UNIVERSITÀ DEGLI STUDI
DI NAPOLI FEDERICO II

Dipartimento di Ingegneria Elettrica e Tecnologie dell'Informazione Corso di Laurea in Informatica

Insegnamento di Laboratorio di Sistemi Operativi

Anno accademico 2020/2021

Tic Tac Toe

Autori:
Stefano Di Nunno (N86002170)
Ciro de Cristofaro (N86002370)

Docenti:
Prof. Marco Faella

INDICE

- Panoramica del programma (Client e Server)
- Compilazione
- Protocollo di comunicazione
- Dettagli implementativi

Panoramica del programma (Client)

Supposto che il server sia già stato avviato precedentemente, il client per poter giocare dovrà, in una fase preliminare, inserire le specifiche per instaurare una nuova connessione al server di gioco.

```
>
>    ./client <SERVER_IP> 5001
>    Insert your nickname: stiv
>
```

Tale fase consiste nel inserire l'indirizzo IP del server, la rispettiva porta di rete ed un nickname.

```
>
>    Tic Tac Toe Online
>
>    [1] Play Game
>    [2] Championship
>    [0] Exit
>
>    Choose an option: ...
>
```

Fatto ciò l'utente si ritroverà di fronte ad un menù come quello appena mostrato, e potrà quindi decidere se: iniziare una nuova partita, visualizzare la classifica o chiudere il programma.

La lista delle azioni che si possono compiere è indicizzata con degli identificativi numerici che, in base all'azione desiderata, l'utente potrà digitare. Digitando `'1'`, per esempio, l'utente potrà iniziare una nuova partita.

```
>
>    Choose an option: 1
>
```

Per poter iniziare una nuova partita è necessario che sul server ci siano attualmente collegati due utenti. Verificata tale precondizione sarà possibile effettuare la partita e quindi visualizzare la board e le mosse del rispettivo avversario.

```
>
>    Waiting your turn
>
>        |     |
>     X  |  X  |  -
>   _____|_____|_____
>        |     |
>     O  |  O  |  -
>   _____|_____|_____
>        |     |
>     -  |  -  |  -
>        |     |
>
>    stiv (X), it's your turn, choose a valid cell (from 1 to 9):
>
```

È possibile effettuare una mossa digitando un numero compreso tra uno e nove. La board di gioco è infatti tradotta nel seguente modo:

```
>
>          |     |
>       1  |  2  |  3
>     _____|_____|_____
>          |     |
>       4  |  5  |  6
>     _____|_____|_____
>          |     |
>       7  |  8  |  9
>          |     |
>
>
```

Per cui digitando `'5'` l'utente occuperà la cella centrale col simbolo (`X` o `O`) assegnatogli precedentemente.

Una volta conclusa la partita, automaticamente ci sarà l'aggiornamento degli score di ogni giocatore. Tali informazioni saranno visibili visualizzando la classifica ovvero digitando l'identificativo `'2'` dal menù.

Panoramica del programma (Server)

Il server ha il ruolo di gestire un numero arbitrario di collegamenti e di partite contemporaneamente. Una volta avviato comparirà la seguente schermata.

```
>
>    Server running...
>
```

Da questo momento sarà possibile visualizzare i client che stanno effettuando delle connessioni al server e visualizzare le corrispondenti richieste. Le richieste possibili sono: richiesta di una nuova partita, richiesta di visualizzazione della classifica e richiesta di disconnessione dal server.

```
>
>    stiv: Connected
>    stiv: Championship request
>    stiv: Connected
>    stiv: New game request
>    ric: Connected
>    ric: New game request
>    stiv: Closing connection
>    ric: Closing connection
>
```

Per poter terminare l’esecuzione del server basterà digitare `CTRL + C`.

Compilazione

Per facilitare la compilazione del programma è stato scritto un piccolo script. Basterà posizionarsi nella cartella del progetto col comando `cd` e successivamente eseguire lo script, ovvero:

```
>
>    cd .../Progetto
>    ./script.sh
>
```

Se già si è posizionati nella cartella del progetto e si vogliono avviare il client ed il server, rispettivamente, i comandi per far ciò saranno:

```
>
>    ./client <SERVER_IP> 5001
>    ./server
>
```

La porta utilizzata di default è la 5001. Cloud server IP: 51.136.61.94

Protocollo di comunicazione tra client e server

Quando il client effettua la connessione al server riceverà uno tra i due seguenti messaggi:

```
>
>    server: New user created
>
```

Questo messaggio appare quando il nickname digitato non era mai stato utilizzato fin ad ora. Per cui il server aggiungerà una nuova entry, con questo nickname, al file della classifica.

```
>
>    server: Welcome back
>
```

Questo messaggio invece indica che questo nickname è già stato utilizzato in precedenza per cui è già presente nel file della classifica e già possiede uno score. Lo score è il numero di vittorie, di sconfitte e di pareggi accumulati in partite precedenti. Trattandosi di un contesto multithreaded, l'accesso al file contenente la classifica avviene in modo sincronizzato, ciò rende sicuro il codice da eventuali fenomeni di race-condition. Il meccanismo di sincronizzazione adottato è quello dei mutex, maggiori dettagli saranno dati nei capitoli successivi.

A questo punto il client sarà in grado di effettuare delle richieste al server e verrà mostrato il menù.

```
>
>    Tic Tac Toe Online
>
>    [1] Play Game
>    [2] Championship
>    [0] Exit
>
>    Choose an option: ...
>
```

Il server resta in attesa che arrivi una nuova richiesta; ogni richiesta è identificata da un identificativo numerico: `'1'` richiesta di gioco, `'2'` richiesta di visualizzazione della classifica e `'0'` richiesta di disconnessione dal server.
Allora l'utente dovrà immettere su stdin uno di questi tre identificativi ed il client automaticamente effettuerà la richiesta.

Se l'utente sceglie di giocare una partita, il server gestirà la richiesta nel seguente modo:

Ci sono due possibilità:

- Non c’è nessun giocatore attualmente disponibile per giocare. Per cui il giocatore che ha appena richiesto di iniziare una nuova partita sarà aggiunto in una lista d’attesa e su stdout del client apparirà il seguente messaggio:

```
>
>   server: Wait another player
>
```

A questo punto il client resterà in attesa finché un altro giocatore non richiederà di effettuare una partita. Solo a questo punto infatti il server potrà svuotare la lista d’attesa ed avviare la nuova partita tra i due giocatori.

- C’è già un giocatore in attesa. Per cui il server svuota la coda e avvia la partita. In questo caso verrà subito assegnato un simbolo (O o X) al giocatore; che attenderà il suo turno per iniziare una nuova partita con l’avversario. Sullo stdout del client appariranno i seguenti messaggi:

```
>
>  You're the player with symbol: O
>  Waiting your turn ...
>
```

Se l’utente fa richiesta di visualizzare lo stato della classifica: in questo caso client e server si scambiano messaggi di dimensione fissa di 128 bytes. Maggiori dettagli saranno dati nei capitoli successivi.

Dettagli implementativi (classifica.h)

La classifica degli score è conservata in un file nominato ‘championship.txt’. Questo file è strutturato nel seguente modo:

```
>
>    nickname1-ww-ll-dd
>    nickname2-ww-ll-dd
>    ...
>
```

Su ogni riga è presente il nickname ed il numero di vittorie, di sconfitte e di pareggi accumulati dal giocatore. Le funzioni atte alla gestione di questo file sono nella libreria `'classifica.h'`.

Come già detto prima, per quanto riguarda la classifica, client e server si scambiano messaggi di 128 byte. A dimostrazione di ciò ecco una parte di codice della funzione `'send_championship'` che si occupa esattamente di questo:

```
>
>    void send_championship(int client_sd) {
>
>       ...
>       // BUFFSIZE == 128, variabile globale
>       // fd1, file descriptor del file della classifica
>       while ((re = read(fd1, buffer, BUFFSIZE)) > 0) {
>
>          int w = write(client_sd, buffer, re);
>
>       ...
>       }
>    ...
>
```

Per cui, da parte del client, c’è una funzione che si occupa di ricevere queste informazioni e stamparle a video. Tale funzione è nominata `show_championship`, ed eccone una parte del codice:

```
>
>   void show_championship(int server_sd) {
>
>      ...
>      // BUFFSIZE == 128, variabile globale
>      while((re = read(server_sd, buffer, BUFFSIZE)) > 0)
  {
>         if(re == -1) {
>            perror("read");
>            exit(1);
>         }
>         buffer[re] = '\0';
>         write(STDOUT_FILENO, buffer, re);
>      }
>   ...
>
```

I giocatori contenuti nella classifica vengono mostrati in ordine di registrazione al gioco e non in base ai punteggi.

Dettagli implementativi (Client – Server)

Il server è in grado di gestire più client in modo concorrente. In particolare il server è di tipo concorrente multi-threaded, ciò vuol dire che ad ogni client sarà dedicato un thread d'esecuzione.

Il processo di gestione di un client è svolto dalla funzione `'manage'`. Come si può notare anche dal codice (in `server.c`),

```
>
>    ...
>    // Server loop
>    while(1) {
>
>     client_len = sizeof(client_addr);
>
>     if ( (client_sd = accept(sd, (struct sockaddr *)
>            &client_addr, &client_len)) < 0 ) {
>        perror("accept");
>        return 1;
>     }
>
>     thread_sd = malloc(sizeof(int));
>     *thread_sd = client_sd;
>
>     pthread_t tid;
>     pthread_create(&tid, NULL, manage, (void *) thread_sd);
>                                    ^^^^^^
>    }
>
```

ad ogni nuova connessione viene lanciato un thread.

Dal momento che, come già detto, il contesto è multi-threaded è stato necessario adottare un meccanismo di sincronizzazione sia per l'accesso al file della classifica sia all'accesso alla struttura dati incaricata alla gestione della lista d'attesa.
In particolare, il meccanismo adottato è stato quello dei mutex.

```
>
> pthread_mutex_t wl_mutex = PTHREAD_MUTEX_INITIALIZER;
> pthread_mutex_t championship_mutex = PTHREAD_MUTEX_INITIALIZER;
>
```

Il `wl_mutex` è il mutex dedicato alla waiting-list. Il `championship_mutex` è il mutex dedicato al file della classifica.
L'utilizzo di tali meccanismi è necessario per non incorrere in fenomeni di race-condition.

```
>
> if(pthread_detach(tid) != 0) {
>         perror("detach error");
>         return 1;
> }
>
```

I thread sono lanciati in detach-state. Nel caso del server infatti non è necessario che alcun thread sia joinable. Quando un detach-thread termina, tutte le risorse occupate fino a quel momento dal thread vengono automaticamente rilasciate al sistema. È buona norma infatti chiamare sempre `pthread_detach` o `pthread_join` per ogni thread che viene creato dall'applicazione.