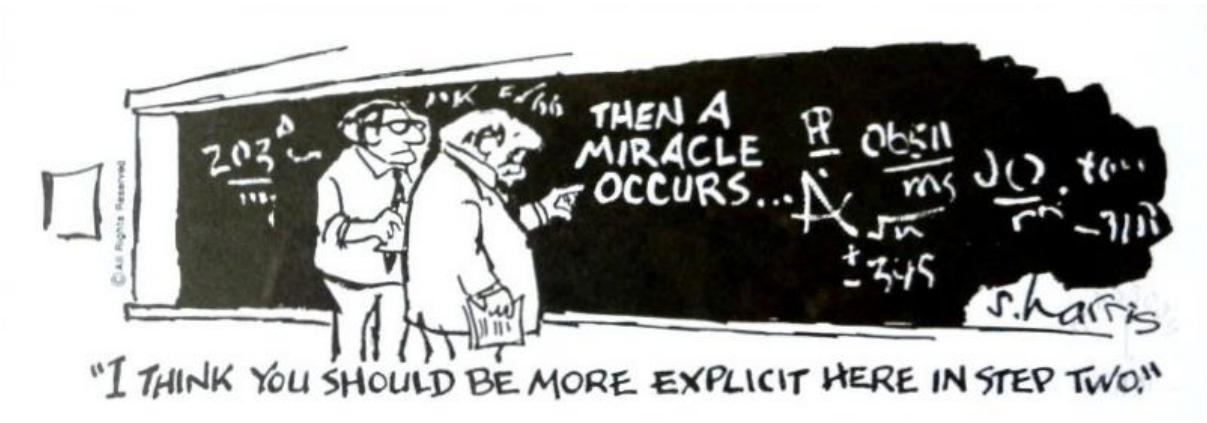
Relazione sullo sviluppo del Progetto di Laboratorio di Sistemi Operativi

SVILUPPATO DA

- Stefano Vittorio Porta – Matricola 859133
- Federico Medina – Matricola 864219
- Alberto Solaro – Matricola 859117

Anno Accademico 2018-2019
Turno 3

1. INTRODUZIONE



La consegna del progetto ha richiesto un'interazione in tempo reale tra un processo **Gestore** e multipli processi **Studente**, i cui programmi (*Gestore.c* e *Studente.c*) sono stati progettati per funzionare in eseguibili separati.

Alcune impostazioni per la personalizzazione dell'esecuzione della simulazione hanno reso chiaro la necessità di progettare anche un semplice lettore di file, che sarà oggetto della sezione seguente.

L'interazione tra gli studenti, necessaria per organizzare la composizione dei gruppi (oggetto del progetto), è stata possibile grazie all'implementazione di metodi di utilità (file *ipc_utils.c*) che hanno aiutato ad interagire con i meccanismi di **System V** (Memoria condivisa, Semafori e Code di messaggi).

Durante l'esecuzione della simulazione si è deciso di stampare diverse informazioni di *debug*. La maggior parte è utilizzabile per tracciare *l'evoluzione della simulazione*, dalla partenza alla conclusione. È possibile incrementare la quantità di informazioni in output ricompilando il programma definendo alcune macro presenti all'inizio dei relativi file *.h*.

2. LETTORE DELLE IMPOSTAZIONI

I due file (.h e .c) di nome *settings_reader* contengono le definizioni e l'implementazione del lettore delle impostazioni.

È stato progettato in modo da leggere un file specificato (*opt.conf*).

Se, avviando il programma principale (*Gestore.r*), non si aggiunge nessun parametro, il file sarà cercato nella cartella in cui si trova il programma, altrimenti cercherà nella path specificata (deve essere specificato anche quale file aprire). In ogni caso, se il file non esiste, il lettore provvederà a generarne una copia partendo da delle impostazioni prefissate.

Il lettore, per ottenere le impostazioni, legge carattere per carattere. Quando viene trovato un simbolo di newline (**\n**), un commento o uno spazio, il valore letto viene salvato nel relativo spazio delle impostazioni.

Se viene trovato un commento (il cui inizio è indicato dal simbolo **#**), il resto della riga è ignorato.

È previsto che il lettore legga le impostazioni necessarie al funzionamento della simulazione in un **ordine prefissato**. I *commenti* generati quando il file di configurazione non esiste possono aiutare a comprendere il significato delle impostazioni.

È stato reso possibile anche modificare i voti di Architettura degli Elaboratori (minimo e massimo) generabili, stessa cosa per la preferenza (minima e massima) di studenti invitabili in un gruppo.

A seconda del numero di preferenze, il lettore si aspetta $\boldsymbol{Massima - Minima + 1}$ percentuali, ognuna dedicata ad una singola preferenza. Queste percentuali sono da includere al fondo del file, dopo tutte le altre impostazioni.

3. GESTORE

Il gestore ha una serie di compiti che dovrà svolgere in sequenza, ed è l'unico in grado di far funzionare correttamente i processi studenti, dato che inizializza tutte le strutture necessarie di **System V**.

Un particolare importante da notare è sull'allocazione della memoria condivisa.

È estremamente importante allocare spazio alle strutture che utilizziamo e, se contengono vettori, anche alla loro capacità massima. Non vi sono stati pochi problemi durante una prima implementazione del progetto, proprio a causa di questo specifico dettaglio.

Sono stati utilizzati dei vettori per le strutture dati per contenere alcune informazioni (i dettagli degli studenti necessari per la simulazione) invece di dei puntatori, in quanto questi ultimi avrebbero portato a errori di *Segmentation Fault*.

Dopo aver inizializzato tutte le informazioni necessarie, il Gestore inizializza gli studenti utilizzando una *fork* seguita, nel caso in cui la system call restituisca 0, da una *execl* che avvia effettivamente il codice di *Studente.c*.

Inizializzati tutti gli studenti, il gestore entra in attesa di un semaforo (chiamato EVERYONE_READY) che sarà decrementato di un'unità anche dagli studenti. Quando il semaforo è azzerato, la simulazione parte.

Il Gestore grazie ad una *sleep* attende *sim_duration* secondi, dopo la quale la simulazione termina.

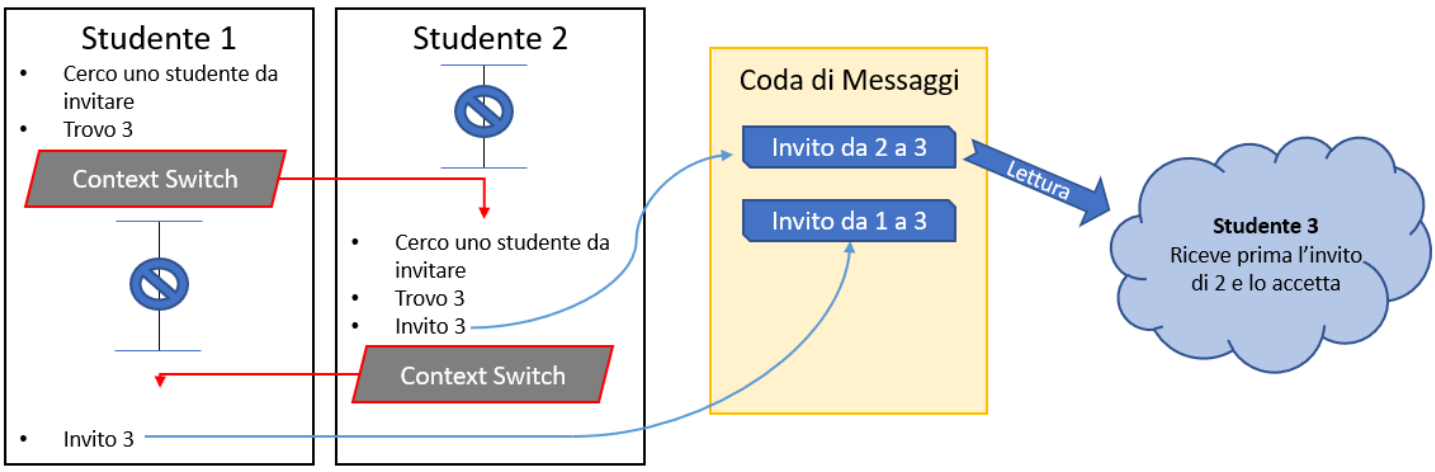
Tempestivamente il Gestore invia agli studenti, grazie a *kill*, un segnale (SIGUSR1) che ordina agli studenti di terminare immediatamente l'interazione con gli altri e di prepararsi a ricevere i voti, che stamperanno una volta pronti.

4. STUDENTI

SYSTEM V

Gli studenti interagiscono tra di loro sfruttando tutti i meccanismi offerti da **System V**:

- **Memoria Condivisa**: per avere informazioni aggiornate sugli altri studenti, abbiamo sfruttato la memoria condivisa. Una struttura dati (*SimulationData*, definita in *types.h*) contiene tutte le informazioni sulla simulazione (impostazioni e dati degli studenti).
  È stato necessario includere diverse informazioni riguardo lo stato degli studenti, tra cui le più essenziali sono
  - Lo **stato attuale** (*status*) codificato con un enumeratore (*StudentStatus*), che identifica in che situazione si trova lo studente (se è disponibile per l'invito, in un gruppo, se è il gestore di un gruppo o se sta attendendo conferma per l'inclusione in un gruppo)
  - Il **voto di Architettura degli Elaboratori**, essenziale per la nostra logica (improntata sull'incremento della media degli studenti)
  - La **preferenza sul numero di studenti nel gruppo** (anch'essa informazione necessaria per incrementare la media)
  - **L'ID dell'owner del gruppo**, essenziale per calcolare il punteggio
  - **L'ID dello studente nel gruppo con il miglior punteggio** (informazione mantenuta solo dall'owner del gruppo)
- **Coda di Messaggi**: Per permettere agli studenti di scambiare **inviti e le seguenti risposte**, una coda di messaggi è resa disponibile dal Gestore. Quando uno studente decide di non invitare più studenti, l'unica cosa che continuerà a fare sarà attendere per messaggi da ricevere, in modo da *evitare che la coda si riempa*.
- **Semafori**: In particolare, un semaforo per studente. Sono essenziali per evitare che si verifichino problemi di *inconsistenza durante l'invito di uno studente in un gruppo*. Può infatti verificarsi (senza questo semaforo dedicato) una race condition (essendo gli studenti non sincronizzati) che potrebbe portare uno studente ad essere invitato in un gruppo, quindi ignorando possibili inviti decisi precedentemente rispetto a quello accettato, ma recapitati dopo.
  Uno studente che trova un possibile invitato cercherà di *riservarlo*, in modo che **nessun altro possa invitarlo prima di lui**, riservando il relativo *semaforo studente*.



Come deve comportarsi 1?

LOGICA PRINCIPALE

Lo studente, durante la simulazione, svolgerà due compiti:

1. Leggerà tutti i messaggi ricevuti (e se accetta un possibile invito smette di leggere attivamente la coda di messaggi, ma la svuoterà da messaggi diretti ad esso ma ormai non più utilizzabili)
2. Cercherà di inviare inviti. Se non riesce ad inviarne, attende per un messaggio, e fino a quando non lo riceverà non proseguirà con l'esecuzione.

Lettura dei messaggi

Durante la lettura dei messaggi, in funzione del tipo di messaggio ricevuto (INVITO, ACCETTAZIONE, RIFIUTO) vengono eseguiti dei metodi appositi.

Se viene ricevuto un invito, verrà eseguito il metodo *acknowledgeInvite*, che decide se accettare o meno l'invito ricevuto.

- Se l'invito è diretto ad uno studente che sta **attendendo risposte ad inviti inviati**, oppure se **non è invitabile** (Owner di gruppo, in attesa di conferma o in gruppo), sarà respinto, ma il numero di rifiuti disponibili allo studente che ha dovuto respingere l'invito **non sarà diminuito** (come da consegna).
- Se invece non abbiamo più rifiuti disponibili, dovremo per forza accettare l'invito ricevuto
- Infine, se nessuna delle altre condizioni è verificata, decidiamo se accettare o meno l'invito con una logica che punta ad incrementare il voto dello studente oltre il voto medio:

$$\frac{Voto\ Massimo\ +\ Voto\ Minimo}{2}$$

Invio degli inviti

Quando dobbiamo invitare uno studente, una logica un po' più complessa viene utilizzata.

Il fulcro della logica implementata è l'incremento della media. Per far ciò, si è deciso di procedere come segue:

Sequenzialmente, partendo dal nostro numero di matricola, iniziamo a scansionare tutti gli studenti, per cercarne uno invitabile. Questa ricerca *verrà eseguita su tutti gli studenti* (se non si trova un candidato) un certo numero di iterazioni (sperimentalmente **7** si è rivelato essere lo sweet spot).

Ogni studente è valutato, **in funzione dell'incremento di voto che si otterrebbe nel gruppo**. Inoltre, si tiene presente anche di quante **iterazioni** di ricerca sono state svolte, in modo da cercare di invitare studenti con voti molto più bassi rispetto al nostro (cosa che incrementa la media se entrano a far parte del nostro gruppo) e *preferibilmente senza penalità*.

Più il contatore delle iterazioni avrà un valore elevato, *più lo studente **incrementerà** la tolleranza*. Ciò permetterà di invitare studenti con voti sempre più vicini al nostro, nel caso non siano disponibili studenti migliori.

COMPORTAMENTI AGGIUNTIVI

Per permettere a tutti gli studenti di incrementare la propria media, abbiamo introdotto un comportamento aggiuntivo.

Dato che anche gli studenti sanno quanto durerà la simulazione, non appena potranno iniziarla, useranno anche (prima della logica principale) un *alarm*, con durata $sim\_duration - 1$.

Il segnale SIGALRM sarà intercettato dalla funzione *simulationAlmostEnded* un secondo prima dell'invio da parte del Gestore del segnale SIGUSR1 che terminerà la simulazione.

In questo modo ci è possibile chiudere il gruppo immediatamente (ci sono alcuni casi particolari in cui non lo facciamo, ad esempio se siamo in un gruppo e non siamo l'owner) ed evitare che lo studente (o il gruppo di studenti) prenda 0.

Questo comportamento permette di incrementare notevolmente la media finale.

5. CONCLUSIONI

La nostra soluzione è stata testata numerose volte, con quantità variabili di studenti, tempi di simulazione e numeri di inviti/rifiuti diversi.

Ci teniamo a ricordare che i voti iniziali sono generati casualmente e la simulazione dipende dalla velocità della macchina su cui è eseguita (senza dimenticare anche la quantità di RAM disponibile e il numero di processi della simulazione e quelli attualmente in esecuzione sul Sistema Operativo).

Abbiamo infatti osservato alcune simulazioni con incrementi molto ridotti, se non nulli, ma la maggior parte delle simulazioni ha sempre portato ad un incremento dei voti.

I test eseguiti sono stati effettuati con alcune modifiche rispetto alle impostazioni standard, in particolare:

- 1000 Studenti
- 10 secondi di simulazione
- 10 inviti massimi
- 6-8 rifiuti massimi

In media la simulazione, eseguita con la nostra soluzione, presenta un incremento medio del voto di Sistemi Operativi rispetto a quello di Architettura degli Elaboratori tra **2.70 e 3** punti.