Syncronizzazione dei processi

I metodi sincronizzati

- Ogni oggetto di Java ha una risorsa di blocco (**lock**)
- Prima di eseguire il codice di un metodo sincronizzato il thread cerca di acquisire il lock
- Quando termina l'esecuzione di un metodo sincronizzato il thread rilascia il lock
- Se il lock é giá acquisito, il thread corrente aspetta
- I metodi non sincronizzati possono eseguire in concorrenza

Condizioni nei metodi synchronized: **if (NOT condizione_attesa) sospendi il thread**

- La sospensione mette il thread in coda d'attesa
- La sospensione avviene con l'istruzione **wait**
- La sospensione é preceduta dal rilascio del **lock** posseduto dal metodo synchronized
- La **wait** deve essere chiamata all'interno di un metodo synchronized
- Se la JVM vede che la **wait** non é eseguita da un thread che possiede il lock, emette una exception

Riattivazione di un thread che si trova nella coda d'attesa

- La riattivazione viene realizzata con l'istruzione **notify**
- Se la coda dei thread in attesa é vuota la notify non ha effetto
- La notify libera il thread ma esso deve competere per riacquisire il lock
- Se la JVM vede che la **notify** non é eseguita da un thread in attesa del lock, emette una exception

I metodi sincronizzati

La **wait** comporta

- Il rilascio del lock
- L'inserimento del thread in esecuzione in coda d'attesa
- Il passaggio del thread allo stato **NotRunnable**

La **notify** comporta

- La liberazione di un thread in coda d'attesa
- Il thread compete per la riacquisizione del lock
- La **notifyall** libera tutti i thread

Esempio nella comunicazione di messaggi

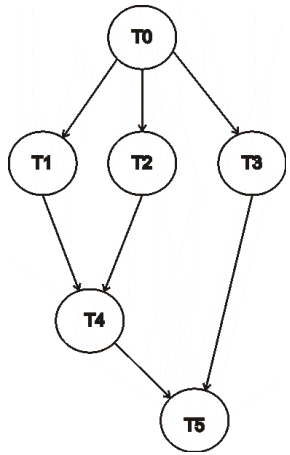
```
class ScambioMessaggi{
  private Messaggio buffer_out, buffer_in ;
  private boolean MessaggioDisponibile=false;
  private boolean BufferDisponibile=true;
  public synchronized void deposita(Messaggio msg){ | public synchronized Messaggio preleva(){
     while(! BufferDisponibile)                      |   while(! MessaggioDisponibile)
       try(wait();) catch (InterruptedException e){}|     try(wait();) catch (InterruptedException e){}
     BufferDisponibile=false;                        |   MessaggioDisponibile=false;
     MessaggioDisponibile=true;                      |   BufferDisponibile=true;
     buffer_out=msg;                                 |   buffer_in=buffer_out;
     notify();                                       |   notify();
  }                                                  |   return(buffer_in);
}                                                    | }
```

Soluzione semaforica alla sincronizzazione

Si supponga di volere imporre al processo P2 di eseguire S2 solo dopo che P1 ha terminato l'esecuzione della istruzione S1. Inizializzando S a zero):

```
--- P1 ------          -------- P2 -------
S1;                    wait(S);
signal(S);             S2;
```

Trovando il semaforo S a zero, il processo P2 viene messo in coda d'attesa, dalla quale viene svegliato solo quando il processo P1 esegue una signal.
Esempio: realizzazione del seguente grafo con la sincronizzazione semaforica



- precedenza di T4: una down sbloccata da T1 e una down sbloccata da T2
- precedenza di T5: una down sbloccata da T3 e una down sbloccata da T4

Sincronizzazione semaforica

```
public class prog {
        public static void main(String[] args) {

          data item= new data(1,2,3,4,5,6);  //3*7-5/6=20.16666
          P1 p1=new P1(item); // istanze
          P2 p2=new P2(item);
          P3 p3=new P3(item);
          P4 p4=new P4(item);
          P5 p5=new P5(item);

          p1.start(); // calcolo concorrente
          p2.start();
          p3.start();
          p4.start();
          p5.start();

          System.out.println(" Fine programma ");
        }
}
```

Sincronizzazione semaforica

```
import java.io.*;
import java.lang.*;
class Semaforo
{
    int value;
    public Semaforo() { value = 0; }
    public Semaforo(int value){this.value = value;}
    public synchronized void  up()
    { value++;  notify(); }
    public synchronized void  down()
    {
      while (value<=0)
        { try { wait(); } catch(InterruptedException e) { }; }
      value--;
    }
}
public class data {
    Semaforo s1,s2,s3,s4,s5;
    public double x, y, z, t, k, a, b, c, d, e, f; // variabili condivise
    public data(){ // costruttori
            x=0; y=0; z=0; t=0; k=0; a=0; b=0; c=0; d=0; e=0; f=0;
    }
    public data(double aa,double bb,double cc,double dd,double ee,double ff) {
        x=0; y=0; z=0; t=0; k=0;
        a=aa; b=bb; c=cc; d=dd; e=ee; f=ff;
        s1=new Semaforo(); s2=new Semaforo();
        s3=new Semaforo(); s4=new Semaforo();
    }
}
```

Sincronizzazione semaforica

```
class P1 extends Thread{
    data item;  public P1(data d) { item=d; } // passaggio dell'oggetto "data" a P1
    public void run(){ item.x=item.a+item.b; System.out.println(" Calcolato x:"+item.x); item.s1.up();}
}
class P2 extends Thread{
    data item;  public P2(data d) { item=d;} // passaggio dell'oggetto "data" a P2
    public void run(){ item.y=item.c+item.d; System.out.println(" Calcolato y:"+item.y); item.s2.up(); }
}
class P3 extends Thread{
    data item;   public P3(data d) {  item=d; } // passaggio dell'oggetto "data" a P3
    public void run(){ item.z=item.e/item.f; System.out.println(" Calcolato z:"+item.z); item.s3.up(); }
}
class P4 extends Thread{
    data item; private double x, y;
    public P4(data d) { item=d; } // passaggio dell'oggetto "data" a P4
    public void run(){
    item.s1.down();item.s2.down();x=item.x;y=item.y;item.t=x*y;System.out.println("Calcolato t:"+item.t);
    item.s4.up();
    }
}
class P5 extends Thread{
    data item; public P5(data d) { item=d; }
    public void run(){
    item.s3.down();item.s4.down(); item.k=item.t-item.z; System.out.println(" Risultato:" + item.k);
    }
}
```

Soluzione SEMAFORICA del produttore/consumatore: il produttore

```
public class P1 extends Thread { // produttore
    private data d; semaforo mutex, pieno, vuoto;
    public P1(data d,semaforo mutex,semaforo pieno,semaforo vuoto){
      this.d=d; this.mutex=mutex; this.pieno=pieno;
      this.vuoto=vuoto;
    }
    public void run (){
       int i=1;
       while(d.conta<50){
           vuoto.down();  //vuoto va inizializzato a 10
           mutex.down();  //mutex va inizializzato a 1
           d.buf[d.IN]=i;
           System.out.println(" P ha inserito "+i
                              +" in  buf["+d.IN+"]");
           d.IN=(d.IN+1)%10;
           i++;  d.conta++;
           mutex.up();
           pieno.up();
          }
    }
```

Soluzione SEMAFORICA del produttore/consumatore: il consumatore

```
public class P2 extends Thread {  // consumatore
    private data d; private int el;
    private semaforo mutex, pieno, vuoto;
    public P2(data d,semaforo mutex,semaforo pieno,semaforo vuoto){
        this.d=d; this.mutex=mutex; this.pieno=pieno;
        this.vuoto=vuoto;
    }
    public void run (){
       while(d.conta<50){
           pieno.down();  //pieno va inizializzato a 0
           mutex.down();  //mutex va inizializzato a 1
           el=d.buf[d.OUT];
           System.out.println("C ha letto buf["+d.OUT+"]="+el+"\n");
           d.OUT=(d.OUT+1)%10; d.conta++;
           mutex.up();
           vuoto.up();
           }
    }
}
```

Il Semaforo

```
public class semaforo extends Thread{
    private int s;
    public semaforo(int n) { s=n; }
    synchronized public void down(){
        if (s<=0) {
            try{wait();} catch(InterruptedException e) {};
        }
        else s--;
    }

     synchronized public void up(){
        s++;
        notify();
    }
}
```

Soluzione errata perche' non e' risolta la mutua esclusione

```
class Cons extends Thread // Classe che realizza il thread del consumatore
{
  Buffer b;  Cons(Buffer p) { this.b=p; }
  public void run() {
    while(true) {
      int el;
      while(b.get_in()==b.get_out()) {System.out.print("");};//Buffer vuoto?
      el=b.get_buf();
      System.out.println("C1 - Ho consumato: " + el + " OUT=" + b.get_out());
      b.put_out((b.get_out()+1)%5);
    }
  }
}
class Prod extends Thread // Classe che realizza il thread produttore
{
  Buffer b; Random r; Prod(Buffer p){ this.b=p; r=new Random(); }
  public void run(){
    int el;
    while (true){
       el=r.nextInt();  // Nuova produzione
       while(b.get_out()==((b.get_in()+1)%5)) {System.out.print("");};//Buffer pieno?
       b.put_buf(el); System.out.println("P  - Ho prodotto:  "+el);
       b.put_in((b.get_in()+1)%5);
    }
  }
}
```

```
class Pc   // Classe principale contentene il main {
  static Buffer buf=new Buffer();
  public static void main(String Arg[])
  {
    Prod  a=new Prod(buf); Cons  b=new Cons(buf); Cons c=new Cons(buf);
    a.start();  b.start();  c.start();
  }
}

P  - Ho prodotto:  93191185
P  - Ho prodotto:  857330274
P  - Ho prodotto:  -1169007862
P  - Ho prodotto:  -740606310
// a questo punto ho generato i primi 4 elementi dell'array
C1 - Ho consumato: 93191185 OUT =0
C1 - Ho consumato: 857330274 OUT =1
C1 - Ho consumato: -1169007862 OUT =2
C1 - Ho consumato: -740606310 OUT =3
// il primo consumatore esegue 4 volte e consuma i 4 elementi
// appena generati
P  - Ho prodotto: 349414068
P - Ho prodotto: 1339793458
// produco altri due elementi
C1 - Ho consumato: 349414068 OUT =4
C2 - Ho consumato: 349414068 OUT =4
// ERRORE: i due consumatori consumano lo stesso elemento!!
...
```

Produttore/consumatore con i monitor di Java.

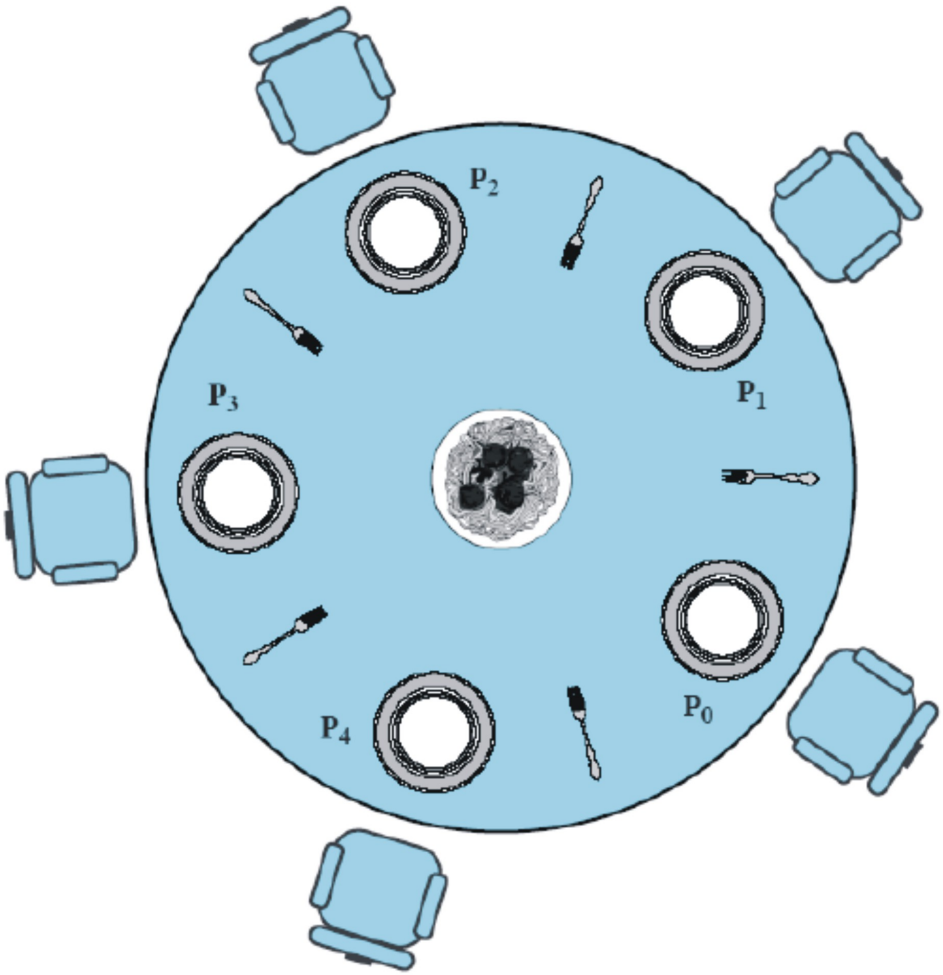
```
class Cons extends Thread {
  Buffer b;  Cons(Buffer p) { this.b=p; }
  public void run()   {
    while(true)     {
      int el=b.get();
      System.out.println("C1 - Ho consumato: " + el);
    }
  }
}

class Prod extends Thread {
  Buffer b;  Random r;
  Prod(Buffer p){ this.b=p; r=new Random(); }
  public void run(){
    while (true){
      int el=r.nextInt(); // Nuova produzione
      b.put(el);
      System.out.println("P  - Ho prodotto:  "+el);
    }
  }
}
```

Produttore/consumatore con i monitor di Java.

```
class Buffer {
  private int buf[]=new int [5];
  private int in; int out;
  Buffer () { in=0; out=0; }
  synchronized int get(){
    while(in==out) // Buffer vuoto?
    { try{ wait(); } catch(InterruptedException e) {}  }
    int el=buf[out];
    out=(out+1)%5;
    notify();
    return el;
  }
  synchronized void put(int el){
    while((in+1)%5==out) // Buffer pieno?
    { try{ wait(); } catch(InterruptedException e) {} }
    buf[in]=el;
    in=(in+1)%5;
    notify();
  }
}
```

La traccia di funzionamento:

```
P - Ho prodotto: -824221474
P - Ho prodotto: -441899243
P  - Hoprodotto: -1772590771
P  - Ho prodotto: 1400672636
//prodotti 4 elementi
C1 - Ho consumato: -824221474
C1 - Ho consumato: -441899243
C1 - Ho consumato: -1772590771
C1 - Ho consumato: 1400672636
// letti 4 elementi. I consumatori si bloccano
...
```


P2
P1
P3
P0
P4

Tentativo di soluzione

```
class Filosofo extends Thread {
    Semaforo sx,dx;     // semafori destro e sinistro
    int numero;         // numero del filosofo
    public Filosofo(int numero,Semaforo sx, Semaforo dx) // costruttore
    { this.sx = sx; this.dx = dx; this.numero = numero;}
    public void run()
    {
      while(true){
        //Pensa
        System.out.println("Filosofo "+numero+" pensa");
        sx.down();
        dx.down();
        System.out.println("Filosofo "+numero+" mangia");
        //mangia
        sx.up();
        dx.up();
      }
    }
}
```

Tentativo di soluzione (cont.)

```
public class main {
    public static void main(String argv[])
    {
    Semaforo s1 = new Semaforo(1); // inizializza semafori
    Semaforo s2 = new Semaforo(1);
    Semaforo s3 = new Semaforo(1);
    Semaforo s4 = new Semaforo(1);
    Semaforo s5 = new Semaforo(1);

    Filosofo f1 = new Filosofo(1,s1,s2); // iniz filosofi
    Filosofo f2 = new Filosofo(2,s2,s3);
    Filosofo f3 = new Filosofo(3,s3,s4);
    Filosofo f4 = new Filosofo(4,s4,s5);
    Filosofo f5 = new Filosofo(5,s5,s1);

    f1.start();f2.start();f3.start();f4.start();f5.start();
    }
}
```

Traccia:

```
# java main
[CUT]
Filosofo 3 mangia
Filosofo 5 mangia
Filosofo 3 pensa
Filosofo 5 pensa
Filosofo 4 mangia
Filosofo 4 pensa
Filosofo 4 mangia
Filosofo 4 pensa
Filosofo 4 mangia
Filosofo 4 pensa
Filosofo 4 mangia
Filosofo 2 mangia
Filosofo 4 pensa
Filosofo 2 pensa
[STALLO]
```

I cinque filosofi: una soluzione funzionante

```
class Filosofo extends Thread {
  Dati dati; int i;
  public Filosofo(int i, Dati dati) {this.dati=dati; this.i=i;}
  public void Prendif(int i) { // Cerca di prendere una forchetta
     dati.mutex.down();
     dati.Stato[i]='A'; Test(i); // A=stato di "Affamato"
     dati.mutex.up();    dati.s[i].down();
  }
  public void Rilasciaf(int i) { // Rilascia la forchetta
     dati.mutex.down();
     dati.Stato[i]='P'; Test(Sx(i)); Test(Dx(i)); // P=stato di "Pensa"
     dati.mutex.up();
  }
  public void Test (int i) {    // controlla se puo' prendere la forchetta
  if(  dati.Stato[i]=='A' && dati.Stato[Dx(i)]!='M' && dati.Stato[Sx(i)]!='M')
       {dati.Stato[i]='M'; dati.s[i].up();} // M=stato di "Mangia"
  }
  public void run()     {
    while(true)     {
       System.out.println("Filosofo "+i+" pensa"); //Pensa
       Prendif(i);
       System.out.println("Filosofo "+i+" mangia"); //Mangia
       Rilasciaf(i);
    }
  }
}
```

Problema dei Lettori/Scrittori

```
class lettore extends Thread {
    buffer dato;
    int numero;          // numero del lettore
    int cont;            // numero di letture
    public lettore(buffer dato,int numero,int cont)
    {this.dato=dato;this.numero=numero;this.cont=cont;}
    public void run()   {
    for (int i=0;i<cont;i++)    {
        dato.mutex.down();
        dato.nlett++;
        if (dato.nlett==1) dato.blocco.down();
        dato.mutex.up();
        System.out.println("Lettore "+numero);
        dato.mutex.down();
        dato.nlett--;
        if (dato.nlett == 0) dato.blocco.up();
        dato.mutex.up();
        }
    }
}
```

Problema dei Lettori/Scrittori (cont.)

```
class scrittore extends Thread {
  buffer dato;        // dati condivisi
  int numero;         // numero dello scrittore
  int cont;           // numero di scritture
  public scrittore(buffer dato,int numero,int cont)
  { this.numero = numero; this.dato = dato; this.cont = cont;  }

  public void run()    {
   for (int i=0;i<cont;i++) {// scrive "cont" volte
      dato.blocco.down();
      System.out.println("Scrittore "+numero+" Num lett "+dato.nlett);
      dato.blocco.up();
    }
  }
}
```

Problema dei Lettori/Scrittori (main)

```
public class main {
    public static void main(String argv[])
    {
    buffer cont= new buffer();
    lettore l1 = new lettore(cont,1,1000); // inzializzo lettori
    lettore l2 = new lettore(cont,2,1000);
    lettore l3 = new lettore(cont,3,1000);
    scrittore s1 = new scrittore(cont,1,5); // inizializzo scrittori
    scrittore s2 = new scrittore(cont,2,5);
    scrittore s3 = new scrittore(cont,3,5);
    l1.start();  // faccio partire i thread
    s1.start();  // in un ordine "misto"
    l2.start();  // per rendere la cosa piu' interessante
    s2.start();
    l3.start();
    s3.start();
    }
}
```

Problema del Barbiere

Uno o piu' barbieri aspettano i clienti. Se non ci sono clienti vanno in wait. Quando arriva un cliente, iniziano a tagliare i capelli.
Se arrivano piú clienti, aspettano nella sala d'aspetto che ha al piú N sedie. Se npn c'é posto, i clienti vano via.
Utilizzabile per descrivere con thread concorrenti il servizio di qualche tipo, con una dimensione della coda d'attesa finita.

Schema di funzionamento:

```
tre semafori: barbiere, cliente, mutex
```

```
  Barbiere
```

```
down(cliente)
down(mutex)
numero clienti --
up(mutex)
up(barbiere)
```

```
  Cliente
```

```
down(mutex)
numero clienti ++
up(mutex)
up(clienti)
down(barbiere)
```

Problema del Barbiere applicato ad un carwash

```
class Car extends Thread
{
    Buffer dato;
    int numero; // numero dell'auto
    public Car(Buffer dato,int numero)
    {  this.numero = numero; this.dato = dato;    }
    public void run() {
    dato.mutex.down();
    if (dato.n < dato.ncar){ // posto disponibile
        dato.n++; // arriva auto
        System.out.println("Auto "+numero+" aspetta carwash");
        System.out.flush();
        dato.auto.up();
        dato.mutex.up();
        dato.wash.down();
     } else  // parcheggio pieno
    {
        System.out.println("Auto "+numero+" parcheggio full");
        System.out.flush();
        dato.mutex.up();
    }
    }
```

```
class Wash extends Thread  {
    Buffer dato;
    int numero; // numero del carwash
    public Wash(Buffer dato,int numero)
    {  this.dato = dato;  this.numero = numero; }
    public void run() {
     while(true)  // carwash lavora sempre
    {
        System.out.println("carwash nr"+numero+" : libero");
        dato.auto.down();
        dato.mutex.down();
        dato.n--; // auto
        System.out.println("carwash "+numero+" sto lavando una macchina");
        dato.wash.up();
        dato.mutex.up();
    }
    }
}
```