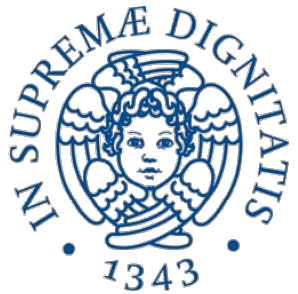
Laboratorio di Informatica

25-10-2016

Damiano Di Francecso Maesa



UNIVERSITÀ DI PISA

Tutor

User

- Studente
- Tesista
- Professore

LaboratorioMain

Tutor

```
public class Tutor {
    //NOTA: gli indici dei computer vanno da 0 a NUMCOMPUTER-1
    // (NON DA 1 a NUMCOMPUTER)
    final int NUMCOMPUTER;
    final ReentrantLock l;
    final Condition professoreInAttesa;
    final Condition studenteInAttesa;
    final Condition[] iesimoOccupato;
    boolean[] computerOccupato;

    public Tutor(int n){
        NUMCOMPUTER=n;
        l=new ReentrantLock();
        professoreInAttesa=l.newCondition();
        studenteInAttesa=l.newCondition();
        iesimoOccupato=new Condition[NUMCOMPUTER];
        for(int i=0;i<NUMCOMPUTER;i++) iesimoOccupato[i]=l.newCondition();
        computerOccupato=new boolean[NUMCOMPUTER];
        for(int i=0;i<NUMCOMPUTER;i++) computerOccupato[i]=false;
    }
```

Tutor

```
private int getComputerLibero(){
    //ritorno l'indice del primo computer non in uso e senza tesisti in attesa
    for(int i=0;i<NUMCOMPUTER;i++)
        if(!computerOccupato[i]&&!l.hasWaiters(iesimoOccupato[i]))
            return i;
    return -1;
}
private int getNumComputerLiberi(){
    int res=0;
    for(int i=0;i<NUMCOMPUTER;i++)
        if(!computerOccupato[i])
            res++;
    return res;
}
```

User

```
public class User implements Runnable {
    final int k;
    final Tutor tutor;
    public User(int i, Tutor t){k=i;tutor=t;}

    public void run(){
        Random r = new Random(System.currentTimeMillis()*2000);
        System.out.println("Inizio utente "+Thread.currentThread().getName());
        try{
        for(int i=0;i<k;i++){
            Thread.sleep(r.nextInt(10000));//attesa prima di inviare la richiesta
            richiediEdUsaComputer(r);
        }
        System.out.println("Fine utente "+Thread.currentThread().getName());
        }catch(InterruptedException e){e.printStackTrace();}
    }
    public void useComputer(Random r){
        try{
            Thread.sleep(r.nextInt(10000));
        }catch(InterruptedException e){e.printStackTrace();}
    }}
```

Studente

```
public class Studente extends User {
    int computerCorrente;
    public Studente(int n, Tutor t){
        super(n,t);
    }
    @Override
    public void richiediEdUsaComputer(Random r){
        this.computerCorrente=tutor.inizioRichiestaStudente();
        useComputer(r);
        tutor.fineRichiestaNoProfessore(computerCorrente);
        System.out.println("Fine studente "+Thread.currentThread(
        ).getName()+" su computer "+computerCorrente);
    }
}
```

Tutor

```
public int inizioRichiestaStudente(){
        l.lock();
        try{try {
            //aspetto se ho professori in coda o non ho computers disponibili
            while(l.hasWaiters(professoreInAttesa)||(getComputerLibero()==-1))
                studenteInAttesa.await();
            int res=getComputerLibero();
            computerOccupato[res]=true;
            return res;
        } catch (InterruptedException ex) {ex.printStackTrace();}
        }finally{l.unlock();}
    return -1;//segnale di errore
    }
    public void fineRichiestaNoProfessore(int s){
        l.lock();
        try{
            //rilascio il computer
            computerOccupato[s]=false;
            //informo che è stato rilasciato se c'è un prof in attesa
            //se non ci sono prof allora se c'è tesista informo lui, altrimenti
            //informo gli studenti che un computer si è liberato
            if(l.hasWaiters(professoreInAttesa))
                professoreInAttesa.signal();
            else if(l.hasWaiters(iesimoOccupato[s]))
                    iesimoOccupato[s].signal();
            else
                studenteInAttesa.signal();
        }finally{l.unlock();}
    }
```

Tesista

```
public class Tesista  extends User {
    final int idx;//indice del computer che gli serve
    public Tesista(int n, Tutor t,int i){
        super(n,t);
        idx=i;
    }
    @Override
    public void richiediEdUsaComputer(Random r){
        tutor.inizioRichiestaTesista(idx);
        useComputer(r);
        tutor.fineRichiestaNoProfessore(idx);
        System.out.println("Fine tesista "+Thread.currentThread(
        ).getName()+" su computer "+idx);
    }
}
```

Tutor

```
public void inizioRichiestaTesista(int idx){
    l.lock();
    try{try {
        //aspetto se ho professori in coda o se il computer è occupato
        while(l.hasWaiters(professoreInAttesa)||computerOccupato[idx])
            iesimoOccupato[idx].await();
        computerOccupato[idx]=true;
    } catch (InterruptedException ex) {ex.printStackTrace();}
    }finally{l.unlock();}
}
//tesista o studente fanno la stessa cosa quando un computer è rilasciato
```

Professore

```
public class Professore  extends User {
    public Professore(int n, Tutor t){
        super(n,t);
    }
    @Override
    public void richiediEdUsaComputer(Random r){
        tutor.inizioRichiestaProfessore();
        useComputer(r);
        tutor.fineRichiestaProfessore();
        System.out.println("Fine professore "+Thread.currentThread(
        ).getName()+" su tutti i computer");
    }
}
```

Tutor

```
public void inizioRichiestaProfessore(){
    l.lock();
    try{try {
        //aspetto se alcuni computers non sono disponibili
        //nota che siccome professoreInAtetsa ha "priorità magiore" allora piano piano
        //i computer si libereranno tutti prima di eseere occupati di nuov da studenti o tesisti
        while(getNumComputerLiberi()!=NUMCOMPUTER)
            professoreInAttesa.await();
        for(int i=0;i<NUMCOMPUTER;i++)
            computerOccupato[i]=true;
    } catch (InterruptedException ex) {ex.printStackTrace();}
    }finally{
        //tolgo lock
        l.unlock();
    }
}
public void fineRichiestaProfessore(){
    l.lock();
    try{
        //rilascio tutti i computer
        for(int i=0;i<NUMCOMPUTER;i++)
            computerOccupato[i]=false;
        //informo che sono tutti liberi se c'è un prof in attesa
        //se non ci sono prof allora informo tutti i tesisti che sono liberi e dopo
        //anche gli studenti. Nota che potrei avere più di un computer libero (dopo
        //che gli eventuali tesisti ne hano occupati alcuni) quindi devo fare signalAll
        if(l.hasWaiters(professoreInAttesa))
            professoreInAttesa.signal();
        else{
            for(int i=0;i<NUMCOMPUTER;i++)
                iesimoOccupato[i].signal();
            studenteInAttesa.signalAll();
        }
    }finally{l.unlock();}}
```

LaboratorioMain

```
public class LaboratorioMain {
    final static int NUMCOMPUTER=20;
    public static void main(String[] args) {
    //LETTURA NUEMRO DI UTENTI DEI VARI TIPI
    Tutor t=new Tutor(NUMCOMPUTER);
    ExecutorService executorS = Executors.newFixedThreadPool(NUMCOMPUTER);
    ExecutorService executorT = Executors.newFixedThreadPool(NUMCOMPUTER);
    ExecutorService executorP = Executors.newFixedThreadPool(NUMCOMPUTER);
    Random r = new Random(System.currentTimeMillis()*3000);
    //generatore per numero random di volte che ogni utente lavora (fissato da me tra 1 e 4)
    //e indice dei computer da assegnare ai tesisti
    //creo gli studenti
    for (int i=1; i<=ns; i++) {
        executorS.execute(new Studente((r.nextInt(4)+1),t));
    }//creo i tesisti
    for (int i=1; i<=nt; i++) {
        //executor.getQueue();
        executorT.execute(new Tesista((r.nextInt(4)+1),t,(r.nextInt(NUMCOMPUTER))));
    }//creo i professori
    for (int i=1; i<=np; i++) {
        //executor.getQueue();
        executorP.execute(new Professore((r.nextInt(4)+1),t));
    }executorS.shutdown();executorT.shutdown();executorP.shutdown();
    while (!executorS.isTerminated()||!executorT.isTerminated()||!executorP.isTerminated()) {
    }
    System.out.println("Finished all Users.");
}}
```

LaboratorioMain

```
//args=new String[3];
//args[0]="3"; args[1]="5";args[2]="20";
if(args.length!=3){
    System.err.println("Must insert exactly three arguments.");
    System.exit(1);
}
int ns=0,np=0,nt=0;
try{
    np=Integer.parseInt(args[0]);
} catch (NumberFormatException e) {
    e.printStackTrace();
    System.err.println("Argument '" + args[0] + "' must be an integer.");
    System.exit(1);
}try{
    nt=Integer.parseInt(args[1]);
} catch (NumberFormatException e) {
    e.printStackTrace();
    System.err.println("Argument '" + args[1] + "' must be an integer.");
    System.exit(1);
}try{
    ns=Integer.parseInt(args[2]);
} catch (NumberFormatException e) {
    e.printStackTrace();
    System.err.println("Argument '" + args[1] + "' must be an integer.");
    System.exit(1);
}
```

Fair ?