5.7 Race conditions are possible in many computer systems. Consider a

banking system that maintains an account balance with two functions:

deposit(amount) and withdraw(amount). These two functions are

passed the amount that is to be deposited or withdrawn from the bank

account balance. Assume that a husband and wife share a bank account.

Concurrently, the husband calls the withdraw() function and the wife

calls deposit(). Describe how a race condition is possible and what

might be done to prevent the race condition from occurring.

Assume that bank account balance is 100 dollars. Husband calls the withdraw() function in order to withdraw 10 dollars from the bank account.After husband withdraw money from bank account, the balance is 90 dollars. However, the woman calls deposit() function to add 10 dollars to the bank account. And the deposit() function takes 100 dollars as initial balance instead of 90 dollars. At the end the balance is 110 dollars, which is wrong. However, the correct balance is 100 dollars.

In order to prevent the race condition, we can use lock mechanism. When husband uses withdraw() function to withdraw money from bank account, the transaction withdraw() will lock the data item—balance, so that other transaction can not read or write the data item—balance. So after transaction withdraw() finishes, the lock held by the transaction will be released. Then transaction deposit() can operate the data item—balance.

**5.8** 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 *Pi* (i == 0 or 1) is shown in Figure 5.21. The other process is *Pj* (j == 1 or 0). Prove that the algorithm satisfies all three requirements for the critical-section problem.

1. Mutual exclusion is achieved by the flags and turn variables. The mutual exclusion means if one process is executing in its critical section, then no other processes can be executing in their critical sections.

Firstly, it’s impossible that flag[i] and flag[j] are both true, because only one process can access to the critical section.

Secondly, if all flags are true, the turn can only be either i or j. Therefore, there is only one process can access the critical section at one time.

Thirdly, only one of flag[i] and flag[j] is true which means that only one process can access to the critical section.

1. Progress means that if one process leaves the critical section, it should release the key to let other process get the key immediately without waiting. In this case, the process which leaves the critical section will set turn to the value of the other process and set flag of itself to false so that the other process can access to the critical section. Therefore, the other process can access the critical section immediately if this process is waiting now.
2. Bound waiting is achieved also by turn variable. In this case, if i and j processes want to access the critical section and only one can access it, after this one exits the critical section, it sets the turn to another process which can let another process enter the critical section. Therefore, the waiting process wouldn’t have to wait indefinitely while the first process repeatedly entered, the waiting process will enter the critical section just after the exit of the other process.