Sistemi Operativi

Prima prova in itinere 04-05-2018 - **Compito A**

Esercizio 1 (12 punti)

Un sistema operativo adotta la politica di scheduling dei thread a code multiple con *prelazione* tra le code. Sono presenti due code, una a priorità 1 (alta priorità) con scheduling round-robin con quanto q=2ms, una coda a priorità 2 (bassa priorità) con scheduling round-robin con quanto q=2ms. Inoltre viene adottata una politica di aging che fa aumentare la priorità se il CPU burst viene eseguito tutto entro il quanto di tempo. Inoltre quando a un thread viene fatta prelazione il thread rimarrà in testa alla coda ready e riceverà un nuovo quanto quando verrà eseguito nuovamente.

Il sistema deve schedulare i seguenti thread tutti presenti in coda ready al tempo 0 ed arrivati in sequenza T_1, T_2, T_3, T_4 e tutti nella coda a bassa priorità e con uso CPU/IO indicati:

T_1 CPU(1ms)/IO(4ms)/CPU(3ms)
T_2 CPU(3ms)/IO(2ms)/CPU(3ms)
T_3 CPU(1ms)/IO(2ms)/CPU(2ms)
T_4 CPU(3ms)/IO(2ms)/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 e
6. gli istanti in cui si ha prelazione tra le code.

Esercizio 2 (18 punti)

In un sistema sono presenti N thread *ArrayGeneratorThread* (con N numero pari) che generano ognuno un array di M > 0 numeri float con valori in [-5, 5). N/2 thread *PScalarThread* prendono il risultato di due thread generator adiacenti e calcolano il prodotto scalare. Il thread *CollectorThread* prende i risultati degli N/2 PScalarThread e ne calcola la somma e la stampa.

Il programma principale chiede in input N e M controllando che siano validi quindi fa partire gli N thread ArrayGeneratorThread, gli N/2 PScalarThread ed il CollectorThread.

Realizzare in Java il sistema descritto usando il metodo join per la sincronizzazione tra thread.

Sistemi Operativi

Prima prova in itinere 04-05-2018 - **Compito B**

Esercizio 1 (12 punti)

Un sistema operativo adotta la politica di scheduling dei thread con una coda con priorità, con prelazione. Ai thread viene assegnato un quanto di tempo q=2ms, se un thread termina il quanto verrà eseguito un altro thread pronto ad essere eseguito a più alta priorità ed in attesa da più tempo. Inoltre viene adottata una politica di aging che fa aumentare la priorità se il CPU burst viene eseguito tutto entro il quanto di tempo. Inoltre quando a un thread viene fatta prelazione il thread viene messo in fondo alla coda ready e riceverà un nuovo quanto quando verrà eseguito nuovamente.

Il sistema deve schedulare i seguenti thread tutti presenti in coda ready al tempo 0 ed arrivati in sequenza T_1, T_2, T_3, T_4 e tutti con priorità 2 (bassa priorità) e con uso CPU/IO indicati:

T_1 CPU(3ms)/IO(3ms)/CPU(3ms)
T_2 CPU(1ms)/IO(3ms)/CPU(1ms)
T_3 CPU(1ms)/IO(1ms)/CPU(3ms)
T_4 CPU(3ms)/IO(2ms)/CPU(1ms)

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 e
6. gli istanti in cui si ha prelazione.

Esercizio 2 (18 punti)

In un sistema sono presenti N>0 thread *ArrayGeneratorThread* che generano ognuno un array di M > 0 numeri interi con valori in [-10, 10]. N-1 thread *DistThread* che prendono il risultato di due thread generator e ne calcolano la distanza $\sqrt{\sum_{i=0}^{M-1}(x[i]-y[i])^2}$ e devono calcolare la distanza tra tra tutti gli array e l'array di posizione 0. Il thread *FindMinThread* prende i risultati degli N-1 thread *DistThread* e trova quello che ha distanza minima e stampa posizione e valore minimo trovato.

Il programma principale chiede in input N e M controllando che siano validi quindi fa partire gli N thread *ArrayGeneratorThread*, gli N-1 *DistThread* ed il *FindMinThread*.

Realizzare in Java il sistema descritto usando il metodo join per la sincronizzazione tra thread ed il metodo *double Math.sqrt(double)* per il calcolo della radice quadrata.