Esercizio sui Semafori

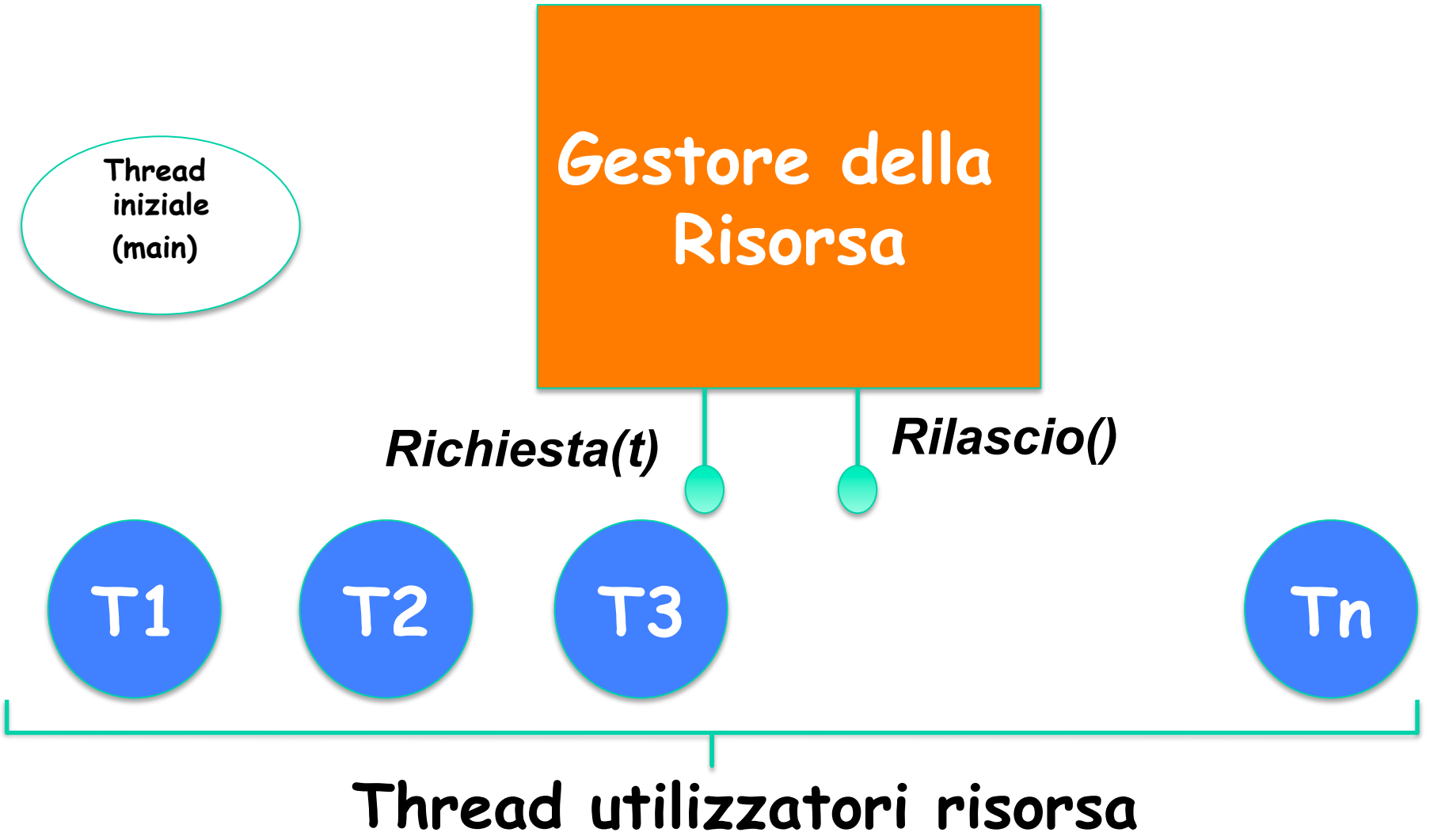
Allocazione di una risorsa con politica prioritaria (SJF)

Esempio 1 - Traccia (1/2)

Si realizzi una applicazione Java che risolva il problema dell'allocazione di una risorsa secondo la politica "*Shortest Job First*":

- **Una sola risorsa** condivisa da più thread
- Ogni thread utilizza la risorsa:
  - In modo **mutuamente esclusivo**
  - In modo **ciclico**
  - Ogni volta, **per una quantità di tempo arbitraria** (stabilita a run-time e dichiarata al momento della richiesta).
- **Politica di allocazione della risorsa**:
  - **SJF**: La precedenza va al thread che intende utilizzarla per il minor tempo.

Impostazione



Impostazione

- **Quali classi ?**
  - **ThreadP**: thread utilizzatori della risorsa; struttura ciclica e determinazione casuale del tempo di utilizzo
  - **Gestore**: mantiene lo stato della risorsa e implementa la politica di allocazione basata su priorità:
    - **Richiesta(t)**: sospensiva se
      - ✓ la risorsa è occupata,
      - ✓ oppure se c'è almeno un processo **più prioritario** (cioè che richiede un tempo minore di t) in attesa
    - **Rilascio()**: rilascio della risorsa ed eventuale risveglio del processo più prioritario in attesa (quello che richiede il minimo t tra tutti i sospesi).
  - **SJF**: classe di test (contiene il main())

Soluzione: classe ThreadP

```
import java.util.Random;

public class ThreadP extends Thread{
      Gestore g;
      Random r;
      int maxt;

      public  ThreadP(Gestore G, Random R, int MaxT)
      {     this.r=R;
            this.g=G;
            this.maxt=MaxT;
      }
```

...classe ThreadP

```
public void run(){
 int i, tau;  long  t;
 try{
   this.sleep(r.nextInt(5)*1000
   tau=r.nextInt(maxt);
   for(i=0; i<15; i++)   {
      g.richiesta(tau);
      this.sleep(tau);
      System.out.print("\n["+i+"]Thread:"+getName()
        +"e ho usato la CPU per "+tau+"ms...\n");
      g.rilascio();
      tau=r.nextInt(maxt);// calcolo nuovo CPU Burst
   }
 }catch(InterruptedException e){}
} //chiude run
}
```

Uso della risorsa.

UN SOLO THREAD ALLA VOLTA!

Impostazione del gestore

Due cause di sospensione:

1. **Accessi al Gestore della risorsa mutamente esclusivi**: 1 **alla volta!** => definisco un semaforo di mutua esclusione

```
semaphore mutex = new Semaphore(1);
```

2. **La risorsa è occupata, oppure c'è almeno un thread più prioritario in attesa:**
   - Quando la risorsa viene liberata deve essere svegliato il processo più prioritario => creiamo un semaforo per ogni livello di priorità:

```
semaphore []codaproc; //1 per ogni liv. Priorità
```

   - Necessità di individuare quanti siano i processo in attesa e la loro priorità:

```
int []sospesi;          //contatori thread sospesi
```

Classe Gestore

```
public class Gestore {
  int n;                    // massimo tempo di uso della risorsa
  boolean libero;
  Semaphore mutex;          //semaforo x la mutua esclusione
  Semaphore []codaproc;     //1 coda per ogni liv. Priorità (tau)
  int []sospesi;            //contatore thread sospesi

  public   Gestore(int MaxTime) {
    int i; this.n=MaxTime;
    mutex = new Semaphore(1);
    sospesi = new int[n];
    codaproc = new Semaphore[n];
    libero = true;
    for(i=0; i<n; i++) {
       codaproc[i]=new Semaphore(0);//semafori "condizione"
       sospesi[i]=0;
    }
  }                    // continua…
```

...classe Gestore

```
/*richiesta per tau ms*/
public void richiesta(int tau){
  int i=0;
  try{
      mutex.acquire();
      while(piu_prio(tau)||libero==false){
            sospesi[tau]++;
            mutex.release();
            codaproc[tau].acquire();
            mutex.acquire();
      }
      libero = false;
      mutex.release();
      }catch(InterruptedException e){}
}
```

...classe Gestore

```
// .. Continua

public void rilascio() {
      int da_svegliare, i;
      try{
            mutex.acquire();
            libero=true;
            da_svegliare = min_sosp();
            if (da_svegliare>=0)
            {     codaproc[da_svegliare].release();
                  sospesi[da_svegliare]--;
            }
            mutex.release();
      }catch(InterruptedException e){}
}
```

Sveglio il processo più prioritario in attesa.

...classe Gestore (metodi utili)

```
private boolean piu_prio(int tau){
        int i=0;
        boolean risposta=false;
        for(i=0; i<tau; i++)
                if (sospesi[i]!=0)
                        return true;
        return risposta;
}

private int min_sosp(){
                int i=0, ris=-1;
                for(i=0; i<n; i++)
                        if (sospesi[i]!=0)
                                return i;
                return ris;
}
```

c'è qualcuno più prioritario di questo thread che userà la risorsa per tau secondi?

Chi è il processo più prioritario (con minor tau) sospeso in coda?

Soluzione: classe sjf

```
import java.util.*;
import java.util.Random;

public class sjf{
 public static void main(String args[]) {
      final int NT=10;//thread
      final int MAXT=500; // quanto di tempo massimo
      int i;
      Random r=new Random(System.currentTimeMillis());
      threadP []TP=new threadP[NT];
      gestore G=new gestore(MAXT);
      for (i=0; i<NT; i++)
            TP[i]=new threadP(G, r, MAXT);
      for (i=0;i<NT; i++)
            TP[i].start();
 }
}
```