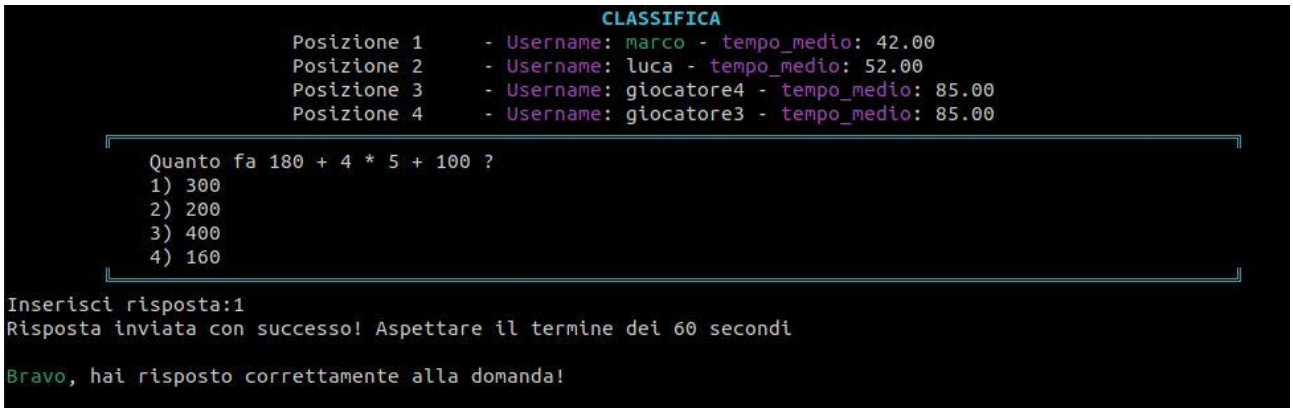
Quando almeno due Client sono connessi e pronti a giocare, viene avviato un countdown di 30 secondi che anticipa la somministrazione delle domande ai giocatori. Esse vengono stampate a schermo e viene richiesto agli utenti di inserire il numero corrispondente alla risposta corretta:

[Screenshot 4] Modalità di somministrazione domanda – Inserimento risposta

In alto viene sempre mostrata la classifica con tutti i Client presenti in gioco aggiornata. Ogni domanda prevede l'attesa dei 60 secondi previsti come tempo limite per rispondere alla domanda stessa e, al termine dei quali, verrà comunicato l'esito della risposta data e processata una nuova domanda per tutti i giocatori, anche per coloro che si sono aggiunti dopo, mentre la partita era già in corso, e sono rimasti in attesa.



[Screenshot 5] Attesa per l'ingresso di nuovi Client



[Screenshot 6] Esito risposta corretta

Possibili esiti alle risposte date
Bravo, hai risposto correttamente alla domanda!
Peccato, la risposta è errata!
Peccato, il tempo a disposizione per rispondere alla domanda è scaduto!

```
Complimenti, hai vinto!

Sei interessato a rigiocare?[Y/N]:
```

[Screenshot 7] Esito gara - Fine partita

Terminato il quiz, viene comunicato ad ogni Client l'esito finale del gioco, al fine di informarlo sull'eventuale vittoria o sconfitta. Viene chiesto, quindi, al giocatore se desidera ricominciare una nuova partita o meno; tale funzionalità permette di resettare la classifica di gioco e avviare direttamente una nuova partita.

```
Server in attesa di connessioni...

Un giocatore si e' connesso;
L'utente luca è registrato al gioco
Un giocatore si e' connesso;
L'utente marco è registrato al gioco
L'utente luca è pronto a giocare!
L'utente marco è pronto a giocare!
Inizio conto alla rovescia per l'avvio della partita

Partita avviata!
Ho ricevuto la risposta 3 dopo 2 secondi
Ho ricevuto la risposta 3 dopo 4 secondi
Ho ricevuto la risposta 4 dopo 8 secondi
Ho ricevuto la risposta 4 dopo 13 secondi
Ho ricevuto la risposta 1 dopo 12 secondi
Ho ricevuto la risposta 1 dopo 14 secondi
Ho ricevuto la risposta 3 dopo 13 secondi
Ho ricevuto la risposta 3 dopo 15 secondi
Ho ricevuto la risposta 4 dopo 2 secondi
Ho ricevuto la risposta 4 dopo 4 secondi
Ho ricevuto la risposta 2 dopo 9 secondi
Ho ricevuto la risposta 2 dopo 13 secondi
Ho ricevuto la risposta 4 dopo 2 secondi
Ho ricevuto la risposta 1 dopo 12 secondi
Ho ricevuto la risposta 2 dopo 19 secondi
Ho ricevuto la risposta 3 dopo 21 secondi
Ho ricevuto la risposta 3 dopo 11 secondi
Ho ricevuto la risposta 4 dopo 17 secondi
Ho ricevuto la risposta 3 dopo 21 secondi
Ho ricevuto la risposta 1 dopo 23 secondi
Partita terminata!
L'utente luca abbandona il gioco
Rimuovo luca dal gioco
L'utente marco abbandona il gioco
Rimuovo marco dal gioco
^C
```

Per quanto riguarda i messaggi **lato Server**, i più comuni sono i seguenti:

- Server in attesa;
- Nuovo giocatore connesso;
- Registrazione al gioco;
- Avviso countdown di inizio;
- Partita avviata;
- Opzione di risposta ricevuta con numero di secondi impiegati dal giocatore;
- Partita terminata;
- Avviso di abbandono gioco e rimozione giocatore;
- Inizio nuova partita;
- Chiusura in corso;
- Altri…

[Screenshot 8] Messaggi lato Server

2. Protocollo di comunicazione

Registrazione username di gioco: Il Server, quando viene eseguito, si mette in ascolto e, ad ogni connessione (`accept`), crea un thread ad hoc per offrire il servizio in modo che più client possano essere gestiti in maniera concorrente. Pertanto, a seguito dell'esecuzione del Client e dell'immissione del nickname, questo viene inviato al Server che ne controllerà la validità e comunicherà l'eventuale corretta (o errata) registrazione. In particolare, il requisito, affinché un nickname risulti valido, riguarda la lunghezza della stringa che non può essere inferiore a tre e superiore a venticinque caratteri e non deve essere già utilizzato da un altro giocatore registrato. La registrazione di un nuovo giocatore corrisponde anche all'immissione dello stesso nella struttura leaderboard.

```
// Registrazione dell'utente
do{
    if(receiveInt(th_params->client_sd, &username_len)==-1){
        perror("Receive username length");
        pthread_exit(NULL);
    }
    if(receiveString(th_params->client_sd, &client_username, username_len)<0){
        perror("Receive username");
        pthread_exit(NULL);
    }
    strcpy(th_params->client_username, client_username);

    added_player=addUserToLeaderboard(&players_leaderboard, th_params->client_username);
    if(sendInt(th_params->client_sd, added_player)<0){
        perror("Confirm username accepted");
        pthread_exit(NULL);
    }
}while(added_player==0);
```

```
do{
    takeSaneString(nickname, 25);
    nickname_len=strlen(nickname);
    if(nickname_len<3){
        write(STDOUT_FILENO, YELLOW_COLOR"\tattenzione"WHITE_COLOR", inser
        for(int i=0;i<nickname_len;i++)
            nickname[i]='\0';
    }else{
        if(sendInt(sd, nickname_len)<0){
            perror("\033[0;31merror\033[0;37m, send username length");
            exit(1);
        }
        if(sendString(sd, nickname, nickname_len)<0){
            perror("\033[0;31merror\033[0;37m, send username");
            exit(1);
        }
        username_registered=0;
        if(receiveInt(sd, &username_registered)<0){
            perror("\033[0;31merror\033[0;37m, server communication");
            exit(1);
        }
        if(username_registered==0){
            write(STDOUT_FILENO, YELLOW_COLOR"\tattenzione"WHITE_COLOR", i
        }
    }
}while(nickname_len<3 || username_registered==0);
```

[Screenshot 9-10] Server-Client / Registrazione username utente

Inserimento in gioco: Un ulteriore loop si verifica quando, dopo aver inserito lo username, viene chiesto al Client la conferma dell'avvio del gioco tramite le scelte [Y/N]. L'immissione della "Y" fa entrare in uno stato di "*Pronto*" il giocatore. Tale stato, tuttavia, non dà immediatamente inizio al gioco poiché:

- se il gioco è già iniziato e i giocatori si trovano a rispondere all'ultima domanda, il nuovo Client non può entrare in partita;
- se il countdown di inizio partita è in corso, il nuovo Client entra nella lista dei giocatori in lobby di attesa;
- se la partita è già iniziata, il nuovo Client entra, momentaneamente, nella lista dei giocatori in lobby di attesa e verrà inserito nella partita in seguito alla somministrazione della successiva domanda ai Client;

Condition variable: Vengono altresì utilizzate le condition variable per sincronizzare i thread.

In particolare, la condition variable `response_player_condition`, è utilizzata per sincronizzare i thread nell'attesa dello scadere dei 60 secondi forniti ad ogni Client per rispondere alla domanda proposta: ogni thread eseguirà una wait sulla condition variable in questione fintantoché il campo "`is_wating`" (dell'oggetto `th_params`, di tipo `ThreadParameters*`) risulta settato ad 1; l'invio al processo di server.c del segnale SIGALARM provocherà l'azione del corrispettivo handler che setterà il campo `is_waiting` di ogni peer thread a 0, consentendone il proseguimento dell'esecuzione.
Altre condition variable utilizzate sono:

`insert_in_game_condition`	Per sincronizzare il thread che gestisce un Client che si vuole unire al gioco quando la partita è già in corso con gli altri thread.
`start_game_condition`	Per consentire l'avvio della partita con un minimo di 2 giocatori.
`countdown_ended_condition`	Per consentire l'avvio della partita soltanto in seguito ad un countdown.
`all_scores_are_updated`	Per sincronizzare l'invio della classifica così che ogni Client possa visualizzarne la stessa versione.

Gestione temporale risposte:

(SERVER) Il tempo per rispondere alle domande è dato da una sveglia impostata ad ogni proposta di domanda all'interno del Server. Tale `alarm` è impostato su 60 secondi per la prima domanda e 60 + 2 secondi nel caso delle domande successive alla prima; tale aggiunta è stata applicata tenendo conto dei 2 secondi in cui il Client visualizza l'esito della risposta data.

(CLIENT) Il tempo per rispondere ad una certa domanda lato Client, è gestito tramite un timeout impostato all'immissione da tastiera della risposta, attraverso l'utilizzo della funzione `select`. Pertanto, se la risposta non viene fornita dal Client entro i 60 secondi, verrà considerata come risposta il valore di default `received_answer = -1` (che corrisponde ad errore).

Per i dettagli implementativi della funzione `int takeAnswer(int type)` si rimanda al capitolo **3.3 Implementazione Client**.

3. Dettagli di implementazione

Le strutture dati utilizzate sono le seguenti:

Player

```
typedef struct Player {
    char username[25];
    int num_answ,
        spent_time,
        is_ready;
    struct Player *next;
} Player;
```

Leaderboard

```
typedef struct Leaderboard {
    Player *head;
    int size;
    int num_ready;
    pthread_mutex_t *mutex;
} Leaderboard;
```

ThreadParameters

```
typedef struct ThreadParameters{
  char client_username[25];
  int tid,
      client_sd,
      is_waiting,
      is_asking;
}ThreadParameters;
```

NumberOfPlayers

```
typedef struct NumberOfPlayers{
        int num_players,
        updated_scores;
    pthread_mutex_t* mutex;
    pthread_cond_t* all_score_are_updated;
}NumberOfPlayers;
```

3.1 Gestione segnali

3.1.1 Gestione segnali: Client

Per quanto riguarda l'uso dei segnali, quelli gestiti (lato Client) dalla funzione sighandler, sono i seguenti.

```
signal(SIGINT, sighandler);
signal(SIGQUIT, sighandler);
signal(SIGABRT, sighandler);
signal(SIGTSTP, sighandler);
```

[Screenshot 11] Client – signal

```
void sighandler(int sig_num){
    write(STDOUT_FILENO, "\n\nE' stata richiesta un'uscita forzata\n", 39);
    if(connection_var!=-1)
        close(connection_var);
    sleep(1);
    exit(0);
}
```

[Screenshot 12] Client – funzione sighandler

3.1.2 Gestione segnali: Server

Per quanto riguarda l'uso dei segnali lato Server, la maggior parte di essi sono ignorati e gestiti direttamente dal Client, ad eccezione di SIGALRM e SIGINT gestiti da due funzioni (handler), ossia *sigAlarmHandler* e *sigIntHandler*.

```
signal(SIGALRM, sigAlarmHandler);
signal(SIGINT, sigIntHandler);
signal(SIGQUIT,SIG_IGN);
signal(SIGTERM,SIG_IGN);
signal(SIGABRT,SIG_IGN);
signal(SIGTSTP,SIG_IGN);
```

[Screenshot 13] Server – signal

```
void sigAlarmHandler(int received_signal){
    if(received_signal == SIGALRM) {
        wakeThreadsUp();
        pthread_cond_broadcast(&response_player_condition);
        pthread_cond_broadcast(&insert_in_game_condition);
    }
}
```

[Screenshot 14] Server – funzione sigAlarmHandler

```
void sigIntHandler(int received_signal){
    if(received_signal == SIGINT){
        write(STDOUT_FILENO, "\n\nChiusura in corso...", 22);
        pthread_mutex_lock(&mutex_sock_d);
        if(sock_desc!=-1)
            close(sock_desc);
        pthread_mutex_unlock(&mutex_sock_d);
        sleep(1);
        write(STDOUT_FILENO, "\n", 1);
        exit(0);
    }
}
```

[Screenshot 15] Server – funzione sigIntHandler

3.2 Implementazione Server

Funzione int sendString(int sd, char* string, int str_length)

```c
int sendString(int sd, char* string, int str_length) {
    int bytes_left = str_length;
    int bytes_written;

    while(bytes_left > 0){
        if ((bytes_written = write(sd, string, bytes_left)) < 0)
            return -1;
        string += bytes_written;
        bytes_left -= bytes_written;
    }
    return 0;
}
```

[Screenshot 16] Server (utility.c) – funzione sendString

Questa funzione è particolarmente importante poiché consente la scrittura di una stringa sul socket. In particolare, viene utilizzata per l'invio di tutte le domande al Client. Essa prende in ingresso il file descriptor del socket su cui scrivere, la stringa che si intende scrivere sul socket e la lunghezza della stringa. Restituisce 0 in caso di successo, -1 altrimenti.

Funzione int setOffset(int fd, int file_dim, int num, int *quiz_len)

```c
int setOffset(int fd, int file_dim, int num, int *quiz_len){
    int file_offset=lseek(fd, 0, SEEK_CUR),
        read_chars = 0,
        nbytes,
        curr_quest=1,
        num_quest,
        read_ints=0;
    char buff[10],
         str_num_quest[30];

    do{
        nbytes=read(fd, &str_num_quest[read_ints], 1);
        if(nbytes == -1)
            return -1;

        read_ints++;
        file_offset++;
    }while(str_num_quest[read_ints-1] != '\n' && file_offset<file_dim);
    str_num_quest[read_ints-1]='\0';
    num_quest=atoi(str_num_quest);
    lseek(fd, file_offset, SEEK_SET);
    if(num_quest==0){
        perror("non vi sono domande disponibili");
        pthread_exit(NULL);
    }
    *quiz_len=num_quest;
    if(num>1){
        if(num<=num_quest){
            do {
                nbytes = read(fd, &buff[read_chars], 1);
                if(nbytes == -1)
                    return -1;

                if (buff[read_chars] == '~') {
                    file_offset+=2;
                    curr_quest++;
                    lseek(fd, file_offset, SEEK_SET);
                }else
                    file_offset++;
            } while (curr_quest!=num && file_offset<file_dim);
        }else
            return -2;
    }
    return file_offset;
}
```

[Screenshot 17] Server (server.c) – funzione setOffset

La funzione *setOffset* imposta il valore dell'offset così da consentire la lettura della domanda n-esima dal file per i giocatori che si uniscono alla partita (verrà eseguita una sola volta da ogni thread).
Ricevuto in ingresso il numero della domanda che dovrà essere somministrata al nuovo giocatore, essa permetterà, ai giocatori che vogliono unirsi alla partita, di farlo in maniera sincrona rispetto agli altri, partendo direttamente dalla domanda successiva rispetto a quella posta ai Client in gioco.

Prende in ingresso il file descriptor del file da cui vengono recuperate le domande, la dimensione in byte del file, il numero di domanda che si vuole poter leggere in seguito e il parametro in cui verrà salvato il numero di domande presenti nel file (posto nella prima riga del file stesso). Ritorna 0 in caso di successo, -1 se si è verificato un errore, -2 se la domanda richiesta non esiste nel file *.txt*

3.3 Implementazione Client

Funzione int takeAnswer(int type)

La funzione takeAnswer gestisce l'inserimento, da parte dell'utente, di una risposta in un certo lasso di tempo. Il parametro in ingresso indica il tipo di risposta che si intende gestire:

- se type=1 la funzione gestisce l'inserimento di una risposta ad una domanda (i cui valori ammessi sono quelli corrispondenti alle opzioni di risposta: da 1 a 4)
- se type!=1 gestisce l'inserimento di una risposta del tipo "Y" o "N"

La funzione ritorna la risposta dell'utente oppure, allo scadere del timeout della select, -1.

```
int takeAnswer(int type) {
    fd_set read_fds;
    FD_ZERO(&read_fds);
    FD_SET(STDIN_FILENO, &read_fds);
    struct timeval timeout;
    timeout.tv_sec = AVAILABLE_TIME+1;
    timeout.tv_usec = 0;
    int givenAns = -1,
        ok;

    if (select(STDIN_FILENO+1, &read_fds, NULL, NULL, &timeout) != 0) {
        char buff[MAXLEN_ANSWER] = { '\0' };
        do{
            ok=1;
            takeSaneString(buff, MAXLEN_ANSWER);
            if(type==1){
                if(!isAPossibleAnswer(buff)){
                    ok=0;
                    write(STDOUT_FILENO, YELLOW_COLOR"Attenzione"WHITE_COLOR", inserire un numero compreso tra 1 e 4: ", 65);
                }
            }else{
                if(strcmp(buff, "Y")!=0 && strcmp(buff, "N")!=0){
                    ok=0;
                    write(STDOUT_FILENO, YELLOW_COLOR"\tAttenzione"WHITE_COLOR", inserire Y o N: ", 43);
                }else{
                    if(strcmp(buff, "Y")==0)
                        return 1;
                    else
                        return -1;
                }
            }
        }while(ok==0);
        givenAns = atoi(buff);
    }
    return givenAns;
}
```

[Screenshot 17] Client (utility.c) – funzione takeAnswer