


Windows Semaphores 

In Windows, semaphores are declared using the API function *CreateSemaphore()*.

```
HANDLE CreateSemaphore (SecurityAttributes,  
                        InitialCount, MaxCount, SemaphoreID);
```

down and up operations are performed using the API functions *WaitForSingleObject()* and *ReleaseSemaphore()*.

Windows provides other synchronization objects such as mutexes and critical sections.

ELC 467– Spring 2020 Lecture 6 Page 7

### Slide 1

The *CreateSemaphore* function returns a handle to the semaphore. Security attributes are used if semaphore will be accessed by other processes. Initial count is the initial value given to the semaphore. A feature that was not present in theoretical definition of semaphores is that a maximum value can be specified. Last argument is a pointer to a variable that holds the semaphore ID. Note that semaphore is declared in a way completely different from normal integer variables.

### Slide 2

*WaitForSingleObject* can be used for waiting for different types of objects, for example another thread. When applied on a semaphore handle, it performs the down operation on this semaphore. *ReleaseSemaphore* performs the up operation, with additional options that can be found in the function documentation.

### Slide 3

Mutexes are binary semaphores used for mutual exclusion.

Example 3

```

DWORD WINAPI Fn1(LPVOID param)
{
    int i;
    while(1){ gotoxy(10,10); i=i+1;
                printf("Thread 1 %d",i);
            }
}

DWORD WINAPI Fn2(LPVOID param)
{
    int i;
    while(1){ gotoxy(10,20); i=i+1;
                printf("Thread 2 %d",i);
            }
}

```

ELC 467— Spring 2020 Lecture 6 Page 8

Slide 4

In this example, we run two threads executing the above two functions. Each function displays the value of an integer which is continuously incremented. gotoxy is a function that moves the cursor to a particular position and line. Each thread writes its continuously increasing count in a different position.


Example 3

Slide 5

Click on the video to see the program running. The output was not as expected. Instead of writing the count value in the same location each time, threads writes the count sometimes in wrong location, or after the output of the other thread. Try to explain this before going to the next slide.

Slide 6

This occurs since thread may be preempted between the gotoxy and the printf functions. When returning, the cursor position, which is a shared variable, was modified by the other thread, causing the output to appear at an unexpected location. We thus expect this to be corrected if we consider the part from gotoxy statement to printf as a critical section.

Windows Semaphores 

```

main ( )
{
    DWORD ThreadID1,ThreadID2;
    char Sem; char c;

    hsem = CreateSemaphore (NULL,1,1,&Sem);

    HANDLE ht1 = CreateThread (NULL,0,Fn1,NULL,0,&ThreadID1);
    HANDLE ht2 = CreateThread (NULL,0,Fn2,NULL,0,&ThreadID2);


    while (c != 'e') {c = getche();}
}

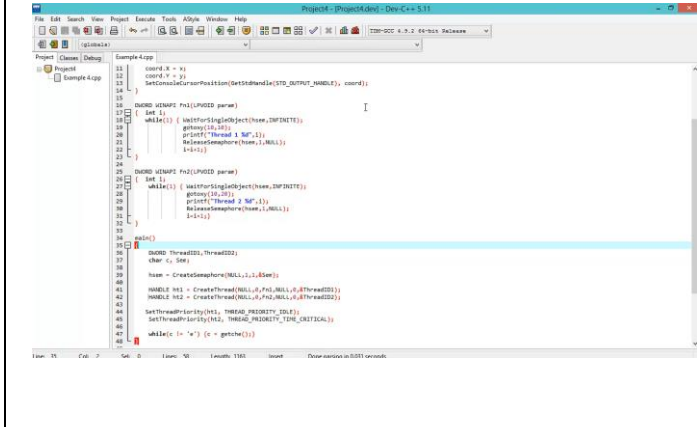
```

ELC 467- Spring 2020 Lecture 6 Page 9

Slide 7

We declare a semaphore with handle hsem. We put the previous critical section between WaitforsingleObject and ReleaseSemaphore operations on hsem, as seen on next slide.

Example 4 



```

11 coord.X = x;
12 coord.Y = y;
13 SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), coord);
14
15
16 DWORD WINAPI Fn1(LPVOID param)
17 {
18     Def();
19     while(1) { WaitForSingleObject(hsem,INFINITE);
20         getch();
21         printf("Thread 1\n");
22         ReleaseSemaphore(hsem,1,NULL);
23         Sleep(1);
24     }
25
26 DWORD WINAPI Fn2(LPVOID param)
27 {
28     Def();
29     while(1) { WaitForSingleObject(hsem,INFINITE);
30         getch();
31         printf("Thread 2\n");
32         ReleaseSemaphore(hsem,1,NULL);
33         Sleep(1);
34     }
35 }
36
37 int Def()
38 {
39     DWORD ThreadID1,ThreadID2;
40     char c;
41     hsem = CreateSemaphore(NULL,1,1,&Sem);
42     HANDLE ht1 = CreateThread(NULL,0,Fn1,NULL,0,&ThreadID1);
43     HANDLE ht2 = CreateThread(NULL,0,Fn2,NULL,0,&ThreadID2);
44     SetThreadPriority(ht1, THREAD_PRIORITY_IDLE);
45     SetThreadPriority(ht2, THREAD_PRIORITY_TIME_CRITICAL);
46     while(c != 'e') {c = getch();}
47 }

```

Slide 8

As expected, the output now appears always in the correct location. Next lecture, we will see the effect of priorities and multiple cores on thread operation. Try to compile, run, and experiment with the examples of this lecture. Programs should run with minor modifications on any C compiler under Windows.