Alcuni problemi classici della concorrenza: 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();
          }
    }
}
```

Alcuni problemi classici della concorrenza: 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();
           }
    }
}
```

Alcuni problemi classici della concorrenza: soluzione SEMAFORICA del produttore/consumatore

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();
    }
}
```

Alcuni problemi classici della concorrenza: tentativo di produttore/consumatore Multi-Thread (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();
  }
}
```

Esempio di traccia di funzionamento

```
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 alternanza stretta

La sezione critica dei consumatori viene protetta con alternanza stretta

```
import java.io.*;
class Cons extends Thread { //consumatore
  Buffer b;   Cons(Buffer p) { this.b=p; }
  public void run()
  {
    while(true)
    {
      while(b.get_blocco()==0) {
          System.out.print("");};
      while(b.get_in()==b.get_out()) {
          System.out.print("");};
      int el=b.get_buf();
      System.out.println("C1 - Ho consumato: " + el);
      b.put_out((b.get_out()+1)%5);
      b.put_blocco(0);
    }
  }
}
```

Produttore consumatore con alternanza stretta

Il corretto funzionamento di questo programma puó essere verificato con la seguente traccia di funzionamento dove si nota che il problema precedente non si verifica piú.

```
P - Ho prodotto: 786473915
P - Ho prodotto: -1670086297
P  - Ho prodotto: -842691321
P  - Ho prodotto: 1807022910
// anche in questo caso ho prodotto i primi 4 elementi
C2 - Ho consumato: 786473915
// il secondo consumatore esegue e legge il primo element
P  - Ho prodotto: -87907585
C1 - Ho consumato: -1670086297
C2 - Ho consumato: -842691321
// i due consumatori eseguono in sequenza e leggono corre
// il secondo e terzo elemento
...
```

Produttore/Consumatore con Peterson

```
// ---- Variabili condivise per l'algoritmo di Peterson -
private int blocco1; int blocco2;  int t;
// ------------------------------------------
class Cons extends Thread {
  Buffer b;   Cons(Buffer p) { this.b=p; }
  public void run()   {
    while(true)    {
      b.put_blocco2(1);
      b.put_t(1);
      while((b.get_blocco1()==1) && (b.get_t()==1)) {
          System.out.print("");};
      while(b.get_in()==b.get_out()) {
          System.out.print("");};
      int el=b.get_buf();
      System.out.println("C1 - Ho consumato: " + el);
      b.put_out((b.get_out()+1)%5);
      b.put_blocco2(2);
    }
  }
}
```

Classe Buffer (Peterson)

```
class Buffer {
  private int buf[]=new int [5];
  private int in;
  private int out;
  private int blocco1;
  private int blocco2;
  private int t;
  Buffer () { in=0; out=0; blocco1=1; blocco2=2; }
  int get_in()       { return in;  }
  int get_out()      { return out; }
  int get_blocco1() { return blocco1; }
  int get_blocco2() { return blocco2; }
  int get_buf()      { return buf[out]; }
  int get_t()        { return t; }
  void put_in       (int a)     { in=a; }
  void put_out      (int a)     { out=a; }
  void put_blocco1 (int a)     { blocco1=a; }
  void put_blocco2 (int a)     { blocco2=a; }
  void put_buf      (int a )    { buf[in]=a; }
  void put_t        ( int a )   { t=a; }
}
```

Produttore/consumatore con i monitor di Java. Classe produttore e classe consumatore

```
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);
    }
  }
}
```

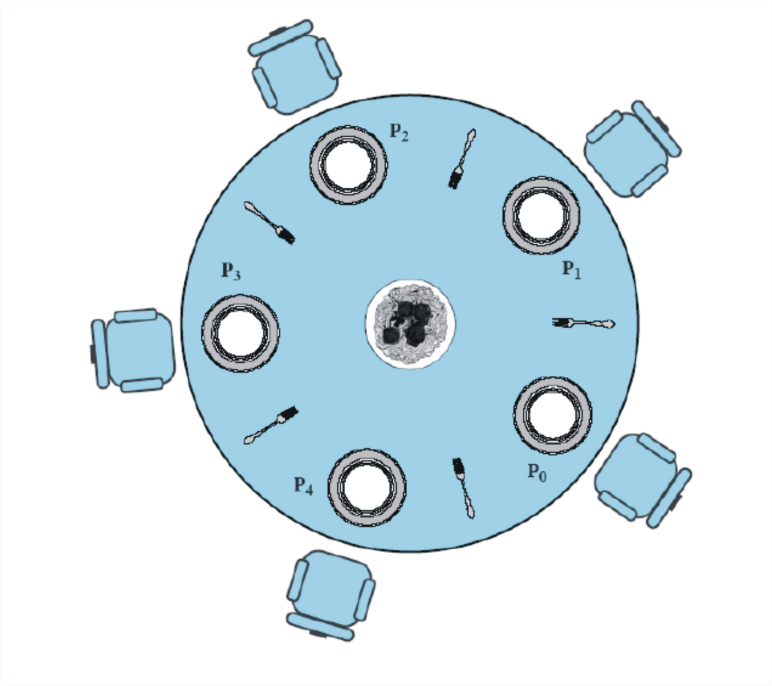
Produttore/consumatore con i monitor di java. Classe condivisa Buffer

```
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
...
```

I cinque filosofi mangiatori di spaghetti: risorse condivise insufficienti



Tentativo di soluzione (sbagliata!)

```
class Filosofo extends Thread {
    Semaforo sx,dx;     // semafori destro e sinistro
    int numero;         // numero del filosofo
    public Filosofo(int numero,Semaforo sx, Semaforo dx)
    { 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();
      }
    }
}
```

I cinque filosofi mangiatori di spaghetti (cont.)

```
public class main {
    public static void main(String argv[])
    {
    Semaforo s1 = new Semaforo(1); // inizializza semafor
    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 (Tanenbaum)

Questa soluzione usa 1) lo STATO DEI FILOSOFI e 2) un semaforo per ogni risorsa (forchetta)

```
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 forch
  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: mentre uso una risorsa, nessuno puó modificarla!

```
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 = c

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

```
public class main {
    public static void main(String argv[])
    {
    buffer cont= new buffer();
    lettore l1 = new lettore(cont,1,1000); // inzializzo
    lettore l2 = new lettore(cont,2,1000);
    lettore l3 = new lettore(cont,3,1000);
    scrittore s1 = new scrittore(cont,1,5); // inizializz
    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                          Cliente

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

Esempio di codice per una stazione di lavamacchine (carwash):

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  // sala piena
    {
        System.out.println("Auto "+numero+" parcheggio full");
        System.out.flush();
        dato.mutex.up();
    }
    }
}
```

Problema del Barbiere applicato ad un carwash

```
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 macchin
        dato.wash.up();
        dato.mutex.up();
    }
    }
}
```