

SISTEMI OPERATIVI IDT/IEL/IIN
INFORMATICA INDUSTRIALE E SISTEMI OPERATIVI IDI
SISTEMI DI ELABORAZIONE p.o.
prova scritta del 18 dicembre 2007

Nome _____
Cognome _____
Matricola _____
Corso di laurea _____

Si consideri il seguente problema:

Tre lavoratori A, B e C devono piantare dei semi e collaborano allo scopo nel modo seguente: A scava i buchi, B mette un seme nel buco, e C riempie il buco.

Esistono i seguenti vincoli alla sincronizzazione del lavoro tra A, B e C:

- B non può piantare un seme a meno che non esista una buca vuota, ma B non si cura di quanto è lontano A (cioè di quante buche ha già creato).
- C non può riempire una buca a meno che una buca esista in cui B ha piantato un seme e che non sia stata ancora riempita. C non si cura di quanto è lontano B.
- C si cura del fatto che A non abbia scavato più di MAX buche avanti di C. Così, se ci sono più di MAX buche non riempite A deve aspettare.
- Esiste solo una pala che sia A che C hanno bisogno di usare per scavare e riempire le buche, rispettivamente.

Si delinei la soluzione per i tre processi che rappresentano A, B e C, usando i semafori Java come meccanismo di sincronizzazione.

Soluzione

```
public class Principale {
    public static final int MAX = 10;

    public static void main( String[] args ) {
        // oggetto buca a cui appartengono i metodi per scavare,
        // piantare e riempire una buca
        Buca buca = new Buca();
        // semafora su cui si sospende A se è MAX buche avanti a C
        Semaphore buche = new Semaphore( Principale.MAX );
        // semaforo di mutua esclusione sulla pala
        Semaphore pala = new Semaphore( 1 );
        // semaforo contatore sul numero di buche scavate ma in cui
        // non è piantato un seme
        Semaphore vuote = new Semaphore( 0 );
        // semaforo contatore sul numero di buche scavate in cui è
        // piantato un seme
        Semaphore piene = new Semaphore( 0 );

        WorkerA A = new WorkerA( buca, buche, pala, vuote );
        WorkerB B = new WorkerB( buca, vuote, piene );
        WorkerC C = new WorkerC( buca, buche, pala, piene );
        A. start();
        B. start();
        C. start();
    }
}

class Buca {
    // costruttore
    public Buca() {
    }

    public void scavaBuca() {
        ...
    }

    public void pianta() {
        ...
    }

    public void riempiBuca() {
        ...
    }
}
```

```

public class WorkerA extends Thread {
    private Buca buca;
    private Semaphore buche;
    private Semaphore pala;
    private Semaphore vuote;

    public WorkerA( Buca buca, Semaphore buche, Semaphore pala,
                   Semaphore vuote ) {
        this.buca = buca;
        this.buche = buche;
        this.pala = pala;
        this.vuote = vuote;
    }

    public void run() {
        while ( true ) {
            try {
                // quando arriva a zero si arresta perché A è MAX buche
                // avanti a C
                buche.acquire();
                // acquisisce la pala
                pala.acquire();
                buca.scavaBuca()
                // aggiunge una buca vuota
                vuote.release();
                pala.release();
            }
            catch( InterruptedException ie ) {
            }
        }
    }
}

```

```

public class WorkerB extends Thread {
    private Buca buca;
    private Semaphore vuote;
    private Semaphore piene;

    public WorkerA( Buca buca, Semaphore vuote, Semaphore piene ) {
        this.buca = buca;
        this.vuote = vuote;
        this.vuote = piene;
    }

    public void run() {
        while ( true ) {
            try {
                vuote.acquire();
                // pianta il seme
                buca.pianta();
            }
        }
    }
}

```

```

        // aggiunge una buca piena
        piene.release();
    }
    catch( InterruptedException ie ) {
    }
}

}

public class WorkerC extends Thread {
    private Buca buca;
    private Semaphore buche;
    private Semaphore pala;
    private Semaphore piene;

    public WorkerC( Buca buca, Semaphore buche, Semaphore pala,
        Semaphore piene ) {
        this.buca = buca;
        this.buche = buche;
        this.pala = pala;
        this.piene = piene;
    }

    public void run() {
        while ( true ) {
            try {
                // attende se non ci sono buche piene
                piene.acquire();
                // acquisisce la pala
                pala.acquire();
                buca.riempiBuca()
                pala.release();
                // permette ad A di creare una buca vuota
                buche.release();
            }
            catch( InterruptedException ie ) {
            }
        }
    }
}

```