6.3 The first known correct software solution to the critical-section problem for *n* processes with a lower bound on waiting of *n* − 1 turns was presented by Eisenberg and McGuire. The processes share the following variables:

```
enum pstate {idle, want_in, in_cs};
pstate flag[n];
int turn;
```

All the elements of `flag` are initially `idle`; the initial value of `turn` is immaterial (between 0 and `n`-1). The structure of process P_i is shown in Figure 6.22. Prove that the algorithm satisfies all three requirements for the critical-section problem.

```
do {
  while (true) {
    flag[i] = want_in;
    j = turn;

    while (j != i) {
      if (flag[j] != idle) {
        j = turn;
      else
        j = (j + 1) % n;
    }

    flag[i] = in_cs;
    j = 0;

    while ( (j < n) && (j == i || flag[j] != in_cs))
      j++;

    if ( (j >= n) && (turn == i || flag[turn] == idle))
      break;
  }

    /* critical section */

  j = (turn + 1) % n;

  while (flag[j] == idle)
    j = (j + 1) % n;

  turn = j;
  flag[i] = idle;

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

Figure 6.22 The structure of process P_i in Eisenberg and McGuire's algorithm.

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。