Sistemi Operativi, Primo Modulo
A.A. 2017/2018
Testo del Primo Homework

Igor Melatti

Come si consegna

Il presente documento descrive le specifiche per l'homework 1. Esso consiste in 2 esercizi, per risolvere i quali occorre scrivere 2 programmi Python (versione 3) che si dovranno chiamare `program01.py` (soluzione del primo esercizio) e `program02.py` (soluzione del secondo esercizio). Per consegnare la soluzione, seguire i seguenti passi:

1. creare una directory chiamata `so1.2017.2018.1.matricola`, dove al posto di `matricola` occorre sostituire il proprio numero di matricola;
2. copiare `program01.py` e `program02.py` in `so1.2017.2018.1.matricola`
3. da dentro quest'ultima directory, creare il file da sottomettere con il seguente comando: `tar cfz so1.2017.2018.1.matricola.tgz program*py`
4. andare alla pagina di sottomissione dell'homework `151.100.17.205/upload/index.php?id_appello=29` e uploadare il file `so1.2017.2018.1.matricola.tgz` ottenuto al passo precedente. **Attenzione:** il suddetto link è raggiungibile solo da indirizzi Sapienza; pertanto, o siete in uno qualsiasi dei laboratori Sapienza, oppure potete settare una VPN come descritto all'URL `https://web.uniroma1.it/sbs/accedi-da-casa/accedi-da-casa-con-bixy#BIXY_info`.

Come si auto-valuta

Per poter autovalutare il proprio homework, è necessario installare VirtualBox (`https://www.virtualbox.org/`), creare una macchina virtuale da 32 bit ed indicare come disco di tale macchina virtuale quello corrispondente a Lubuntu 14.04-3, come scaricabile da `http://www.osboxes.org/lubuntu/`. È necessario installare gawk, in quanto in questa distribuzione di Lubuntu c'è invece mawk. Per farlo, è sufficiente dare i seguenti comandi: `sudo apt-get update && sudo`

`apt-get upgrade && sudo apt-get install gawk` (rispondere NO alla domanda sul passare alla versione successiva di Lubuntu).

Si consiglia di configurare la macchina virtuale con NAT per la connessione ad Internet, e di settare una "Shared Folder" (cartella condivisa) per poter facilmente scambiare files tra sistema operativo ospitante e Lubuntu. Si consiglia inoltre di installare le "Guest Additions" nel seguente modo: dapprima dare il comando `sudo apt-get install dkms` da un terminale all'interno di Lubuntu, poi scegliere "Insert Guest Additions CD Image" dal menu di VirtualBox, e poi di nuovo da terminale di Lubuntu scrivere `cd /media/osboxes/VBOXADDITIONS*; sudo ./VBoxLinuxAdditions.run`. Infine, riavviare Lubuntu.

All'interno di tale macchina virtuale, scaricare il pacchetto per l'autovalutazione (*grader*) dall'URL `http://twiki.di.uniroma1.it/pub/SO/SO1213AL/SistemiOperativi12CFUModulo1Canale120172018/grader.1.tgz`, e copiarlo in una directory con permessi di scrittura per l'utente attuale. All'interno di tale directory, dare il seguente comando:

```
tar xfzp grader.1.tgz && cd grader.1
```

È ora necessario copiare il file `so1.2017.2018.1.matricola.tgz` descritto sopra dentro alla directory attuale (ovvero, `grader.1`). Dopodiché, è sufficiente lanciare `bash grader.1.sh` per avere il risultato: senza argomenti, valuterà tutti e 2 gli esercizi, mentre con un argomento pari ad i valuterà solo l'esercizio i (in quest'ultimo caso, è sufficiente che il file `so1.2017.2018.1.matricola.tgz` contenga solo l'esercizio i).

Esercizio 1

Si implementi un simulatore di uno scheduler round-robin a breve termine (dispatcher) per un singolo microprocessore ad un singolo core. A tal proposito, scrivere una classe SchedulerSimulator che abbia i seguenti metodi:

- costruttore con 2 argomenti: `quantum` e `max_procs`. Gli argomenti hanno il seguente significato: `quantum` è il quanto di tempo (in secondi) e `max_procs` è il numero massimo di processi che possono essere accettati dallo scheduler. Si può assumere che, una volta settati, `quantum` e `max_procs` non vengano mai cambiati. Il tempo interno del simulatore va settato a 0.

- `advance_time(t)`: avanza il tempo del simulatore di `t` secondi (da notare che potrebbe dover eseguire zero, uno o più context switch). Assumere che tutte le operazioni di I/O possano essere eseguite in parallelo (ad esempio perché vengono fatte su dispositivi diversi).

- `add_proc(code_io)`: aggiunge un nuovo processo (come ready), nell'attuale tempo del simulatore. L'argomento `code_io` è una lista di tempi di esecuzione $[t_0, ..., t_{n-1}]$, tali che ogni t_i indica per quanto tempo verrà eseguito il processo prima di arrivare alla prossima istruzione di I/O se i è pari, mentre indica quanto il device di I/O impiega per servire l'attuale operazione di I/O se i è dispari. Questo metodo deve tornare un pid per il processo, inteso come numero intero tra 1 e `max_procs`. Se ci sono già `max_procs` processi che competono per l'esecuzione, questo metodo deve tornare `None`, senza aggiungere il processo.

- `get_ready()`: ritorna la lista dei processi attualmente nella coda dei processi ready, dove ogni elemento della lista è un dizionario con 2 chiavi: `pid` e `code_io`. L'ordinamento della lista dev'essere quello dell'attuale coda dei ready: ovvero, al posto 0 c'è l'elemento che si trova sul fronte della coda (cio, che verrebbe estratto per primo), al posto 1 il secondo e così via. Ovviamente, rispetto a quello indicato con `add_proc`, il `code_io` dev'essere opportunamente modificato tenendo conto dell'avanzamento del tempo (ovvero, delle chiamate ad `advance_time`).

- `get_blocked()`: ritorna la lista dei processi blocked al tempo attuale, con lo stesso formato di `get_ready()`.

- `get_running()`: ritorna il processo attualmente in esecuzione, sempre come dizionario con chiavi `pid` e `code_io`; se nessun processo è attualmente in esecuzione, allora deve tornare `None`.

Non considerare le problematiche degli scheduler a medio e lungo termine; pertanto considerare il modello di processo a 5 stati.

Nel confrontare 2 float o double, assumere che siano uguali se la loro differenza è minore di 10^{-5}.

Un tempo di I/O o di CPU si esaurisce solo quando il tempo del simulatore lo *supera* (non è sufficiente che lo eguagli). Fa eccezione solo l'*ultimo* tempo di CPU: lì è sufficiente che il tempo del simulatore eguagli il tempo di CPU (con il margine d'errore discusso prima), e il processo risulterà terminato.

Assumere che i processi che da blocked possono ritornare ready non generino un interrupt per lo scheduler: pertanto, verranno aggiunti in coda ai ready solo quando scade il prossimo timeout (o c'è la prossima richiesta di I/O), e dopo la gestione dei ready stessi. Se però non c'è nessun processo running (e quindi nessun processo ready) e qualche processo blocked può essere riportato a ready, lo si porta a ready senza aspettare il timeout.

Se necessario, si possono aggiungere altri metodi alle classi.

Per gli esempi vedere il file `esempi/test01.py` e il suo output atteso `esempi/test01.out`.

AVVERTENZE: non usare caratteri non ASCII, come le lettere accentate; non importare moduli che non sono nella libreria standard o che non sono forniti nella cartella dell'homework; non modificare i moduli importati. Se questo file è più grande di 100KB o il grader non termina entro 5 minuti, il punteggio dell'esercizio è zero.

Esercizio 2

Si implementi un simulatore di un gestore di memoria virtuale con paginazione. A tal proposito, scrivere una classe MemorySimulator che abbia i seguenti metodi:

- costruttore con i seguenti argomenti:
  - il numero di bytes `M` contenuti in memoria;
  - il numero di bytes `Mp` contenuti in memoria ed utilizzabili come spazio utente;
  - il numero di bytes `S` contenuti nella parte di memoria secondaria che può essere usata per la gestione della memoria virtuale;
  - il numero di bytes `P` di ciascuna pagina;
  - lista `l` avente un elemento per ogni processo in competizione per l'esecuzione (e che genera richieste in memoria). L'elemento `l[`i`]` contiene il numero massimo di frame che possono essere usati dal processo i. Tali frame si posizionano nell'ordine dato, ma dopo la memoria usata per il sistema operativo: il primo processo ha `l[0]` frame in memoria a partire da `M - Mp`, il secondo gli `l[1]` frame immediatamente successivi, eccetera;
- `handle_request(addr, i)`: gestisce una nuova richiesta di un indirizzo (logico) `addr` fatta da un processo `i`, con $0 \leq$ `i` $< N$. Ritorna una coppia `(frame, ind_fis)`, dove `ind_fis` è l'indirizzo fisico corripondente ad `addr`, e `frame` è l'indice della pagina dove si trova `ind_fis` (la prima pagina ha indice 0);
- `get_memory()`: ritorna una lista `m` contenente un elemento per ogni frame; `m[`f`]` dev'essere `(`p`, `i`)`, dove p è il numero di pagina caricato dentro il frame f (`None` se il frame di indice f non è usato) ed i è l'indice del processo cui la pagina p appartiene. Non mostrare i frame che non sono stati assegnati ad alcun processo dalla lista `l` passata al costruttore
- `get_stats()`: ritorna una coppia `(`p_1`, `p_2`)`, dove p_1 è il numero di page hit fino ad ora, e p_2 è il numero di page miss fino ad ora.

Si possono fare le seguenti assunzioni:

- chi usa la classe chiamerà solamente i metodi descritti sopra;
- sia `M` che `Mp` sono multipli di `P`;
- `P` ed `S` sono potenze di 2;
- la somma degli elementi di `l` è al più uguale al numero di frames in `Mp` bytes;
- gli "indici dei processi" cui si fa riferimento sopra sono quelli della lista `l`;

- inizialmente, la memoria è vuota;
- la memoria di sistema non è paginata;
- quando viene chiamato il costruttore, nessun frame di memoria è usato;
- il numero di processi non aumenta e non diminuisce, ed è sempre almeno 1;
- l'algoritmo da usare per la sostituzione delle pagine è quello dell'orologio;
- non c'è il TLB;
- usare la politica dell'allocazione fissa per il memory resident set;
- i processi usati dal costruttore restano in vita per sempre e non se ne aggiungono altri.

Se necessario, si possono aggiungere altre funzioni ausiliarie.

Per gli esempi vedere il file `esempi/test02.py` e il suo output atteso `esempi/test02.out`.

AVVERTENZE: non usare caratteri non ASCII, come le lettere accentate; non importare moduli che non sono nella libreria standard o che non sono forniti nella cartella dell'homework; non modificare i moduli importati. Se questo file è piu' grande di 100KB o il grader non termina entro 5 minuti, il punteggio dell'esercizio è zero.