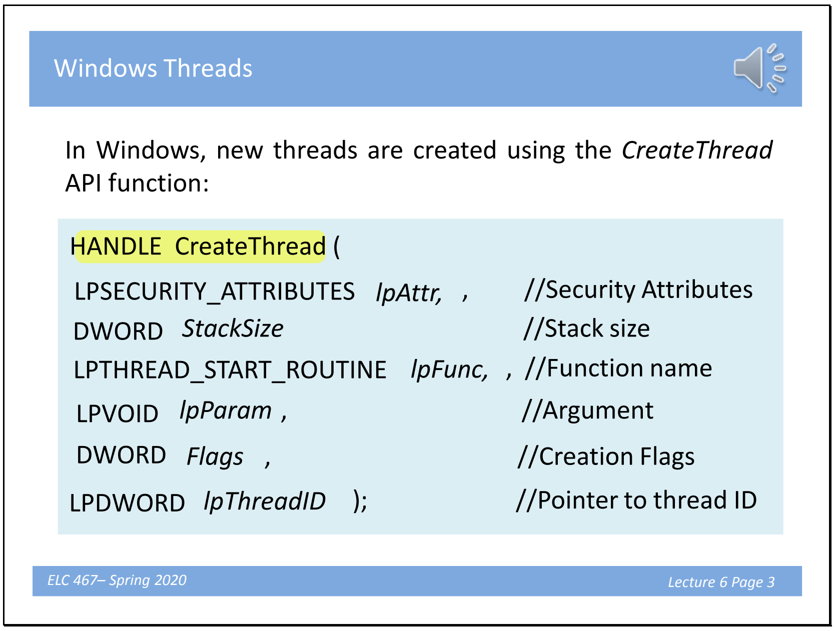


Windows Threads

In Windows, new threads are created using the *CreateThread* API function:

```
HANDLE  CreateThread (
 LPSECURITY_ATTRIBUTES  lpAttr,  ,        //Security Attributes
 DWORD  StackSize                         //Stack size
 LPTHREAD_START_ROUTINE  lpFunc,  ,       //Function name
 LPVOID  lpParam ,                        //Argument
 DWORD  Flags  ,                          //Creation Flags
LPDWORD  lpThreadID   );                  //Pointer to thread ID
```

ELC 467– Spring 2020   Lecture 6 Page 3

Slide 1

As an example of using the concepts studied so far, we consider the Windows operating system as a case study. We start by studying how to create a new thread in a Windows process. A new thread is created using the C function CreateThread, which has a meaningful name as all Windows functions. API stands for Applications Programming Interface, and API functions are the system functions that can be used by programs.

Slide 2

Function returns a handle, which is a pointer to the new created thread. System is treated as a collection of objects (processes, threads, windows, files .etc.), each pointed to by a handle. We refer to the thread later by this handle, e.g. to suspend thread, wait for the thread,… etc.

Slide 3

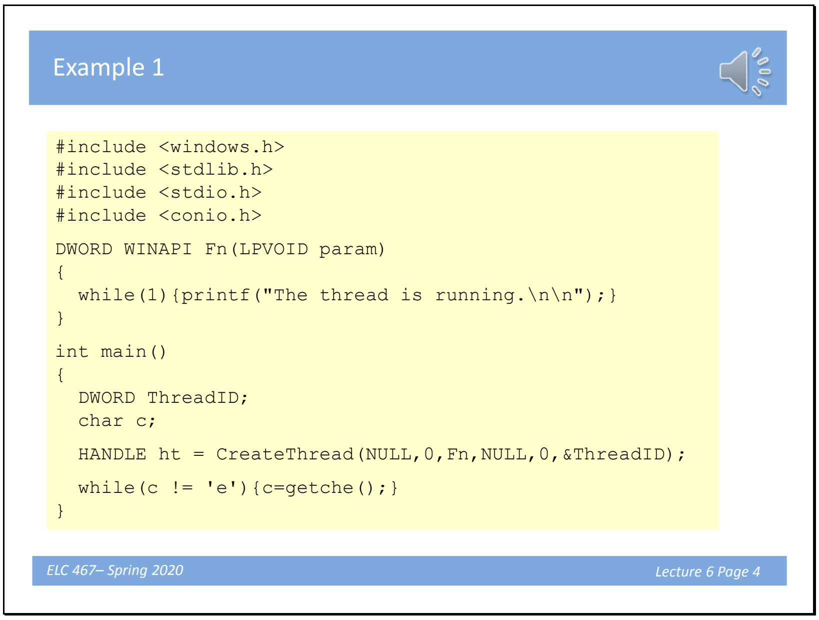
Function takes six arguments as follows. Security attributes defines how thread can be accessed by other processes. We will not make use of this in our examples, so we put the first argument to NULL. Second parameter is a double word specifying the size of stack reserved for the thread. We put it in our examples to zero, which gives it a default value.

Slide 4

Third argument specifies a function that will run in the new thread. If this function has any arguments, these are passed in the fourth argument of CreateThread.

Slide 5

A number of Creation flags can be put to control how the thread is created. For example, will it run immediately or will be suspended when created and started later. Possible flags can be found in the help pages of any compiler or online. Last argument passes a pointer to a variable that holds the unique id given to thread by the system.



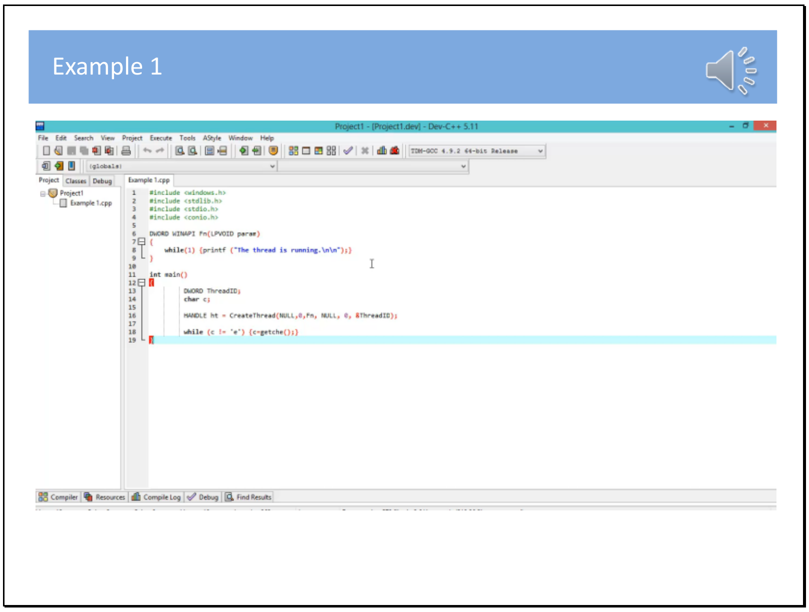
Example 1

```
#include <windows.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

DWORD WINAPI Fn(LPVOID param)
{
  while(1){printf("The thread is running.\n\n");}
}

int main()
{
  DWORD ThreadID;
  char c;

  HANDLE ht = CreateThread(NULL,0,Fn,NULL,0,&ThreadID);

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

ELC 467– Spring 2020 Lecture 6 Page 4

Slide 6

In this simple example, we create a thread that runs a function that just repeatedly displays “The thread is running”. Read the program carefully. Note that we must include windows.h header file to use API functions. Some compilers require the function run in a thread to be declared in a particular form with the word WINAPI. Main program and hence the created thread ends when e is pressed.



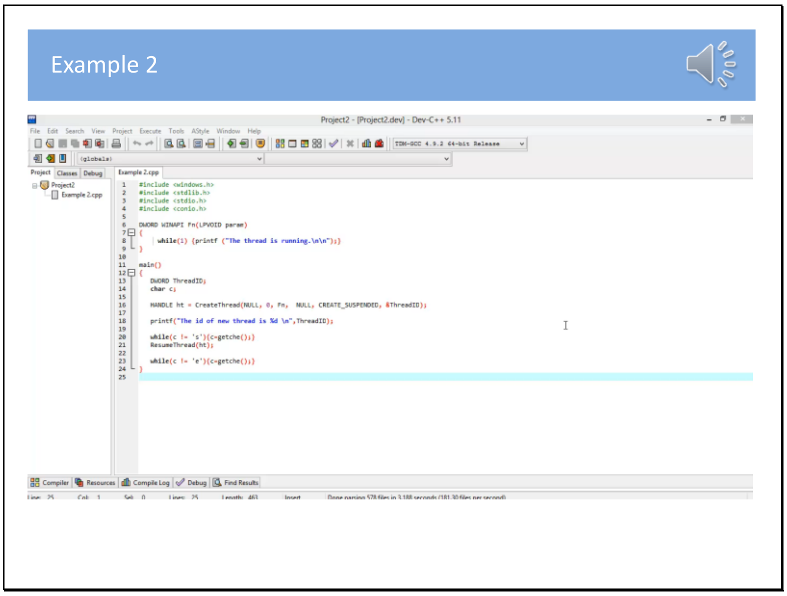
Example 1

Project1 - [Project1.dev] - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Release
(globals)
Project Classes Debug
Project1
Example 1.cpp
Compiler Resources Compile Log Debug Find Results

```
#include <windows.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

DWORD WINAPI Fn(LPVOID param)
{
    while(1) {printf ("The thread is running.\n\n");}
}

int main()
{
        DWORD ThreadID;
        char c;

        HANDLE ht = CreateThread(NULL,0,Fn, NULL, 0, &ThreadID);

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

Slide 7

The program is run here as a console application on dev-c++ compiler. Click on the video to see the program running.



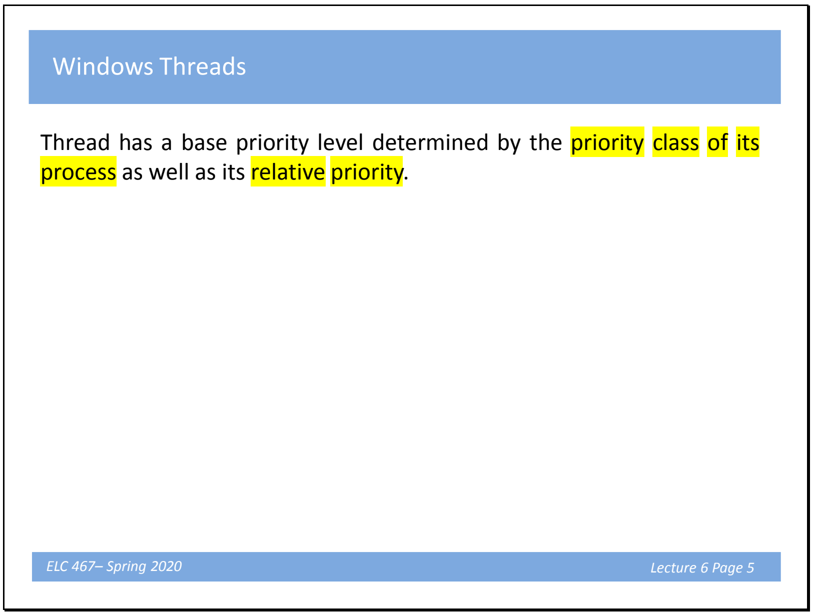
Example 2

Project2 - [Project2.dev] - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Release
(globals)
Project Classes Debug
Project2
Example 2.cpp
Compiler Resources Compile Log Debug Find Results

```
#include <windows.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

DWORD WINAPI Fn(LPVOID param)
{
    while(1) {printf ("The thread is running.\n\n");}
}

main()
{
   DWORD ThreadID;
   char c;

   HANDLE ht = CreateThread(NULL, 0, Fn,  NULL, CREATE_SUSPENDED, &ThreadID);

   printf("The id of new thread is %d \n",ThreadID);

   while(c != 's'){c=getche();}
   ResumeThread(ht);

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

Slide 8

This example is similar but the thread is created suspended. It is started when ‘s’ is pressed, using the function ResumeThread. Note that we refer to the thread by its handle, returned when created. The thread id is also displayed. Click on the video to see the program running.



Windows Threads

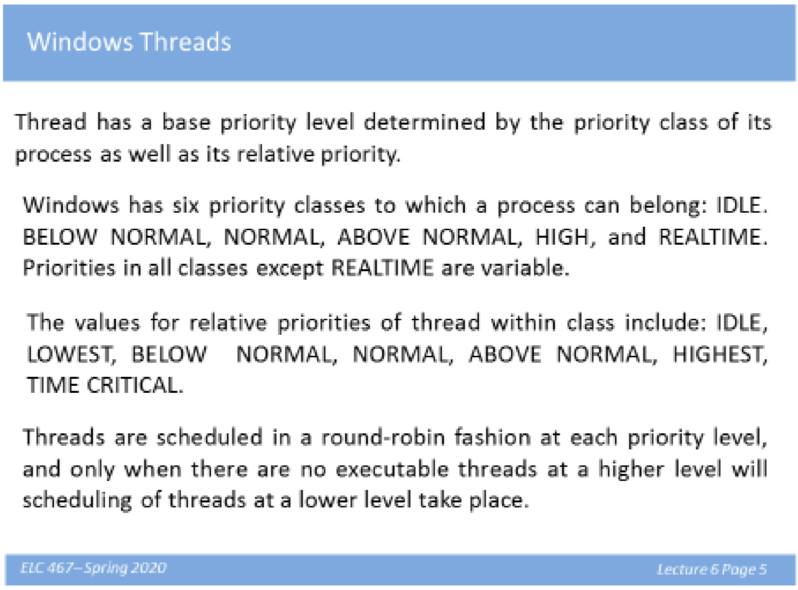
Thread has a base priority level determined by the priority class of its process as well as its relative priority.

ELC 467– Spring 2020

Lecture 6 Page 5

Slide 9

Each Windows thread has a priority level, which is an integer between 0 and 31. 31 is the highest priority level. Thread starts with a base priority level that may be changed later. The priority thread is determined relative to the priority class of its process.



Windows Threads

Thread has a base priority level determined by the priority class of its process as well as its relative priority.

Windows has six priority classes to which a process can belong: IDLE. BELOW NORMAL, NORMAL, ABOVE NORMAL, HIGH, and REALTIME. Priorities in all classes except REALTIME are variable.

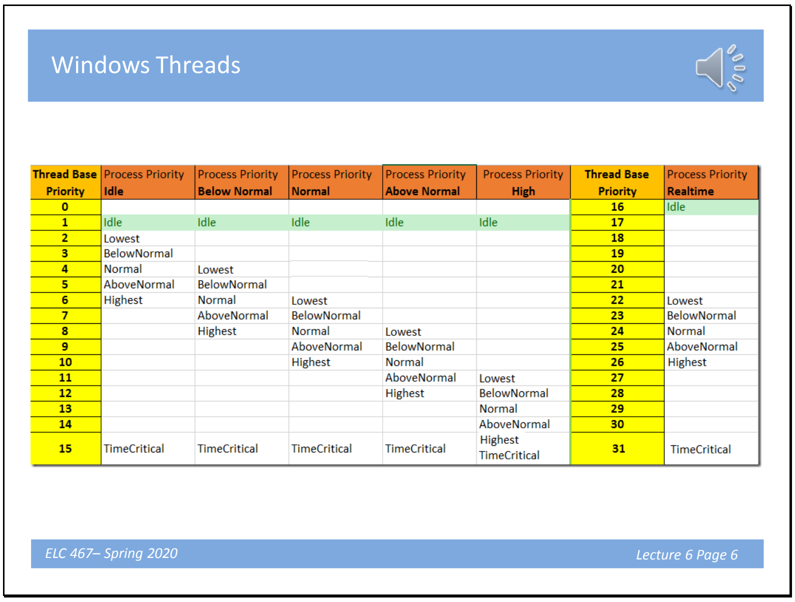
The values for relative priorities of thread within class include: IDLE, LOWEST, BELOW NORMAL, NORMAL, ABOVE NORMAL, HIGHEST, TIME CRITICAL.

Threads are scheduled in a round-robin fashion at each priority level, and only when there are no executable threads at a higher level will scheduling of threads at a lower level take place.

ELC 467–Spring 2020

Lecture 6 Page 5

Slides 10-11



Windows Threads

Thread Base Priority	Process Priority **Idle**	Process Priority **Below Normal**	Process Priority **Normal**	Process Priority **Above Normal**	Process Priority **High**	Thread Base Priority	Process Priority **Realtime**
0						16	Idle
1	Idle	Idle	Idle	Idle	Idle	17	
2	Lowest					18	
3	BelowNormal					19	
4	Normal	Lowest				20	
5	AboveNormal	BelowNormal				21	
6	Highest	Normal	Lowest			22	Lowest
7		AboveNormal	BelowNormal			23	BelowNormal
8		Highest	Normal	Lowest		24	Normal
9			AboveNormal	BelowNormal		25	AboveNormal
10			Highest	Normal		26	Highest
11				AboveNormal	Lowest	27	
12				Highest	BelowNormal	28	
13					Normal	29	
14					AboveNormal	30	
15	TimeCritical	TimeCritical	TimeCritical	TimeCritical	Highest TimeCritical	31	TimeCritical

ELC 467– Spring 2020 *Lecture 6 Page 6*

Slide 12

This table shows the priority level of a thread as function of process priority class and thread relative priority.