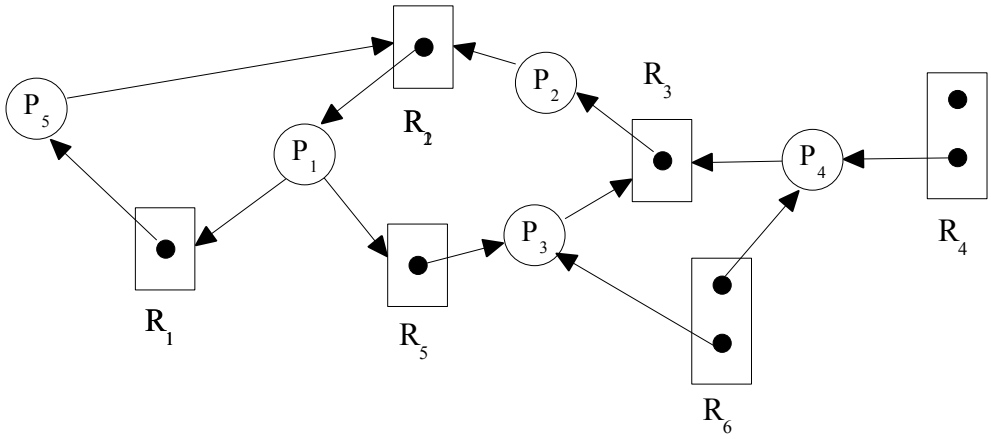
Sistemi Operativi

PROVA SCRITTA INTERMEDIA – 07 giugno 2011

Esercizio 1. (punti 10)

Dato il seguente grafo di allocazione delle risorse:

- indicare gli eventuali cicli presenti, i processi e le risorse coinvolte;
- verificare se esiste stallo ed eventualmente quali sono i processi coinvolti.



Esercizio 2. (punti 20)

In un sistema eseguono 5 thread T0, T1, T2, T3 e T4. I thread devono andare in esecuzione secondo l'ordine T0, T1, T2, T3 e T4. Durante la propria esecuzione ogni thread chiama il metodo `doWork()` per svolgere il lavoro assegnato. L'esecuzione dei 5 thread corrisponde ad un ciclo.
L'esecuzione termina quando il numero di cicli eseguiti è uguale a 10 (ogni thread esegue 10 volte). Quando questo accade tutti i thread devono uscire dai rispettivi metodi `run()` e terminare.
Per risolvere i problemi di sincronizzazione è consentito l'uso dei soli **metodi sincronizzati Java**.

Soluzione

Esercizio 1.

- Nel grafo sono presenti due cicli:
  P_1->R_1->P_5->R_2 e P_2->R_2->P_1->R_5->P_3->R_3->P_2
- Nel sistema esiste uno stallo. I processi coinvolti sono P1, P2, P3, P4, P5.

Esercizio 2.

La soluzione proposta usa notifyAll() per svegliare tutti i thread sospesi su uno stesso oggetto. Una soluzione più efficiente avrebbe potuto usare punti si sospensione e notifica diversi per ogni thread in modo da risvegliare solo un thread alla volta.

```
package compito_20110607;

public class Principale {
	public static final int nthread = 5;
	public static final int ncicli = 10;

	public static void main( String args[] ) {
		Turn turn = new Turn( nthread );
		for ( int i=0; i<nthread; i++ ) {
			Worker wk = new Worker( i, turn, ncicli );
			wk.start();
		}
	}
}

public class Turn {
	private int turn;	// turno
	private int count;	// numero di esecuzioni
	private int nthread;

	public Turn( int nt ) {
		turn = 0;
		count = 0;
		nthread = nt;
	}

	public synchronized void startWork( int id ) {
		while ( turn%nthread != id ) {
			try {
				wait();
			}
			catch( InterruptedException ie ) {
			}
		}
	}

	public synchronized void endWork( int id ) {
		turn = (turn + 1) % nthread;
		count ++;
		if ( count%nthread == 0 ) {
			System.out.println( "fine ciclo " + count/nthread + " count =
" + count );
			System.out.flush();
		}
		notifyAll();
	}
}
```

```
public class Worker extends Thread {
	private int id;
	private Turn turn;
	private int ncicli;

	public Worker( int id, Turn t, int ncicli ) {
		this.id = id;
		this.ncicli = ncicli;
		turn = t;
	}

	public void run() {
		int count = 0;
		while ( count < ncicli ) {
			turn.startWork( id );
			doWork();
			turn.endWork( id );
			count ++;
		}
		System.out.println( "termina thread " + id + " count " + count );
		System.out.flush();
	}

	public void doWork() {
		System.out.println( "esegue thread " + id );
		System.out.flush();
		try {
			Thread.sleep( (long)Math.random()*2000 );
		}
		catch( InterruptedException ie ) {
		}
	}
}
```