the mutex. However, because we provide a value of NULL, we do not name the mutex. If successful, CreateMutex() returns a HANDLE to the mutex lock; otherwise, it returns NULL.

In Section 6.8.2, we identified dispatcher objects as being either signaled or nonsignaled. A signaled object is available for ownership; once a dispatcher object (such as a mutex lock) is acquired, it moves to the nonsignaled state. When the object is released, it returns to signaled. Mutex locks are acquired by invoking the WaitForSingleObject() function, passing the function the HANDLE to the lock and a flag indicating how long to wait. The following code demonstrates how the mutex lock created above can be acquired:

```c
WaitForSingleObject(Mutex, INFINITE);
```

The parameter value INFINITE indicates that we will wait an infinite amount of time for the lock to become available. Other values could be used that would allow the calling thread to time out if the lock did not become available within a specified time. If the lock is in a signaled state, WaitForSingleObject() returns immediately, and the lock becomes nonsignaled. A lock is released (moves to the signaled state) by invoking ReleaseMutex(), such as:

```c
ReleaseMutex(Mutex);
```

### Win32 Semaphores

Semaphores in the Win32 API are also dispatcher objects and thus use the same signaling mechanism as mutex locks. Semaphores are created as follows:

```c
#include <windows.h>

HANDLE Sem;
Sem = CreateSemaphore(NULL, 1, 5, NULL);
```

The first and last parameters identify a security attribute and a name for the semaphore, similar to what was described for mutex locks. The second and third parameters indicate the initial value and maximum value of the semaphore. In this instance, the initial value of the semaphore is 1, and its maximum value is 5. If successful, CreateSemaphore() returns a HANDLE to the mutex lock; otherwise, it returns NULL.

Semaphores are acquired with the same WaitForSingleObject() function as mutex locks. We acquire the semaphore `Sem` created in this example by using the statement:

```c
WaitForSingleObject(Semaphore, INFINITE);
```

If the value of the semaphore is > 0, the semaphore is in the signaled state and thus is acquired by the calling thread. Otherwise, the calling thread blocks indefinitely—as we are specifying INFINITE—until the semaphore becomes signaled.