11.7 Summary

In this chapter, we introduced the concept of threads and discussed the POSIX.1 primitives available to create and destroy them. We also introduced the problem of thread synchronization. We discussed five fundamental synchronization mechanisms—mutexes, reader–writer locks, condition variables, spin locks, and barriers—and we saw how to use them to protect shared resources.

Exercises

11.1 Modify the example code shown in Figure 11.4 to pass the structure between the threads properly.

11.2 In the example code shown in Figure 11.14, what additional synchronization (if any) is necessary to allow the master thread to change the thread ID associated with a pending job? How would this affect the `job_remove` function?

11.3 Apply the techniques shown in Figure 11.15 to the worker thread example (Figures 11.1 and 11.14) to implement the worker thread function. Don't forget to update the `queue_init` function to initialize the condition variable and change the `job_insert` and `job_append` functions to signal the worker threads. What difficulties arise?

11.4 Which sequence of steps is correct?

1. Lock a mutex (`pthread_mutex_lock`).
2. Change the condition protected by the mutex.
3. Signal threads waiting on the condition (`pthread_cond_broadcast`).
4. Unlock the mutex (`pthread_mutex_unlock`).

or

1. Lock a mutex (`pthread_mutex_lock`).
2. Change the condition protected by the mutex.
3. Unlock the mutex (`pthread_mutex_unlock`).
4. Signal threads waiting on the condition (`pthread_cond_broadcast`).

11.5 What synchronization primitives would you need to implement a barrier? Provide an implementation of the `pthread_barrier_wait` function.