Elaborato 2: System Call A.A. 2020/2021

L'elaborato consiste di due applicazioni in linguaggio "C", "*F4Server.c*" e "*F4Client.c*", che sfruttano le system call SYSTEMV, implementando il gioco "FORZA 4", basato sulle regole del classico gioco da tavolo.

(Il gioco è implementato con alcune varianti ed in grado di funzionare in ambiente UNIX/LINUX da 2 utenti).

Strutture di gioco comuni ad entrambi i programmi

Giocatori:

```
typedef struct giocatore_s{ //struttura utilizzata per gestire i giocatori
        char nome[50]; //stringa col nome del giocatore
        int status; //intero che gestisce la partita (0->ancora in corso, 1->vincitore, -1->perdente,2->matrice piena, parità)
        char simbolo; //il gettone che il giocatore usa nel campo da gioco
        int mossa; //variabile per memorizzare le varie giocate
        int pid; //pid di ogni giocatore, gestito da client diversi
}giocatore_t;
```

Molte delle variabili che fanno parte della struttura dei giocatori vengono comunicate tra client e server attraverso delle FIFO.

Casella:

```
typedef struct casella_s{
        char valore; //gettone presente nella casella (" "->vuoto, "X"->esempio)
        int id; //ogni casella ha un numero ( |1|2|3|4|5| a capo |6|7|8|9|10| ecc.)
        int idDiagonale; //id della casella successiva nella diagonale (per il controllo del vincitore in diagonale)
}casella_t;
```

Il campo è un array bidimensionale di caselle in memoria condivisa, gestito tramite un puntatore da entrambi i programmi:

```
casella_t (*campo)[n] = shmat(ShmID,NULL,0);
```

La struttura era stata progettata a questo modo per permettere di implementare il controllo diagonale, che non è stato fatto.

Funzioni comuni ad entrambi i programmi

Void stampaCampo(casella_t campo[][n],int n,int m)

Questa funzione stampa il campo, utilizzando il suo essere un array bidimensionale per stampare ordinatamente secondo righe e colonne, attraverso due for, uno stampa le colonne, uno gestisce le righe.

Int calcolaRiga(casella_t campo[][n],int n,int mossa)

Calcola, data la colonna in cui si vuole inserire il gettone, qual è la prima riga libera, iniziando a controllare ovviamente dall'ultima. Ritorna l'indice della riga.

F4Server.c

Funzionamento generale: il server gestisce la creazione del campo e la gestione del gioco, compresi i turni client/server, ad ogni turno comunica ad entrambi i client se hanno vinto o perso, se il gioco è ancora in corso o se la matrice di gioco risulta piena (caso in cui viene dichiarato il pareggio). Vediamo ogni funzione con attenzione ai punti più importanti.

Funzioni:

Void segnale(int)

Gestisce il segnale ctrl^c: se viene premuto una sola volta viene stampato un avviso che comunica che se verrà premuto di seguito una seconda volta (5 secondi per farlo) il gioco verrà terminato.

Se viene premuto una seconda volta, al termine dei 5 secondi il server comunica ad entrambi i client la terminazione del gioco, tramite un segnale (SIGXFSZ). Si occupa di eliminare la memoria condivisa, le fifo create e i semafori, infine termina egli stesso (exit (0)).

Se invece non viene premuto, torna al punto in cui era, ctrl^c non viene però azzerato, quindi alla pressione di un secondo ctrl^c il gioco termina senza stampare alcun avviso.

Void abbandono(int)

Se uno dei client abbandona la partita (tramite la pressione del doppio ctrl^c) il server riceve un segnale (SIGUSR1). Il server si occupa anche in questo caso di eliminare la memoria condivisa, i semafori, e la sua fifo per poi uscire.

Void stampaCampo(casella_t campo[][n],int n,int m)

Come sopra

ControllaRiga, ControllaColonna e MatricePiena

Ad ogni turno il server dovrà verificare che la matrice non sia piena, per poter continuare a giocare, e se uno dei due client ha vinto.

char controllaRiga(casella_t campo[][n],int n,int m)

char controllaColonna (casella_t campo[][n],int n,int m)

ControllaRiga e ControllaColonna contano quanti simboli uguali di seguito trovano (rispettivamente nella riga e nella colonna) tramite due for, quando la variabile conta (che tiene appunto conto di quanti valori identici di seguito vengono trovati) arriva a 4, ritornano il char contenuto nelle caselle uguali, così che il server possa identificare un vincitore.

int matricePiena(casella_t campo[][n],int n,int m)

Al contrario MatricePiena ritorna 0 quando trova uno spazio nei char " ", cioè quando c'è almeno una casella vuota, e 1 quando non ne trova nessuno, quindi il campo è pieno.

Int creaChiave

Crea la chiave per semafori e memorie condivise tramite la funzione ftok.

Void campoVuoto(casella_t campo[][n],int n,int m)

Inizializza il campo all'inizio di ogni partita: ogni casella viene impostata come vuota (" "), le viene assegnato il suo id e il suo idDiagonale (calcolato come il valore dell'id + il numero di colonne +1).

Int calcolaRiga(casella_t campo[][n],int n,int mossa)

Come sopra.

Main

////////// -IMPOSTAZIONI DI BASE- //////////

Il server inizia gestendo la riga di comando con cui viene lanciato. Infatti esso si aspetta di essere lanciato come:

./F4Server.c <int num colonne> <int num righe> <char simbolo giocatore 1> < char simbolo giocatore 2>

E stampa un avviso del comando corretto quando non viene fatto (ovvero quando gli argomenti sono troppo pochi) - es. ./F4Server 5 5 X O

Dopo di che inizia ad impostare le cose necessarie:

-Verifica che la grandezza del campo rispetti quella minima (5x5) altrimenti continua a chiederla;

-Imposta come simboli dei due giocatori (creati come variabili globali) quelli inseriti, e come status iniziale 0 (ovvero partita in corso);

-Crea il segmento di memoria condivisa e ci associa il puntatore, necessario alla gestione del campo.

////////// -CREAZIONE E GESTIONE DELLE FIFO E DEI SEMAFORI- //////////

Il passo successivo è la creazione della FIFO del Server e l'apertura delle FIFO dei client.

Il Server legge i nomi dei due giocatori (scritti sulla Fifo del Server dai Client) e poi apre le due fifo client o una sola, infatti tra le opzioni di gioco c'è la modalità Single Player (ovvero Giocatore Singolo) che viene passato come NO dal primo giocatore, se vuole giocare da solo. In base a questo, tutto il programma

utilizza una variabile (giocaDaSolo) per gestire le due modalità. Leggerà inoltre il pid del client (utile alla gestione dei segnali).

Se il giocatore è in modalità single player, il server aprirà solo la fifo "primoPathToClientFIFO" e la utilizzerà per comunicare:

-simbolo del giocatore,

-grandezza del campo,

-proprio pid (sempre per la gestione dei segnali)

Se i giocatori sono due, il server aprirà anche "secondoPathToClientFIFO" comunicando le stesse cose con in aggiunta il pid dell'avversario (come sopra).

Le FIFO rimangono aperte per comunicare lo status dei giocatori all'avanzare della partita.

Infine il server crea il semaforo necessario alla gestione dei turni Server/Client.

////////// -CICLO DI GIOCO- //////////

I turni di gioco sono gestiti tramite un Do-While, la condizione necessaria alla perpetuazione del ciclo è che gli status dei giocatori siano uguali e pari a 0 (quindi la partita è ancora in corso):

All'inizio di ogni turno il Server imposta il semaforo su –1, così infatti si blocca e dovrà attendere di essere sbloccato da un Client. Una volta sbloccato il Server esegue il controllo della Matrice:

-se trova la matrice piena (matricePiena()) stampa un messaggio di avviso e comunica ad entrambi i client (o al giocatore in modalità single player, tramite le rispettive fifo) che il loro status passa a 2 (ovvero matrice piena e partita terminata in parità) per poi interrompere il ciclo.

-altrimenti passa ad eseguire i controlli di riga, colonna e diagonale (non presente): le funzioni ritornano un char assegnato alla variabile vincitore. Viene confrontato con i due simboli.

-se trova un vincitore, e questo ha il simbolo del primo giocatore comunica 1 sulla prima fifo e –1 sulla seconda, per poi interrompere il ciclo; se è uguale al simbolo del secondo giocatore fa l'esatto opposto.

-infine se le funzioni hanno ritornato 0 (quindi diverso da entrambi i simboli) si può continuare a giocare: in modalità single player modificherà casualmente la matrice (viene generato un numero casuale tra le possibili colonne e calcolata la prima riga disponibile, tramite l'apposita funzione) e comunicherà al giocatore che può continuare a giocare, altrimenti comunicherà ad entrambi che possono continuare (status = 0).

////////// -CHIUSURA DELLA PARTITA- //////////

All'uscita dal ciclo vengono chiuse/rimosse le fifo, eliminata la memoria condivisa ed il semaforo, così da poter cominciare una nuova partita.

F4Client.c

Funzionamento generale: il client gestisce il gioco dal lato dell'utente giocatore: si occupa di stampare la matrice ad ogni turno e inserire il gettone del giocatore; comunica poi lo status del giocatore durante la partita.

Funzioni:

Void segnale(int)

Gestisce il segnale ctrl^c: se viene premuto una sola volta viene stampato un avviso che comunica che se verrà premuto di seguito una seconda volta (5 secondi per farlo) il gioco verrà terminato.

Se viene premuto una seconda volta, al termine dei 5 secondi il server comunica ad entrambi i client la terminazione del gioco, tramite un segnale (SIGUSR1). Si occupa di eliminare le fifo create per poi terminare (exit (0)).

Se invece non viene premuto, torna al punto in cui era, ctrl^c non viene però azzerato, quindi alla pressione di un secondo ctrl^c il gioco termina senza stampare alcun avviso.

Void abbandono(int)

Se uno dei client abbandona la partita (tramite la pressione del doppio ctrl^c) il client, come abbiamo visto sopra, riceve un segnale (SIGUSR1). Dato che il server e il client che ha abbandonato si occupano di ogni pulizia, il client stampa semplicemente un messaggio di avviso prima di uscire.

Void terminazione(int)

Se è il server a terminare la partita, i client ricevono un segnale (SIGXFSZ), in questo caso è il server che si occupa di tutte le "pulizie" del caso, quindi i l client stampa semplicemente un messaggio di avviso prima di uscire.

Void stampaCampo(casella_t campo[][n],int n,int m)

Come sopra

Void modificaCampo(casella_t campo[][n],int colonna,int riga,char simbolo)

Modifica semplicemente il campo secondo i parametri ricevuti (l'operazione è eseguita in una funzione, poiché nel ciclo di gioco entrava in contrasto con altre operazioni, generando un mal funzionamento).

Int calcolaRiga(casella_t campo[][n],int n,int mossa)

Come sopra.

Main

////////// -IMPOSTAZIONI GENERALI- //////////

Il client inizia gestendo la riga di comando con cui viene lanciato, si aspetta che esso sia:

./F4Client.c <stringa nome utente> (optional ->) <NO> (modalità single player)

E stampa un avviso del comando corretto quando non viene fatto (ovvero quando gli argomenti sono troppo pochi) - es. ./F4Client Giocatore1 NO

Dopo di che inizia ad impostare le cose necessarie, ovvero crea la variabile per la gestione del giocatore (giocatoreClient) e vi copia il nome inserito;

////////// -CREAZIONE E GESTIONE DELLE FIFO E DEI SEMAFORI- //////////

Il passo successivo è l'apertura della fifo del Server, su cui viene scritto il nome del giocatore. In modalità single player (sempre gestita dalla variabile giocaDaSolo), il client trasferisce anche la stringa “NO” (provato ad utilizzare * e ‘*’ ma venivano entrambi letti come riferimenti all’ultimo file aperto nella directory di riferimento dal server). In seguito notifica anche il proprio pid al server.

Il Client poi apre la Fifo Client: essendo possibile la creazione di due fifo “primo PathToClienFIFO” e “secondoPathToClientFIFO”, il client verifica la presenza della prima: se esiste vuol dire che c’è già un giocatore e crea la seconda. In entrambi i casi vengono letti:

-simbolo del giocatore,

-grandezza del campo,

-pid del server (per la gestione dei segnali),

-pid dell’avversario, se presente (come sopra).

-chiave per poter accedere alla memoria condivisa

Nota: alla creazione delle fifo viene impostato anche il valore della variabile turno, necessaria a gestire l’alternarsi dei due turni tra i client. Il valore di 1 viene dato a chi apre per primo la fifo, che sarà quindi il primo a giocare e il valroe di due viene assegnato nel caso in cui sia il secondo a giocare.

Accede al segmento di memoria condivisa e ci associa il puntatore, necessario alla gestione del campo.

Infine, il client accede al semaforo creato dal server per la gestione dei turni Server/Client, grazie ad un altro passaggio tramite fifo della chiave.

////////// -TURNO EFFETTIVO DI GIOCO- //////////

I turni di gioco sono gestiti tramite un Do-While: a condizione necessaria alla perpetuazione del ciclo è che lo status del giocatore sia pari a 0 (quindi la partita è ancora in corso):

All’inizio di ogni turno il Client verifica quale valore ha la variabile turno. Se questa è impostata su 2, il giocatore rimane in attesa che sia giocata una mano dall’avversario (o dal computer) questa attesa avviene poiché il client attende che il server scriva il nuovo valore del suo status; il server però è momentaneamente bloccato dal semaforo, che verrà sbloccato solo dal client che gioca il turno. Una volta letto il valore dello status il valore della variabile turno passa a 1, così che al prossimo ciclo sarà questo client a giocare.

Se la variabile turno ha invece valore 1, è il turno del giocatore:

-viene stampata la matrice, così che il giocatore possa decidere in quale colonna inserire il proprio gettone,

-viene chiesto di inserire da tastiera il valore della colonna, il valore deve risultare coerente con la grandezza del campo e deve essere inserita una colonna non piena. In entrambi i casi viene richiesto l’inserimento, finché questo non è coerente,

-viene calcolata la riga necessaria (come sopra) e inserito il gettone,

-viene ristampata la matrice, con il gettone inserito,

-viene sbloccato il semaforo che bloccava il server, così che possa effettuare i controlli necessari,

-viene letto il nuovo status,

-se il giocatore non è in modalità single player viene impostata la variabile turno a 2, così che il prossimo turno tocchi all'avversario.

Finito questo if-else (relativo al turno), ve n'è un altro relativo al nuovo status:

-se legge 2 la matrice è stata trova la matrice piena (dal server), stampa quindi un messaggio di avviso che la partita è finita in parità,

-se legge 1 la partita è stata vinta, stampa quindi un messaggio di avviso,

-se legge –1 la partita è stata vinta dall'avversario, stampa quindi un messaggio di avviso

-se nessuno di questi if è verificato lo status sarà uguale a 0 e il ciclo continuerà ad essere eseguito fin quando la condizione non verrà più rispettata.

////////// -CHIUSURA DELLA PARTITA- //////////

All'uscita dal ciclo vengono rimosse le fifo, eliminato il collegamento alla memoria condivisa, così da poter cominciare una nuova partita.