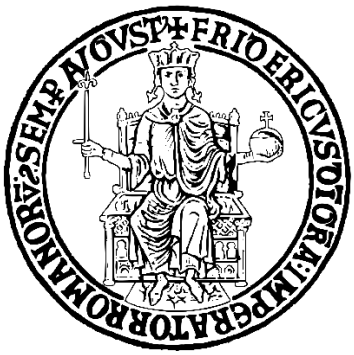
Università degli Studi di Napoli Federico II

Scuola Politecnica e delle Scienze di Base

Dipartimento di Ingegneria Elettrica e Tecnologie dell'Informazione



Corso di Laurea in Informatica

Laboratorio di Sistemi Operativi

Anno accademico 2020/2021

Specifica, progettazione e implementazione della traccia "Tic-Tac-Toe"

Consegna: Marzo

Autori:

Ciccarelli Antonio N86002985

Balzano Giuseppe N86002857

Gruppo N.39

Docenti:

Faella Marco

Cutugno Francesco

Grazioso Marco

INDICE

1. Guida alla compilazione e manuale d'uso

1.1 LATO SERVER IN C

1. Scaricare il sorgente presente nella cartella "SERVER" dalla seguente repository GitHub:

   https://github.com/Peppebalzanoo/LSO_2021

2. Compilare il file Server.c presente nella directory utilizzando il comando gcc da shell:

```
1. gcc -Wall -pthread server.c -o Server
```

3. Avviare il file creato al termine della compilazione come semplice eseguibile shell:

```
1. ./Server
```

1.2 LATO CLIENT IN ANDROID

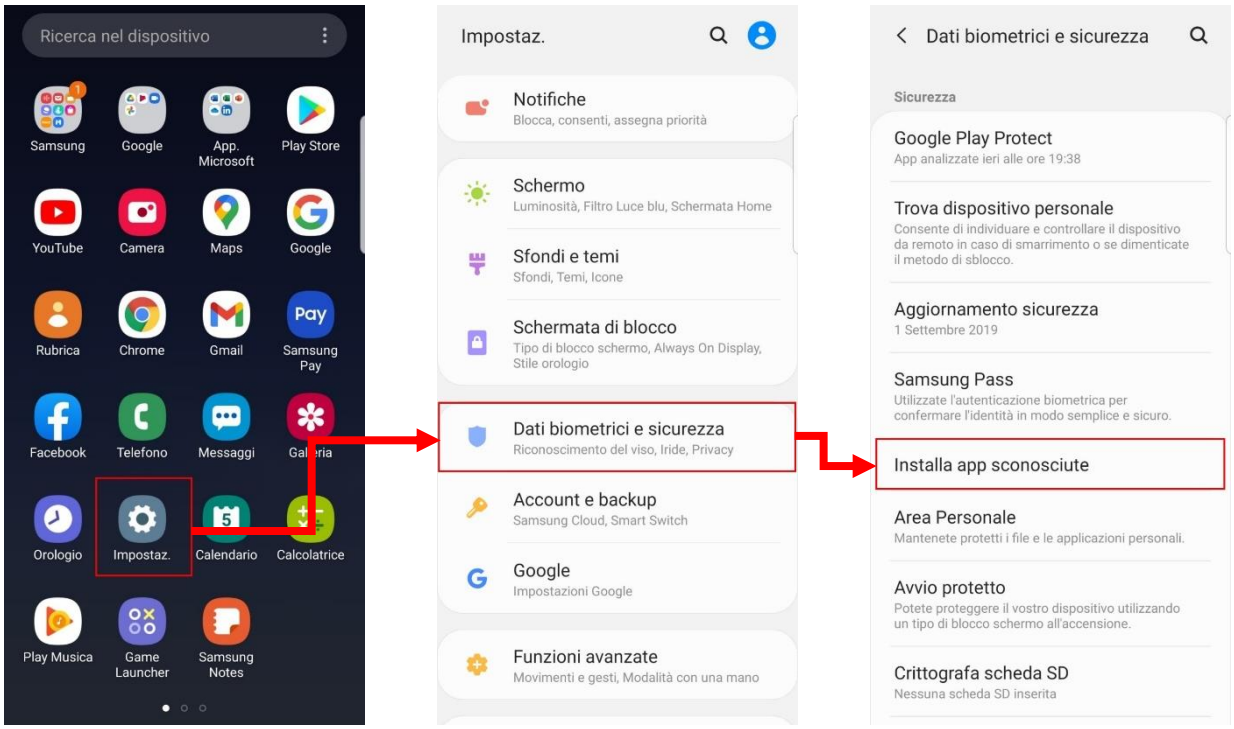
1. Scaricare sul proprio smartphone il file APK presente nella directory "APK" dalla seguente repository GitHub:

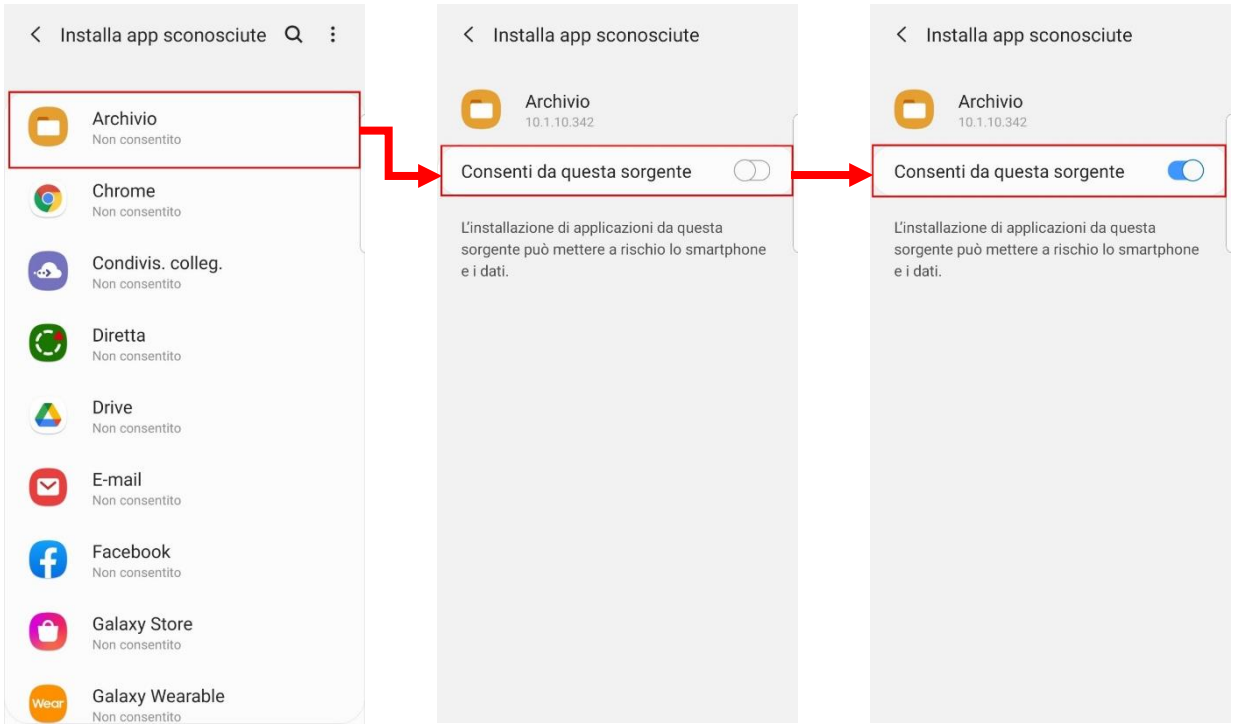
   https://github.com/Peppebalzanoo/LSO_2021

2. Codice Java nella cartella "CLIENT" alla seguente repository GitHub:

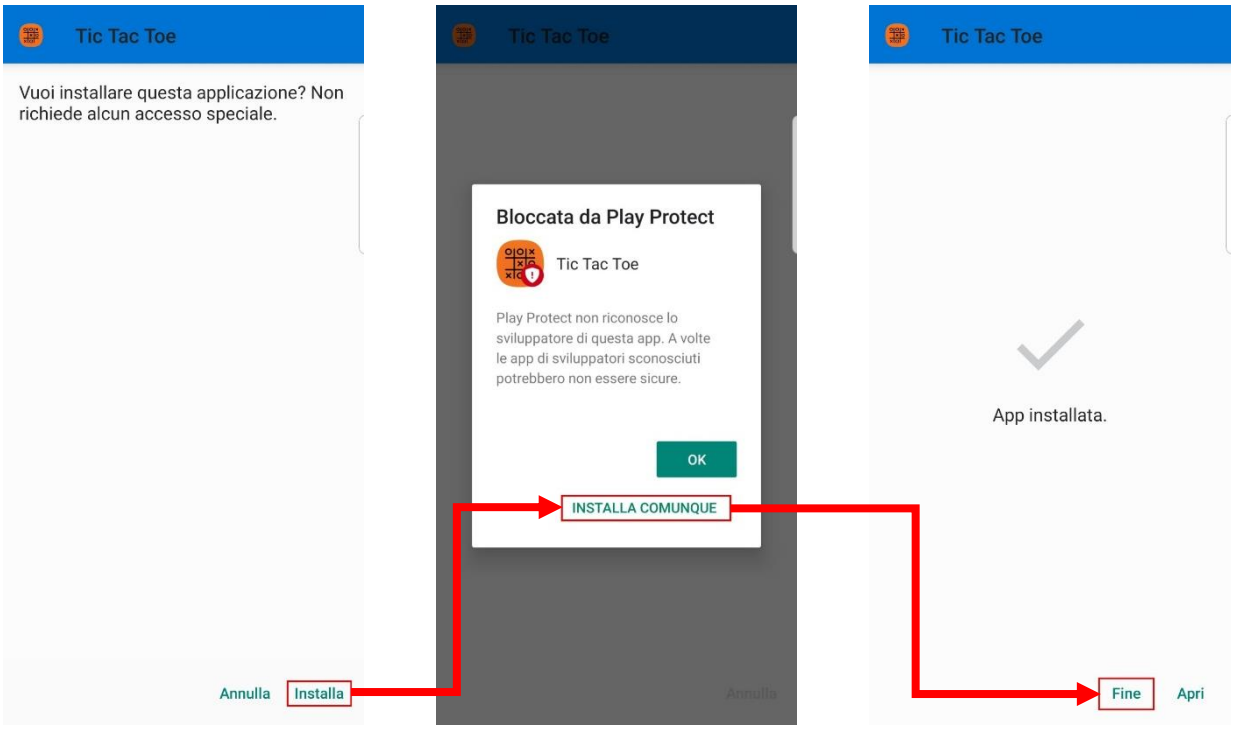
   https://github.com/Peppebalzanoo/LSO_2021

3. Accedere alle impostazioni del proprio smartphone ed abilitare nella sezione sicurezza l'opzione per consentire l'installazione di applicazioni non ufficiali da sorgenti sconosciute





4. Procedere ora all'installazione dell'applicazione attraverso il file APK precedentemente scaricato



5. All'avvio dell'applicazione l'utente potrà scegliere diverse modalità di gioco: **SINGLE PLAYER, MULTIPLAYER OFFLINE** e **MULTIPLAYER ONLINE.**

1.2.1 SINGLE PLAYER

Questa modalità permette ad un singolo utente di sfidare in una o più partite un Player CPU.

Al click sul bottone "Single Player" verrà immediatamente iniziata una partita. Ad ogni mossa dell'utente ne corrisponderà una da parte dell'avversario virtuale.

All'utente verrà assegnato segno "X" ed al Player CPU segno opposto "O".

Quest'ultimo farà la propria mossa utilizzando un algoritmo capace di scegliere in maniera del tutto casuale una casella vuota tra quelle disponibili.

Per il controllo dello stato della partita e per l'avanzamento della stessa, ad ogni mossa verrà utilizzato un algoritmo in grado di capire se vi è stata una mossa vincente e di permettere l'assegnazione della vittoria, sconfitta o pareggio all'utente.

Ad ogni partita conclusa per :

a) "Vittoria" verrà assegnato un punto all'utente;
b) "Sconfitta" verrà assegnato un punto al Player CPU;
c) "Pareggio" non verrà assegnato alcun punto.

Al termine della partita l'utente potrà scegliere se rigiocare oppure se tornare alla schermata principale e cambiare modalità di gioco.

Questa modalità non richiede nessuna connessione al SERVER.

1.2.2 MULTIPLAYER OFFLINE

Questa modalità permette a due utenti di sfidarsi in una o più partite attraverso l'utilizzo del medesimo smartphone.

Al click sul bottone "Multiplayer Offline" verrà immediatamente iniziata una partita. Ad ogni mossa del primo utente ne corrisponderà una da parte dell'avversario.

Al primo utente verrà assegnato segno "X" ed all'altro utente verrà assegnato segno opposto "O".

Per il controllo dello stato della partita e per l'avanzamento della stessa, ad ogni mossa verrà utilizzato un algoritmo in grado di capire se vi è stata una mossa vincente e di permettere l'assegnazione della vittoria o pareggio ai due utenti.

Ad ogni partita conclusa per :

a) "Vittoria X" verrà assegnato un punto all'utente vincitore con segno "X";
b) "Vittoria O" verrà assegnato un punto all'utente vincitore con segno "O";

c) “Pareggio” non verrà assegnato alcun punto.

Al termine della partita gli utenti potranno scegliere se rigiocare oppure se tornare alla schermata principale e cambiare modalità di gioco.

Questa modalità non richiede nessuna connessione al SERVER.

1.2.3 MULTUPLAYER ONLINE

Questa modalità permette a due utenti di sfidarsi in una partita da remoto.

Questa modalità richiede all’utente obbligatoriamente una connessione ad INTERNET ed al SERVER.

Se il Server non è disponibile verrà mostrato il messaggio di avviso “Errore di connessione…”, lasciando l’utente alla scherma principale dell’applicativo.

Se il Client dell’utente non dispone di una connessione ad INTERNET verrà mostrato il messaggio di avviso “Errore di connessione…”.

Se il Client dell’utente dispone di una connessione ad INTERNET e per qualsiasi motivo, a partita in corso, dovesse perderla o eventualmente disabilitarla volontariamente, verrà mostrato il messaggio di avviso “Errore di connessione…”.
Si effettuerà quindi la disconnessione dal Server e verrà assegnata la vittoria al suo avversario.

Se il Server, a partita in corso , si dovesse arrestare a causa di una qualsiasi anomalia, ai Client dei due utenti verrà mostrato il messaggio di avviso “Oops qualcosa è andato storto…” reindirizzandoli alla schermata principale.

Al click sul bottone “Multiplayer Online” verrà effettuata la connessione al Server, il quale metterà in attesa l’utente per la ricerca di un avversario.
Il Server provvederà all’abbinamento di due utenti alla medesima partita.

A lato Server, quest’ultimo assocerà “X” al primo Client connesso e segno “O” al secondo.
A lato Client, entrambi gli utenti di una partita giocheranno con segno “X” ma riceveranno la mossa dell’avversario con segno opposto “O”.

Ogni Client avrà a disposizione 30 secondi per effettuare la propria mossa.
Nel caso in cui a termine di tale quanto di tempo, il Client in questione non dovesse ancora effettuare la mossa, verrà mostrato un messaggio di avviso “Sconfitta per inattività”.

Si effettuerà quindi la disconnessione dal Server e verrà assegnata la vittoria al suo avversario.

Per il controllo dello stato della partita e per l'avanzamento della stessa, ad ogni mossa verrà utilizzato un algoritmo in grado di capire se vi è stata una mossa vincente e di permettere l'assegnazione della vittoria, sconfitta o pareggio ai due utenti.

Ad ogni partita conclusa con risultato di pareggio verrà assegnato "Pareggio" ad entrambi i Client, altrimenti verrà assegnata "Vittoria" al Client vincitore e "Sconfitta" al Client perdente.

Nel caso in cui uno dei due Client durante la partita dovesse abbandonare volontariamente chiudendo l'applicativo, verrà assegnata la vittoria al suo avversario.

Al termine della partita i due utenti si disconnetteranno dal Server, tornando alla schermata principale per poter scegliere se continuare a giocare ancora oppure no.

2. Protocollo applicativo tra client e server

Per lo scambio di informazioni tra Client e Server il protocollo applicativo utilizzato prevede:

2.1 LATO CLIENT

A lato Client l'uso di un oggetto denominato "input" di tipo BufferedReader ed un oggetto denominato "output" di tipo DataOutputStream.

- La classe DataOutputStream farà in modo che non ci si debba preoccupare della codifica dei dati spediti, in quanto utilizza il *Network Byte Order* per la memorizzazione dei dati.

  Per la trasmissione di tali dati il Client utilizza il metodo writeBytes(String s) il quale scrive una stringa sullo stream collegato come una sequenza di byte. Quindi può essere utilizzato per trasmettere dei dati in formato ASCII a un dispositivo.

  La lunghezza della stringa non viene scritta nello stream, questo però non è un problema in quanto l'implementazione adottata prevede che il Server sappia a priori quanti bytes leggere, in quanto il Client provvederà alla trasmissione di una stringa formata da un solo carattere, che corrisponderà al numero di casella in cui l'utente desidera inserire il proprio segno.

- La classe BufferedReader è una classe dedicata alla lettura di buffers, essa è stata scelta a discapito della classe DataInputStream in quanto prevede il metodo readline(). Tale metodo consente al Server di non dover prima inviare la lunghezza della stringa da trasmettere, in quanto il metodo readline() legge una sequenza di caratteri fino al carattere "\n" o "\r".

2.2 LATO SERVER

A lato Server per assicurare la corretta ricezione delle informazioni da parte dei Client e la corretta trasmissione di queste si prevede la conversione dei dati dalla rappresentazione interna dei Server alla rappresentazione *Network Byte Order* attraverso l'uso delle funzioni htons() e htonl().

- La scrittura dei dati avviene attraverso la system call write che scriverà una sequenza di caratteri terminante con uno "\n" cosicché il Client attraverso l'uso del metodo readline() possa riceverla correttamente.

- La lettura dei dati avviene attraverso la system call read che leggerà sempre 1 byte, in quanto i diversi Client invieranno una stringa formata da un solo carattere che corrisponderà ad un numero da 0 a 8. Tale numero farà riferimento alla cella in cui l'utente ha inserito il proprio segno.

Inoltre, si è utilizzato il protocollo di trasferimento TCP a discapito dell'UDP, in quanto considerato più sicuro ed affidabile.

3. DETTAGLI LATO SERVER

3.1 STRUTTURA DEL SERVER

Per la realizzazione del Server si è utilizzato il linguaggio C facendo riferimento solo alla libreria STANDARD dello stesso e prevedendo l'uso esclusivo delle principali systems call e dei principali meccanismi di sincronizzazione studiati a lezione.

Il Server è di tipo multithreading, come richiesto espressamente da traccia, ed è in grado di gestire ed accogliere più Client simultaneamente.

Quest'ultimo è risiedente su piattaforma cloud Microsoft Azure ed è accessibile all'indirizzo IP pubblico 104.40.215.231 e porta 20000.

La macchina virtuale che ospita tale server è la Standard_B1s (1 CPU virtuale, 1 GB di memoria) con sistema operativo Ubuntu Server 18.04 LTS-Gen1 residente in Europa occidentale.

La struttura del Server è la classica utilizzata per il protocollo di trasferimento scelto, nel nostro caso TCP, prevedendo sia l'utilizzo di sockaddr_in, che è la generica struttura degli indirizzi Socket Address per IPv4 e sia le necessarie conversioni di porta e di indirizzo IPv4 dalla rappresentazione interna degli hosts alla rappresentazione di rete comunemente chiamata *Network Byte Order*.

Si è previsto quindi l'uso delle funzioni definite nell'include <netnet/in.h> come htons (Host To Network Short) e htonl (Host To Network Long).

```
1. struct sockaddr_in server_addr;
2. struct sockaddr_in client_addr;
3. socklen_t client_len = sizeof(client_addr);
4. server_addr.sin_family = AF_INET;
5. server_addr.sin_port = htons(PORT);
6. server_addr.sin_addr.s_addr = htonl(INADDR_ANY);
```

AF_INET specifica la famiglia di protocolli utilizzata, ovvero IPv4.

INADDR_ANY specifica la conversione generica di indirizzi IPv4.

3.2 SYSTEM CALLS

La chiamata di sistema socket() definisce un canale di comunicazione bidirezionale progettato proprio per comunicazioni Client-Server. Essa fornisce un file descriptor attraverso il quale è possibile effettuare le System Calls read, write, close etc.

I socket permettono di specificare il tipo di comunicazione attraverso dominio e stile.

- Il dominio di una socket equivale alla scelta di una famiglia di protocolli. Nel nostro caso utilizziamo PF_INET che corrisponde alla famiglia IPv4.

- Lo stile di comunicazione definisce il tipo di canale logico instaurato per la comunicazione, ossia le caratteristiche della trasmissione. Nel nostro caso utilizziamo SOCK_STREAM per avere un canale di comunicazione bidirezionale a flusso, con connessione, sequenziale ed affidabile.

La funzione bind () assegna un socket a un indirizzo. Quando un socket viene creato utilizzando socket (), gli viene assegnata solo una famiglia di protocolli, ma non gli viene assegnato un indirizzo. Questa associazione con un indirizzo deve essere eseguita con la chiamata di sistema bind () prima che il socket possa accettare connessioni ad altri host.

Essa accetta tre argomenti:

1. Server_sd, un descrittore che rappresenta il socket su cui eseguire il bind.
2. Server_addr, un puntatore a una struttura sockaddr che rappresenta l'indirizzo a cui collegarsi.
3. Un campo socklen_t che specifica la dimensione della struttura sockaddr.

La funzione listen() mette il socket in modalità ascolto in attesa di nuove connessioni.
Essa accetta due argomenti:

1. server_sd, il descrittore che si mette in modalità ascolto
2. NUMBERS_CONNECTION_CLIENTS_IN_WAIT, specifica quante connessioni possono essere in attesa di essere accettate

```
1.    server_sd = socket(PF_INET, SOCK_STREAM, 0);
2.    if(server_sd==-1){
3.       printError("[SOCKET] Errore creazioen socket\n");
4.       exit(EXIT_FAILURE);
5.    }
6.
7.    if(bind(server_sd, (struct sockaddr *) &server_addr, sizeof(server_addr))==-1){
8.       printError("[SOCKET] Errore bind socket\n");
```

```
9.          exit(EXIT_FAILURE);
10.     }
11.
12.     if(listen(server_sd, NUMBERS_CONNECTION_CLIENTS_IN_WAIT)==-1){
13.         printError("[SOCKET] Errore listen socket\n");
14.         exit(EXIT_FAILURE);
15.     }
```

La funzione accept() è utilizzata dal Server in caso di comunicazione orientata alla connessione e restituisce un nuovo socket descriptor che il Server utilizzerà per effettuare la comunicazione con un client.

Essa prende 3 argomenti:

1. Server_sd è una socket creata in precedenza e su cui il Server è in ascolto.
2. Client_addr è un parametro in cui verrà restituito l'indirizzo del client che ha effettuato la connessione.
3. Client_len è un parametro in cui verrà restituita la dimensione dell'indirizzo del client che ha effettuato la connessione.

Dopo un'attenta analisi del dominio e appurato che prima di poter iniziare ogni partita il Server deve accettare una connessione con due Clients si è deciso di utilizzare due accept() consecutive.

Questo perché essendo a conoscenza del fatto che le accept() sono funzioni bloccanti, siamo sicuri che il Server non continuerà la propria esecuzione e quindi non effettuerà la creazione di un thread dedicato alla singola partita, fino a quando non ci saranno due Clients connessi e disposti a giocare.

Sapendo che questo può richiedere del tempo e che la ricerca di un avversario non è immediata, si è previsto l'utilizzo di una schermata di caricamento lato Client che viene visualizzata non appena il Client effettua la connessione al Server.

Se quest'ultima non dovesse avvenire per un qualche motivo sarà reso noto all'utente, attraverso un toast, che si è verificato un errore di connessione.

Una volta stabilite le connessioni con entrambi i Clients, il Server provvede alla creazione di un thread per la partita che vede come protagonisti i due Clients appena connessi.

Il thread per la partita viene creato attraversa la system call pthreat_create(), la quale prende 4 parametri:

1. Tid, se la chiama ha successo tid punterà al thread id.
2. Il secondo parametro permette di specificare gli attributi del thread, in questo caso non ce ne sono.
3. Connection_handler è l'indirizzo della funzione di avvio del thread.
4. Partita è una struttura appositamente creata che contiene i file descriptor delle due connessioni sopra accettate.

```
1.  client_sd = accept(server_sd, (struct sockaddr*)&client_addr, &client_len);
2.  if(client_sd==-1){
3.      printError("[CONNESSIONE] Errore accept\n");
4.      exit(EXIT_FAILURE);
5.  }
6.  printLog("[CONNESSIONE] Client connesso\n");
7.
8.  client_sd2 = accept(server_sd, (struct sockaddr*)&client_addr2, &client_len2);
9.  if(client_sd2==-1){
10.     printError("[CONNESSIONE] Errore accept\n");
11.     exit(EXIT_FAILURE);
12. }
13. printLog("[CONNESSIONE] Client connesso\n")
14. pthread_create(&tid,NULL,connection_handler,(void*)partita)
```

La funzione readWrite è il cuore di tutta la comunicazione Client-Server.

Quando questa viene invocata il Server effettua una chiamata alla system call select() per controllare se sulla socket "user1" è possibile effettuare un'operazione di lettura.

Ricordiamo che user1 in questa funzione corrisponderà sempre alla socket del Client che dovrà effettuare una trasmissione di dati al Server, ovvero il Client che deve giocare.

La funzione select() prende 5 parametri:

1. (user1)+1 è il numero massimo di descrittori controllati dalla funzione.
2. Specifica che la socket user1 è pronta per un'operazione di lettura, "set" è un puntatore ad una variabile di tipo fd_set, il quale è un tipo di dato che rappresenta l'insieme dei descrittori.
3. Specifica che la socket user1 è pronta per un'operazione di scrittura, nel nostro caso NULL.
4. Specifica il verificarsi di eccezioni, nel nostro caso NULL.

5. Specifica il valore massimo che la funzione attende per individuare un descrittore pronto, nel nostro caso 30 secondi.

Se entro il timeout la system call select ritorna la socket user1, si proseguirà normalmente con l'esecuzione poiché siamo certi che su tale socket sarà possibile effettuare una lettura.

Quindi il Server resta in attesa su una chiama read() riferita al primo client passato come parametro.

Questo perché la prima volta che viene chiamata tale funzione, siamo sicuri di attendere una mossa dal Client giusto in quanto è sempre il primo Client connesso al Server a giocare per primo in partita.

Una volta che il Server legge la mossa fatta dal Client passato come primo parametro, esso attraverso la funzione insertMatrix() provvede a salvare tale mossa in una matrice che tiene traccia delle giocate dei due Clients.

Attraverso la funzione checkWinner(), la quale fa uso della suddetta matrice, il Server controlla se si è verificata una condizione di vittoria. Al termine di tale funzione il Server provvede a inoltrare la mossa, precedentemente letta, al Client passato come secondo parametro della funzione readWrite().

Al termine di quest'ultima il codice che ha invocato tale funzione provvede a controllare se c'è stato un vincitore oppure bisogna continuare a giocare.

```
1.    int readWrite(int user1, int user2, int matrix[][3], int a){
2.
3.        char client_message[10];
4.        bzero(client_message, 10);
5.        int posButton;
6.        int checkWin = 0;
7.        fd_set set;
8.        struct timeval timeout;
9.        int rev;
10.
11.       FD_ZERO(&set);
12.       FD_SET(user1, &set);
13.       timeout.tv_sec=30;
14.
15.       rev = select( (user1)+1, &set, NULL, NULL, &timeout);
16.       if(rev == -1){
17.
18.           printError("[SELECT] Errore select\n");
19.
20.       } else if(rev == 0){
21.
```

```
22.           printError("[SELECT] Timer scaduto\n");
23.           return -1;
24.
25.       } else {
26.
27.           ssize_t read_size = read(user1, client_message, 1);
28.           if(read_size == -1){
29.               printError("[READ] Errore lettura\n");
30.               return -2;
31.           }else if(read_size == 0){
32.               return -1;
33.           }
34.
35.           printLog("[READ] Read effettuata correttamente\n");
36.
37.           posButton = atoi(client_message);
38.
39.           insertMatrix(matrix, posButton, a);
40.
41.           checkWin = checkWinner(matrix);
42.
43.
44.           ssize_t write_size = write(user2, strcat(client_message, "\n"), 2);
45.           if(write_size==-1){
46.               printError("[WRITE] Errore scrittura\n");
47.               return -2;
48.           }
49.           printLog("[WRITE] Write effettuata correttamente\n");
50.
51.
52.       }
53.
54.
55.    return checkWin;
56.
57. }
```

4. DETTAGLI LATO CLIENT

4.1 CONNESSIONE AL SERVER

Quando l'utente desidera effettuare una partita online e clicca sul bottone "Multiplayer Online", il Client proverà ad effettuare una connessione al Server chiamando il seguente metodo run().

Il primo punto per instaurare la suddetta connessione è la creazione di una socket.

Fatto ciò, quest'ultima effettuerà la vera e propria connessione al Server attraverso l'uso del metodo connect() il quale prende due argomenti:

1. new InetSocketAddress: classe che implementa un IP Socket Address e ne crea uno a partire da un hostname un numero di porta.
2. Time-out: il tempo entro il quale la socket deve effettuare la connessione al Server.

Una volta stabilita la connessione, si provvederà alla creazione di un oggetto di tipo DataOutputStream "o" e di un oggetto di tipo BufferReader "in".

Il primo è utilizzato per la trasmissione dei dati al Server mentre il secondo è utilizzato per la ricezione di dati dal Server.

```
1.    public void run() {
2.
3.        try {
4.
5.            socket = new Socket();
6.
7.            socket.connect(new InetSocketAddress
8.                (ClientConnection.SERVER_IP, ClientConnection.SERVERPORT), 500);
9.
10.           o = new DataOutputStream(socket.getOutputStream());
11.
12.           in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
13.
14.       } catch (UnknownHostException e1) {
15.           e1.printStackTrace();
16.       } catch (IOException e1) {
17.           e1.printStackTrace();
18.       }
19.
20.   }
```

4.2 COMUNICAZIONE CON IL SERVER

Ogni qualvolta l'utente effettua una mossa, verrà invocata la funzione send() la quale provvederà alla trasmissione dei dati al Server.

Tale trasmissione avviene attraverso la creazione un nuovo Thread "t1" e facendo un'ovveride del suo metodo run, il quale provvede alla vera e propria trasmissione dei dati utilizzando il metodo writeBytes().

Quest'ultimo scrive una stringa sullo stream collegato come una sequenza di byte.

Subito dopo viene invocato il metodo flush() il quale si occupa di svuotare il flusso di output e forzare la scrittura dei byte memorizzati nel buffer.

Al termine della trasmissione, si provvede ad interrompere il Thread "t1" attraverso l'uso di del metodo interrupt().

```
1.    public void send(String str){
2.
3.        t1 = new Thread(new Runnable() {
4.
5.            @Override
6.            public void run() {
7.
8.                try {
9.                    o.writeBytes(str);
10.                   o.flush();
11.               } catch (IOException e) {
12.                   e.printStackTrace();
13.               }
14.
15.               t1.interrupt();
16.           }
17.       });
18.
19.       t1.start();
20.
21.   }
```

Ogni qualvolta il Server trasmetterà dati al Client, quest'ultimo sarà in ascolto e pronto a riceverli attraverso la seguente funzione rec().

Quest'ultima provvede a leggere i datai trasmetti dal Server utilizzando il metodo readLine(), il quale legge una sequenza di caratteri fino al carattere "\n" o "\r".

Una volta fatto ciò, si provvede al "replace" del carattere speciale terminatore "\n" e al return di quanto letto.

```
1.      public String rec(){
2.
3.          try{
4.              return in.readLine().replace("\n", "");
5.          } catch (IOException e) {
6.              e.printStackTrace();
7.              return null;
8.          }
9.
10.     }
```