1. Assumiamo un array di 100 interi A inizializzato con [0,0,0,0,0,0] condiviso da thread che appartengono a tre tipi:
- thread di tipo 1: ciclicamente generano un numero random k ed eseguono l'operazione
  for(int i = 0; i<A.length; i++) {A[i] = A[i] + k + i;}, che deve essere indivisibile sul dato A.
- thread di tipo 2: ciclicamente generano un numero random k ed effettuano l'operazione
  for(int i = A.length -1; i >= 0; i=i-2) {A[i] = A[i] + k - i;}, che deve essere indivisibile sul dato A.
- thread di tipo 3: ciclicamente eseguono
  int x = 0; for(int i = 0; i<A.length; i++) {x=x+ A[I]; System.out.println(x)}.

Usando i semafori con la semantica tradizionale, scrivere il codice dei 3 tipi di thread, rispettando i seguenti vincoli: un thread può essere in waiting su un semaforo solo se ciò è necessario per garantire le indivisibilità delle operazioni dei thread di tipo 1 e 2, oppure se è necessario per evitare race condition su variabili condivise. Inoltre, quando un thread di tipo 1 termina la propria operazione indivisibile su A, se vi sono thread in attesa, viene data priorità a thread di tipo 2, poi a thread di tipo 3, poi a thread di tipo 1.
Analogamente, quando un thread di tipo 2 termina la propria operazione indivisibile su A, se vi sono thread in attesa, viene data priorità a thread di tipo 1, poi a thread di tipo 3, poi a thread di tipo 2.

2. Assumiamo un array di int A inizializzato con [30,10,20] e condiviso da due thread.
Un thread esegue l'istruzione A[0] = A[0] + A[1].
L'altro thread esegue l'istruzione A[0] = A[0] * A[2].
Si argomenti in modo formale se possono verificarsi race condition su A.

3. Si spieghino i concetti di linking statico e loading statico.

Esercizio 1.

Variabili:

wrk12: numero thread di tipo 1 oppure 2 che stanno lavorando. Valore iniziale 0. Valori possibili: 0,1.
wrk3: numero thread di tipo 3 che stanno lavorando. Valore iniziale 0. Valori possibili: 0,1,2,3,……

tw1: numero thread di tipo 1 in waiting. Valore iniziale 0.
tw2: numero thread di tipo 2 in waiting. Valore iniziale 0.
tw3: numero thread di tipo 3 in waiting. Valore iniziale 0.

Semafori:

mutex. Valore iniziale 1. Serve per garantire accesso alle variabili condivise di cui sopra in sezioni critiche.

s1: Valore iniziale 0. Serve per mettere in waiting i thread di tipo 1.
s2: Valore iniziale 0. Serve per mettere in waiting i thread di tipo 2.
s3: Valore iniziale 0. Serve per mettere in waiting i thread di tipo 3.

Thread tipo 1:

```
while(true){
 // some work having nothing to do with our array
  wait(mutex);
  if(wrk12>0 ll wrk3>0){
      tw1++; signal(mutex); wait(s1);}
  else{
      wrk12++; signal(mutex);
  }

  k= …..
  for(int i = 0; i<A.length; i++) {A[i] = A[i] + k + i;},

  wait(mutex);
  wrk1- -
  if(tw2>0){tw2 - - ; wrk12++; signal(s2);}
  else{
      while(tw3>0){tw3- - ; wrk3++; signal(s3);}
      if (wrk3==0 & tw1>0){tw1- -; wrk12++; signal(s1); }
  }
  signa(mutex);
  // some work having nothing to do with our array
}
```

Thread tipo 2: analogo a thread di tipo 1.

Thread tipo 3:

```
 while(true){
    // some work having nothing to do with our array
    wait(mutex);
    if(wrk12>0){
        tw3++; signal(mutex); wait(s3);}
    else{
       wrk3++; signal(mutex);
    }

    int x = 0; for(int i = 0; i<A.length; i++) {x=x+ A[I]; System.out.println(x)}.
```

```
    wait(mutex);
    wrk3- - ;
    if(wrk3==0 & tw1>0){tw1 - - ; wrk12++; signal(s1);}
    if(wrk3==0 & wrk12==0 & tw2>0){tw2 - - ; wrk12++; signal(s2);}
    signa(mutex);
    // some work having nothing to do with our array
}
```

Funzione che formalizza l'operazione del primo thread (cioè A[0] = A[0] + A[1]):

f1([a, b, c]) = [a+b, b, c].

Nota: in particolare, vale che

f1([30, 10, 20]) = [30+10, 10, 20] = [40, 10, 20] e

f1([600, 10, 20]) = [600+10, 10, 20] = [610, 10, 20].

Funzione che formalizza l'operazione del secondo thread (cioè A[0] = A[0] * A[2]):

f2([a, b, c]) = [a*c, b, c].

Nota: in particolare, vale che

f2([30, 10, 20]) = [30*20, 10, 20] = [600, 10, 20] e

f2([40, 10, 20]) = [40*20, 10, 20] = [800, 10, 20].

Valori ammissibili su A, con A inizializzato con [x, y, z], cioè valori che otterremmo in caso di esecuzione sequenziale delle due operazioni:

f2(f1([x, y, z])) = f2([x+y, y, z]) = [(x+y) * z, y, z]

e

f1(f2([x, y, z])) = f1([x*z, y, z]) = [(x*z) + y, y, z]

Nel nostro esempio:

f2(f1([30,10,20])) = [(30 + 10) * 20, 10, 20] = [40 * 20, 10, 20] = [800, 10, 20]

f1(f2([30,10,20])) = [(30 * 20) + 10, 10, 20] = [600 + 10, 10, 20] = [610, 10, 20]

Possiamo avere r.c. perchè l'array può assumere, per esempio, il valore [40, 10, 20] che è diverso dai due ammissibili.