

Laboratorio di Sistemi Operativi

Progetto II-III

Programmazione C – Programmazione di Sistema

Anno 2010/2011

Giorgio Aresu 103121

Sara Bressan 101039

Premessa

L'intero progetto è nato dalla fusione di due progettini di laboratorio assegnatici durante quest'anno accademico.

Il progetto riguardante la programmazione in C chiedeva di sviluppare uno a scelta tra due programmi: il primo per giocare a Tic-Tac-Toe in multiplayer e l'altro per poter giocare contro un'intelligenza artificiale.

Anche per la parte di programmazione di sistema si poteva scegliere una tra due opzioni, entrambe da realizzare in linguaggio C:

- sviluppare due programmi, Tic-Tac-Toe_Multiplayer_Server e Tic-Tac-Toe_Multiplayer_Client che, utilizzando le chiamate di sistema, permettano a due persone collegate allo stesso sistema da diverse console (reali o virtuali), di giocare a tic-tac-toe
- sviluppare due programmi, sempre di tipo client-server per poter giocare a tic-tac-toe da remoto contro un'intelligenza artificiale gestita dal server.

Era richiesto che l'input fosse ricevuto tramite delle coordinate indicanti la posizione desiderata, ma è stato concesso dal docente di utilizzare lo schema del tastierino numerico, che abbiamo deciso di adottare.

Il progetto si è rivelato presto interessante, tant'è che abbiamo deciso di realizzare tutti i punti senza limitarci a scegliere quali svolgere, dando luogo quindi a un unico progetto che permetta di effettuare tutte le varianti di partita, cioè singolo e multigiocatore sia in locale che online (con la possibilità di giocare da pc collegati a internet o appartenenti a qualunque rete)

Sviluppo del progetto

Per non essere troppo dispersivi, abbiamo deciso di realizzare due soli file eseguibili: un client e un server. Per la parte grafica del client ci siamo serviti della libreria *ncurses*, che fornisce numerose funzioni di base con cui costruire l'interfaccia.

Il progetto è composto da otto file sorgenti:

- main.c con relativo header: nucleo del progetto, contiene la sola funzione main che si occupa di indirizzare il tipo di partita, e chiamare tutte le funzioni necessarie, in base alla scelta dell'utente. Avvia le partite in locale e la parte client di quelle in rete, con l'ausilio delle funzioni contenute nei file aus.
- aus.c con relativo header: contiene la definizione delle strutture utilizzate nel gioco, tutte le funzioni utilizzate in main.c e quelle in comune col server.
- server.c con relativo header: contiene il codice del server e tutte le sue funzioni ad eccezione di quelle comuni col client che sono definite in aus.c.
- precompDefs.h: contiene solo e esclusivamente le costanti di precompilazione.
- makefile: contiene i comandi per la compilazione dei file.

All'avvio del programma troviamo un menù nel quale è possibile scegliere il tipo di partita, avviare un server sul pc, leggere le informazioni sul gioco ed uscire. Scegliendo di avviare un server viene aperto un altro terminale nel quale si esegue il server.

Abbiamo creato delle strutture per definire il giocatore (*player*), le celle (*cell*) e le righe nel menù (*menuItem*):

- *player*: è composta dal nome, di dieci caratteri più '\0', da una variabile booleana (*hasCross*) che indica il simbolo utilizzato dal giocatore e un'altra variabile booleana (*hasSCP*) che indica il colore scelto. Il giocatore all'inizio di qualsiasi tipo di partita può decidere il proprio simbolo e il colore; quando si gioca in multiplayer locale il giocatore1 decide con che simbolo e colore giocare, e al giocatore2 automaticamente viene assegnato simbolo e colore rimasti, mentre in quello in rete ognuno può scegliere il colore e il simbolo desiderati e sarà il client a visualizzare le mosse dell'avversario in base alle preferenze del proprio giocatore.
- *cell*: è formata da due variabili *y* e *x* che indicano le coordinate della cella all'interno della griglia e da un intero che indica lo status della cella, che può assumere tre valori: 0 se libera,

- 1 se occupata dal giocatore1 e 2 se occupata dall'altro (giocatore2 o IA).
- *menuItem*: è composta da delle variabili y, x che indicano le coordinate di quella voce rispetto all'area che le conterrà e da un vettore di char che contiene l'etichetta della voce.

L'intelligenza artificiale da noi implementata ha una strategia piuttosto semplice. Come prima cosa controlla se con qualche mossa può vincere o impedire all'avversario di farlo, altrimenti decide dove mettere il segno in base alle percentuali specificate nel file precomDefs.h, più sono elevate più sarà difficile vincere, ma anche monotono il gioco (tenderà a finire quasi sempre in parità o con la sua vittoria).

Il controllo della vittoria è implementato nella funzione **checkVictory**, che restituisce un valore booleano, 1 se si ha vinto e 0 altrimenti. La funzione si occupa di controllare le varie colonne, righe e diagonali per decretare il vincitore.

Il server si occupa di accettare connessioni e avviare nuovi processi a cui passarle per restare sempre libero di accettarne di nuove; analizzando gli aspetti di ogni partita verrà spiegato come funziona più in dettaglio.

Per evitare di dover attendere potenzialmente all'infinito la risposta del server (recv e send sono chiamate bloccanti), abbiamo introdotto dei timeout per la ricezione di messaggi in alcuni punti critici; per esempio se un giocatore avesse avviato una partita in multiplayer in rete, il client sarebbe rimasto bloccato fino alla ricezione di altre connessioni, e per uscire dal gioco l'utente avrebbe dovuto chiudere brutalmente il programma. Per evitarlo abbiamo impostato in quel punto un timeout alla ricezione di dati del socket per permettere, con un ciclo, di leggere eventuale input da tastiera durante l'attesa, e quindi riconoscere la pressione di F2 per tornare al menu principale interrompendo la connessione.

Analizziamo ora gli aspetti principali di ogni tipo di partita:

Giocatore vs IA (locale)

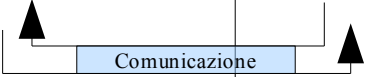
In questa modalità di gioco, dopo aver chiamato **initalizeGame** per inizializzare il gioco, la funzione **drawInGame** si occupa di disegnare il tavolo di gioco e, in base al turno (viene scelto a caso chi deve iniziare), fa giocare l'intelligenza artificiale oppure il giocatore, controllando la correttezza dell'input (celle già occupate). Infine assegna la vittoria o il pareggio. Con la funzione **drawNewGame** viene chiesto se si vuole effettuare un'altra partita o tornare al menù principale.

Multiplayer (locale)

Come per il giocatore singolo, initializeGame e drawInGame preparano la partita, ma questa volta **drawInGame** non alterna l'input da tastiera con l'IA bensì l'input da tastiera di entrambi i giocatori.

Giocatore vs IA (via rete)

Per spiegare il funzionamento della partita singleplayer via rete usiamo questa tabella con pseudocodice:

Server	Processo Figlio	Client 1
accept() <i><scambio di messaggi></i> recv(client1) fork() <i><pronto per nuova partita></i>	 <i>Processo Avviato</i> <i><inizio partita></i> 	connect() <i><scambio di messaggi></i> send(server, "single") <i><inizio partita></i>

Il server inizia a scambiare messaggi di identificazione con il client che si collega. Quando il client richiede una partita in singleplayer, il server crea un nuovo processo, che si occupa di gestire la partita in tutti i suoi aspetti. Appena lanciato il nuovo processo il server rimane in attesa di nuove connessioni. Al termine della partita, in caso di disconnessione del client o di errori di connessione il processo figlio (quello che gestiva la partita) viene terminato.

Multiplayer (via rete)

Server	Processo Figlio	Client 1	Client 2
waitingPid=0 accept() <i><scambio di messaggi></i>		connect() <i><scambio di messaggi></i> send(server, "multi")	
recv(client1) fork()	<i>Processo Avviato</i>		
waitingPid=PidFiglio accept() <i><scambio di messaggi></i>	<i>sigStop()</i> <i>Processo Bloccato</i>		connect() <i><scambio di messaggi></i> send(server, "multi")
recv(client2) waitingPid≠0 sigCont(figlio)	<i>Risveglio Processo</i>		
send(client2, riconnetti) lock <i>Attesa di Sblocco</i>	accept() <i><scambio di messaggi></i>		recv(server) disconnect()
<i>Ripresa</i> <i><pronto per nuova partita></i>	unlock(server) <i><inizio partita></i>	<i><inizio partita></i>	connect() <i><scambio di messaggi></i> <i><inizio partita></i>

```

sequenceDiagram
    participant S as Server
    participant PF as Processo Figlio
    participant C1 as Client 1
    participant C2 as Client 2

    Note over S: waitingPid=0  
accept()  
<scambio di messaggi>
    Note over C1: connect()  
<scambio di messaggi>  
send(server, "multi")
    Note over PF: Processo Avviato
    Note over S: recv(client1)
    Note over S: fork()
    Note over S: waitingPid=PidFiglio  
accept()  
<scambio di messaggi>
    Note over PF: sigStop()  
Processo Bloccato
    Note over C2: connect()  
<scambio di messaggi>  
send(server, "multi")
    Note over S: recv(client2)  
waitingPid≠0  
sigCont(figlio)
    Note over PF: Risveglio Processo
    Note over S: send(client2, riconnetti)  
lock  
Attesa di Sblocco
    Note over PF: accept()  
<scambio di messaggi>
    Note over S: Ripresa  
<pronto per nuova partita>
    Note over PF: unlock(server)  
<inizio partita>
    Note over C1: <inizio partita>
    Note over C2: connect()  
<scambio di messaggi>  
  
<inizio partita>

    S-->>PF: Comunicazione
    PF-->>C1: Comunicazione
    C1-->>S: Comunicazione
    S-->>C2: Comunicazione
    
```

Quando arriva questa seconda richiesta il server risveglia il processo, che si pone in attesa di connessioni, mentre il server si blocca. Viene richiesto al secondo client di riconnettersi, affinché la connessione stavolta avvenga non col server ma col processo figlio. Appena il client si è riconnesso¹, il processo informa il server di riprendere a accettare connessioni. A questo punto la partita inizia sul processo figlio, che è in grado di comunicare con entrambi i client (ma i client non possono bypassare il processo per comunicare direttamente tra loro), mentre il server è pronto a accettare altri client.

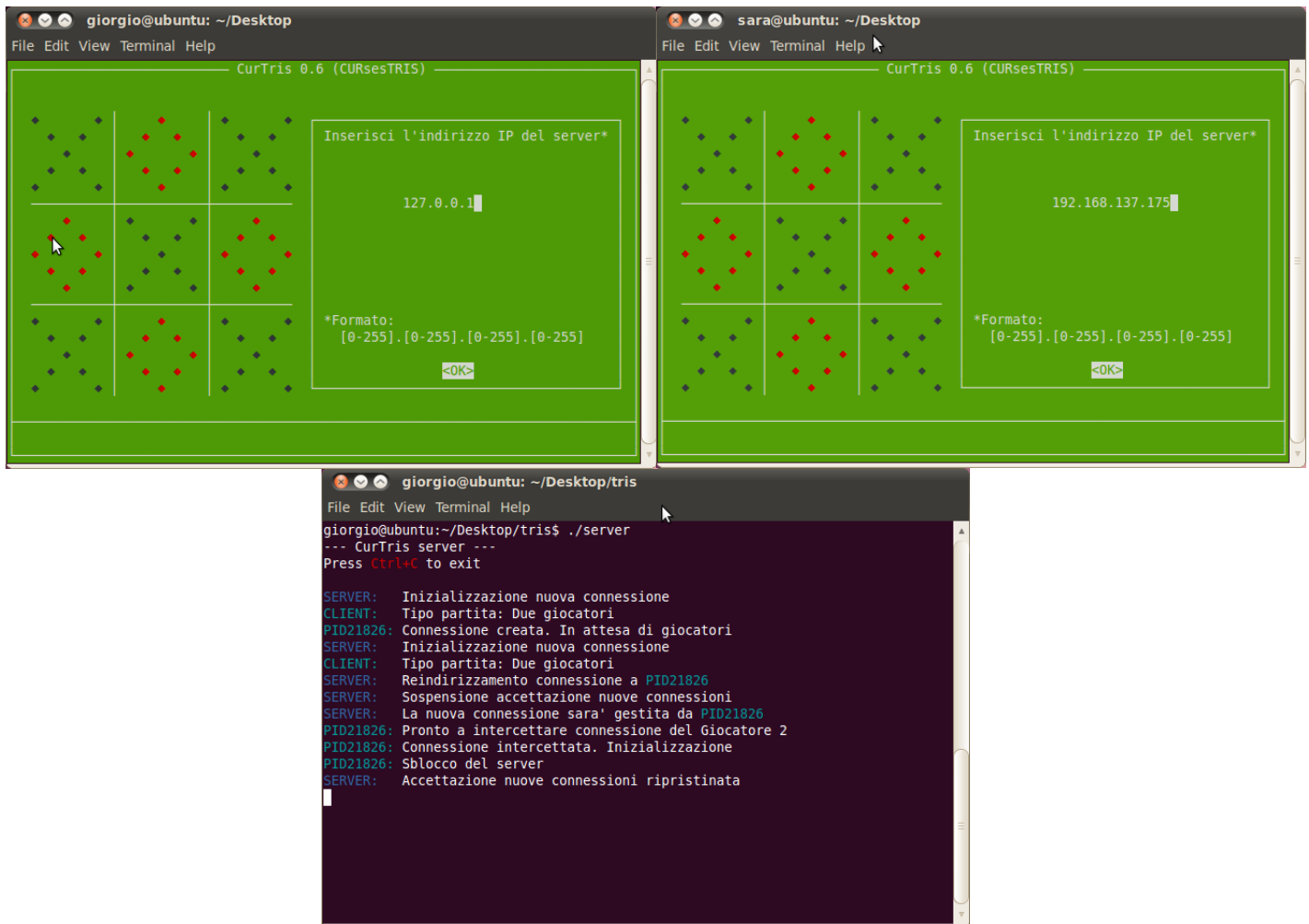
1 - Supposto che nessun altro client si connetta in questo breve lasso di tempo, altrimenti il nuovo client verrebbe catapultato a sua insaputa in una partita e il client che si riconnetterà verrà considerato come un nuovo client

Test

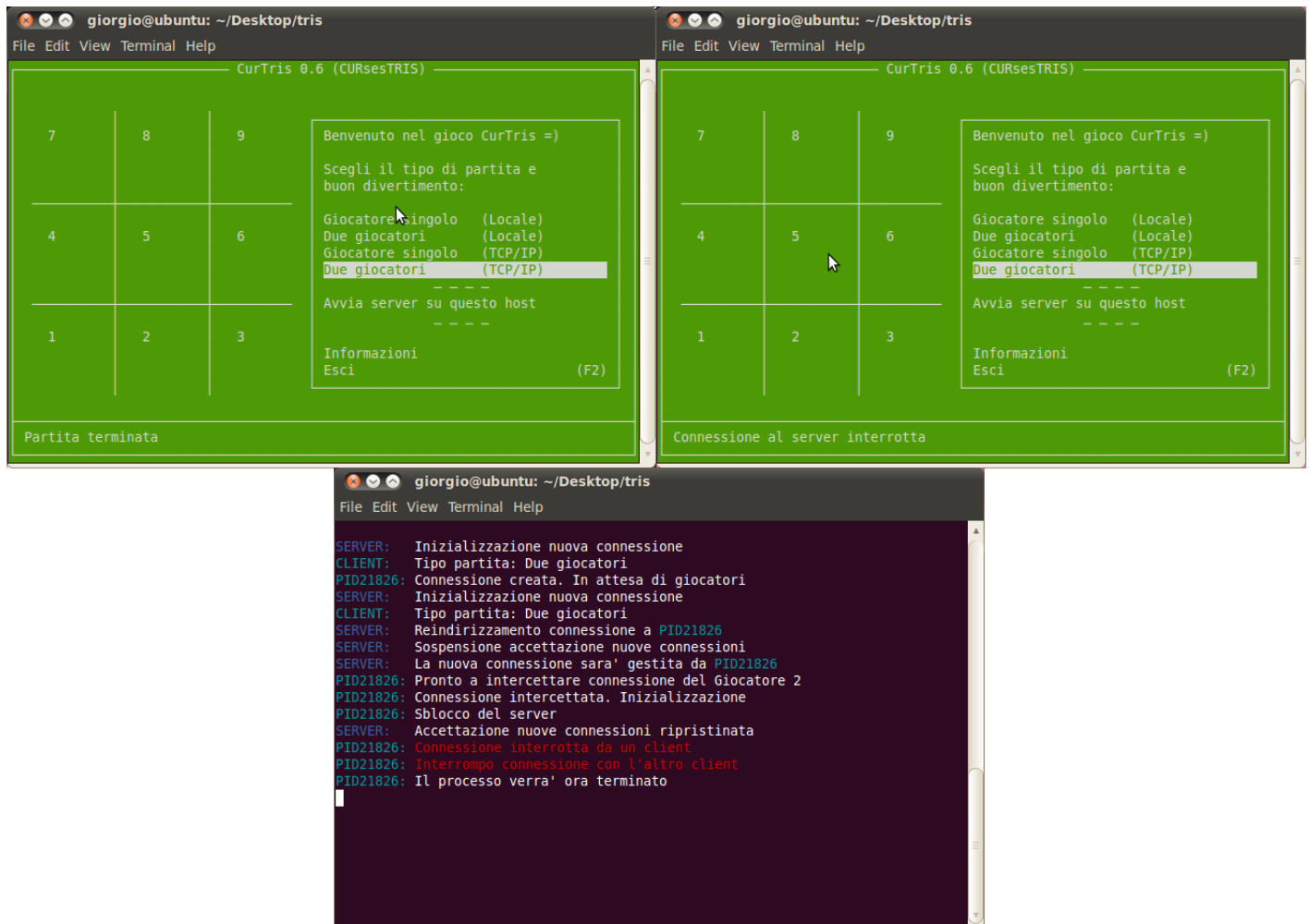
Per controllare l'effettivo funzionamento del programma abbiamo eseguito alcuni test sia in locale avviando un server e i vari client, sia tramite collegamento wi-fi avviando un server su un pc e provando a connettersi tramite l'altro e viceversa. Per maggiore semplicità riporteremo solo degli screenshot dei test più significativi.

Connessione tra due pc: un client si connette all'indirizzo di loopback essendo sullo stesso pc del server, mentre l'altro usa l'indirizzo IP in cui è in esecuzione il server.

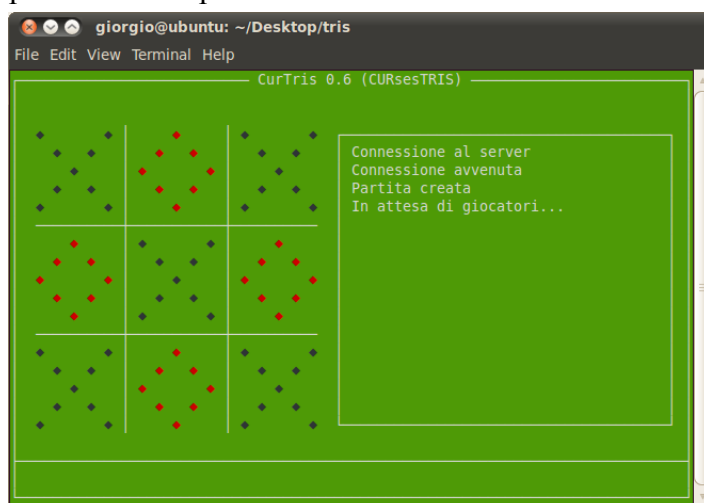
Quando entrambi i giocatori sono connessi la partita inizia regolarmente, quindi le comunicazioni avvengono correttamente.



Un client si disconnette durante la partita: durante la partita (indipendentemente da chi sia il turno) un client si disconnette. Il server se ne accorge e chiude la connessione. Alla prima mossa dell'altro client questo si accorge che la connessione è stata chiusa e torna al menù principale.

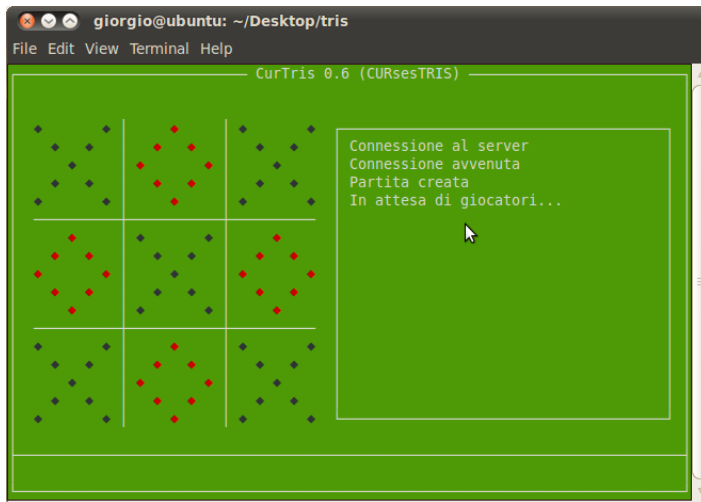


Un client si disconnette durante l'attesa di un secondo giocatore: durante l'attesa il giocatore decide di disconnettersi. Quando avviene una nuova connessione al server questo se ne accorge e termina il processo della partita mai iniziata.



Il giocatore disconnette premendo F2

Un altro client si connette al server:



Il server termina il vecchio processo e ne avvia un altro.

