Sistemi Operativi (modulo II / B) **CON SOLUZIONE** 6 giugno 2018

Esercizio 1

Si considerino `MAX` thread `utente` che prendono un numero e attendono il loro turno nel modo seguente:

```
int numero = 1; // numero, variabile globale
thread utente(i) {
    int prendi_numero; // variabile locale in cui viene copiato il numero

    numero = numero + 1; // incrementa il numero globale
    prendi_numero = numero; // copia in locale il numero

    // attende che venga chiamato prendi_numero
}
```

1. Mostrare un'esecuzione in cui `numero` vale N e due thread prendono il numero N+1
   Normalmente l'incremento di `numero` viene implementato caricando il valore in un registro, incrementandolo e salvando il risultato in memoria. Se due thread leggono N nel proprio registro entrambi computeranno e scriveranno in memoria N+1, perdendo, di fatto, un incremento.
2. Mostrare un'esecuzione in cui `numero` vale N e due thread prendono il numero N+2
   Potrebbe accadere che due thread incrementino in successione `numero` portandolo a N+2 prima di copiarlo in `prendi_numero`. In tal caso N+1 verrebbe "saltato".

3. Proporre una soluzione ai due problemi precedenti basata su semafori
   E' sufficiente utilizzare un semaforo `mutex` e inserire `P(mutex)` e `V(mutex)` prima e dopo l'incremento di numero e la copia in `prendi_numero`. In questo modo le interferenze ai punti precedenti non saranno possibili perché solo un thread alla volta potrà incrementare `numero` e copiarlo nella propria variabile locale `prendi_numero`.

Esercizio 2

Considerare tanti thread di due tipi: `rosso` o `blu`. Progettare un Monitor `M` che sincronizzi i thread in modo che solo thread dello stesso tipo (`rosso` o `blu`) possano entrare contemporaneamente in sezione critica

```
thread rosso {
   M.entraRosso();
   <sez critica>
   M.esceRosso();
}
```

```
thread blu {
   M.entraBlu();
   <sez critica>
   M.esceBlu();
}
```

Scrivere lo pseudocodice del Monitor illustrandone il funzionamento

```
Monitor M {
  int N_rossi=0, N_blu=0; // numero di rossi e blu in sezione critica

  void entraRosso() {
      while(N_blu>0)
          wait(); // attende se ci sono blu
      N_rossi++;  // un rosso entra
  }
  void esceRosso() {
      N_rossi--; // un rosso esce
      if (N_rossi==0) // l'ultimo sveglia i blu
          notifyAll();
  }
```

```
  void entraBlu() {
      while(N_rossi>0)
          wait(); // attende se ci sono rossi
      N_blu++;    // un blu entra
  }
  void esceBlu() {
      N_blu--;
      if (N_blu==0) // l'ultimo sveglia i rossi
          notifyAll();
  }
```

Il monitor proposto utilizza due variabili `N_rossi` e `N_blu` per contare il numero di thread rossi e blu in sezione critica. Quando un `rosso` vuole entrare attende se ci sono thread `blu` in sezione critica. Se non ci sono thread blu incrementa `N_rossi` e accede. Quando esce decrementa `N_rossi` e se è l'ultimo rosso notifica eventuali thread `blu` tramite una `notifyAll()`. I thread `blu` si comportano in modo simmetrico.