Sistemi Operativi

Compito 03-06-2021

Esercizio 1 (10 punti)

Un sistema operativo adotta la politica di scheduling dei thread a code multiple con *prelazione* tra le code. Sono presenti due code, una ad alta priorità con scheduling round-robin con quanto q=1ms e una coda a bassa priorità con scheduling FCFS. Inoltre quando un thread nella coda FCFS viene prelazionato questo rimane in testa alla coda dei thread pronti.

Il sistema deve schedulare i seguenti thread con tempi di arrivo, priorità e uso CPU/IO indicati:

T_1 T_{arrivo}=2 pri=H CPU(2ms)/IO(3ms)/CPU(2ms)

T_2 T_{arrivo}=2 pri=H CPU(2ms)/IO(3ms)/CPU(2ms)

T_3 T_{arrivo}=1 pri=L CPU(2ms)/IO(3ms)/CPU(2ms)

T_4 T_{arrivo}=0 pri=L CPU(3ms)/IO(3ms)/CPU(3ms)

(al tempo 2 si consideri che T_1 preceda T_2 nella coda ready)

Si determini: il **diagramma di Gantt**, il **tempo di attesa** medio, il **tempo di ritorno** medio, il **tempo di risposta** medio e il numero di cambi di contesto

Esercizio 2 (20 punti)

Si vuole realizzare il seguente sistema:

Sono presenti N *ClientThread* che producono delle richieste e M *WorkerThread* che le devono gestire e produrre un risultato. Le richieste sono inserite in una *RequestQueue* limitata a K posizioni.

Il *WorkerThread* iterativamente preleva una richiesta e produce una risposta e per farlo usa una risorsa A per tempo T1 e quindi una risorsa B che tiene per tempo T2 prima di rilasciarle entrambe (le risorse iniziali sono NA e NB). Il *ClientThread* dopo aver inviato il messaggio aspetta la risposta da parte del *WorkerThread*.

Ogni *ClientThread* effettua iterativamente le chiamate e determina per ogni chiamata il tempo dall'inserimento nella coda alla ricezione della risposta (usare System.currentTimeMillis() per misurare i tempi) quindi stampa il valore inviato, il valore ricevuto ed il tempo impiegato.

Per il testing il *ClientThread* invia nella richiesta un numero casuale in floating point in [0,100) e il *WorkerThread* restituisce il numero moltiplicato per 2.

Il programma principale deve far partire tutti i thread quindi dopo 10 secondi deve interrompere tutti i thread facendo in modo di liberare le risorse acquisite. Quando tutti i *ClientThread* hanno terminato stampare per ogni *ClientThread* il tempo minimo, massimo e medio delle chiamate (evitando di conteggiare la chiamata finale che è stata interrotta) e il numero di risorse presenti.

Realizzare in java il sistema descritto usando i ***semafori*** per la sincronizzazione tra thread.