ultima modifica 11/08/2017

Scheduler Simulator

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 fornire una comparativa tra i due approcci.

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

Funzionalità

All'avvio il simulatore (che implementa il main) dovrà creare 3 processi:

1. master
2. scheduler_preemptive
3. scheduler_not_preemptive

Il processo "master" legge da file i job da eseguire e li inoltra ad entrambi gli scheduler.

Lo scheduler con preemption adotta la strategia Round Robin, la durata del quanto (time slice) verrà fornita come parametro al simulatore, mentre lo scheduler senza preemption adotta la strategia Shortest Job Next (SJN).

Con "Shortest Job" è inteso il Job con la minor somma della lunghezza di istruzioni che sono ancora da eseguire.
Uno scheduler termina nel momento in cui non ci sono più job da eseguire.

A prescindere dalla strategia, quando un'istruzione si sospende a causa di una operazione bloccante, un altro job deve essere schedulato.

Entità

Il processore di ogni scheduler ha due core. Perciò bisognerà utilizzare 2 thread (reali) per simulare l'esecuzione di 2 job in parallelo. Ogni core avrà quindi il proprio clock simulato con un numero intero che incrementa ad ogni ciclo.

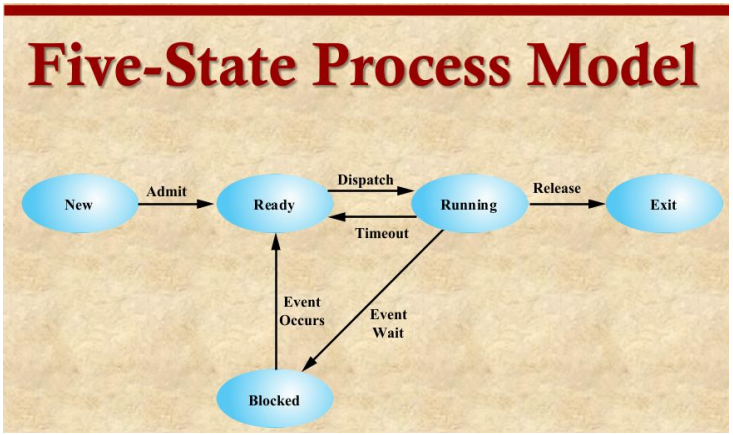
Un'istruzione è composta da:

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

Un job è definito da:

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

Tutti i job letti da file avranno stato "new".
Uno scheduler NON deve mandare in esecuzione un job se: *clock(core)* < *clock(job)*
Un job termina 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 job, al termine della scansione del file, i job devono essere inviati ai due scheduler nelle stesso ordine in cui sono stati letti.
4. -q | --quantum: al durata di un quanto di tempo (misurato in cicli di clock) sullo scheduler con preemption

Ad esempio:

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

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 ai 10 files con la lista dei job (ordinati per "arrival_time") e delle relative istruzioni.

Ogni linea incomincia con "j" o con "i"

- j -> job
    - j, id, arrival_time
    - seguito dalle sue "i"struzioni
- i -> istruzione
    - i, type_flag, length, io_max

Ad esempio, un file con il seguente contenuto:

```
j,0,2
i,1,6,6
i,1,3,8
j,1,23
i,0,4,0
```

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

I file sono nominati: x_jobs.dat con x=01, 02, … , 10
I test verranno effettuati utilizzando il numero del file come quanto di tempo, ad esempio il file "07_jobs.dat" verrà testato con "-q 7".

Output

Ogni volta che cambia lo stato di un job ogni scheduler dovrà loggare l'evento sul file corrispondente e l'output dovrà essere RIGOROSAMENTE formattato come descritto:

```
core,clock,job_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 con tutti i sorgenti e corredato di Makefile. Il comando “`make all`” dovrà generare un file eseguibile “`simulator`”.

Nel file zip dovrà anche essere incluso un file "AUTHORS.txt" (attenzione, è case sensitive) in cui 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).