

**UNIVERSITA’ DEGLI STUDI DI TORINO**

Dipartimento di Informatica

Corso di Sistemi Operativi

**Elaborato finale del Laboratorio di Sistemi Operativi**

Autore: Annalisa Sabatelli

Matr. 866879

# Introduzione

Il progetto prevede la realizzazione di un codice basato sulla programmazione concorrente che simuli lo scambio di messaggi tra studenti con lo scopo di formare dei gruppi di studio di una ipotetica classe del corso di Sistemi Operativi coerentemente a quelli che sono criteri di preferenza di ognuno e coerentemente ad una serie di vincoli e regole definiti dalla consegna.

Sulla base dei gruppi così formati, allo scadere di un lead time preliminarmente stabilito, si assegna a ciascun studente il voto dell’esame di “Sistemi Operativi”, che dipenderà sostanzialmente dalla corrispondenza tra le caratteristiche del gruppo finale di appartenenza e le proprie preferenze iniziali e il voto conseguito nell’esame di “Architettura di elaboratori”.

L’obiettivo è quello di sfruttare a favore le regole di formazione dei gruppi e di assegnamento finale dei voti agli studenti affinché la media dei voti dell’esame di “Sistemi Operativi” sia migliore rispetto a quella di partenza di “Architettura degli Elaboratori”.

# Struttura del programma

Il programma è composto dai seguenti file:

* **Opt.conf:** il file di configurazione contiene, come richiesto nella consegna, indicazioni sul numero massimo di inviti (*nof\_invites*) che ciascun studente può effettuare per formare il proprio gruppo e la ripartizione, espressa in percentuale sul numero totale di studenti, della preferenza di questi relativamente alla numerosità del gruppo a cui vorrebbero appartenere (gruppo 2, gruppo 3, gruppo 4 dove 2, 3 e 4 indicano proprio la numerosità del gruppo). L’assegnamento di questi valori è arbitrario. Nel corso delle varie simulazioni effettuate con lo scopo di testare il codice, è stata effettuata una basilare analisi di sensitività di questi.
* **Header.h:** il file header è utilizzato per includere le librerie C necessarie e contenere i parametri globali utilizzati dai vari processi e la definizione delle strutture dati. In particolare, all’interno dell’header è indicato il numero di studenti che compongono la classe (POP\_SIZE), il lead time a disposizione degli studenti per formare i gruppi di studio (SIM\_TIME), e i codici identificativi utilizzati dalle primitive IPC. Anche in questo caso, effettuando un’analisi di sensitività sui parametri SIM\_TIME (da 5 a 60 secondi) e POP\_SIZE (da 100 a qualche migliaia) si assiste a prestazioni diverse del programma.
* **ProgramManager.c:** il file ProgramManager.c contiene un menù di partenza che offre all’utente d tre alternative: premere ‘*G*’ per dare avvio al programma e alla formazione dei gruppi; premere *‘?’* per visualizzare guida & credits; premere *‘Q’* per uscire.
* **CorsoMaster.c:** il file CorsoMaster.c contiene il codice del processo padre al quale, in sintesi, sono assegnati i seguenti compiti:
  + Lettura del file opt.conf e memorizzazione dei dati in apposite variabili;
  + Utilizzo delle primitive *\*get* degli strumenti di IPC per creare semafori, code di messaggi e memoria condivisa che saranno poi utilizzati anche dai processi figli;
  + Creazione di *POP\_SIZE* processi figli rappresentanti ciascuno uno studente tramite la primitiva *fork();*
  + Avvio e terminazione della simulazione per la formazione di gruppi tramite l’invio di segnali appositi ai processi figli;
  + Invio di un segnale di pre alarm superato il 75% del *SIM\_TIME* per sollecitare la chiusura dei gruppi ed evitare voti nulli;
  + Ricezione da ciascun processo figlio di un messaggio contenente il proprio voto finale di Sistemi Operativi al termine della simulazione;
  + Invio di un segnale per causare la terminazione dei processi figli;
  + Eliminazione di tutte le strutture IPC create;
  + Stampa dell’elenco di voti risultante e calcolo della media complessiva.
* **Studente.c:** il filestudente.c contiene il codice che modellizza il comportamento di uno studente. Pur essendo un processo figlio creato da CorsoMaster.c, l’esecuzione della primitiva *execve()* permette di avere codici di esecuzione distinti tra padre e figli. Ogni studente è caratterizzato da:
  + *Matricola:* identificatore numerico univoco del singolo studente. In questo caso si è scelto di assegnare alla matricola il valore del pid del processo.
  + *Voto di Architettura di elaboratori:* generato secondo una distribuzione uniforme tra 18 e 30;
  + *Laboratorio di appartenenza:* T3 o T4 a seconda che la matricola sia rispettivamente dispari o pari;
  + Preferenza sulla numerosità del gruppo di studio al quale si vorrebbe appartenere (gruppi da 2, 3 o 4 componenti;

Questi dati sono memorizzati sia in una struttura dati locale a ciascun processo che in una *shared memory* in modo che siano accessibili globalmente da tutti i processi. La *shared memory* è stata pensata come un vettore di strutture, in cui ogni posizione è assegnata ad uno studente. Oltre agli attributi caratteristici indicati sopra, ciascuna struttura della *shared memory* ha anche altri campi funzionali alla creazione e chiusura dei gruppi e alla differenziazione tra *follower* e *leader* di un gruppo:

* Stato\_g: è lo stato del gruppo che può essere *closed “C”* oppure *open “O”*;
* Stato\_s: è lo stato dello studente che può essere *free “F”* oppure *assigned “A”*;
* Tipo\_componente: differenzia i *leader “L”* dai *follower “F”* di un gruppo;
* Nome\_gruppo: è il nome identificativo di ciascun gruppo e corrisponde alla matricola del leader.

Una volta ricevuto il segnale di inizio simulazione dal processo padre, gli studenti si scambiano messaggi per formare gruppi di studio agendo secondo un possibile algoritmo di ottimizzazione definito coerentemente con le regole e i vincoli indicati nella consegna. Alla fine del *SIM\_TIME* i processi studente comunicano il voto risultante di Sistemi Operativi al processo padre e, su indicazione di questo, terminano.

# Algoritmo di formazione dei gruppi

Il programma può considerarsi suddiviso in due fasi principali:

* **Fase di inizializzazione:** il corsoMaster legge dal file opt.conf le informazioni contenute, crea i processi figli e questi, regolati da un semaforo, accedono in scrittura alla shared memory e la inizializzano con le loro informazioni caratteristiche.
* **Fase di formazione dei gruppi:** il corsoMaster avvia la simulazione tramite l’invio di un segnale ai processi figli che iniziano a scambiare messaggi su una coda per formare i gruppi secondo un algoritmo stabilito.

L’algoritmo definito e implementato per consentire ai processi studenti la formazione dei gruppi è finalizzato ad ottimizzare la media complessiva del voto finale di Sistemi Operativi in modo che questa sia maggiore rispetto alla media dei voti di Architettura degli elaboratori.

Si è scelto di suddividere i processi studenti in tre macro gruppi sulla base del voto di Architettura di elaboratori e di assegnare ad ogni gruppo un comportamento specifico:

* **Gruppo 1: studenti con voto compreso tra 18 e 21**

Non invitano ma sono solo in attesa di essere invitati da studenti appartenenti al gruppo 3, ossia con voto compreso tra 27 e 30. Una volta ricevuto l’invito, se ancora liberi, accettano rispondendo con un messaggio di conferma ed aggiornano il proprio stato nella memoria condivisa da *free* (F) ad *assigned* (A). Se sono ancora liberi al momento del verificarsi del pre-allarme decidono di chiudere il gruppo da soli per evitare voti nulli.

* **Gruppo 2: studenti con voto compreso tra 22 e 27**

Possono sia proporsi come *leader* ed invitare che rimanere in attesa di ricevere un invito e far parte di un gruppo come *follower*. All’avvio del *SIM\_TIME,* controllano la coda di messaggi. Se non sono presenti messaggi allora un processo casuale con probabilità del 50% decide chi può proporsi come leader ed invitare e chi, invece, rimane comunque in attesa di un messaggio.

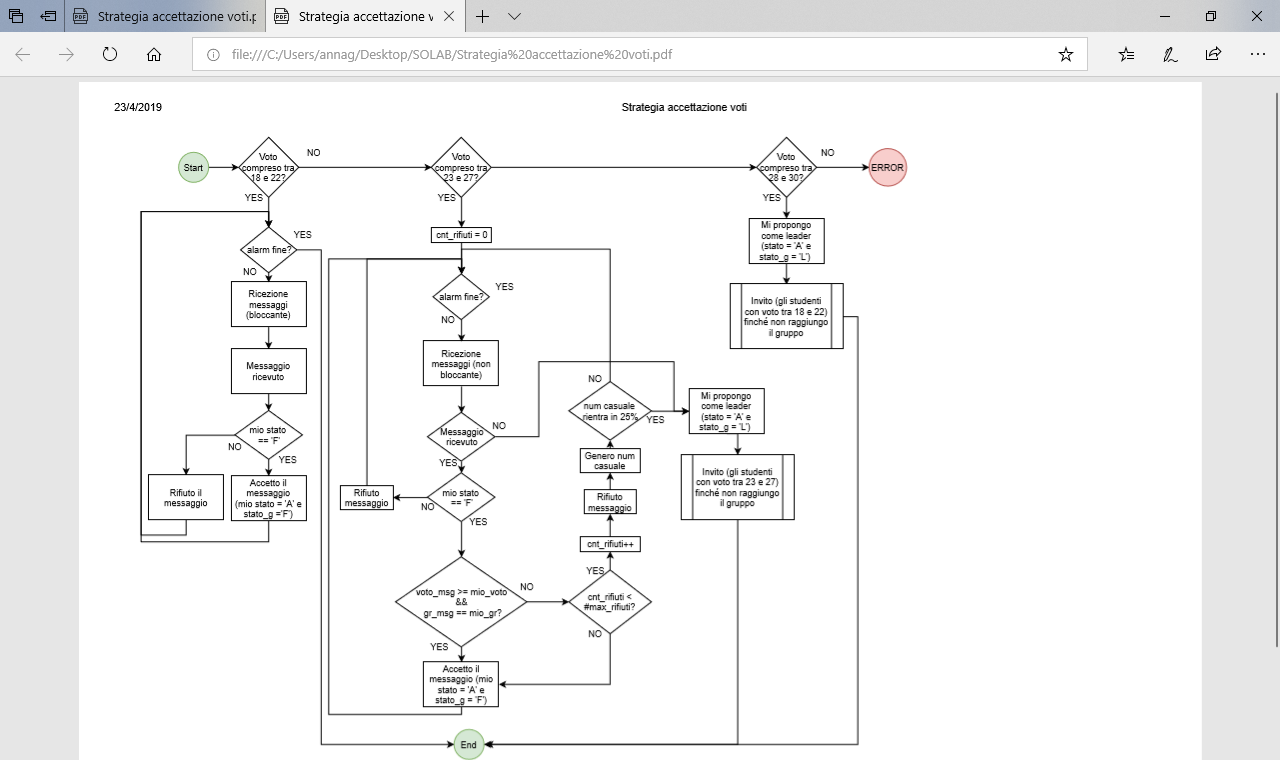
Se ci si propone come leader, lo studente scrive “*L*” nel campo “*tipo\_componente*” della memoria condivisa in modo da evitare che altri possano invitarlo e incomincia a scorrerla ricercando altri studenti a cui mandare inviti.

Ciascuno nel ricevere l’invito, controlla i dati principali del mittente (voto di Architettura di elaboratori e numerosità del gruppo che ha intenzione di formare) e può decidere se rifiutarlo o accettarlo. In entrambi i casi comunica la propria scelta la mittente tramite un apposito messaggio.

Se lo accetta, aggiorna il proprio stato nella memoria condivisa da *free* (F) ad *assigned* (A) e inserisce nel campo “*nome\_gruppo*” la matricola dello studente *leader* da cui ha ricevuto l’invito. Al contrario, esiste un numero massimo di rifiuti *MAX\_REJECT* superato il quale si è costretti ad accettare. Al verificarsi del pre-allarme, i leader dei gruppi non ancora completi e tutti coloro che non appartengono a nessun gruppo decidono di chiudere il gruppo per evitare voti nulli modificando lo stato del gruppo in memoria condivisa da *Open* (O) a *Closed* (C).

* **Gruppo 3: studenti con voto compreso tra 28 e 30**

Ogni studente si propone subito come *leader* di un gruppo e invita studenti appartenenti al gruppo 1. Nel momento in cui raggiunge la numerosità desiderata per il proprio gruppo oppure termina il numero massimo di inviti che può spedire (*nof\_invites*) oppure riceve il segnale di pre-allarme chiude il gruppo modificando lo stato del gruppo in memoria condivisa da *Open* (O) a *Closed* (C). La scelta di associare in un gruppo un voto alto con diversi voti bassi è finalizzata a neutralizzare i voti bassi ed alzare la media complessiva della classe dato che, in caso di gruppo chiuso alla fine del *SIM\_TIME*, secondo le regole indicate, a tutti i membri viene assegnato il massimo tra i voti di partenza.



# Esecuzione del programma

### Compilazione

Per compilare il programma è sufficiente spostarsi sulla directory bin dove è presente il *makefile*.

> cd bin

> make all

Se non si dispone di make, è possibile usare la shell *compila.sh.*

> cd bin

> ./compila.sh

### Esecuzione

Eseguire il ProgramMaster attraverso il comando

>./programManager