SISTEMI OPERATIVI IIN, IDT, IEL, AUS

prima prova in itinere - 26.05.2009

Nome: _____ Cognome: _____

Esercizio 1. (punti 10)

Un insieme di processi è caratterizzato dai tempi di CPU burst e di IO burst, e dalle priorità riportate di seguito (a numeri più piccoli corrispondono priorità maggiori):

P_1: priorità = 1 CPU(2) / IO(3) / CPU(3)
P_2: priorità = 3 CPU(1) / IO(1) / CPU(3)
P_3: priorità = 2 CPU(3) / IO(2) / CPU(2)
P_4: priorità = 2 CPU(2) / IO(3) / CPU(1)

I processi sono arrivati nell'ordine P_1, P_2, P_3, P_4 e al tempo iniziale sono tutti presenti nelle rispettive code di ready.

I processi sono schedulati in accordo ad un algoritmo con *code multiple,* con prelazione tra le code. Ad ogni coda corrisponde una priorità ed ogni coda è gestita da un algoritmo *Shortest Job First* (SJF).

Applicando la precedente politica di scheduling si determinino:

- il diagramma di *Gantt*;
- i tempi di attesa e di ritorno medi;
- il numero di cambi di contesto.

Esercizio 2. (punti 20)

Scrivere il programma Java che opera secondo la logica seguente:

- Il metodo main avvia N thread (N=10), identificati da un numero progressivo tra 0 e N-1. I thread sono divisi in produttori pari (ProducerEven) e dispari (ProducerOdd) che estendono una classe comune Producer (essa stessa già estensione della classe Thread).
- I thread condividono un array di oggetti, di dimensione N istanziato nel metodo main. I thread pari inseriscono nell'array un oggetto di tipo Date (classe java.util.Date), quelli dispari un oggetto di tipo Integer (java.lang.Integer) e quindi terminano. Ogni thread inserisce nella posizione dell'array corrispondente al suo identificatore.
- Un thread consumatore riceve in ingresso il vettore di oggetti da cui preleva prima gli oggetti di posizione pari e poi quelli dispari, stampando a video il contenuto. Ovviamente il thread consumatore prima di prelevare il contenuto dell'array deve assicurarsi che non sia nullo.

Nota: un oggetto di tipo data è generato chiamando il costruttore Date() della classe Date. Un oggetto di tipo intero è generato chiamando il costruttore Integer(int) della classe Integer.

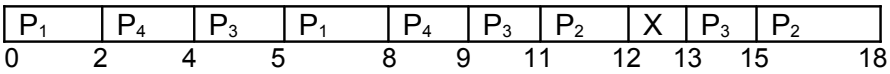
```
// Classe Producer
public class Producer extends Thread {
      protected int id;
      protected Object[] objects;

      public Producer( int i, Object[] obj ) {
            id = i;
            objects = obj;
      }
}
```

Soluzione

Esercizio 1.

Diagramma di Gantt:



All'istante 5, P_3 torna in ready con ancora CPU(2). Torna in esecuzione in 10, dopo P_4 che ha un CPU burst pù breve (CPU(1)).
Tempi di attesa e di ritorno medi:

$T_A = (T_A(P_1) + T_A(P_2) + T_A(P_3) + T_A(P_4))/4 = ((0) + (11+2) + (4+4) + (2+1))/4 = (13 + 8 + 3)/4 = 24/4 = 6$ ms

$T_R = (T_R(P_1) + T_R(P_2) + T_R(P_3) + T_R(P_4))/4 = (8 + 18 + 15 + 9)/4 = 50/4 = 12.5$ ms

Numero cambi di contesto: $N_{cs} = 8$

Esercizio 2.

```
import java.util.Date;

public class Principale {
	// numero produttori e dimensione vettore oggetti prodotti
	public static final int N = 10;

	public static void main( String[] args ) {
		// vettore di oggetti in cui producono i produttori
		Object[] objects = new Object[N];

		// crea e avvia un array di thread produttori
		Producer[] producers = new Producer[N];
		for ( int i=0; i<N-1; i+=2 ) {
			producers[i] = new ProducerEven( i, objects );
			producers[i].start();
			producers[i+1] = new ProducerOdd( i+1, objects );
			producers[i+1].start();
		}
		// crea ed avvia il thread consumatore
		Consumer consumer = new Consumer( producers, objects );
		consumer.start();
	}
}

public class Producer extends Thread {
	protected int id;
	protected Object[] objects;

	public Producer( int i, Object[] obj ) {
		id = i;
		objects = obj;
	}
}
```

```
// thread produttore pari
class ProducerEven extends Producer {
      public ProducerEven( int i, Object[] obj ) {
            super( i, obj );
      }

      public void run() { // produce una data
            objects[id] = new Date();
      }
}

// thread produttore dispari
class ProducerOdd extends Producer {
      public ProducerOdd( int i, Object[] obj ) {
            super( i, obj );
      }

      public void run() { // produce una data
            objects[id] = new Integer(id);
      }
}

// thread consumatore
class Consumer extends Thread {
      private Producer[] producers;
      private Object[] objects;

      public Consumer( Producer[] p, Object[] obj ) {
            producers = p;
            objects = obj;
      }

      public void run() {
            for ( int i=0; i<producers.length; i+=2 ) {
                  try { // attende il termine produttore pari
                        producers[i].join();
                        System.out.println("Producer "+ i +" terminated");
                        System.out.println( "Consumed: " + objects[i] );
                  }
                  catch( InterruptedException ie ) {
                  }
            }
            for ( int i=1; i<producers.length; i+=2 ) {
                  try { // attende il termine produttore dispari
                        producers[i].join();
                        System.out.println("Producer "+ i +" terminated");
                        System.out.println( "Consumed: " + objects[i] );
                  }
                  catch( InterruptedException ie ) {
                  }
            }
      }
}
```

Per come è formulato il problema è accettabile anche un'attesa attiva in sostituzione del metodo join() (while (object[i] == NULL);) che produce una soluzione corretta anche non efficiente.