

/*****

Università della Calabria - Corso di Laurea in Ingegneria Informatica -
A.A. 2003/2004

Proposta di soluzione per l'esame di Sistemi Operativi - 7 gennaio 2005 -
Traccia A

NOTE:

Il sistema descritto può essere modellato mediante due classi:

- Processo: una classe thread usata per rappresentare i processi;
- Canale: una classe monitor che gestisce le richieste dei processi,
e modifica lo stato dei processi e delle risorse disponibili (le linee).

Un semplice main è utilizzato per creare il monitor e per istanziare ed avviare un insieme di oggetti processo.

Ogni processo effettua ciclicamente una trasmissione verso un altro processo. Ogni trasmissione può essere descritta mediante tre fasi:

1. inizia la trasmissione con il processo desiderato;
2. trasmette per un certo tempo;
3. termina la trasmissione.

I passi 2 e 3 sono eseguiti soltanto se il processo non rinuncia alla trasmissione (come spiegato nella traccia).

I passi 1 e 3 devono essere richiesti al monitor, il quale conosce le risorse disponibili (numero di linee libere) nonché lo stato di tutti i processi (liberi o occupati).

Quindi il monitor deve esportare un metodo per avviare la trasmissione (metodo inizia) ed un

metodo per terminare la trasmissione (metodo termina). Questi metodi devono essere synchronized,

in quanto al loro interno si modifica il valore delle diverse variabili utilizzate per tenere

traccia dello stato del sistema. Si ricorda che i metodi sincronizzati sono acceduti in mutua

esclusione. I processi non devono quindi eseguire il comando sleep all'interno di metodi

sincronizzati, altrimenti le trasmissioni vengono eseguite in sequenza, e non in parallelo.

Un classico schema wait/notifyAll è utilizzato nei metodi inizia e termina. In particolare,

la wait è inclusa in un ciclo while perché ogni processo risvegliato deve verificare (tramite

il metodo privato possoTrasmettere) di essere proprio il processo che ne ha diritto. Tra tutti

i processi risvegliati (dalla notifyAll invocata nel metodo termina), soltanto un processo uscirà

dal ciclo while, mentre tutti gli altri ritorneranno in wait.

*****/

```
import java.util.Vector;
```

```
public class Prova2A {
    public static void main (String args[]) {
        int numeroProcessi = 100;
        int numeroLinee = numeroProcessi/3;
        Canale canale = new Canale (numeroProcessi, numeroLinee);
        for (int i = 0; i < numeroProcessi; i++) {
            Processo processo = new Processo (i, numeroProcessi, canale);
            processo.start();
        }
    }
}
```

```

class Processo extends Thread {
    private int mioID;
    private int numeroProcessi;
    private Canale canale;

    public Processo (int mioID, int numeroProcessi, Canale canale) {
        this.mioID = mioID;
        this.numeroProcessi = numeroProcessi;
        this.canale = canale;
    }

    public void run () {
        while (true) {
            attesa (1000,10000); // tra una trasmissione e la successiva
            attende tra 1 e 10 secondi
            int IDRicevente = generaIDRicevente();
            boolean esito = canale.inizia (mioID, IDRicevente);
            if (esito == true) {
                attesa (20000,50000); // la trasmissione dura tra 20 e 50
                secondi
                canale.termina(mioID, IDRicevente);
            }
        }
    }

    private void attesa (int min, int max) {
        try {
            sleep ((int)(Math.random()*(max-min)+min) );
        } catch (InterruptedException e){ System.err.println (e); }
    }

    private int generaIDRicevente () {
        // genera l'ID del processo al quale trasmettere, evitando che
        // coincida con l'ID del processo che trasmette
        int IDGenerato;
        do {
            IDGenerato = (int)(Math.random()*numeroProcessi);
        } while (IDGenerato == mioID);
        return IDGenerato;
    }
}

```

```

class Canale {
    private int numeroProcessi;
    private int numeroLinee;
    private int lineeLibere;
    private boolean processoLibero[]; // processoLibero[i] == true se e
    solo se il processo i-esimo è attualmente libero
    private int IDAtteso[]; // IDAtteso[i] contiene l'ID del processo con
    cui l'i-esimo processo è in attesa di comunicare
    private Vector codaAttesa;

    public Canale (int numeroProcessi, int numeroLinee) {
        this.numeroProcessi = numeroProcessi;
        this.numeroLinee = numeroLinee;
        lineeLibere = numeroLinee;
        processoLibero = new boolean[numeroProcessi];
        IDAtteso = new int[numeroProcessi];
        for (int i=0; i < numeroProcessi; i++) {
            processoLibero[i] = true;
            IDAtteso[i] = -1;
        }
        codaAttesa = new Vector();
    }

    public synchronized boolean inizia (int IDTrasmittente, int
    IDRicevente) {
        if (processoLibero[IDTrasmittente] == false) // il processo
        trasmittente risulta attualmente occupato (quindi non può

```

```

        trasmettere!)
        return false; // rinuncia
    System.out.println (IDTrasmittente+" prova a trasmettere a "+
    IDRicevente);
    if (lineeLibere > 0 && processoLibero[IDRicevente] == false) { //
    il canale non è saturo ma il ricevente è occupato
        System.out.println (IDTrasmittente+" rinuncia perche' "+
        IDRicevente+" e' occupato");
        return false; // rinuncia
    }
    if (lineeLibere > 0 && processoLibero[IDRicevente] == true) { // il
    canale non è saturo ed il ricevente è libero
        processoLibero[IDTrasmittente] = false;
        processoLibero[IDRicevente] = false;
        lineeLibere--;
        System.out.println (IDTrasmittente+
        " inizia la trasmissione verso "+IDRicevente+
        " (linee libere = "+lineeLibere+"");
        return true; // la trasmissione ha avuto inizio e il processo
        lascia il monitor
    }
    // a questo punto il canale è saturo, quindi il processo deve
    mettersi in coda
    System.out.println (IDTrasmittente+
    " va in coda perche' il canale e' saturo");
    // faccio un inserimento ordinato (così la coda contiene gli ID in
    ordine crescente)
    boolean inserito = false;
    for (int i=0; i < codaAttesa.size() && !inserito; i++)
        if ( ((Integer)codaAttesa.elementAt(i)).intValue() >
        IDTrasmittente) {
            codaAttesa.insertElementAt(new Integer(IDTrasmittente),i);
            inserito = true;
        }
    if (!inserito)
        codaAttesa.add(new Integer(IDTrasmittente)); // aggiunge in
        coda
    IDAtteso[IDTrasmittente] = IDRicevente;
    while (!possoTrasmettere(IDTrasmittente,IDRicevente)) {
        try {
            wait();
        } catch (InterruptedException e) {System.err.println(e);}
    }
    // il processo è uscito dalla coda, quindi può trasmettere
    processoLibero[IDTrasmittente] = false;
    processoLibero[IDRicevente] = false;
    lineeLibere--;
    System.out.println (IDTrasmittente+
    " esce dalla coda ed inizia la trasmissione verso "+IDRicevente+
    " (linee libere = "+lineeLibere+"");
    return true; // la trasmissione ha avuto inizio e il processo
    lascia il monitor
}

private boolean possoTrasmettere (int mioID, int IDRicevente) {
    if (lineeLibere == 0 || processoLibero[IDRicevente] == false)
        return false;
    // se c'e' qualcuno, tra quelli in coda prima di me (cioè con ID
    minore), il cui ricevente sia libero, allora non è il mio turno
    int miaPosizioneInCoda = codaAttesa.indexOf(new Integer(mioID)); //
    trovo la mia posizione in coda
    for (int i=0; i < miaPosizioneInCoda; i++) {
        int trasmittente = ((Integer)codaAttesa.elementAt(i)).intValue
        (); // il processo trasmittente in posizione i nella coda
        d'attesa
        int ricevente = IDAtteso[trasmittente]; // il processo a cui
        vuole trasmettere il processo che si trova in posizione i
        if (processoLibero[ricevente] == true)
            return false;
    }
    // è il mio turno; mi rimuovo dalla coda
    codaAttesa.removeElementAt(miaPosizioneInCoda);
}

```

```
        IDAtteso[mioID] = -1;
        return true;
    }

    public synchronized void termina (int IDTrasmittente, int IDRicevente)
    {
        processoLibero[IDTrasmittente] = true;
        processoLibero[IDRicevente] = true;
        lineeLibere++;
        System.out.println (IDTrasmittente+
            " termina la trasmissione verso "+IDRicevente+" (linee libere = "+
            lineeLibere+"));
        notifyAll(); // poichè si liberano una linea e due processi,
        sveglio tutti quelli che sono in wait perchè verifichino se possono
        trasmettere
    }
}
```