Sistemi Operativi

Compito 10-06-2020

Esercizio 1 (10 punti)

Un sistema operativo adotta la politica di scheduling dei thread a code multiple con priorità e prelazione tra le code. Sono presenti due code, la coda A ad alta priorità con scheduling round-robin con quanto q=3ms e la coda B a bassa priorità con scheduling round-robin con quanto q=4ms. Inoltre se un thread della coda B nel suo CPU brust non usa tutto il quanto di tempo viene promosso nella coda A, mentre un thread della coda A che usa tutto il quanto di tempo verrà spostato nella coda B. In caso di prelazione da altra coda il thread prelazionato viene messo in fondo alla coda dei thread pronti.

Il sistema deve schedulare i seguenti thread arrivati in sequenza T_1, T_2, T_3, T_4, nelle code indicate e con uso CPU/IO indicati:

T_1	coda=A	CPU(2ms)/IO(6ms)/CPU(2ms)
T_2	coda=A	CPU(5ms)/IO(2ms)/CPU(2ms)
T_3	coda=B	CPU(3ms)/IO(2ms)/CPU(2ms)
T_4	coda=B	CPU(5ms)/IO(4ms)/CPU(3ms)

Si determini:

1. il **diagramma di Gantt**,
2. il **tempo di attesa** medio,
3. il **tempo di ritorno** medio,
4. il **tempo di risposta** medio,
5. il numero di cambi di contesto

Esercizio 2 (20 punti)

Si vuole realizzare in java una classe *PriorityResourceManager* che gestisce *N* risorse. Le richieste possono essere fatte da *M* thread Requester, nella richiesta il thread specifica il suo id e la priorità della richiesta, un numero tra 0 e *P*-1 con 0 la priorità più alta. Se la risorsa non è disponibile la richiesta viene accodata (con la priorità indicata), quando la risorsa sarà rilasciata verrà concessa al thread in attesa a più alta priorità e a parità di priorità a quello in attesa da più tempo. Se invece al momento della richiesta la risorsa è disponibile il primo thread che la trova disponibile la acquisisce.

Ogni thread Requester R_i (*i*=0..M-1) iterativamente richiede una risorsa a priorità *i* se *i*<*P* e a priorità *P*-1 se *i* >= *P*, aspetta *T1*>0 secondi, rilascia la risorsa e aspetta *T2*>=0 secondi.

Il programma principale deve far partire i thread *Requester* distanziati di un secondo e dopo un minuto fermare i thread in modo che rilascino l'eventuale risorsa posseduta e stampi il numero di volte che ogni thread ha acquisito la risorsa. Nel caso di *N* = 2, *M* = 4 e *P* = 3 si determinino due valori per *T1* e *T2* per i quali almeno un thread entra in starvation e altri due valori di *T1* e *T2* per i quali non si ha starvation.

Realizzare in Java il sistema descritto usando i metodi sincronizzati per la sincronizzazione tra thread.