

/*****

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 B

NOTE:

Il sistema descritto può essere modellato mediante due classi:
- Utente: una classe thread usata per rappresentare gli utenti;
- CentraleTelefonica: una classe monitor che gestisce le richieste degli utenti,
e modifica lo stato degli utenti e delle risorse disponibili (le linee).
Un semplice main è utilizzato per creare il monitor e per istanziare ed avviare un insieme di oggetti utente.
Ogni utente effettua ciclicamente una telefonata verso un altro utente.
Ogni telefonata può essere descritta mediante tre fasi:
1. chiama l'utente desiderato;
2. conversa per un certo tempo;
3. chiude la conversazione.
I passi 2 e 3 sono eseguiti soltanto se l'utente non rinuncia alla telefonata (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 gli utenti (liberi o occupati).
Quindi il monitor deve esportare un metodo per avviare la telefonata (metodo chiama) ed un metodo per terminare la telefonata (metodo chiudi). 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. Gli utenti non devono quindi eseguire il comando sleep all'interno di metodi sincronizzati, altrimenti le telefonate vengono eseguite in sequenza, e non in parallelo.
Un classico schema wait/notifyAll è utilizzato nei metodi chiama e chiudi. In particolare, la wait è inclusa in un ciclo while perché ogni utente risvegliato deve verificare (tramite il metodo privato possoChiamare) di essere proprio l'utente che ne ha diritto. Tra tutti gli utenti risvegliati (dalla notifyAll invocata nel metodo chiudi), soltanto un utente uscirà dal ciclo while, mentre tutti gli altri ritorneranno in wait.

*****/

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

```
public class Prova2B {  
    public static void main (String args[]) {  
        int numeroUtenti = 100;  
        int numeroLinee = 40;  
        CentraleTelefonica centrale = new CentraleTelefonica (numeroUtenti,  
            numeroLinee);  
        for (int i = 0; i < numeroUtenti; i++) {  
            Utente utente = new Utente (i, numeroUtenti, centrale);  
            utente.start();  
        }  
    }  
}
```

```

}

class Utente extends Thread {
    private int mioNumero;
    private int numeroUtenti;
    private CentraleTelefonica centrale;

    public Utente (int mioNumero, int numeroUtenti, CentraleTelefonica
centrale) {
        this.mioNumero = mioNumero;
        this.numeroUtenti = numeroUtenti;
        this.centrale = centrale;
    }

    public void run () {
        while (true) {
            attesa (1000,10000); // tra una chiamata e la successiva
            attende tra 1 e 10 secondi
            int numeroCorrispondente = generaNumeroCorrispondente();
            boolean esito = centrale.chiama (mioNumero,
            numeroCorrispondente);
            if (esito == true) {
                attesa (20000,50000); // la chiamata dura tra 20 e 50
                secondi
                centrale.chiudi(mioNumero, numeroCorrispondente);
            }
        }
    }

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

    private int generaNumeroCorrispondente () {
        // genera il numero da chiamare, evitando che coincida con il
        numero del chiamante
        int numeroGenerato;
        do {
            numeroGenerato = (int)(Math.random()*numeroUtenti);
        } while (numeroGenerato == mioNumero);
        return numeroGenerato;
    }
}

class CentraleTelefonica {
    private int numeroUtenti;
    private int numeroLinee;
    private int lineeLibere;
    private boolean utenteLibero[]; // utenteLibero[i] == true se e solo se
l'utente i-esimo è attualmente libero
    private int numeroAtteso[]; // numeroAtteso[i] contiene il numero
dell'utente con cui l'i-esimo utente è in attesa di comunicare
    private Vector codaAttesa;

    public CentraleTelefonica (int numeroUtenti, int numeroLinee) {
        this.numeroUtenti = numeroUtenti;
        this.numeroLinee = numeroLinee;
        lineeLibere = numeroLinee;
        utenteLibero = new boolean[numeroUtenti];
        numeroAtteso = new int[numeroUtenti];
        for (int i=0; i < numeroUtenti; i++) {
            utenteLibero[i] = true;
            numeroAtteso[i] = -1;
        }
        codaAttesa = new Vector();
    }
}

```

```

public synchronized boolean chiama (int numeroChiamante, int
numeroChiamato) {
    if (utenteLibero[numeroChiamante] == false) // l'utente chiamante
        risulta attualmente occupato (quindi non può chiamare!)
        return false; // rinuncia
    System.out.println (numeroChiamante+" prova a chiamare "+
numeroChiamato);
    if (lineeLibere > 0 && utenteLibero[numeroChiamato] == false) { //
il canale non è saturo ma il chiamato è occupato
        System.out.println (numeroChiamante+" rinuncia perche' "+
numeroChiamato+" e' occupato");
        return false; // rinuncia
    }
    if (lineeLibere > 0 && utenteLibero[numeroChiamato] == true) { //
il canale non è saturo ed il chiamato è libero
        utenteLibero[numeroChiamante] = false;
        utenteLibero[numeroChiamato] = false;
        lineeLibere--;
        System.out.println (numeroChiamante+
" inizia la conversazione con "+numeroChiamato+
" (linee libere = "+lineeLibere+"));
        return true; // la chiamata ha avuto inizio e l'utente lascia
il monitor
    }
    // a questo punto il canale è saturo, quindi l'utente deve mettersi
in coda
    System.out.println (numeroChiamante+
" va in coda perche' il canale e' saturo");
    codaAttesa.add(new Integer(numeroChiamante));
    numeroAtteso[numeroChiamante] = numeroChiamato;
    while (!possoChiamare(numeroChiamante,numeroChiamato)) {
        try {
            wait();
        } catch (InterruptedException e) {System.err.println(e);}
    }
    // l'utente è uscito dalla coda, quindi può chiamare
    utenteLibero[numeroChiamante] = false;
    utenteLibero[numeroChiamato] = false;
    lineeLibere--;
    System.out.println (numeroChiamante+
" esce dalla coda ed inizia la conversazione con "+numeroChiamato+
" (linee libere = "+lineeLibere+"));
    return true; // la chiamata ha avuto inizio e l'utente lascia il
monitor
}

private boolean possoChiamare (int mioNumero, int numeroChiamato) {
    if (lineeLibere == 0 || utenteLibero[numeroChiamato] == false)
        return false;
    // se c'e' qualcuno, tra quelli in coda prima di me, il cui
chiamato sia libero, allora non è il mio turno
    int miaPosizioneInCoda = codaAttesa.indexOf(new
Integer(mioNumero)); // trovo la mia posizione in coda
    for (int i=0; i < miaPosizioneInCoda; i++) {
        int chiamante = ((Integer)codaAttesa.elementAt(i)).intValue();
        // il chiamante in posizione i nella coda d'attesa
        int chiamato = numeroAtteso[chiamante]; // l'utente desiderato
dal chiamante che si trova in posizione i
        if (utenteLibero[chiamato] == true)
            return false;
    }
    // è il mio turno; mi rimuovo dalla coda
    codaAttesa.removeElementAt(miaPosizioneInCoda);
    numeroAtteso[mioNumero] = -1;
    return true;
}

public synchronized void chiudi (int numeroChiamante, int
numeroChiamato) {
    utenteLibero[numeroChiamante] = true;
    utenteLibero[numeroChiamato] = true;
}

```

```
        lineeLibere++;
        System.out.println (numeroChiamante+" chiude la conversazione con "
        +numeroChiamato+" (linee libere = "+lineeLibere+"");
        notifyAll(); // poichè si liberano una linea e due utenti, sveglio
        tutti quelli che sono in wait perchè verifichino se possono
        chiamare
    }
}
```