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:

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

All the elements of \( \text{flag} \) are initially \( \text{idle} \); the initial value of \( \text{turn} \) is immaterial (between 0 and \( n - 1 \)). The structure of process \( P_i \) is shown in Figure 6.26. Prove that the algorithm satisfies all three requirements for the critical-section problem.

6.5 A file is to be shared among different processes, each of which has a unique number. The file can be accessed simultaneously by several processes, subject to the following constraint: The sum of all unique numbers associated with all the processes currently accessing the file must be less than \( n \). Write a monitor to coordinate access to the file.

6.10 Why do Solaris, Linux, and Windows XP use spinlocks as a synchronization mechanism only on multiprocessor systems and not on single-processor systems?

6.14 Show that the two-phase locking protocol ensures conflict serializability.

6.19 Describe two kernel data structures in which race conditions are possible. Be sure to include a description of how a race condition can occur.

6.24 Windows Vista provides a new lightweight synchronization tool called slim reader–writer locks. Whereas most implementations of reader–writer locks favor either readers or writers, or perhaps order waiting threads using a FIFO policy, slim reader–writer locks favor neither readers nor writers, nor are waiting threads ordered in a FIFO queue. Explain the benefits of providing such a synchronization tool.
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 6.26  The structure of process $P_i$ in Eisenberg and McGuire's algorithm.