Esercizio 1

Considerare la seguente proposta (non corretta) di soluzione al problema della sezione critica:

```
boolean pronto[2] = {false,false};

thread T0 {                          thread T1 {
  pronto[0] = true;                    pronto[1] = true;
  while(pronto[1]) {}                  while(pronto[0]) {}
    < sezione critica >                  < sezione critica >
  pronto[0] = false;                   pronto[1] = false;
}                                    }
```

1. Dare una giustificazione del fatto che T0 e T1 non possono mai essere entrambi in sezione critica:
   Supponiamo che T0 stia entrando in sezione critica e che T1 cerchi di entrare anch'esso in sezione critica. Necessariamente pronto[0] deve essere true, perché T0 è già dopo il ciclo while e quindi dopo l'assegnamento di pronto[0]. Questo implica che T1 si bloccherà nel proprio ciclo while. Naturalmente, lo stesso accade se T1 sta entrando in sezione critica e T0 cerca anch'esso di entrare. Di conseguenza T0 e T1 non possono essere entrambi in sezione critica.

2. Mostrare una esecuzione in cui T0 e T1 stallano:
   Se T0 supera l'assegnamento di pronto[0] a true e viene schedulato T1 che anch'esso assegna pronto[1] a true, i due thread rimarranno bloccati nei rispettivi cicli while, senza possibilità di uscita. Ognuno attende un evento che può essere effettuato solo dall'altro thread, in una classica situazione di attesa circolare (stallo).

Esercizio 2

Progettare un Monitor `S` che permetta ai tre thread seguenti di scrivere in modo sincronizzato la parola PROMOSSO:

```
thread PRO {              thread MO {               thread SSO {
   print "PRO";              S.attendi(0);             S.attendi(1);
   S.notifica(0);            print "MO";               print "SSO";
}                            S.notifica(1);          }
                          }
```

Scrivere lo pseudocodice del Monitor illustrandone il funzionamento:

```
Monitor S {
  Boolean notificato[2] = {false, false}; // 2 punti di attesa

  void notifica(int i) {
    notificato[i] = true; // memorizza la notifica i-esima
    notifyAll();          // notifica tutti i thread (non usiamo condition)
  }
  void attendi(int i) {
    while(!notificato[i]) // attende la notifica i-esima
      wait();
  }
```

Il monitor proposto utilizza un array di booleani notificato[2] per memorizzare le due diverse notifiche (0 e 1). Il metodo notifica(i) mette a true la rispettiva notifica (notificato[i]) e sveglia eventuali thread in attesa. Non utilizziamo variabili condition (come in Java) e quindi notifichiamo tutti i thread con notifyAll. Il metodo attendi, mette il thread in attesa della relativa notifica utilizzando un classico ciclo while con una wait. Il ciclo while rimette in thread in attesa nel caso venga svegliato senza che sia presenta la rispettiva notifica.