CS415 Module 3 Part C - Thread Safety and Thread Level Storage

Athens State University

Contents

1 Windows and Threads

Processes in Windows

Processes and services provided by Windows are relatively simple and general purpose

- Implemented as objects
- Created as new process or copy of existing
- Process may contain one or more threads
- Both processes and thread objects have built-in synchronization capabilities

Windows process design is driven by the need to provide support for a variety of OS environments. Processes supported by different OS environments differ in a number of ways, including the following:

- How processes are named
- Whether threads are provided within processes
- How processes are represented
- How process resources are protected
- What mechanisms are used for interprocess communication and synchronization
- How processes are related to each other

Accordingly, the native process structures and services provided by the Windows Kernel are relatively simple and general purpose, allowing each OS subsystem to emulate a particular process structure and functionality. Important characteristics of Windows processes are the following:

- Windows processes are implemented as objects.
- A process can be created as new process, or as a copy of an existing process.
- An executable process may contain one or more threads.
- Both process and thread objects have built-in synchronization capabilities.

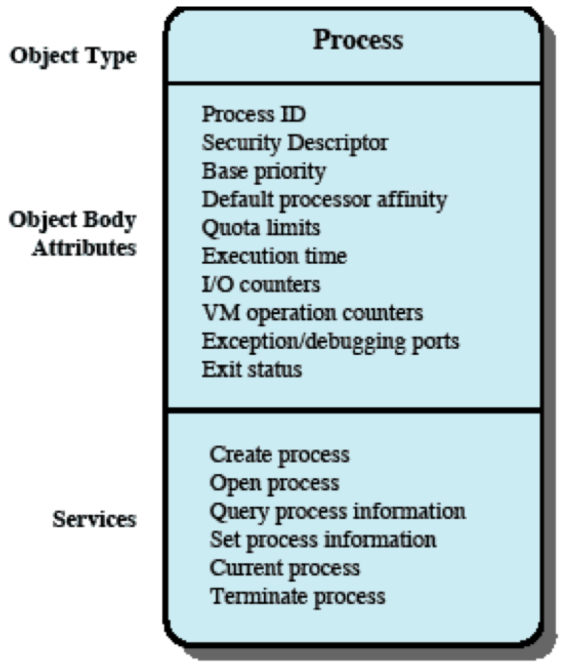
Process and Thread Objects in Windows

Windows makes use of two types of process-related objects

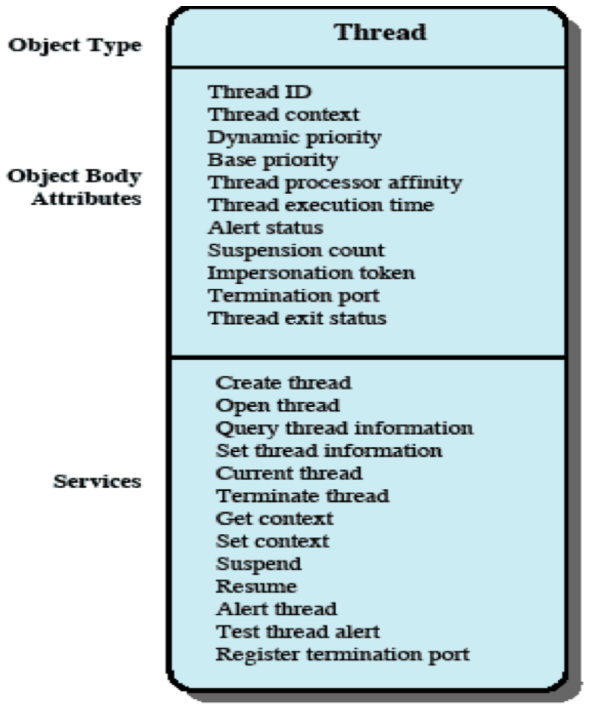
- Processes
  - An entity corresponding to a user job or application that owns resources
- Threads
  - A dispatchable unit of work that executes sequentially and is interruptible

The object-oriented structure of Windows facilitates the development of a general- purpose process facility. Windows makes use of two types of process-related objects: processes and threads. A process is an entity corresponding to a user job or application that owns resources, such as memory and open files. A thread is a dispatchable unit of work that executes sequentially and is interruptible, so that the processor can turn to another thread.

Process and Thread Objects in Windows



(a) Process object

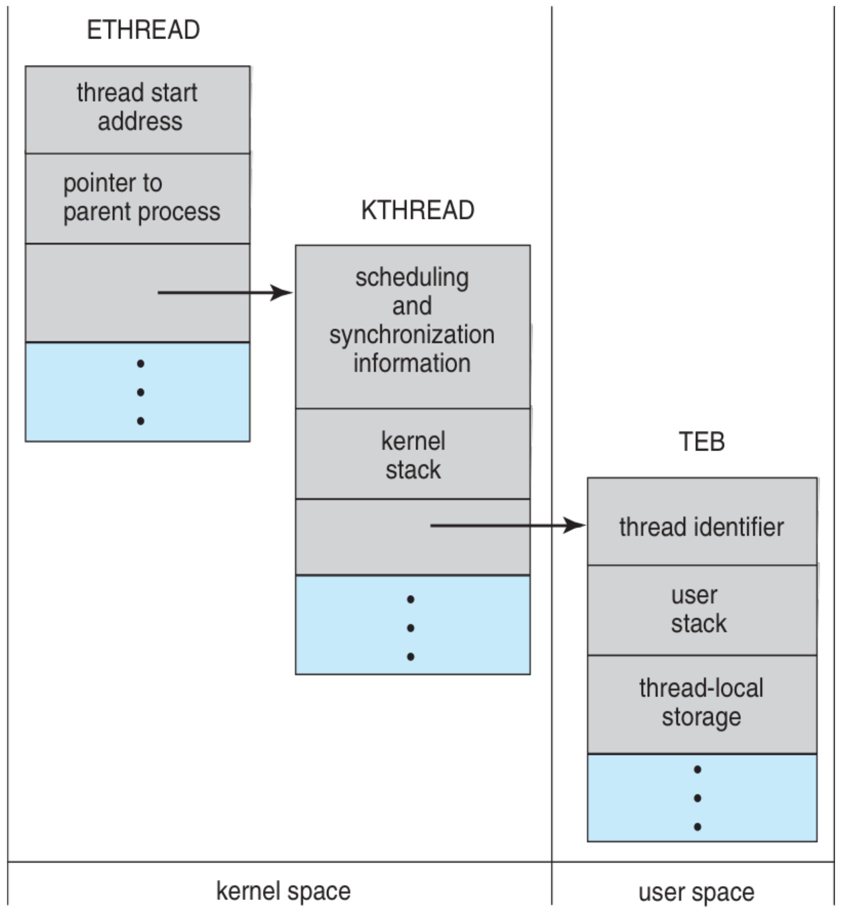


(b) Thread object

Windows process must contain at least one thread to execute. That thread may then create other threads. Note that some of the attributes of a thread resemble those of a process. In those cases, the thread attribute value is derived from the process attribute value. For example, the thread processor affinity is the set of processors in a multiprocessor system that may execute this thread; this set is equal to or a subset of the process processor affinity.

Note that one of the attributes of a thread object is context, which contains the values of the processor registers when the thread last ran. This information enables threads to be suspended and resumed. Furthermore, it is possible to alter the behavior of a thread by altering its context while it is suspended.

Process and Thread Data Structures



Using Win32 Process and Thread API

```
#include <windows.h>
#include <stdio.h>
DWORD Sum; /* data is shared by the thread(s) */

/* the thread runs in this separate function */
DWORD WINAPI Summation(LPVOID Param)
{
  DWORD Upper = *(DWORD*)Param;
  for (DWORD i = 0; i <= Upper; i++)
    Sum += i;
  return 0;
}

int main(int argc, char *argv[])
{
  DWORD ThreadId;
  HANDLE ThreadHandle;
  int Param;

  if (argc != 2) {
    fprintf(stderr,"An integer parameter is required\n");
    return -1;
  }
  Param = atoi(argv[1]);
  if (Param < 0) {
    fprintf(stderr,"An integer >= 0 is required\n");
    return -1;
  }
```

Using Win32 Process and Thread API

```
    /* create the thread */
    ThreadHandle = CreateThread(
      NULL, /* default security attributes */
      0, /* default stack size */
      Summation, /* thread function */
      &Param, /* parameter to thread function */
      0, /* default creation flags */
      &ThreadId); /* returns the thread identifier */

    if (ThreadHandle != NULL) {
       /* now wait for the thread to finish */
      WaitForSingleObject(ThreadHandle,INFINITE);

      /* close the thread handle */
      CloseHandle(ThreadHandle);

      printf("sum = %d\n",Sum);
    }
  }
```

2 Java Threads

Threads in Java

- A combined model threading library is provided as part of the Java Class Library
- User threads are managed by the JVM, the JVM implements threads using the native OS
- A Java app implements threads in two steps:
  - Extending the `Thread` class
  - Implementing the `Runnable` interface

The `Runnable` interface in Java takes the form:

```java
public interface Runnable {
  public abstract void run();
}
```

```java
class Sum {
   private int sum;
   public int getSum() { return sum;}
   public int setSum() { this.sum = sum;}
}
class Summation implements Runnable {
  private int upper;
  private Sum sumValue;
  public Summation(int upper, Sum mySumValue) {
    this.upper = upper;
    this.sumValue = mySumValue;
  }
  public void run() {
   int sum = 0;
   for (int i - 0; i<= upper; i++) sum += i;
   sumValue.setSum(sum);
  }
}
```

```
public class SumDriver {
  public static void main(String[] args){
    if (args.length > 0) {
      Sum Object = new Sum();
      int upper = Integer.parseInt(args[0]) ;
      Thread thrd = new Thread(new Summation(upper, sumObject));
      thrd.start();
      try {
        thrd.join();
        System.out.println("Sum of " + upper + " is " + sumObject.getSum());
      } catch (InterruptedException ie) {}
    }
  }
}
```