UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II
SCUOLA POLITECNICA E DELLE SCIENZE DI BASE
DIPARTIMENTO DI INGEGNERIA ELETTRICA E TECNOLOGIE DELL'INFORMAZIONE

CORSO DI LAUREA IN INFORMATICA
INSEGNAMENTO DI LABORATORIO DI SISTEMI OPERATIVI

Anno Accademico 2020/2021

Specifica, progettazione e implementazione di un applicativo client-server per giocare a "Tic Tac Toe"

Docente

Prof. Faella Marco

Autori

Carmine Grimaldi – N86002551

Michele Ambrosanio – N86002568

N° Gruppo: 41

Sommario

Per la visione dei sorgenti si rimanda al repository GitHub

https://github.com/carmine-grimaldi/Tic-Tac-Toe

Descrizione del progetto

L'applicazione è stata scritta utilizzando il linguaggio C e la comunicazione tra Client e Server avviene attraverso socket TCP.

Il progetto consiste nella realizzazione di un'applicazione che consenta a diversi client di giocare tra di loro a "Tic Tac Toe".

Il server, unico per tutti i possibili client, gestisce simultaneamente le richieste e ne verifica la loro conformità, ed eventualmente le sezioni di gioco.

In particolare, un client può effettuare tre possibili operazioni nel menù:

1. Iniziare una nuova partita
2. Visualizzare la classifica attuale
3. Effettuare la disconnessione

Il Server effettua un log a schermo delle informazioni riguardanti i Client loggati e tiene traccia delle sezioni di gioco in corso.

1.Guida alla compilazione e manuale d'uso

Segue una descrizione sintetica dell'applicativo, senza entrare nei dettagli tecnici, che permette la comprensione e l'utilizzo del programma.
Innanzitutto è necessario compilare i due file.c presenti all'inizio della directory: Server.c e Client.c

```
Server: gcc –pthread Server.c –o Server
```

```
Client: gcc –pthread Client.c –o Client
```

Dopo le due compilazioni, si verranno a creare i rispettivi file Server.out e Client.out pronti ad essere eseguiti da shell.

```
1. ./Server <Porta>
```

L'indirizzo del server va specificato come parametro al client.

```
2. ./Client <Indirizzo IP> <Porta>
```

Nel momento in cui verrà avviato il file Client.out, comparirà all'utente una schermata di Login che richiederà il nickname con il quale identificarlo.

Nel momento in cui il nickname soddisferà i vincoli di accesso legati alla sintassi di quest'ultimo, la schermata di gioco verrà visualizzata attraverso un menù con varie opzioni selezionabili, tra cui l'inizio di una nuova partita, la visualizzazione della classifica in tempo reale degli utenti loggati in base ad i loro risultati, ed infine, la possibilità di disconnettersi, terminando la relativa esperienza di gioco.

Il funzionamento del Server

Come detto in precedenza, il server deve gestire le connessioni dei client.

Per gestire le connessioni, si utilizza un array di record di tipo "clientTable", il quale ha una dimensione massima definita dalla costante.

```
#define CLIENT_LIMIT 10
```

Alla connessione di un client, il server controlla se non è stato raggiunto il limite massimo di client connessi, se tale limite non è stato raggiunto, il client verrà accettato e si terrà aggiornato il numero di client connessi tramite un messaggio a video, e sarà creato un thread detached per quello specifico client.

Se invece il limite di client connessi è stato raggiunto, il client che ha effettuato la richiesta verrà rifiutato e terminerà subito la sua esecuzione, stampando un messaggio di errore a schermo.

```
typedef struct {
        int sd;
        int opponentSd;
        int status;
        int victories;
        int draws;
        char *nickname;
} clientTable;
```

All'accettazione di un client viene dedicato un record di tipo "clientTable" all'interno dell'array contenente tutte le informazioni necessarie a soddisfare le varie richieste.

Il funzionamento del Client

Iniziare una nuova partita

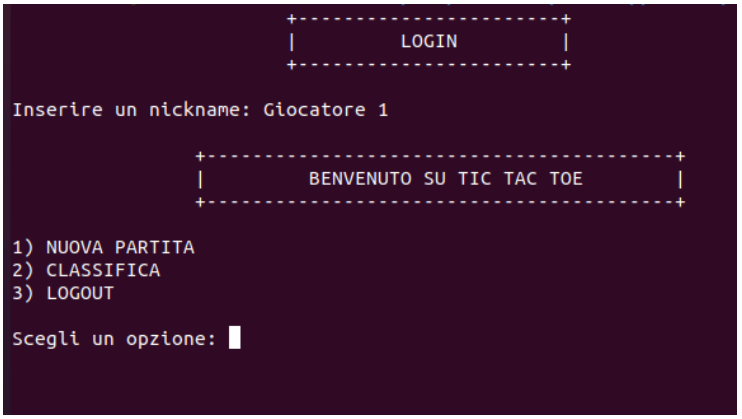


Fig.1

Digitando “1” nel menù (**Fig.1**) si ricercherà un altro giocatore con cui iniziare una nuova partita, nel caso in cui non ci siano altri client connessi, il giocatore resterà in attesa (dalla quale può uscire digitando nuovamente “1”).

Quando la partita inizierà, verrà visualizzato a schermo il nickname dell’avversario, dopodiché uno dei due giocatori avrà la possibilità di fare la prima mossa mentre l’altro rimarrà in attesa di ricevere la mossa dell’avversario.

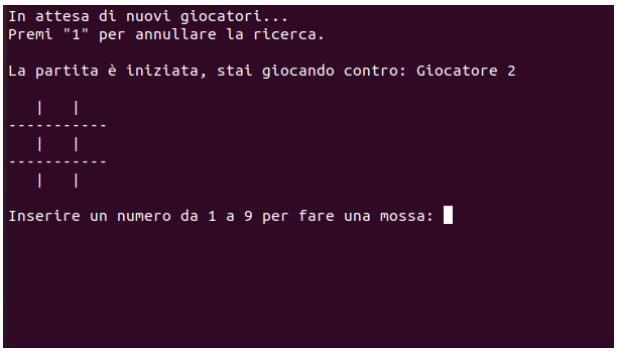


Fig.2

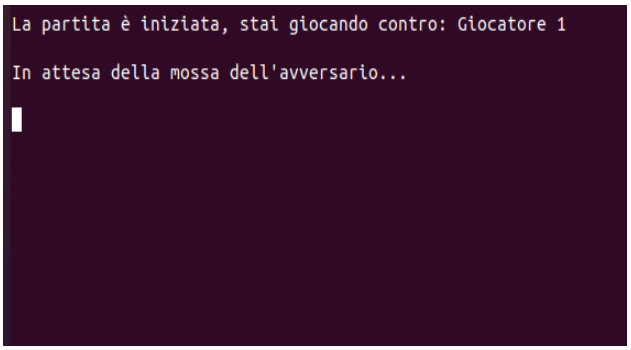


Fig.3

Il giocatore che inizierà la partita si vedrà comparire a schermo la griglia **(Fig.2)** di gioco relativa a “Tic Tac Toe”. Le mosse verranno inviate al Server interpretando la griglia come il tastierino numerico da 1 a 9 **(Fig.4)**, mentre il suo avversario rimarrà in attesa di ricevere la mossa **(Fig.3)**.

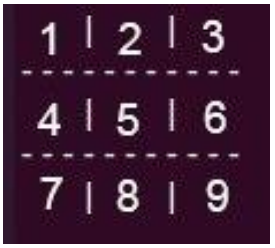


Fig.4

Nel momento in cui l'avversario riceverà la suddetta mossa, le schermate dei due giocatori si scambieranno con l'unica differenza relativa alla griglia di gioco aggiornata, ed a quel punto toccherà all'avversario fare la prossima mossa.

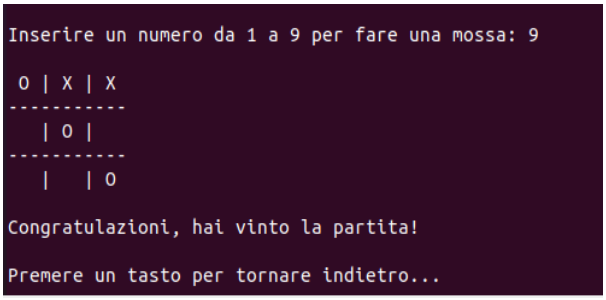


Fig.5

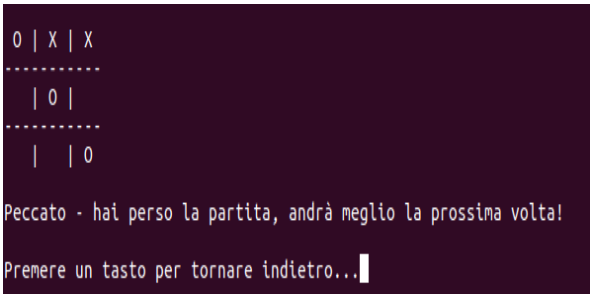


Fig.6

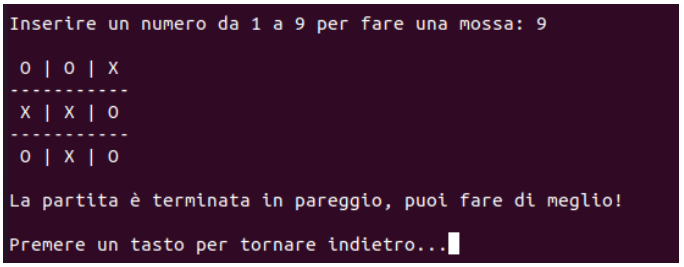


Fig.7

Nel momento in cui verrà effettuato il "Tris", la partita giungerà al termine, facendo visualizzare ai relativi giocatori impegnati nella partita l'eventuale vittoria **(Fig.5)** e sconfitta **(Fig.6)**.

Nel caso in cui le mosse copriranno l'intera griglia e non sarà avvenuto il "Tris", i giocatori visualizzeranno la stessa schermata, ossia quella di pareggio **(Fig.7)**.

Al termine della partita attraverso la digitazione di un carattere qualsiasi si ritornerà al menù principale.

Visualizzazione Classifica e funzionamento

Digitando “2” nel menù si visualizzerà la classifica **(Fig.8)** dei giocatori online in quel momento. La classifica fornirà informazioni su vittorie e pareggi per tutto l’arco di tempo in cui i giocatori hanno continuato a giocare senza disconnettersi. Quest’ultima verrà ordinata per numero di vittorie e, a parità di quest’ultime, per numero di pareggi.

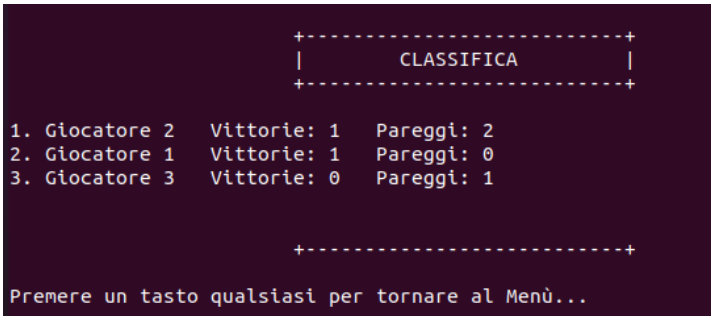


Fig.8

Disconnessione

Digitando “3” nel menù principale si verrà disconnessi **(Fig.9)** inviando un messaggio di avviso al server che si occuperà dello smaltimento delle informazioni relative al giocatore.

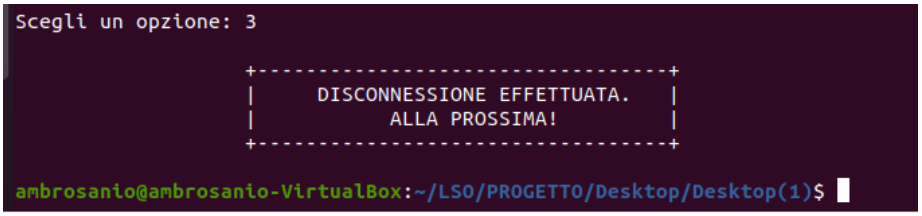
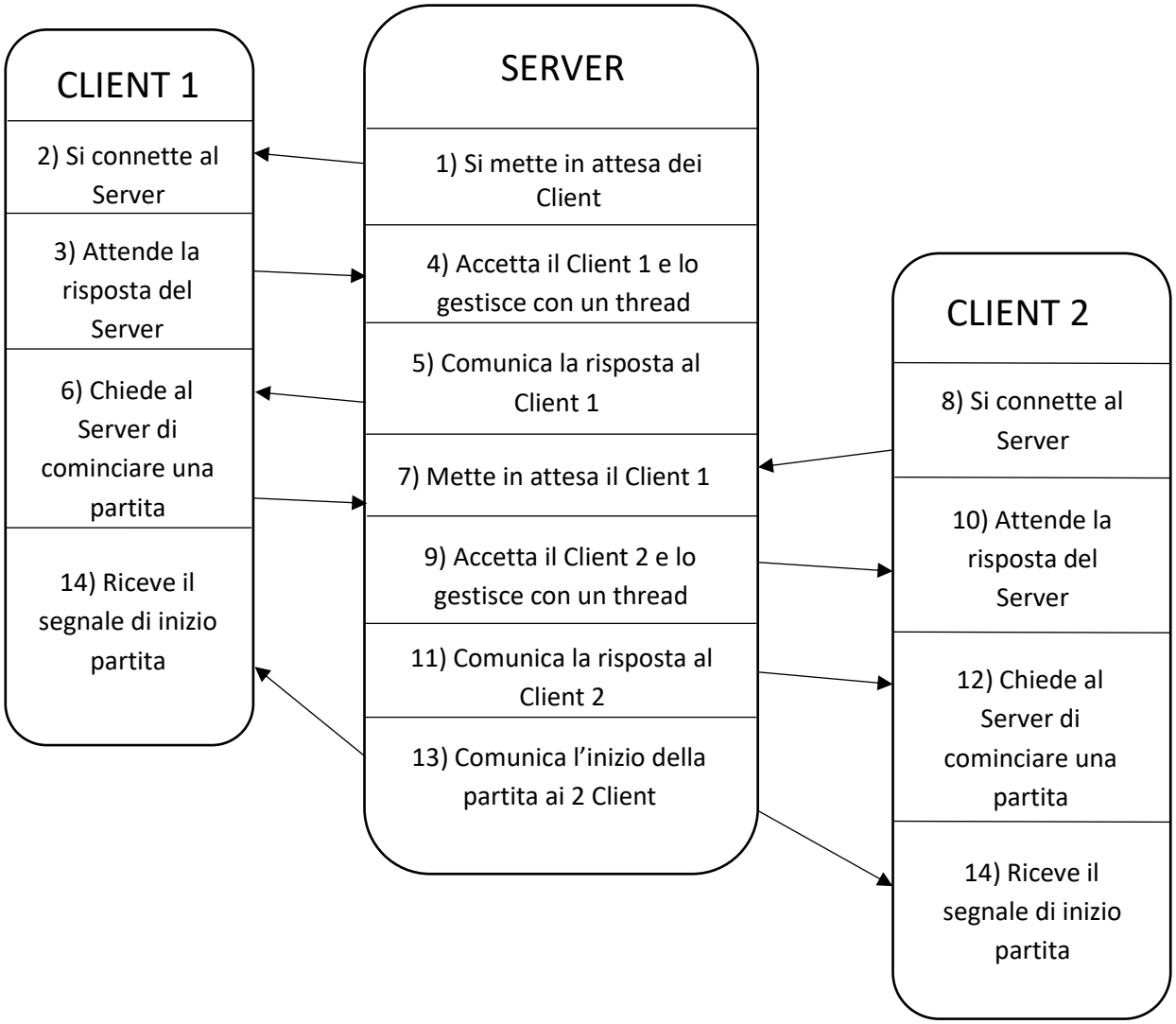


Fig.9

2.Protocollo di comunicazione Client-Server

Implementazione della connessione Client – Server



Per la gestione della pre-connessione, Client e Server si scambiano tra di loro dei messaggi. Una volta che il Client ha inoltrato la richiesta di connessione al Server, quest'ultimo verifica se il limite di connessioni è stato superato.

La comunicazione avviene in questo modo: il client invia una stringa al server che identifica un'operazione da compiere (tali identificativi sono riportati qui di seguito, queste costanti sono definite in **constantsInCommon.h**), il server risponde di conseguenza inviando la risposta al client sempre attraverso una stringa in un buffer.

```
#define CONNECTION_ACCEPTED "accepted"
#define CONNECTION_REFUSED "refused"

#define NICKNAME_ACCEPTED "yes"
#define NICKNAME_REFUSED "no"

#define ABORT_FIND_MATCH_REQUEST '#'
#define SEARCH_ABORTED_MSG "search aborted"

#define OPPONENT_NICKNAME_REQUEST '@'

#define RANKING_REQUEST '/'

#define MATCH_IS_OVER_REQUEST '?'

#define CLIENT_DISCONNECTED_REQUEST '^'

#define TRY_TO_PLAY_REQUEST '*'

#define ILLEGAL_REQUEST "illegal request"

#define CLIENT_WON_MATCH '1'
#define CLIENT_MATCH_TIE '2'

#define MOVE_INSERTED_MSG "ok"

#define WAIT_FOR_MATCH_TO_BEGIN "wait"

#define FIRST_CLIENT_TO_PLAY_MSG "start1"
#define SECOND_CLIENT_TO_PLAY_MSG "start2"

#define CONNECTION_LOST_MSG "connectionLost"
```

Fig.10

Sempre in merito al discorso della pre-connessione, in caso di limite raggiunto, il Server invierà al Client la stringa “refused”, mentre, in caso contrario, invierà la stringa “accepted”. Una volta ricevuta una delle due stringhe il Client leggerà quest’ultima e si comporterà di conseguenza. Nel caso della ricezione della stringa “refused”, verrà visualizzato a video il seguente messaggio: “Il server è attualmente pieno, riprovare più tardi”, dopodiché terminerà l’esecuzione del Client. In caso contrario verrà visualizzata la schermata di login del gioco.

Nel caso in cui la connessione è stata accettata, il Server incrementa una variabile globale che rappresenta il numero di Client attualmente connessi, dopodiché, dedicherà una cella all’interno dell’array di record con tutte le informazioni necessarie a gestire le eventuali richieste del Client.

Ovviamente, trattandosi di un contesto multithreaded, tutte le modifiche all’array dedicato alle connessioni, che è condiviso tra i threads, sono sincronizzate tramite un mutex globale, evitando così situazioni di race condition.

Terminazione del Client

Quando un Client, per qualsiasi ragione, chiude la comunicazione, il Server gestisce l’errore tramite un segnale della socket, procedendo quindi alla disconnessione di quel Client.

Quest’ultimo ricevendo il segnale di disconnessione, recupera la posizione del client dall’array, libera quella posizione e infine decrementa il numero di client connessi.

Terminazione del Server

Il Server può raggiungere lo stato di terminazione solo tramite riga di comando digitando la stringa “abort”, la quale viene riconosciuta da un thread in background lanciato all’inizio dell’esecuzione del Server.

3.Dettagli implementativi

Gestione della concorrenza

Il software, potendo gestire diversi client contemporaneamente, è tenuto a dover affrontare problematiche dovute alla concorrenza delle operazioni, in particolare la lettura e la scrittura sull’array di record globale.

Ad esempio: un primo client desidera cercare un avversario per iniziare una partita, mentre viene effettuata l'operazione, un secondo client desidera di visualizzare la classifica, la quale richiede l’accesso al medesimo array.

Per evitare queste problematiche, è stato utilizzato il mutex “clientConnMutex” con le opportune system call pthread_mutex_lock() e pthread_mutex_unlock().

Poiché la system call read() è bloccante, esiste un unico thread per ogni Client che si occupa di recepire le richieste ricevute tramite tale system call, dopodiché le smista a chi di dovere.

Dettagli ritenuti interessanti – Server

Funzione rankingInsertionSort()

```
void rankingInsertionSort(clientTable * client[], int size)
{
    int i, j;
    clientTable * temp;

    for(i=1; i<size; i++)
    {
        temp = client[i];
        j = i-1;

        while(j>=0 && ((client[j]->victories < temp->victories) ||
(client[j]->victories == temp->victories && client[j]->draws < temp->draws)))
        {
            client[j+1] = client[j];
            j--;
        }
        client[j+1] = temp;
    }
}
```

Tale algoritmo risulta essere una variante di InsertionSort() adattata per ordinare l'array dato in input per ordine di numero di vittorie e, a parità di quest'ultime, per numero di pareggi (informazioni contenute nella struct "clientTable"). L'algoritmo è quindi utilizzato ai fini di soddisfare la richiesta opzionale della classifica. È stato scelto insertionSort() poiché su array di piccole dimensioni risulta essere l'algoritmo ottimale, avendo scelto come limite di Client iniziale un massimo di 10 client connessi contemporaneamente.

Dettagli ritenuti interessanti – Client

Funzione Play()

```
void play(int sd, char buf[], char opponentSymbol, char mySymbol){
    int n;
    int movesCnt = 0;

    if(opponentSymbol == 'X')
```

```
        movesCnt++;

    while((n=read(sd, buf, 100)) > 0){
        clean_stdin();

        if(strcmp(buf, "connectionLost") == 0) {
            printf("\nIl tuo avversario si è disconnesso, partita
                      terminata.\n\n");
            printf("Premere un tasto per tornare indietro...");
            getchar();
            printf("\n");
            break;
        }
        addMoveToMatrix(buf[0], opponentSymbol);
        movesCnt++;

        if(movesCnt >= 5) { //La partita potrebbe essere terminata
            if(checkResult(movesCnt, opponentSymbol) == 1){
                printf("\n");
                printMatrix();
                memset(buf, 0, 100);
                buf[0] = '?';
                buf[1] = '0';
                if(write(sd, buf, 2) < 2){
                    perror("write");
                    exit(1);
                }
                printf("\n\nPeccato - hai perso la partita, andrà meglio la
                            prossima volta!\n\n");
                printf("Premere un tasto per tornare indietro...");
                int c;
                while ((c = getchar()) != '\n' && c != EOF) {}
                getchar();

                printf("\n");
                break;
            }else if(checkResult(movesCnt, opponentSymbol) == 2){
                printf("\n");
                printMatrix();
                memset(buf, 0, 100);
                buf[0] = '?';
                buf[1] = '2';
                if(write(sd, buf, 2) < 2){
                    perror("write");
                    exit(1);
```

```
            }
            printf("\n\nLa partita è terminata in pareggio, puoi fare di
                        meglio!\n\n");
            printf("Premere un tasto per tornare indietro...");
            int c;
            while ((c = getchar()) != '\n' && c != EOF) {}
            getchar();
            printf("\n");
            break;
         }
    }
    printf("\n");
    printMatrix();

    printf("\n\nInserire un numero da 1 a 9 per fare una mossa: ");

    if(!selectAndSendYourMove(mySymbol, sd)){
        printf("\nIl tuo avversario si è disconnesso, partita
                  terminata.\n\n");
        printf("Premere un tasto per tornare indietro...");
        int c;
        while ((c = getchar()) != '\n' && c != EOF) {}
        getchar();
        printf("\n");
        break;
    }
    movesCnt++;
    if(movesCnt >= 5){ //La partita potrebbe essere terminata
        if(checkResult(movesCnt, mySymbol) == 1){ //Ho vinto la partita
            printf("\n");
            printMatrix();
            memset(buf, 0, 100);
            buf[0] = '?';
            buf[1] = '1';
            if(write(sd, buf, 2) < 2){
                perror("write");
                exit(1);
            }
            printf("\n\nCongratulazioni, hai vinto la partita!\n\n");
            printf("Premere un tasto per tornare indietro...");
            int c;
            while ((c = getchar()) != '\n' && c != EOF) {}
            getchar();
            printf("\n");
            break;
```

```
            }else if(checkResult(movesCnt, opponentSymbol) == 2){
                printf("\n");
                printMatrix();
                memset(buf, 0, 100);
                buf[0] = '?';
                buf[1] = '2';
                if(write(sd, buf, 2) < 2){
                    perror("write");
                    exit(1);
                }
                printf("\n\nLa partita è terminata in pareggio, puoi fare di
                            meglio!\n\n");
                printf("Premere un tasto per tornare indietro...");
                int c;
                while ((c = getchar()) != '\n' && c != EOF) {}
                getchar();
                printf("\n");
                break;
            }
        }
        //Stampa nuova matrice
        printf("\n");
        printMatrix();
        printf("\n\nIn attesa della mossa dell'avversario...\n");
        memset(buf, 0, 100); //Prima della prossima read
    }
    if(n <= 0) {
        perror("read");
        exit(1);
    }
}
```

Tale funzione si occupa di gestire lo scambio di mosse dei due giocatori durante la partita attraverso le system call read() e write(), che sfruttano inoltre il Server come tramite per comunicare tra di loro.

La funzione ha la responsabilità di riconoscere anche la stringa “connectionLost” che indica la disconnessione dell’avversario durante la partita, in quel caso play() forza l’interruzione del gioco lasciando i dati relativi alle vittorie e pareggi inalterati. La funzione play() necessita del Socket descriptor ai fini della comunicazione con il Server, necessita poi di un buffer che conterrà ogni volta la mossa espressa in un numero da 1 a 9 e, infine, prende in input il simbolo associato al giocatore corrente e quello associato all’avversario, per poter procedere all’inserimento delle mosse nella griglia di gioco. La funzione interessata ha inoltre il compito di verificare l’avvenuto “Tris” a partire dal momento in cui sono state raggiunte almeno 5 mosse.