Process and Thread Management (Microsoft Vista and Microsoft Windows 10)

Farah Sardouk, Msc Student in ECE
Student Number: 153101013
Kemerburgaz University

What is a Process?

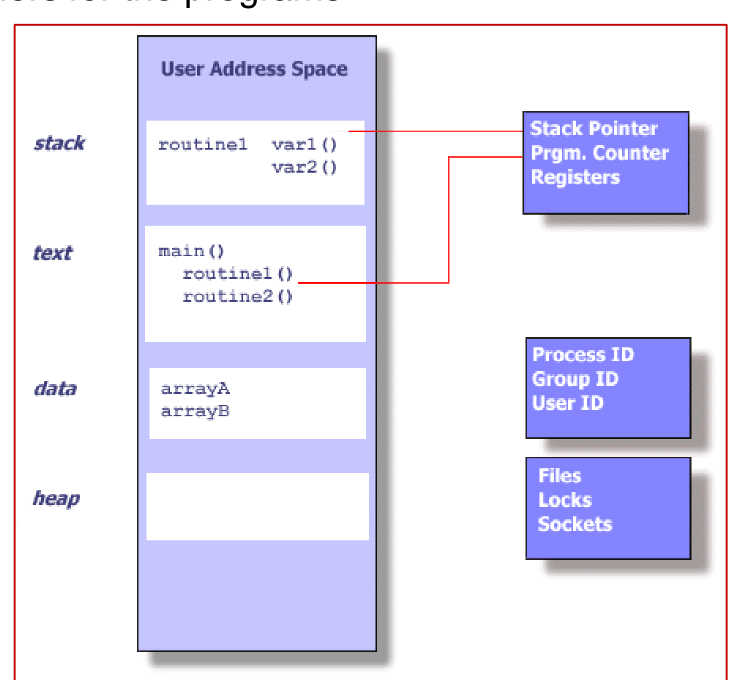
A process is an instance of the computer program that is created **by OS** to run your **Main** program.

In Windows Vista processers are containers for the programs.

It Contains all the information to be handled by the OS:

- Process ID, process group ID, user ID, and group ID.
- Environment
- ***Scheduling Properties (priority, etc.)***.
- Program instructions
- ***Registers,*** ***Stacks***
- File descriptors
- ***Signal actions***
- Shared libraries
- Inter-process communication tools

  shared memory , message queues,
  semaphores , pipes,



What's New About Processes? (MS 10)

- Each *process* provides the resources needed to execute a program.

- A process has:

a virtual address space, executable code, open handles to system objects, a security context, a unique process identifier, environment variables, a priority class, minimum and maximum working set sizes, and at least one thread of execution.

- Each process is started with a single thread, often called the *primary thread*, but can create additional threads from any of its threads.

What is a Thread?

A thread is created **by OS** to run a stream of instructions.

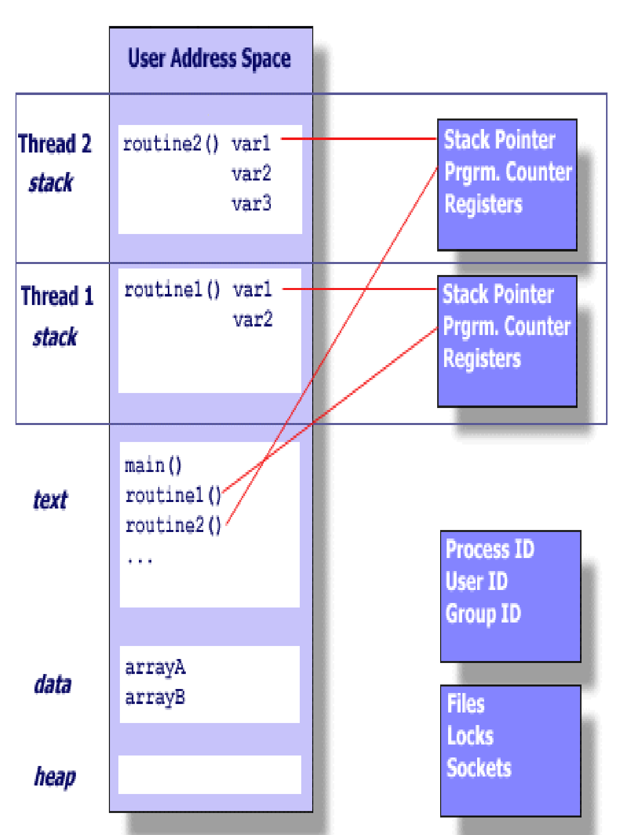
- Threads contain smaller set of information:
  - *Scheduling Properties*
  - *Registers*
  - *Stacks*
  - *Signal actions*
- Light-Weight

Most overhead accomplished through maintaining its process.

- Exists within a process and uses the process resources

- It has its own independent flow of control *(as long as its parent process exists and the OS supports it)*

- Duplicates only essential resources for independent scheduling.



What’s New about Threads in Microsoft?

- A *thread* is the entity within a process that can be scheduled for execution.

- . All threads of a process share its virtual address space and system resources.
 In addition, each thread maintains exception handlers, a scheduling priority, thread local storage, a unique thread identifier, and a set of structures the system will use to save the thread context until it is scheduled.

- The *thread context* includes the thread's set of machine registers, the kernel stack, a thread environment block, and a user stack in the address space of the thread's process. Threads can also have their own security context, which can be used for impersonating clients.

- **Microsoft Windows supports *preemptive multitasking*, which creates the effect of simultaneous execution of multiple threads from multiple processes. On a multiprocessor computer, the system can simultaneously execute as many threads as there are processors on the computer.**

Jobs: A *job object* allows groups of processes to be managed as a unit.

Job objects are namable, securable, sharable objects that control attributes of the processes associated with them.

Operations performed on the job object affect all processes associated with the job object.

All process threads created in the process will also be in the job.

Problems: **one process can be in one job**, there will be a conflict if many jobs attempt to manage the same process.

Fibers

A *fiber* is a unit of execution that must be manually scheduled by the application. Fibers run in the context of the threads that schedule them.

Fibers are created by **allocating a stack** and **a user-mode fiber data** structure for storing registers. Fiber data can also be created **independently** of threads. Fibers **will not run until another running fiber** in thread **make explicitly call** SwitchToFiber.

Pros:
It is easier and takes fewer time to switch between fibers than switching between threads.

Cons : It needs **a lot of synchronization** to make sure fibers do not interface with each other.

Solution: create **threads as much as** processors to run them, and affinities the threads to run only on a distinct set of available **processors.**

What's New About Processes and Threads?

Windows 7 and Windows Server 2008 R2 include the following new programming elements for processes and threads.

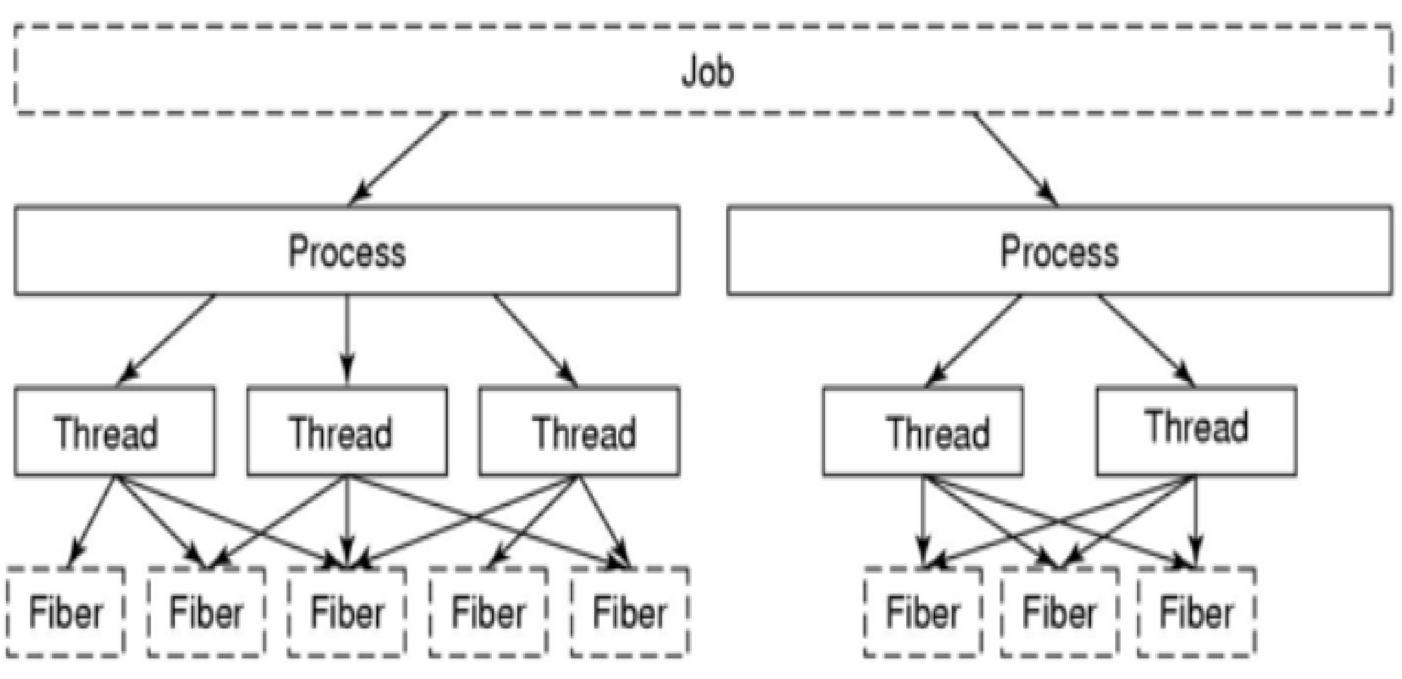
New Capabilities

The 64-bit versions of Windows 7 and Windows Server 2008 R2 support more than 64 logical processors on a single computer. User-mode scheduling (UMS) is a lightweight mechanism that applications can use to schedule their own threads.

The 64-bit versions of Windows 7 and Windows Server 2008 R2 and later versions of Windows support more than 64 logical processors on a single computer. This functionality is not available on 32-bit versions of Windows.

Systems with more than one physical processor or systems with physical processors that have multiple cores provide the operating system with multiple logical processors. A *logical processor* is one logical computing engine from the perspective of the operating system, application or driver. A *core* is one processor unit, which can consist of one or more logical processors. A *physical processor* can consist of one or more cores. A physical processor is the same as a processor package, a socket, or a CPU.

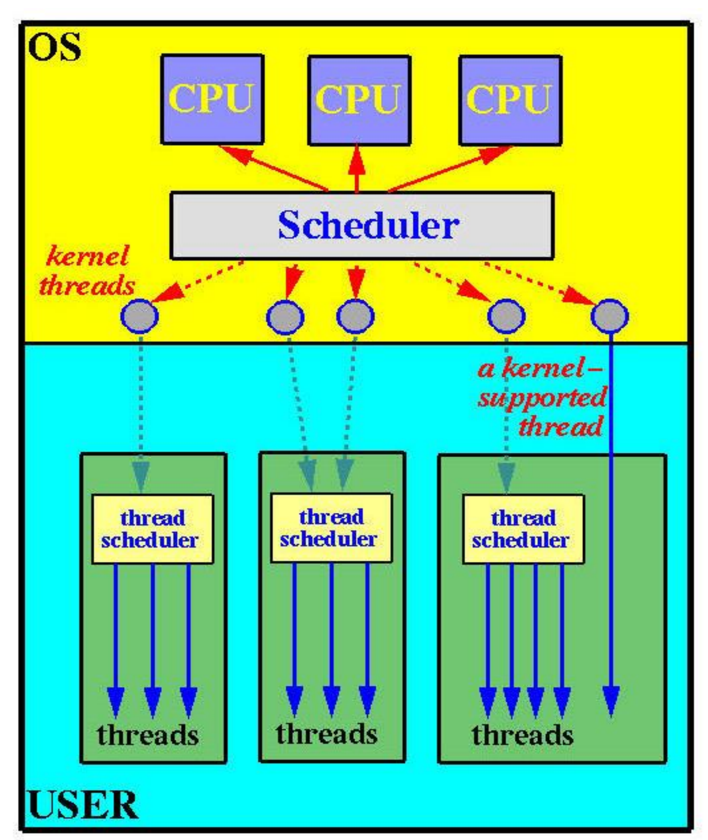
Jobs And Fibers



Summary

Name	Description
Job	Collection of processes that share quotas and limits
Process	Container for holding resources
Thread	Entity scheduled by the kernel
Fiber	Lightweight thread managed entirely in user space

User And Kernel Thread



User and Kernel Thread

1- User Threads:

- User threads are supported at the user level. The kernel is not aware of user threads.
- A library provides all support for thread creation, termination, joining, and scheduling.
- There is no kernel intervention, and, hence, user threads are usually more efficient.
- Unfortunately, since the kernel only recognizes the containing process (of the threads), if one thread is blocked, every other threads of the same process are also blocked because the containing process is blocked.

User and Kernel Threads

2- Kernel threads

- Kernel threads are directly supported by the kernel. The kernel does thread creation, termination, joining, and scheduling in kernel space.
- Kernel threads are usually slower than the user threads.
- However, blocking one thread will not cause other threads of the same process to block. The kernel simply runs other threads.
- In a multiprocessor environment, the kernel can schedule threads on different processors.

Process creation

- **Automatically by the system:**

  - System initialization
  - Application and background process

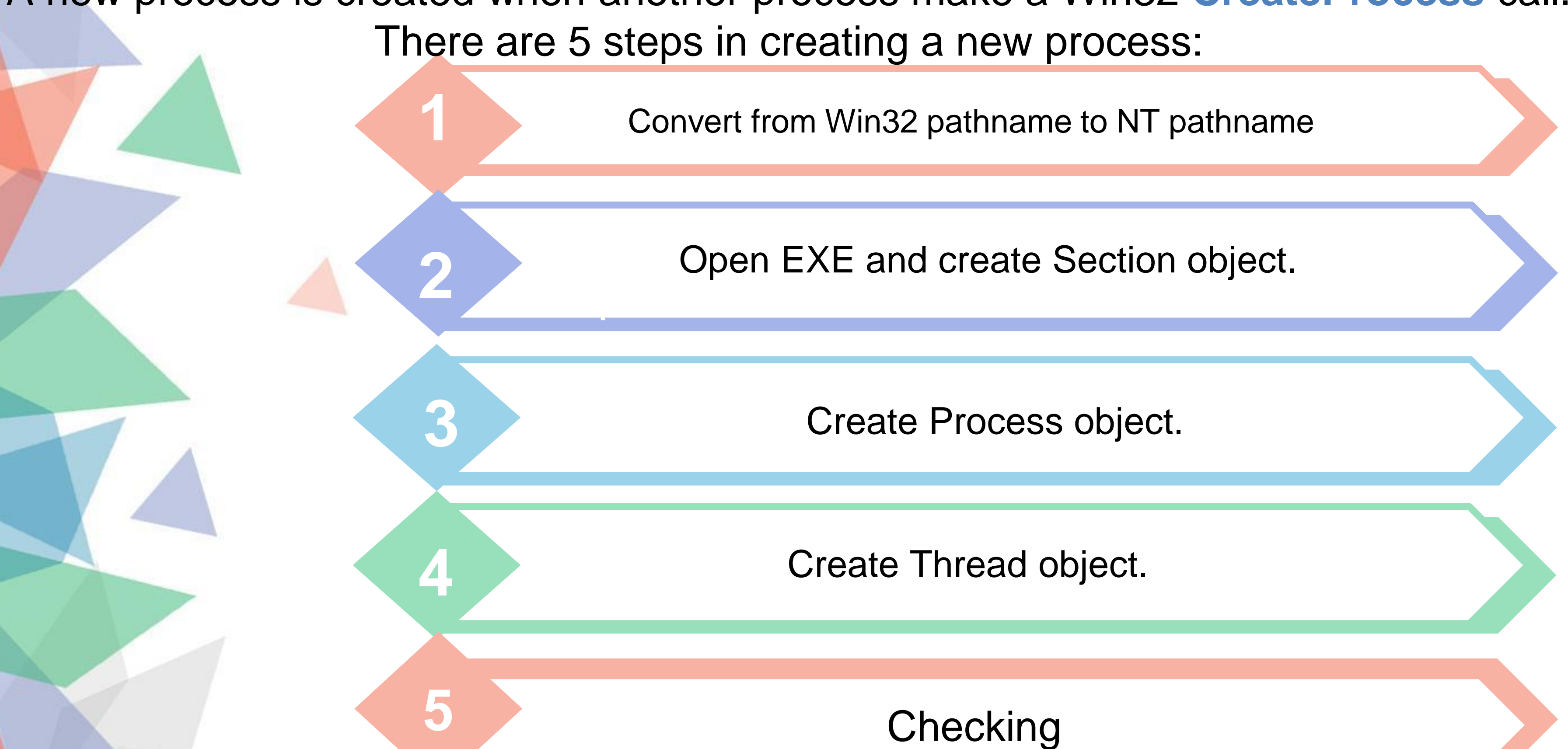
- **By another process**

  - A system call for process creation

  - Context: A process needs some computation
    - An user request (by command or interact with an icon)

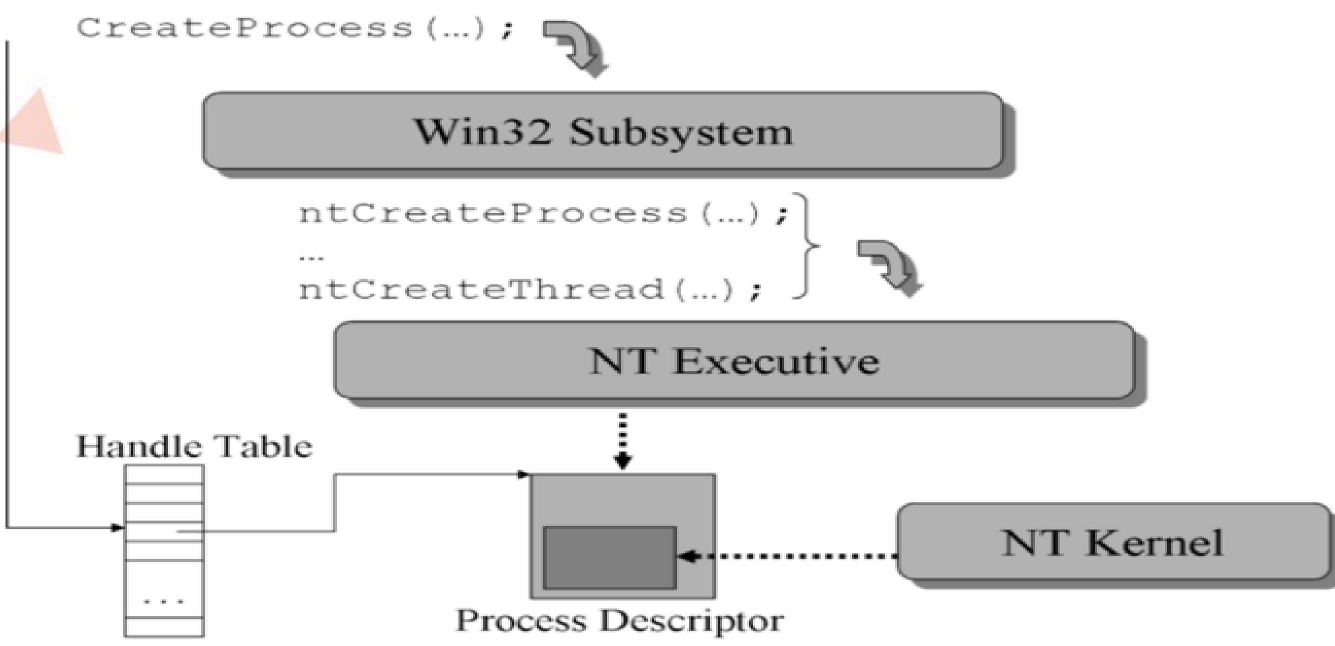
Implementation of Processes and Threads

A new process is created when another process make a Win32 **CreateProcess** call. There are 5 steps in creating a new process:



Process Creation(Continue)

New Child Processes are created by another process (the Parent Process) at system boot time.



What's new about Process Creation

The **CreateProcess** function creates a new process, which runs independently of the creating process. However, for simplicity, the relationship is referred to as a parent-child relationship.

If **CreateProcess** succeeds, it returns a **PROCESS_INFORMATION** structure containing handles and identifiers for the new process and its primary thread. The thread and process handles are created with full access rights, although access can be restricted if you specify security descriptors. When you no longer need these handles, close them by using the **CloseHandle** function.

You can also create a process using the **CreateProcessAsUser** or **CreateProcessWithLogonW** function. This allows you to specify the security context of the user account in which the process will execute.

Scheduling

➢Windows schedules threads, not processes.

➢The Scheduler is called when:

❖An I/O operation completes
❖specified waiting time expires

➢Scheduling is preemptive, priority-based, and round-robin at the highest priority

➢Processes/Threads can specify affinity mask to run only on certain processors:
SetProcessAffinityMask(),
➢SetThreadAffinityMask(), …

Scheduling in Microsoft Windows

Windows Server 2008, Windows Vista, Windows Server 2003 and Windows XP: Processor groups are not supported.

When the system starts, the operating system creates processor groups and assigns logical processors to the groups. If the system is capable of hot-adding processors, the operating system allows space in groups for processors that might arrive while the system is running.

The operating system minimizes the number of groups in a system. For example, a system with 128 logical processors would have two processor groups with 64 processors in each group, not four groups with 32 logical processors in each group.

Scheduling Algorithm

Threads are scheduled to run based on their scheduling priority.
Each thread is assigned a specific scheduling priority.
The priority levels range from **zero** (lowest priority) to **31** (highest priority), correspondingly associated with **32 queues**.
Base priority (of a thread) = F(priority class, priority level) = constant.
Dynamic priority = Base priority + Boost Amount, is used to determine which thread to execute.

		Win32 process class priorities					
		Real-time	High	Above Normal	Normal	Below Normal	Idle
Win32 thread priorities	Time critical	31	15	15	15	15	15
	Highest	26	15	12	10	8	6
	Above normal	25	14	11	9	7	5
	Normal	24	13	10	8	6	4
	Below normal	23	12	9	7	5	3
	Lowest	22	11	8	6	4	2
	Idle	16	1	1	1	1	1

Scheduling Algorithm

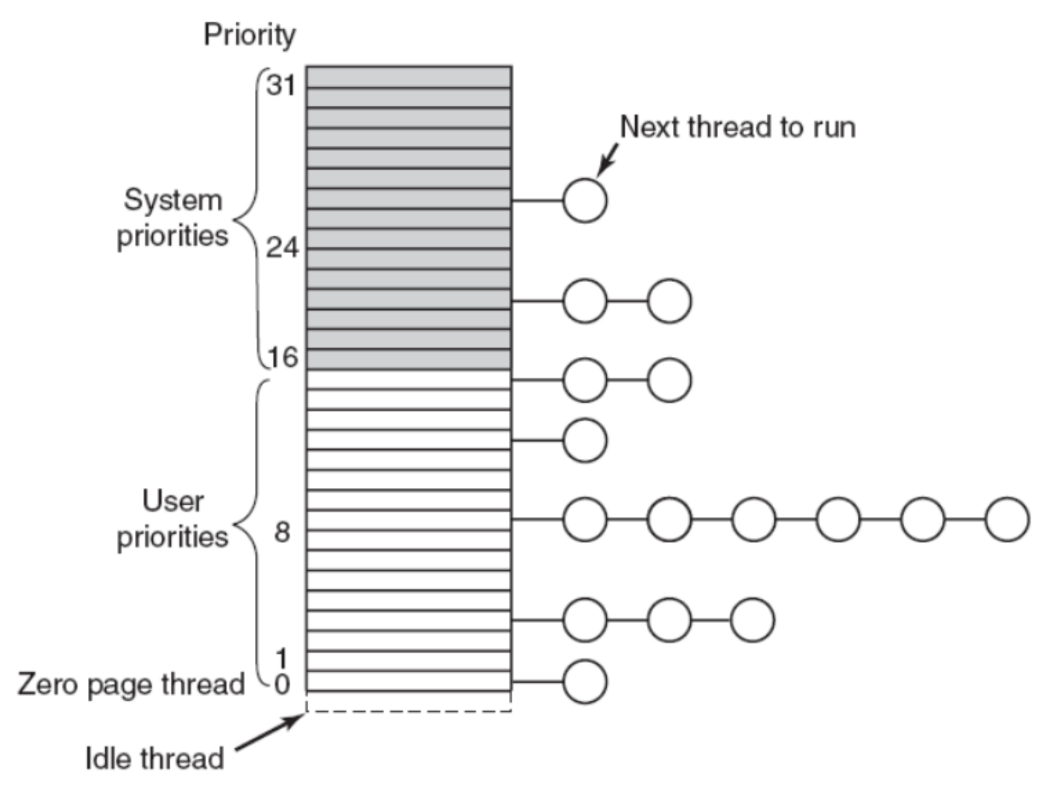
The system treats all threads with the same priority equally.

The system assigns time slices in a round-robin fashion to all threads with the highest priority.

❖If none of these threads are ready to run, the system assigns time slices in a round-robin fashion to all threads with the next highest priority.

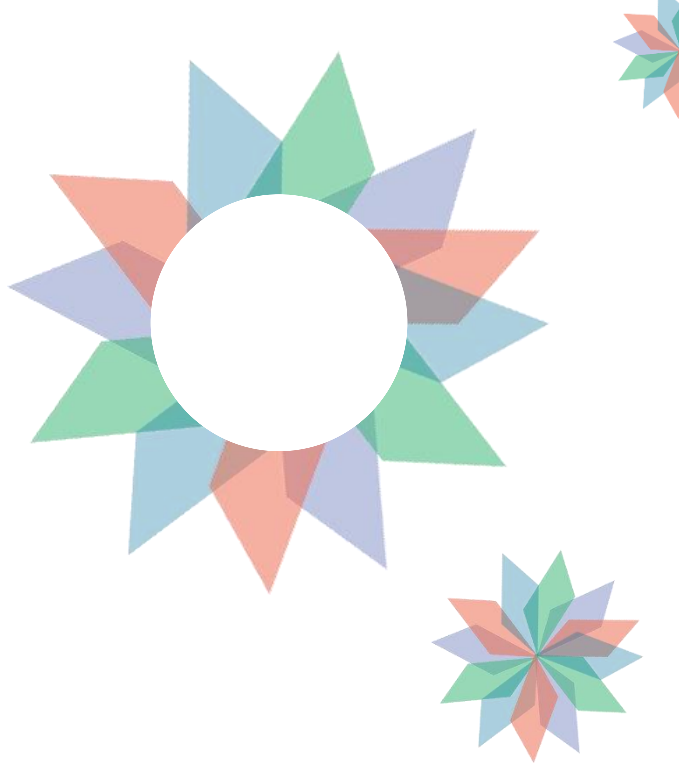
❖If a higher-priority thread becomes available to run, the system ceases to execute the lower-priority thread (without allowing it to finish using its time slice), and assigns a full time slice to the higher-priority thread.

Scheduling



References

- http://softwareblogs.intel.com/2006/10/19/why-windows-threads-are-better-than-posix-threads/
- https://msdn.microsoft.com/en-us/library/windows/desktop/ms682661(v=vs.85).aspx
- https://msdn.microsoft.com/en-us/library/windows/desktop/ms684161(v=vs.85).aspx
- https://msdn.microsoft.com/en-us/library/windows/desktop/ms681917(v=vs.85).aspx
- https://msdn.microsoft.com/en-us/library/windows/desktop/ms681917(v=vs.85).aspx



Thanks For
Listening

Thank You!
By Farah Sardouk