1. 依照 6.2 的題目敘述，證明其演算法滿足三個 critical-section problem 需求

```
do {
   flag[i] = true;

   while (flag[j]) {
      if (turn == j) {
         flag[i] = false;
         while (turn == j)
            ; /* do nothing */
         flag[i] = true;
      }
   }

      /* critical section */

   turn = j;
   flag[i] = false;

      /* remainder section */
} while (true);
```

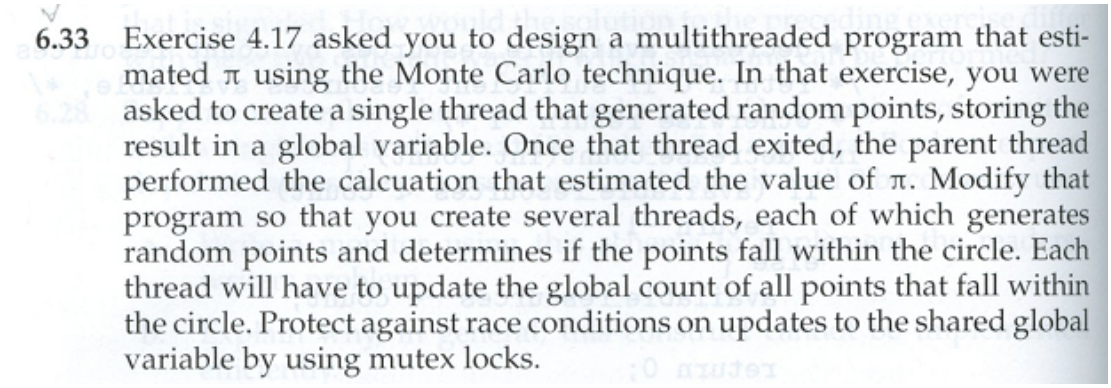
Figure 6.21 The structure of process P_i in Dekker's algorithm.

6.2 The first known correct software solution to the critical-section problem for two processes was developed by Dekker. The two processes, P_0 and P_1, share the following variables:

```
boolean flag[2]; /* initially false */
int turn;
```

The structure of process P_i (i == 0 or 1) is shown in Figure 6.21. The other process is P_j (j == 1 or 0). Prove that the algorithm satisfies all three requirements for the critical-section problem.

2. 題目要求修改原本的單執行緒的程式，設計一個多執行緒的程式，使用 ***Mutex Locks*** 方法估算圓周率 π 的值。
   原本的程式只有一個執行緒負責生成隨機點，並將結果存儲在一個全域變數中。在該執行緒結束後，父執行緒進行計算，估算出圓周率的值。
   現在需要修改程式，創建多個執行緒，每個執行緒負責生成隨機點並判斷這些點是否落在圓內。



6.33 Exercise 4.17 asked you to design a multithreaded program that estimated π using the Monte Carlo technique. In that exercise, you were asked to create a single thread that generated random points, storing the result in a global variable. Once that thread exited, the parent thread performed the calcuation that estimated the value of π. Modify that program so that you create several threads, each of which generates random points and determines if the points fall within the circle. Each thread will have to update the global count of all points that fall within the circle. Protect against race conditions on updates to the shared global variable by using mutex locks.

** 請使用 ***Pthreads Mutex Locks***，除了 main thread 外，請再生出 5 個 threads，每個 thread 產生 1000 個 random points 後結束，main thread 最後再驗收成果且計算出 π 值。