EII
Escuela de Ingeniería Informática de Oviedo

C. de Valdes Salas, 11
Tel. 98 510 27 96
eii@uniovi.es
33007 Oviedo

- Process Management Subsystem in Windows Operating System -

PROFESORADO:
Miguel Riesgo Albizu - albizu@uniovi.es

AUTOR:
Guillermo Facundo Colunga - uo236856@uniovi.es

Table of contents

Introduction

This report was proposed by the lecturers of the subject and developed by students. From the first point was clear that this was intended to be a technical document, so with that premise we've done a technical report describing and explaining the process management subsystem in Windows Operating System.

In order to develop such a report we, the students, distributed the work in such a way that each student develop its own points and review the others so every point is at least reviewed twice, one by the student who developed and another by the complementary student.

Finally we include an small review of the work performed by each student during the development of the essay:

Guillermo Facundo Colunga. Given that the other member of the team did not show any kind of interest in the development of the essay he is direct responsable of the whole report. *Personal Comment: I would like to have another student in the group so I can get feedback and another point of view. The essay its not as good as it could be.*

Basics of Operating Systems

We will define and Operating System (OS), as the piece of software that manages the computer hardware and software resources. And provides common services for computer programs.

In modern OSs, like Windows, there is a need to make the computer system efficient because we will run many different tasks at the same time and that could affect performance. That is the reason why operative systems implement a subsystem that manages how processes are created, stored, executed and terminated.

There's no doubt that there is a wide variety of Operating Systems in the market, but we have focussed the essay on windows OS due to the simple fact that is the most used operative system for personal and profesional computers (without taking into account servers and smart-devices).

Windows OS

We've already saw the basics of operating systems and why we've choose windows to perform the essay. Now lets try to give non expert users an approach to Windows Operating System.

Windows borns in 1985 with its firs release Windows 1.0. As its name indicates its and operating system and belongs to the group of graphical interface OSs, that means that Windows provides a "user friendly" graphical interface so the users can interact with the OS by means of mouse and graphics.

Besides Windows had many versions from its first release until now it can be categorize under the following architectures versions: **Windows 1.x**, **Windows 3.x**, **Windows 9x** and **Windows NT** that includes from XP to Windows 10. In this report, we will choose the Windows NT architecture because is the one that more releases cover and the current one.

Process and Thread objects

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.**

Processes

The official definition of a process if *a set of activities that interact to achieve a result*[1]. But we will use the following extended definition: A process is an entity that has an owner, owns resources, such memory and interacts with the OS to achieve a goal.

In Windows process is represented by an object whose general structure is shown in the following figure.

Object Type	Process
Object Body Attributes	Process ID Security descriptor Base priority Default processor affinity Quota limits Execution time I/O counters VM operation counters Exception / debugging ports Exit status
Services	Create process Open process Query process information Set process information Current process Terminate process

Each process is defined by a number of attributes and encapsulates a number of actions, or services, that it may perform. A process will perform a service upon receipt of the appropriate message; the only way to invoke such a service is by means of messages to a process object that provides that service.

Threads

A thread is a dispatch-able unit of work that executes sequentially and is interruptible, so that the processor can turn to another thread. As a process in Windows a thread is object-implemented, the following figure illustrates how it is implemented:

[1] Processes definition from Wikipedia Process entry.

Object Type	Thread
Object Body Attributes	Thread ID Thread context Dynamic priority Base priority Thread processor affinity Thread execution time Alert status Suspension count Impersonation token Termination port Thread exit status
Services	Create thread Open thread Query thread information Set thread information Current thread Terminate thread Get context Set context Suspend Resume Alert thread Test thread alert Register termination port

Process and Thread Creation

Each process and thread in Windows is represented by an object whose structure has already been shown. So, when Windows creates a new process, it uses the object class, or type, defined for the Windows process as a template to generate a new object instance. At the time of creation, attribute values are assigned. Table 3.1 gives a brief definition of each of the object attributes for a process object.

A Windows process must contain at least one thread to execute. That thread may then create other threads. In a multiprocessor system, multiple threads from the same process may execute in parallel. Figure 4.13b depicts the object structure for a thread object, and Table 3.2 defines the thread object attributes. 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. This information enables threads to be suspended and resumed. Furthermore, it is possible to alter the behavior of a thread by altering its context when it is suspended.

Table 3.1 Windows Process Object Attributes

Process ID	A unique value that identifies the process to the operating system.
Security Descriptor	Describes who created an object, who can gain access to or use the object, and who is denied access to the object.
Base priority	A baseline execution priority for the process's threads.
Default processor affinity	The default set of processors on which the process's threads can run.
Quota limits	The maximum amount of paged and nonpaged system memory, paging file space, and processor time a user's processes can use.
Execution time	The total amount of time all threads in the process have executed.
I/O counters	Variables that record the number and type of I/O operations that the process's threads have performed.
VM operation counters	Variables that record the number and types of virtual memory operations that the process's threads have performed.
Exception/debugging ports	Interprocess communication channels to which the process manager sends a message when one of the process's threads causes an exception.
Exit status	The reason for a process's termination.

Table 3.2 Windows Thread Object Attributes

Thread ID	A unique value that identifies a thread when it calls a server.
Thread context	The set of register values and other volatile data that defines the execution state of a thread.
Dynamic priority	The thread's execution priority at any given moment.
Base priority	The lower limit of the thread's dynamic priority.
Thread processor affinity	The set of processors on which the thread can run, which is a subset or all of the processor affinity of the thread's process.
Thread execution time	The cumulative amount of time a thread has executed in user mode and in kernel mode.
Alert status	A flag that indicates whether the thread should execute an asynchronous procedure call.
Suspension count	The number of times the thread's execution has been suspended without being resumed.
Impersonation token	A temporary access token allowing a thread to perform operations on behalf of another process (used by subsystems).
Termination port	An interprocess communication channel to which the process manager sends a message when the thread terminates (used by subsystems).
Thread exit status	The reason for a thread's termination.

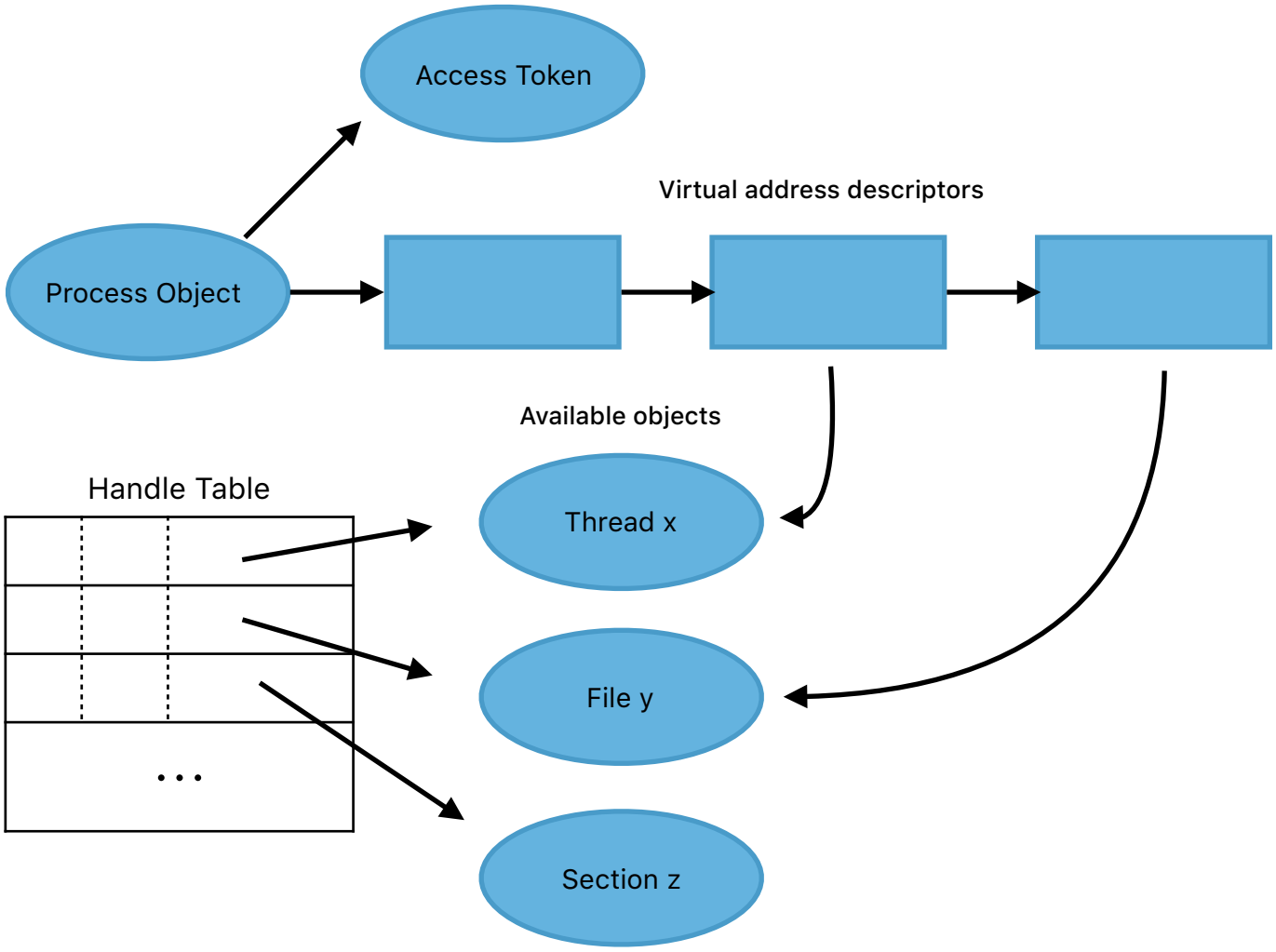
Process and Thread Execution

The following figure illustrates the way in which a process relates to the resources it controls or uses. Each process is assigned a security access token, called the primary token of the process. When a user first logs on, Windows creates an access token that includes a security ID for the user. Every process that is created by or runs on behalf of this user has a copy of this access token.

Windows uses the token to validate the user's ability to access secured objects or to perform restricted functions on the system and on secured objects. The access token controls whether the process can change its own attributes. In this case, the process does not have a handle opened to its access token. If the process attempts to open such a handle, the security system determines whether this is permitted and therefore whether the process may change its own attributes.

Also related to the process is a series of blocks that define the virtual address space currently assigned to this process. The process cannot directly modify these structures but must rely on the virtual memory manager, which provides a memory-allocation service for the process.

Finally, the process includes an object table, with handles to other objects known to this process. One handle exists for each thread contained in this object. next figure shows a single thread. In addition, the process has access to a file object and to a section object that defines a section of shared memory.



Example. In Windows the fundamental process management function is `CreateProcess`, which creates a process with a single thread. It is necessary to specify the name of an executable program file as part of the `CreateProcess` call. For UNIX users, `CreateProcess` is, however, similar to the common UNIX sequence of successive calls to `fork` and `execl` (or one of five other `exec` functions).

Multithreading

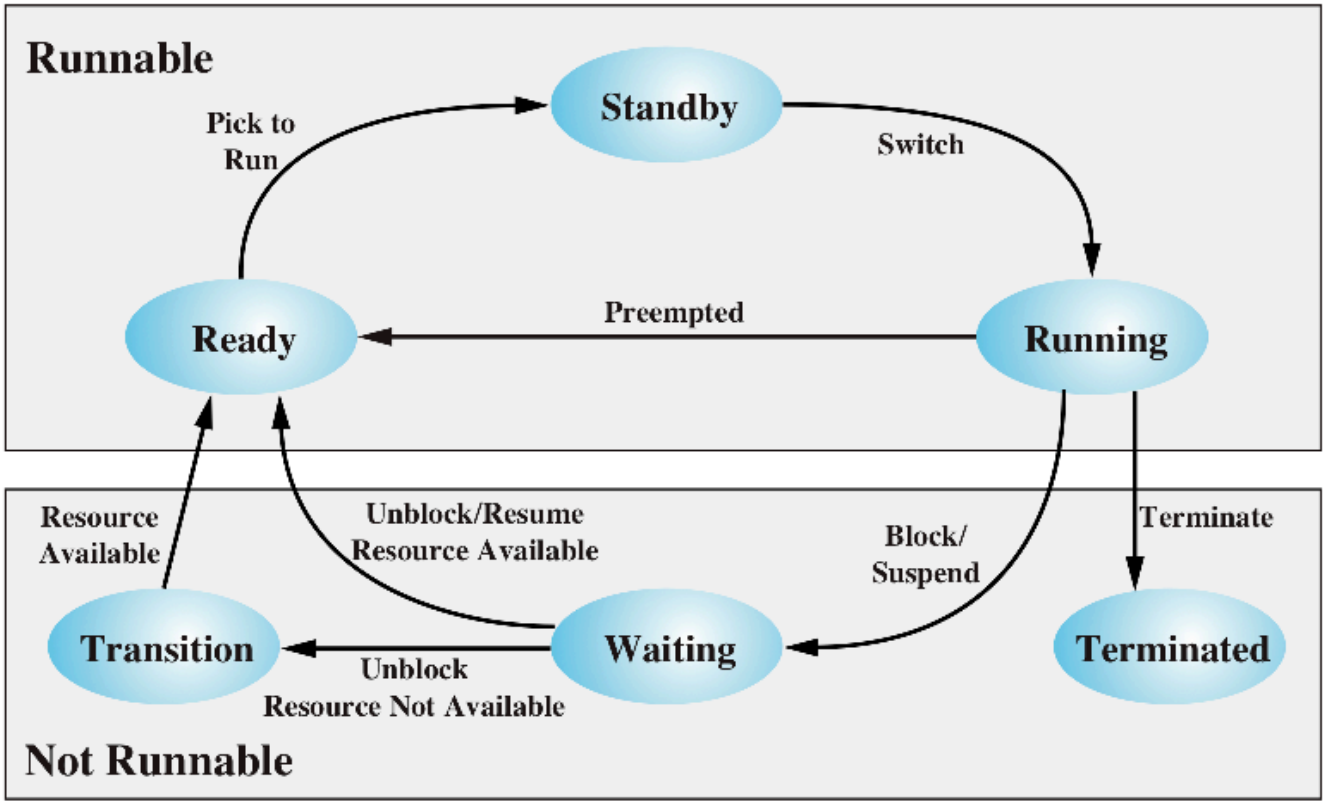
Taking into account that nowadays multithreading is consider indispensable for an OS we couldn't continue without giving some general ideas about the multithreading in Windows OS.

Windows supports concurrency among processes because threads in different processes may execute concurrently. Moreover, multiple threads within the same process may be allocated to separate processors and execute simultaneously. A multithreaded process achieves concurrency without the overhead of using multiple processes. Threads within the same process can exchange information through their common address space and have access to the shared resources of the process. Threads in different processes can exchange information through shared memory that has been set up between the two processes.

An object-oriented multithreaded process is an efficient means of implementing a server application. For example, one server process can service a number of clients. Each client request triggers the creation of a new thread within the server.

Thread States

An existing Windows thread is in one of six states: Ready, StandBy, Running, Waiting, Transitioning and Terminated. The stated flow is displayed in the following chart and the states are defined after the figure.



Ready. May be scheduled for execution. The microkernel dispatcher keeps track of all ready threads and schedules them in priority order.

Standby. A standby thread has been selected to run next on a particular processor. The thread waits in this state until that processor is made available. If the standby thread's priority is high enough, the running thread on that processor may be preempted in favor of the standby thread. Otherwise, the standby thread waits until the running thread blocks or exhausts its time slice.

Running. Once the microkernel performs a thread or process switch, the standby thread enters the running state and begins execution and continues execution until it is preempted, exhausts its time slice, blocks, or terminates. In the first two cases, it goes back to the ready state.

Waiting. A thread enters the waiting state when (1) it is blocked on an event (e.g., I/O), (2) it voluntarily waits for synchronization purposes, or (3) an environment subsystem directs the thread to suspend itself. When the waiting condition is satisfied, the thread moves to the Ready state if all of its resources are available.

Transitioning. A thread enters this state after waiting if it is ready to run but the resources are not available. For example, the thread's stack may be paged out of memory. When the resources are available, the thread goes to the Ready state.

Terminated. A thread can be terminated by itself, by another thread, or when its parent process terminates. Once housekeeping chores are completed, the thread is removed from the system, or it may be retained by the executive1 for future reinitialization.

Process and Thread termination

As a generalization we can state that a runnable terminates its execution when passes from the state **running** to the state **terminated**, in this chapter we will see the different ways this can be achieved.

Termination after succeed

Imagine you have a program that computes the addition of A and B, both integer numbers to keep it simple. So you run the program with 4 and 1 and the program computes 5. In that case the OS will make a call to `TerminateProcess` function that will change the state of the thread that contains the process from Running to Terminated, after that will free the resources of the process and will let the OS to continue with its tasks.

In that particular case the TerminateProcess function succeed will return a cero value, else a non-cero value and the function `GetLastError` should be called to get the error that make the function crash. That will be shown in the next point.

Termination after error in process/thread execution

Now imagine the following program implemented in your favorite language:

```
// Index to look for.
counter = 1

// Array of numbers with 4 elements
numbers = int[4]

numbers[0] = 0
numbers[1] = 1
numbers[2] = 2
numbers[3] = 3

// Trying to print the 4th element in the array of numbers
print(numbers[4])
```

As you can see the program described before will try to access an array memory that is not allocated so will abruptly end the execution of the program and send a code to the OS to inform that the process ended with an “Exception”. In this case something like segment violation or forbidden memory access will be send to the OS with the appropriate code. After that the OS is able to continue with other tasks.

Conclusions

Modern operating systems faces the challenging problem of given support to an infinite variety of programs and environments so need to implement mechanisms that, without knowing anything about the environment nor the program, are able to execute the processes as a single thread or multithread.

Regarding to Windows OS, the architecture of NT has served them so well these past years but that doesn't mean it can be used or adapted forever, they need to take a big steep and change the core of their OS in order to provide a more flexible environment with higher performance.

Glossary

Object-oriented	Relates to an environment where the real world ideas are represented as objects stored in memory.
Hardware	Piece of physical equipment that can run a piece of software.
Software	Piece of code that can be executed by a piece of hardware.
Kernel	Central part of a the software that usually is used as a foundation for other calls from the same or other software.
Microkernel	A kernel that by its size its categorize as micro.
Handle Table	A data structure that emulates a real world table, this table holds the information about the process, thread and resources.

References

Wikipedia, Inc. Microsoft Windows BNF: cb14400232k. [20/03/2018]

Wikipedia, Inc. Núcleo (informática). Translated by Guillermo Facundo Colunga. [20/03/2018]

The Windows Operating System, William Stallings. [20/03/2018]

Operating Systems: Internals and Design Principles, Fifth Edition Prentice Hall, 2005, ISBN 0-13-147954-7.