Relazione del progetto di laboratorio

Corso Sistemi Operativi 2018-2019

Andrea Malgaroli, matricola 823429
Roberto Sterpone, matricola 822326,
Lorenzo Tabasso, matricola 812499,

1 Organizzazione del codice

Il programma è organizzato in 3 file.c (*manager.c*, *student.c*, *utility.c*) che contengono le differenti funzioni, raggruppate a seconda della tipologia del processo che le esegue. È inoltre presente un header file chiamato *common.h*. I due nuclei del programma sono i due file *manager.c* e *student.c* i quali contengono ognuno un *main()* e diverse funzioni che rendono possibile l'esecuzione della simulazione. In particolare, in questa simulazione il Manager crea tanti studenti quanto specificato nella macro POP_SIZE tramite una chiamata a una funzione execve() all'interno di un ciclo for. Gli studenti (figli del manager) non appena vengono creati e sbloccati da parte del padre, iniziano la loro computazione.

2 IPCs

Una volta avviato il programma vengono create le seguenti IPC:

1. Un'area di memoria condivisa nella quale è presente una matrice, in cui ogni riga contiene: il PID dello studente, l'id del gruppo dello studente, il numero di studenti del gruppo, il voto di Architettura degli elaboratori, il voto di Sistemi operativi (inizialmente settato a 0) e la preferenza dello studente sul numero di membri del gruppo ideale.
2. Tre code di messaggi : due usate dagli studenti per spedire e ricevere gli inviti, e una usata dagli studenti per comunicare al manager la composizione del gruppo poco prima del termine della simulazione.
3. Un semaforo "RW" inizializzato e usato da ogni studente (con flag IPC_PRIVATE) al fine di sincronizzarsi nel leggere/scrivere sulle code di messaggi.
4. Un semaforo "Children" inizializzato dal Manager, al fine di sbloccare gli Student nel momento in cui il cronometro della simulazione si è avviato, e alla fine della simulazione per consentire agli Student di terminare spontaneamente.

3 Simulazione

Manager.c

Il Manager è il responsabile per la creazione dei processi Student, esso infatti crea POP_SIZE processi Student tramite l'esecuzione della system call *execve().* Una volta creati i processi Student, essi rimangono bloccati fino all'aggiunta da parte del padre di una risorsa sul semaforo "Children", in modo tale che il Manager possa, prima di avviare la simulazione, inizializzare correttamente tutte le risorse necessarie, quali la coda di messaggi "Message_Queue_Parent" e l'area di memoria condivisa con il suo contenuto. In particolare, l'area di memoria condivisa contiene la matrice *marks* la quale verrà usata alla fine della simulazione dal Manager per calcolare i voti da assegnare (usando la funzione *compute_marks()*) e dagli Student per controllare il voto assegnato loro dal manager in seguito all'andamento della simulazione. Il Manager come ultima cosa prima di terminare spontaneamente, attende che tutti gli Student terminino e pubblica:

- Il numero di studenti per ogni voto di Architettura degli Elaboratori e voto medio di Architettura degli Elaboratori.
- Il numero di studenti per ogni voto del progetto di Sistemi Operativi e il voto medio del progetto di Sistemi Operativi.

Nota: è stata inserita una sleep(2) alla riga 156 del codice di *manager.c* semplicemente per agevolare la stampa a video dei voti, in virtù di ciò anche quando sembra che il programma non stia terminando, in realtà sta continuando ad eseguire, poiché vi è questa sleep() che "pulisce" il terminale nel display dei voti.

Student.c

Dopo la definizione delle variabili utili alla simulazione, i processi Student iniziano a inviare inviti ad altri studenti per la formazione dei gruppi. Iniziano spedendo inviti fino al limite consentito loro dal file di configurazione *opt.conf*, poi si mettono in ascolto sulla coda di messaggi in attesa di altri inviti o risposte ad inviti precedentemente inviati.
Parlando come fossi uno studente, nel caso in cui qualche altro studente accettasse il mio invito, non potrò accettare altre proposte nè rispondere positivamente a inviti ricevuti successivamente.
Scrittura e lettura si susseguono fino allo scadere del tempo *sim_time* oppure al sino al raggiungimento del numero di membri desiderati per il gruppo.
Usciti dal ciclo while principale, gli Student *Leader* (quelli con la variabile *leader* inizializzata a *1*) invieranno al Manager i dati del proprio gruppo, in modo tale da consentire a quest'ultimo di calcolare i voti finali, e si metteranno in attesa al semaforo per terminare dopo il calcolo dei voti da parte del Manager.
Quando il processo Manager sblocca le risorse del semaforo ogni processo Student pubblica le proprie informazioni e termina.

Utility.c

Essenzialmente, in questo file sono presenti le varie funzioni per la creazione delle varie IPCs (coda di messaggi, semafori e memoria condivisa), la funzione che implementa il timer (*start_timer()*), le funzioni per la richiesta e il rilascio di una risorsa su un semaforo, una funzione per deallocare tutte le IPCs, le funzioni usate per il calcolo delle probabilità e delle preferenze sul numero di membri del gruppo ideale, e infine tutte le funzioni utili per le strutture dati usate.