#include <semaphore.h>
sem_t mutex;

/* create the semaphore */
sem_init(&mutex, 0, 1);

/* acquire the semaphore */
sem_wait(&mutex);

/*** critical section ***/

/* release the semaphore */
sem_post(&mutex);

Figure 6.31

In this example, by passing the flag 0, we are indicating that this semaphore can only be shared by threads belonging to the same process that created the semaphore. A nonzero value would allow other processes to access the semaphore as well. In this example, we initialize the semaphore to the value 5.

In Section 6.5, we described the classical wait() and signal() semaphore operations. Pthreads names the wait() and signal() operations sem_wait() and sem_post(), respectively. The code example shown in Figure 6.31 creates a binary semaphore mutex with an initial value of 1 and illustrates its use in protecting a critical section.

Win32

Details concerning thread creation using the Win32 API are available in Chapter 4. Please refer to that chapter for specific instructions.

Win32 Mutex Locks

Mutex locks are a type of dispatcher object, as described in Section 6.8.2. The following illustrates how to create a mutex lock using the CreateMutex() function:

```c
#include <windows.h>
HANDLE Mutex;
Mutex = CreateMutex(NULL, FALSE, NULL);
```

The first parameter refers to a security attribute for the mutex lock. By setting this attribute to NULL, we are disallowing any children of the process creating this mutex lock to inherit the handle of the mutex. The second parameter indicates whether the creator of the mutex is the initial owner of the mutex lock. Passing a value of FALSE indicates that the thread creating the mutex is not the initial owner; we shall soon see how mutex locks are acquired. The third parameter allows naming of