Relazione Progetto

Multiplayer Game

Sistemi Operativi 1

2014/15



Willliams Rizzi Alessandro Bacchiega

Mat. 165098 Mat. 167391

Introduzione

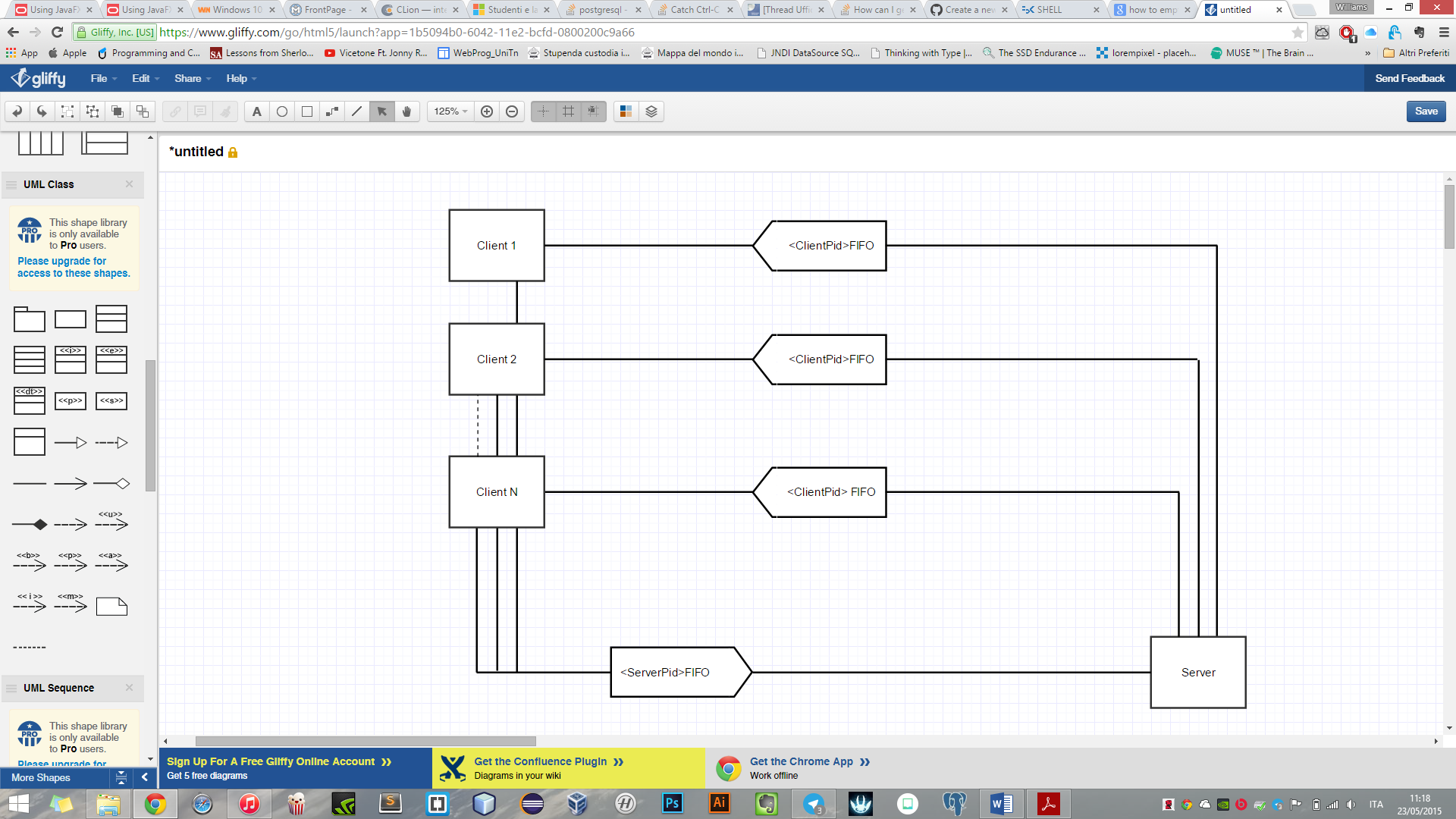
Il nostro progetto consiste nel creare un sistema client-server che ha il compito di simulare un gioco multiplayer nel quale un server sottopone ad ogni client una somma di due numeri compresi tra 0 e 99, ogni client deve tentare di dare la risposta corretta prima degli altri, fatto questo chi ha dato un risposta corretta guadagna un punto, mentre chi sbaglia ne perde uno, si passa quindi al round successivo fino a quando non si ha un vincitore.

Analizzando la consegna abbiamo ritenuto necessaria una distinzione logica fra le diverse entità (e relativi ruoli) partecipanti ad un’ipotetica partita. Sono stati individuati due attori, il primo dei quali è unico e gestisce il corretto svolgimento del gioco, vale a dire il *server*, mentre il secondo identifica un generico partecipante alla sessione di gioco, il *client*.

Organizzazione della rete di comunicazione

Per permettere lo scambio di messaggi che tra i clients ed il server, sia durante le fasi di inizializzazione della partita che durante lo svolgimento del gioco, si è deciso di organizzare la comunicazione all'interno del programma mediante una *pipe* particolare, detta *named pipe* o FIFO, uno strumento che consente lo scambio di messaggi fra processi che non hanno in comune antenati. In particolare, per la realizzazione di questo progetto è stato scelto di impiegare:

* Una FIFO condivisa (chiamata *MultiPlayerFifo*) mediante la quale i clients effettuano la domanda di iscrizione al server inviandogli il PID.
* Una FIFO dedicata per ciascun client (chiamata *<ClientPid>Fifo*) il cui nome è il PID del processo client. Su tale FIFO il client legge i numeri da sommare;
* Una FIFO condivisa ed unica (chiamata *<ServerPid>Fifo*) il cui nome è il PID del processo server. Su tale FIFO i clients scrivono al server il risultato della somma.



Struttura dei messaggi

Per rendere la comunicazione più sicura abbiamo deciso di codificare tutti i messaggi in transito sulle FIFO e decodificare gli stessi tramite la libreria “comunicazione.h” secondo la seguente struttura:

CARATTERE SPAZIATORE

END

SECONDA INFORMAZIONAE

PRIMA INFORMAZIONE

INIT

In init troviamo un carattere inizializzatore che definirà la sematica del messaggio, infatti un messaggio potrà essere di carattere comunicativo o numerale, a seconda del tipo di informazione in arrivo la funzione che si occupa di decodificare il messaggio avrà un valore di ritorno differente.

Struttura del programma

Successivamente si è passati ad un’analisi approfondita del gioco, mettendo in evidenza le diverse fasi che lo compongono, forti delle osservazioni suesposte.

* Avvio dei processi server e client
* Iscrizione
* Gioco
* Terminazione

### Avvio dei processi server e client

Durante la fase di avvio il server verifica la presenza di eventuali altre istanze di se verificando la presenza in /src/fifo della *MultiPlayerFifo* e, in caso affermativo, termina immediatamente notificando un messaggio di errore.

Passato questo controllo crea a sau volta la *MultiPlayerFifo* e la FIFO che ha come nome il PID del proprio processo (*<ServerPid>Fifo*), canali sui quali effettuerà solamente operazioni di lettura.

Terminate queste operazioni, il server si pone in ascolto delle iscrizioni dei clients, necessarie per dare inizio alla partita.

Durante la fase di avvio il client, qualora riscontri la presenza della FIFO adibita alle iscrizioni, crea come primo passo la FIFO sulla quale riceve informazioni dal server, il cui nome è il PID del proprio processo (*<ClientPid>Fifo*). Il passo successivo consiste nell'inviare al server il PID per poter essere accettato nella sessione di gioco. Qualora, invece, la ricerca del canale di comunicazione, la *MultiPlayerFifo,* non vada a buon fine, allora il processo termina mostrando un messaggio di errore all'utente.

### Iscrizione

La registrazione al gioco può ritenersi completata solamente una volta che il server, ricevuto il PID di un client, gli risponde inviando a sua volta il proprio PID se sono ancora presenti posti disponibili oppure la stringa “full” se è già stato raggiunto il numero massimo di giocatori. Se i clients sono ammessi alla fase di gioco, conoscendo il PID del server, possono aprire in modalità scrittura la *<ServerPid>Fifo*.

Il server accetta un numero massimo di partecipanti pari al valore del parametro --max <max> settato da riga di comando al momento dell’esecuzione tramite la funzione standard getOptLong. Richieste successive una volta raggiunto il massimo numero di giocatori sono rifiutate.

### Gioco

Nella fase di gioco il server si occupa della generazione di numeri interi casuali che invia a ciascun client. Una volta letti, i clients li stampano a schermo chiedendo all’utente di inserirne la somma. Dopo che l’utente ha digitato il numero viene generata la stringa da inviare al server ed incapsulata tramite l’apposita funzione clientToServer contenuta in “comunicazione.h”.

Successivamente è compito del server processare le risposte tramite la funzione di decodifica disponibile in comunicazione.h dei clients ed assegnare loro l’adeguato punteggio in base alla correttezza o meno del risultato.

### Terminazione

Il server controlla il punteggio di ciascun giocatore una volta ricevuta la prima risposta corretta. Se uno fra i punteggi dei giocatori eguaglia o supera quello settato da riga di comando mediante il parametro --win <win> sempre tramite la funzione getOptLong, la partita termina e viene dichiarato vincitore il giocatore che ha raggiunto tale punteggio.

Quando un giocatore vince la partita termina ed il server ha il compito di inviare a ciascun partecipante un messaggio d'avviso che farà terminare i relativi processi.

Considerazioni finali

Tra tante possibili vie implementative abbiamo deciso di utilizzare il più possibili gli strumenti visti a lezione, quindi limitato l’utilizzo delle thread al necessario, infatti un’altra implementazione potrebbe essere stata quella di istanziare una thread ad-hoc per ogni client partecipante al gioco e creare una thread supplementare per dirigere il gioco. Il che avrebbe reso tutte le comunicazioni molto più sicure e l’handling delle situazioni critiche di gioco come ad esempio la morte prematura di un entità molto più semplice e sicura.

Un altro principio che abbiamo ritenuto fondamentale nell’implementazione è stato quello di cercare di utilizzare meno strumenti possibili in modo da rendere l’esecuzione del programma il più possibile lineare, il codice molto leggibile e l’utilizzo della memoria minimo, motivo peraltro che sta alla base della scelta del linguaggio C come standard per questi progetti.

Abbiamo inoltre optato per utilizzare comandi molto semplici per quanto riguarda l’interazione con l’utente in modo da rendere la curva di apprendimento pressoché inesistente.

Una piccola nota che trovo interessante riportare è che il nostro software è quasi completamente auto contenuto e non sporca in alcun modo la memoria, infatti tutte le FIFO vengono create in una sotto directory all’interno di src, questo in vista di una possibile installazione da parte di un utente medio, il quale per eliminare il nostro programma e ogni sua traccia non dovrà fare altro che spostare in cestino la cartella dello stesso.