ultima modifica 29/01/2019

Scheduler Simulator
Documento di specifiche dei requisiti

Introduzione

Realizzare un programma (d'ora in poi "simulatore") in C che simuli due scheduler.
Uno con preemption, l'altro senza.
L'obiettivo della simulazione è quello di scrivere un documento (max 3 pagine A4 singola colonna in formato PDF) di report in cui venga descritta la comparativa tra i due scheduler scelti.

Il programma deve simulare le entità istruzione, task, processore, core, clock e scheduler.

Funzionalità

All'avvio il simulatore (che implementa il main) dovrà creare (fork) 2 processi:

1. scheduler_preemptive
2. scheduler_not_preemptive

Il processo del simulatore deve quindi leggere i task da eseguire da file (vedi paragrafo "input") e li deve inoltrare ad entrambe gli scheduler.

Lo studente dovrà scegliere i due algoritmi di scheduling a piacere tra quelli visti a lezione, ma sarà gradita l'iniziativa di un approfondimento personale su altri algoritmi presenti in letteratura.

Uno scheduler termina nel momento in cui non ci sono più task da eseguire.

A prescindere dalla strategia (preemption/non-preemption), quando un'istruzione si sospende a causa di una operazione bloccante, un altro task deve essere schedulato.

Entità

Il processore di ogni scheduler ha due core simulati con 2 thread "reali" per implementare l'esecuzione di 2 task in parallelo. Ogni core simulerà il contatore dei clock ticks con un numero intero senza segno che parte da zero e incrementa ad ogni ciclo.

Con il termine clock non si intende il segnale di clock, ma il CPU time, misurato appunto in “clock ticks”.

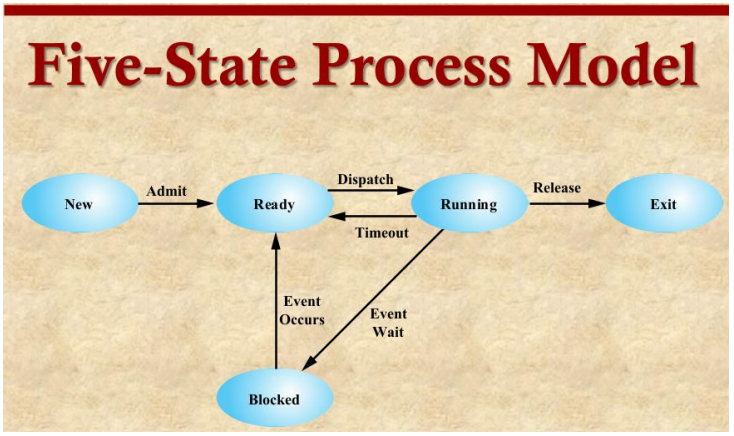
Un’istruzione è composta da:

- *type_flag*:
  - Flag 0 := istruzione di calcolo, ovvero non bloccante;
  - Flag 1 := operazione di I/O, ovvero bloccante.
- *length*:
  - se l'istruzione non è bloccante indica la sua durata di esecuzione in cicli di clock
  - se l’istruzione è bloccante, il task si blocca per un numero randomico compreso tra 1 e *length* cicli di clock

Un task è una lista ordinata di istruzioni. La scelta della lista (circolare, doppiamente linkata, ecc.) è a discrezione dello studente ma deve essere giustificata nel documento di report.

Un task in esecuzione viene gestito tramite una struttura dati, la Task Control Block (TCB), definita da:

- *id*: id univoco
- pc: program counter (puntatore alla prossima istruzione da eseguire)
- *arrival_time*: numero intero che descrive a quale ciclo di clock il task deve essere considerato dallo scheduler
- *instr_list*: la lista delle istruzioni da eseguire.
- *state*: lo stato (new, ready, running, blocked, exit)



I task dovranno rispettare il Five-State model dei processi, quindi tutti i task letti da file avranno inizialmente lo stato “new”.

Uno scheduler NON deve mandare in esecuzione un task se: *clock(core)* < *clock(task->arrival_time)*

Un task termina (stato exit) nel momento in cui non ha più istruzioni da eseguire.

Parametri da linea di comando

Il simulatore accetta i seguenti parametri (*obbligatori*) da linea di comando:

1. -op | --output-preemption: il file di output con i risultati dello scheduler con preemption
2. -on | --output-no-preemption: il file di output con i risultati dello scheduler senza preemption
3. -i | --input: il file di input contenente la lista dei task, al termine della scansione del file, i task devono essere inviati ai due scheduler nelle stesso ordine in cui sono stati letti.
4. -h | --help: stampa l'helper di come utilizzare il programma

Ad esempio:

```
./simulator -op out_pre.csv -on out_not_pre.csv -i 6_tasks.dat
```

Se questi parametri non vengono forniti, il simulatore termina mostrando un messaggio che descriva l'utilizzo del simulatore.

Input

Su aulaweb è presente il link all'archivio compresso con 5 files di input contenenti la lista dei task, ordinati per "arrival_time" (e saranno sempre ordinati, non dovete ordinarli!), e le relative istruzioni.

Ogni linea incomincia con "t" o con "i"

- t -> task
    - t, id, arrival_time
    - seguito dalle sue "i"struzioni
- i -> istruzione
    - i, type_flag, length

Ad esempio, un file con il seguente contenuto:

```
t,0,2
i,1,6
i,1,3
t,1,23
i,0,4
```

Contiene 2 task, il primo formato da due istruzioni bloccanti, il secondo da un'istruzione non bloccante.

I file sono nominati: x_tasks.dat con x=01, 02, … , 05

Da linea di comando, per scompattare l'archivio, usate atool:

```
$ sudo apt install atool
$ aunpack input_tasks.7z
```

Output

Ogni volta che cambia lo stato di un task, lo scheduler dovrà loggare l'evento (con il nuovo stato) sul file (quindi un file per ogni scheduler, perciò sono necessari 2 file di output) specificato da linea di comando e l'output dovrà essere **RIGOROSAMENTE** formattato come descritto:

```
core,clock,task_id,stato
```

Esempi:

```
core0,123,456,running
core1,42,33,exit
```

Questo formato si chiama CSV.

Specifiche di consegna

Ogni progetto dovrà essere consegnato in un file zip, senza cartelle, contenente solo i seguenti files:

- sorgenti (*.c)
- headers (*.h)
- Makefile
- REPORT.pdf
- AUTHORS.txt

Il Makefile deve essere configurato in modo tale che il comando "`make all`" generi un file eseguibile "`simulator`".
Ricordo che separare correttamente il progetto su più file sorgenti aumenta la qualità del codice.

Il progetto può essere svolto singolarmente o in coppia; in quest'ultimo caso entrambi devono contribuire sia alla scrittura del codice, sia del documento.

Il documento di report "REPORT.pdf" deve contenere:

- descrizione astratta del funzionamento del simulatore

- il confronto tra i due algoritmi di scheduling scelti (sono graditi i grafici)
- giustificazione delle scelte implementative (vedi “considerazioni finali”)
- problematiche incontrate (e risolte) durante lo sviluppo

Nel file "AUTHORS.txt" devono essere specificati "matricola,cognome,nome" degli autori del progetto.

Ad esempio:
123,rossi,mario
456,verdi,luigi

Attenzione: un progetto che non compila (come sistema operativo per i test userò la stessa macchina virtuale con XUbuntu che ho fornito a inizio corso) oppure che genera un output mal formattato non sarà corretto.
Ogni progetto sarà sottoposto a controlli antiplagio (consiglio: non è sufficiente rinominare variabili/funzioni per eludere i controlli). Dato che il progetto è simile a quello dell’anno scorso, saranno controllati anche con i progetti dell’anno passato.

Nel fare i test ho impostato un timeout di 3 minuti (se il simulatore è scritto correttamente riuscite a stare sotto il minuto anche con un processore di 10 anni fa), quindi un simulatore che non termini la sua esecuzione nel tempo indicato sarà considerato sbagliato.
Controllate con il comando “time” la durata dell’esecuzione del vostro programma, altrimenti investigate con i comandi “strace”, “ltrace” e “valgrind” visti a lezione.

```
$ time ./simulator -op 1.txt -on 2.txt -i /home/student/test/input/01_tasks.dat -q
1
real 0m19.174s
user 0m17.996s
sys 0m8.312s
```

Qui trovate la spiegazione dettagliata dell'output:
https://stackoverflow.com/questions/556405/what-do-real-user-and-sys-mean-in-the-output-of-time1

Considerazioni finali

Nell'affrontare un corner case, se non violate le specifiche descritte in questo documento, potete risolvere il problema come volete, basta che giustifichiate la vostra decisione nel documento di report.