Corso di laurea in informatica

Laboratorio di Sistemi Operativi

Manuale Forza 4

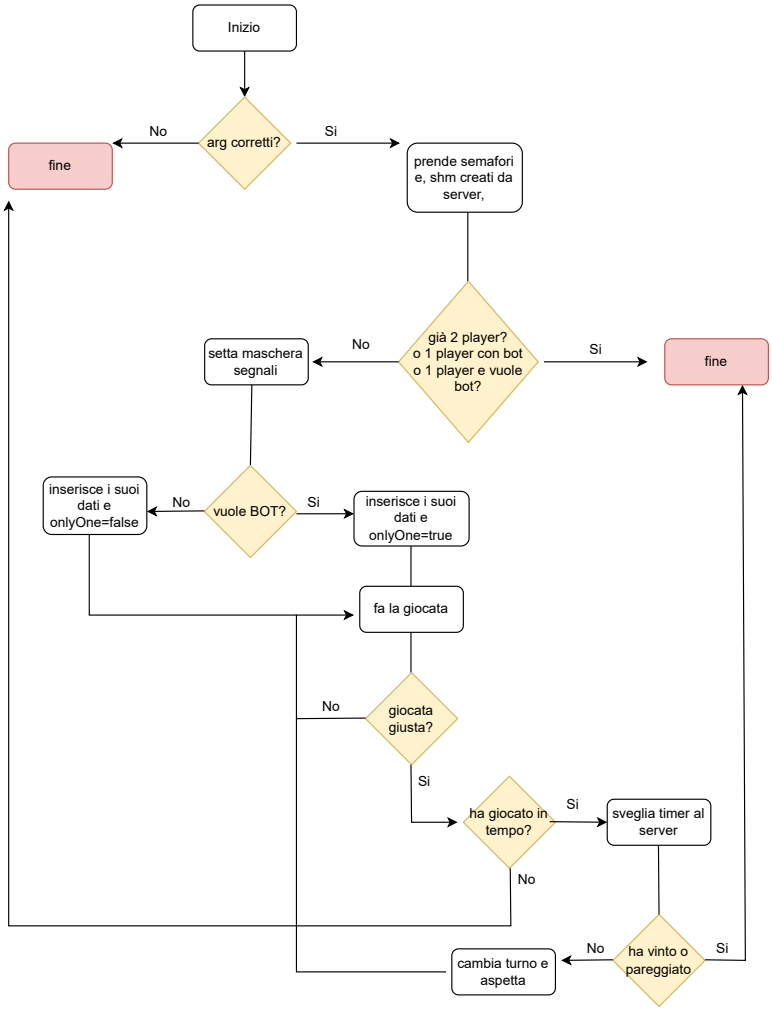
Alberto Ondei (VR471795)
2022/23

Indice

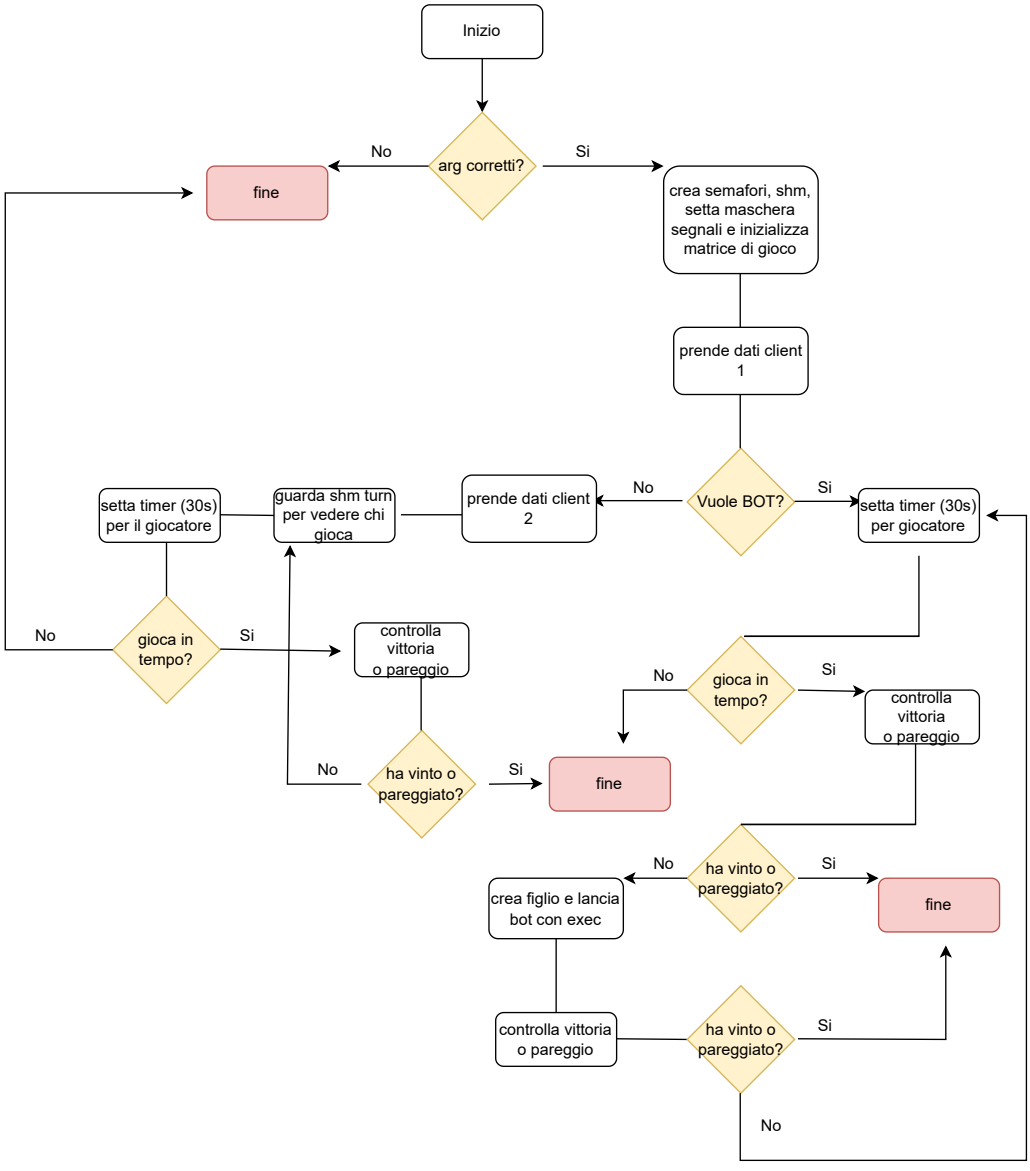
1 Diagrammi di flusso

Nei due diagrammi di fluso (client e server) verrà rappresentato ad alto livello come si comportano i processi per avere un'idea di come funzionano.

1.1 Diagramma client



1.2 Diagramma server



2 Descrizione scelte progettuali

Per gestire il progetto mi sono bastate system call relative a segnali, semafori, memoria condivisa, gestione dei processi figli e exec.

2.1 Descrizione generale

Il primo processo che verrà chiamato sarà il processo server e deve essere eseguito nel seguente modo: "./F4Server righe colonne forma1 forma2", se i parametri inseriti non sono 5 (tenendo conto di ./F4Server) verrà generato errore.

Il processo server compierà queste azioni in ordine:

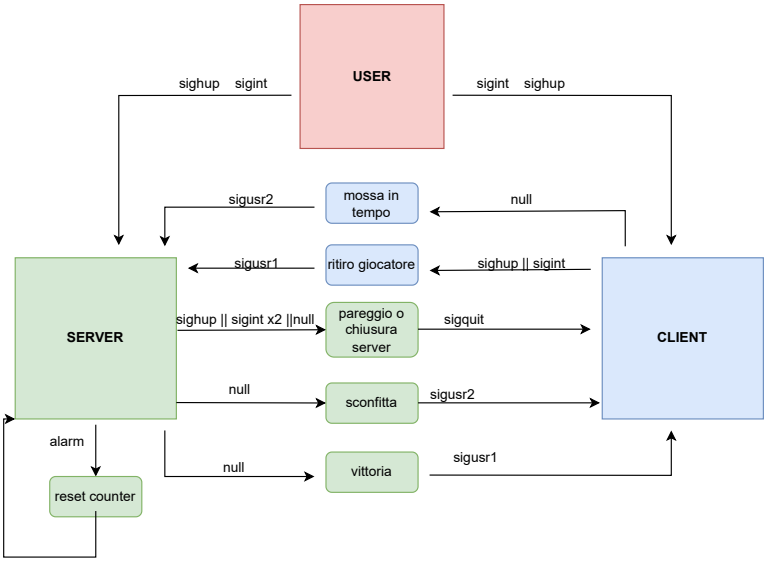
1. Crea il set di semafori, setta la maschera di segnali, crea tutte le memorie condivise di cui avrà bisogno e inizializza la matrice di gioco vuota (con tutti spazi)
2. Aspetta che arrivi il primo giocatore e quando arriva prende i dati condivisi nella struttura condivisa setClient
3. Nella struttura condivisa Information ci va ad aggiungere i dati del client appena arrivato
4. Controlla se vuole giocare col bot
5. Se si arbitra la partita tra il client e client bot (processo che verrà eseguito ad ogni iterazione)
6. Se no aspetta che arrivi il secondo giocatore e quando arriva prende i dati condivisi nella struttura condivisa setClient
7. Arbitra la partita tra i due client

Il processo client invece deve essere eseguito nel seguente modo: "./F4Client nomeClient '*'", dove '*' è facoltativo nel caso voglia giocare col bot.

Esso compierà queste azioni in ordine:

1. Prende set di semafori dal server
2. Aspetta che il server crei tutte le memorie condivise
3. Prende le memorie condivise dal server
4. Setta la maschera di segnali
5. Se vuole bot gioca la partita contro il bot
6. Altrimenti inserisce i suoi dati nella struttura condivisa setClient e gioca contro l'altro client

2.2 Segnali



2.3 Creazione e rimozione IPC

Per generare chiavi IPC ho usato la system call ftok, mettendo come file di riferimento quello del server e inserendo un numero che incrementavo ogni volta che mi serviva un'altra IPC.

Inoltre tramite l'uso di sincronizzazione tra client e server ho fatto in modo che l'unico a creare le IPC sia il server, mentre il client le prende.

Quindi il server usa una funzione speciale che ha attivate le flag IPC_CREAT (per crearle) e IPC_EXCL (in modo che se fosse già presente nel sistema, dia errore e termini il programma).

La rimozione delle IPC viene fatta solo dal server

2.4 Memoria condivisa

- "**struct setClient** *client": memoria condivisa che ha come scopo principale quello di capire chi è il primo giocatore e chi il secondo, quindi di **distinguere** i due client.

  In sostanza il primo client che riuscirà a inserire i dati dentro questa struttura verrà considerato primo player.

Questo è stato possibile grazie alla **sincronizzazione** tra i due client e tra il server tramit l'uso di semafori, perchè mentre il primo client inserisce i dati il secondo non può arrivare e inserire i suoi prima che il server non li abbia letti.

- "**Information** *cinfo": memoria condivisa in cui vengono inserite informazioni generali, nelle info verranno copiati i pid e i nomi client inseriti precedentemente nella setClient, e inoltre verrà messo anche il pid del server, in modo che i client possano inviargli segnali.
- "**Shape** *shape" vengono inserite le forme di gioco inserite come parametro, serviranno ai client in quanto quando faranno la mossa riempiranno la casella di gioco con la loro forma
- "**int** *turn" in questa variabile il client che sta giocando scriverà che è il suo turno (0 per client 1 e 1 per client 2) cosi il server capisce chi ha fatto l'ultima mossa
- "**int** *howMany": mi dice quanti client stanno giocando, inizializzata a 0 dal server, se i client sono piu di due termina, oppure se un giocatore sta già giocando col bot non può esserci un altro giocatore e se viene inserito termina, oppure se c'è già un giocatore che aspetta non può essere inserito un giocatore che vuole il bot e termina.
- "**int** *timeRunOut": viene usata per una questione estetica, non ha scopi logici, in sostanza se il server nota che è scaduto il tempo, setta questa variabile a 1 e manda la kill di sconfitta a chi ha fatto scadere il tempo (e di vittoria all'altro) e settando questa variabile a 1 i client possono stampare a video il motivo della sconfitta (quindi per tempo scaduto)
- "**int** *isRetired": anche questa ha solo scopi estetici, viene settata dal client e il server stampa a video chi si è ritirato (se si è ritirato qualcuno)

2.5 Semafori

Ho usato i semafori come visti a teoria di sistemi operativi, quindi ho solo usato funzionalità che incrementano il valore e lo decrementano (P e V), non ho usato altre funzionalità (come quella di aspettare che un semaforo venga messo a 0) perchè non mi veniva naturale.

L'uso dei semafori si può distinguere in 3 categorie:

- **Distinguere i client**: In questa fase bisogna distinguere chi è il primo client e chi il secondo, guardando chi inserisce per primo i dati in struttura condivisa "setClient".

  - MUTEX: inizializzato a 1, serve per fare in modo che i client inseriscano i dati in setClient in modo mutualmente esclusivo
  - EMPTY: inizializzato a 0, il server non prende i dati in setClient finchè qualcuno non li inserisce, quando il player li inserisce fa libera empty in modo da dare il via libera al server di prendere i dati
  - Dopo che il server prende i dati del primo client libera MUTEX, di prima e da il via libera al secondo possibile client di entrare
  - WAIT_SECOND: il primo client aspetta che arrivi il secondo client prima di andare avanti
  - wait_second verrà liberato dal server quando quest'ultimo prende i dati del secondo client messi in setClient

- Fase di gioco:
  - BC1: blocca il client 1 quando non è il suo turno e verrà sbloccato quando il client 2 ha finito di giocare
  - BC2: blocca il client 2 quando non è il suo turno e verrà dal client 1 dopo che il server ha fatto il check di vittoria o pareggio (non ha senso liberarlo prima che il server faccia il check)
  - BS: blocca il server: viene usato 2 volte:
    1. La prima volta blocca il server per aspettare che il client faccia la giocata, e dopo fa il check di vittoria o pareggio
    2. La seconda volta viene bloccato per aspettare che il client setti la variabile condivisa "turn" al turno giusto, altrimenti potrebbero esserci problemi di scheduling e quando il server fa il check lo fa col "turn" sbagliato
    3. Nel bot ASSEST viene usato per bloccare il client, mentre BS per bloccare il server
  - BC: blocca il client (quello che sta giocando indipendentemente da 1 o 2) che aspetta il server faccia il check dopo la giocata

- Altri semafori:
  - START: usato per fare in modo che il server crei le shared memory prima che il client le vada a prendere, volendo questo semaforo si potrebbe evitare dato che il server deve essere il primo processo a essere runnato e la creazione delle IPC è una delle prime cose che fa, però l'ho messo per sicurezza
  - MUTEX_2: usato per aumentare o diminuire howmany in modo mutualmente esclusivo tra i due client, non potevo usare MUTEX per questione di scheduling, dato che successivamente viene usato per altro.