

6.1 Race conditions are possible in many computer systems. Consider a banking system with two methods: `deposit(amount)` and `withdraw(amount)`. These two methods are passed the amount that is to be deposited or withdrawn from a bank account. Assume that a husband and wife share a bank account and that concurrently the husband calls the `withdraw()` method and the wife calls `deposit()`. Describe how a race condition is possible and what might be done to prevent the race condition from occurring.

```
#define MAX_PROCESSES 255
int number_of_processes = 0;

/* the implementation of fork() calls this function */
int allocate_process() {
    int new_pid;

    if (number_of_processes == MAX_PROCESSES)
        return -1;
    else {
        /* allocate necessary process resources */
        ++number_of_processes;

        return new_pid;
    }
}

/* the implementation of exit() calls this function */
void release_process() {
    /* release process resources */
    --number_of_processes;
}
```

Figure 6.23 Allocating and releasing processes.

6.14 Consider the code example for allocating and releasing processes shown in Figure 6.23.

- Identify the race condition(s).
- Assume you have a mutex lock named `mutex` with the operations `acquire()` and `release()`. Indicate where the locking needs to be placed to prevent the race condition(s).

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 calculation 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.

**** using Pthreads Mutex Locks**