Operating Systems: Internals and Design Principles

Chapter 4
Threads

Seventh Edition

By William Stallings

Processes and Threads

Traditional processes have two characteristics:

Resource Ownership

Process includes a virtual address space to hold the process image

- the OS provides protection to prevent unwanted interference between processes with respect to resources

Scheduling/Execution

Follows an execution path that may be interleaved with other processes

- a process has an execution state (Running, Ready, etc.) and a dispatching priority and is scheduled and dispatched by the OS
- Traditional processes are *sequential*; i.e. only *one* execution path

Processes and Threads

- ***Multithreading*** **-** The ability of an OS to support multiple, concurrent paths of execution within a single process
- The unit of resource ownership is referred to as a ***process*** or ***task***
- The unit of dispatching is referred to as a ***thread*** or ***lightweight process***

Single Threaded Approaches

- A single execution path per process, in which the concept of a thread is not recognized, is referred to as a single-threaded approach
- MS-DOS, some versions of UNIX supported only this type of process.

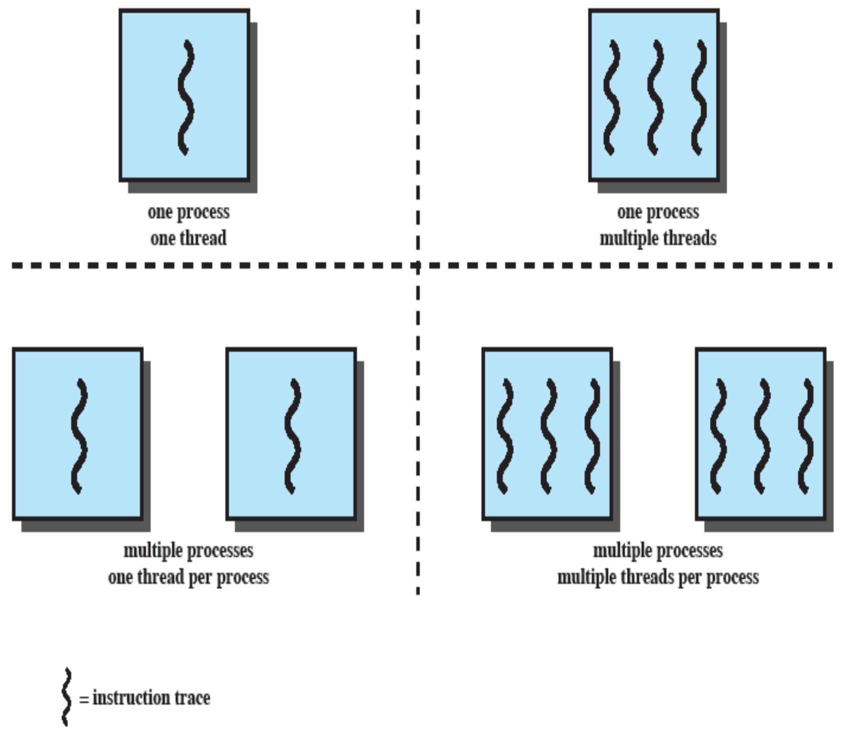


Figure 4.1 Threads and Processes [ANDE97]

Multithreaded Approaches

- The right half of Figure 4.1 depicts multithreaded approaches
- A Java run-time environment is a system of *one* process with multiple threads; Windows, some UNIXes, support *multiple* multithreaded processes.

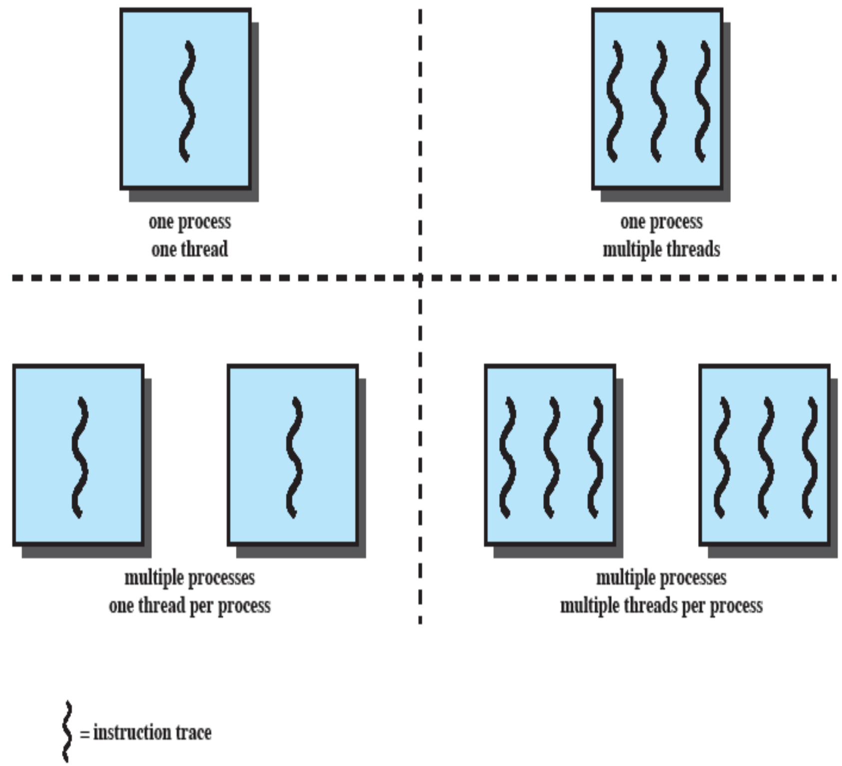
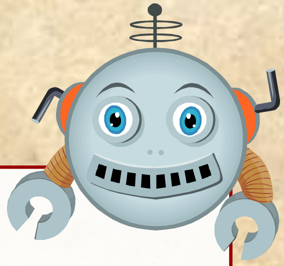


Figure 4.1 Threads and Processes [ANDE97]

Processes

- In a multithreaded environment the process is the unit that owns resources and the unit of protection.
  - i.e., the OS provides protection at the process level
- Processes have
  - A virtual address space that holds the process image
  - Protected access to
    - processors
    - other processes
    - files
    - I/O resources

One or More Threads in a Process



Each thread has:

- an execution state (Running, Ready, etc.)
- saved thread context when not running (TCB)
- an execution stack
- some per-thread static storage for local variables
- access to the shared memory and resources of its process (all threads of a process share this)

Threads vs. Processes

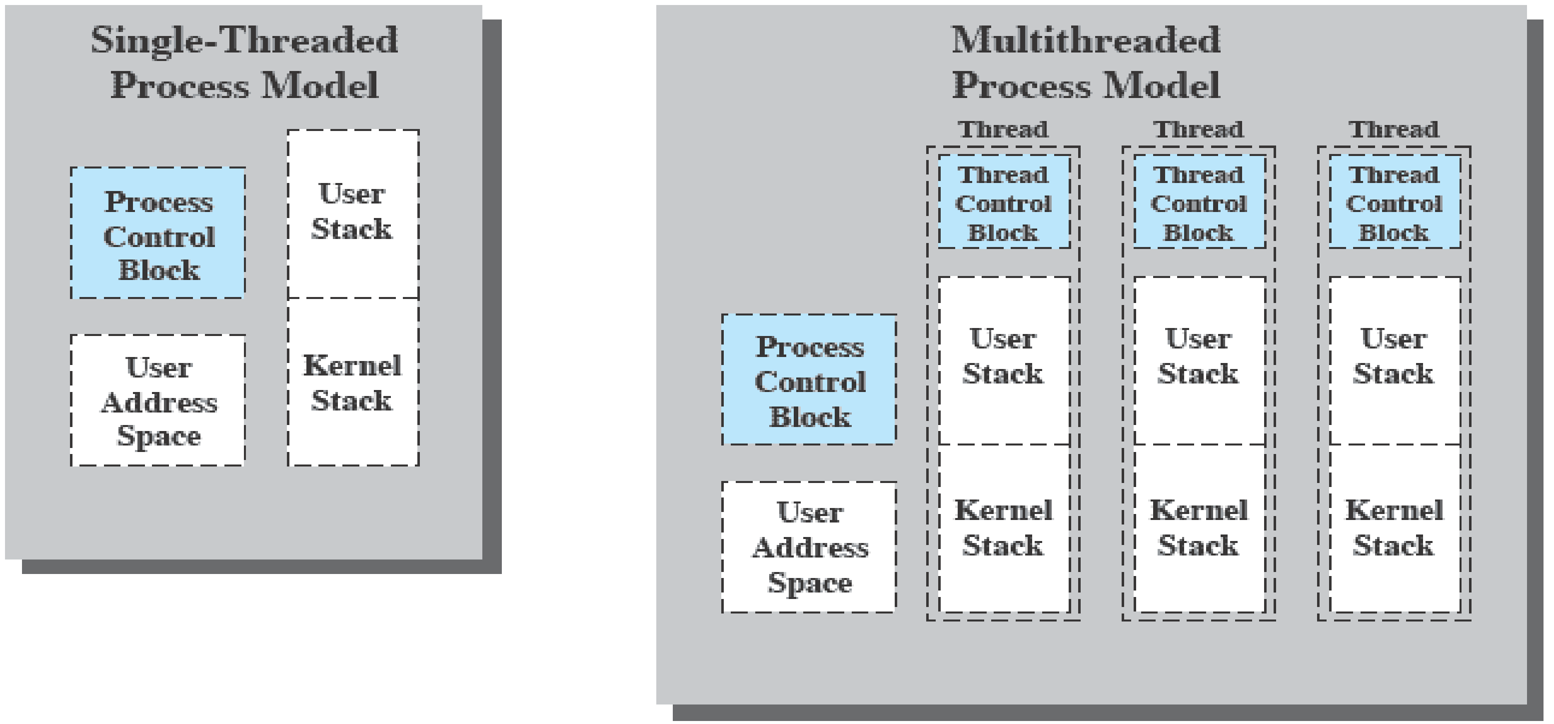


Figure 4.2 Single Threaded and Multithreaded Process Models

Benefits of Threads

- Takes less time to create a new thread than a process
- Less time to terminate a thread than a process
- Switching between two threads takes less time than switching between processes
- Threads enhance efficiency in communication between programs

Thread Use in a Single-User System

- Foreground and background work
- Asynchronous processing
- Speed of execution
- Modular program structure

Threads

- In an OS that supports threads, scheduling and dispatching is done on a thread basis

Most of the state information dealing with execution is maintained in thread-level data structures

- suspending a process involves suspending all threads of the process
- termination of a process terminates all thr within the process

Thread Execution States

The key states for a thread are:

- Running
- Ready
- Blocked

Thread operations associated with a change in thread state are:

- Spawn (create)
- Block
- Unblock
- Finish

Multithreading on a Uniprocessor

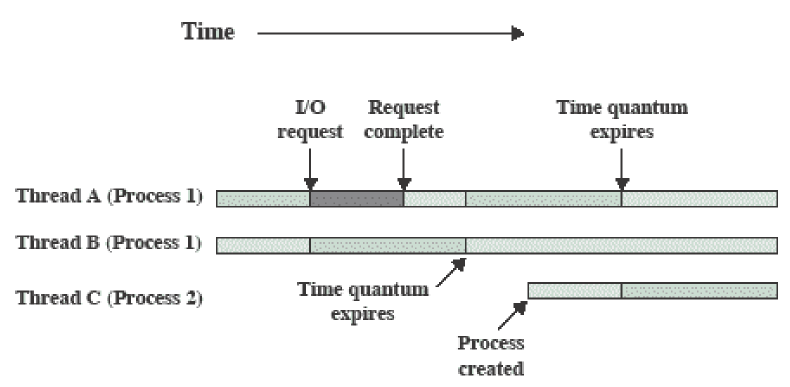
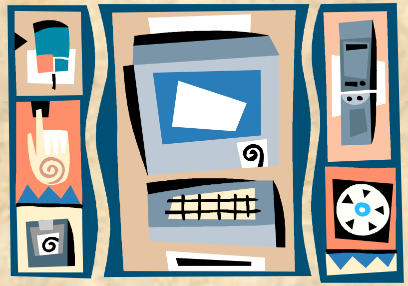


Figure 4.4 Multithreading Example on a Uniprocessor



Thread Synchronization

- It is necessary to synchronize the activities of the various threads
  - all threads of a process share the same address space and other resources
  - any alteration of a resource by one thread affects the other threads in the same process

Relationship Between Threads and Processes

Threads:Processes	Description	Example Systems
1:1	Each thread of execution is a unique process with its own address space and resources.	Traditional UNIX implementations
M:1	A process defines an address space and dynamic resource ownership. Multiple threads may be created and executed within that process.	Windows NT, Solaris, Linux, OS/2, OS/390, MACH
1:M	A thread may migrate from one process environment to another. This allows a thread to be easily moved among distinct systems.	Ra (Clouds), Emerald
M:N	Combines attributes of M:1 and 1:M cases.	TRIX

Table 4.2 Relationship between Threads and Processes

Multiple Cores & Multithreading

- Multithreading and multicore chips have the potential to improve performance of applications that have large amounts of parallelism
- Gaming, simulations, etc. are examples
- Performance doesn't necessarily scale linearly with the number of cores …

Amdahl's Law

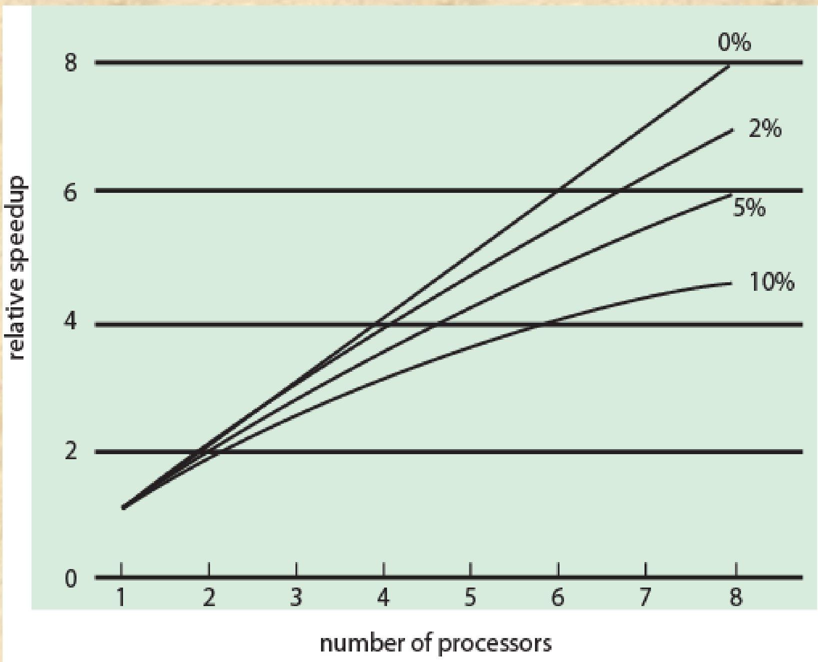
- Speedup depends on the amount of code that must be executed sequentially
- Formula:

$$\text{Speedup} = \frac{\text{time to run on single processor}}{\text{time to execute on N || processors}}$$

$$= \frac{1}{(1 - f) + f / N}$$

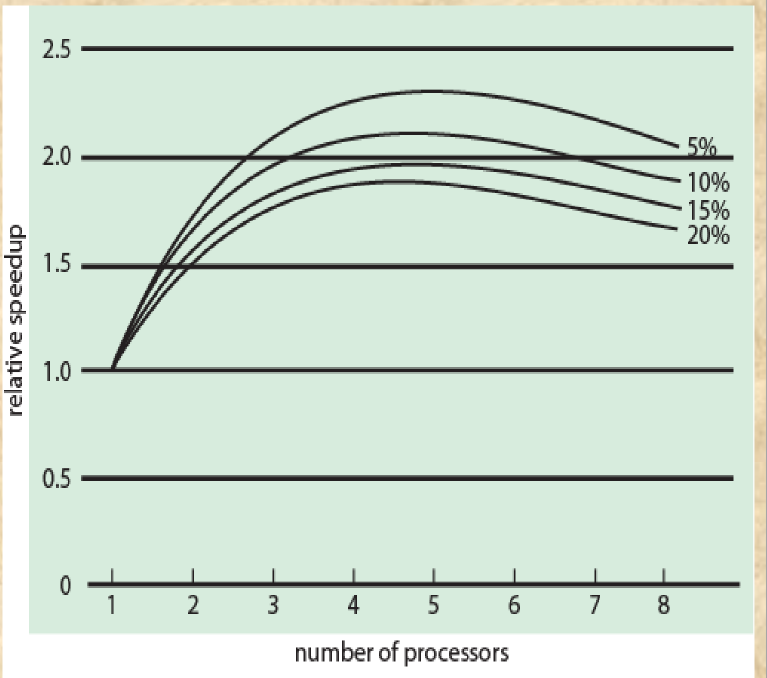
(where *f* is the amount of parallelizable code)

Performance Effect of Multiple Cores



(a) Speedup with 0%, 2%, 5%, and 10% sequential portions

Figure 4.7 (a)



(b) Speedup with overheads

Figure 4.7 (b)

Windows Processes

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

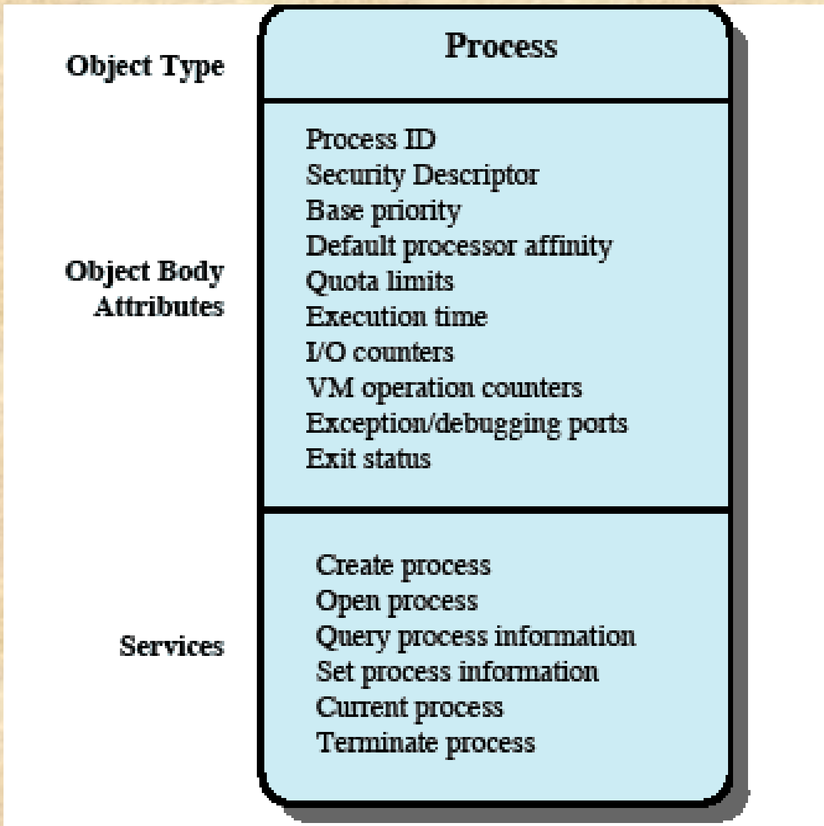
- implemented as objects
- created as new process or a copy of an existing
- an executable process may contain one or more threads
- both processes and thread objects have built-in synchronization capabilities

Process and Thread Objects

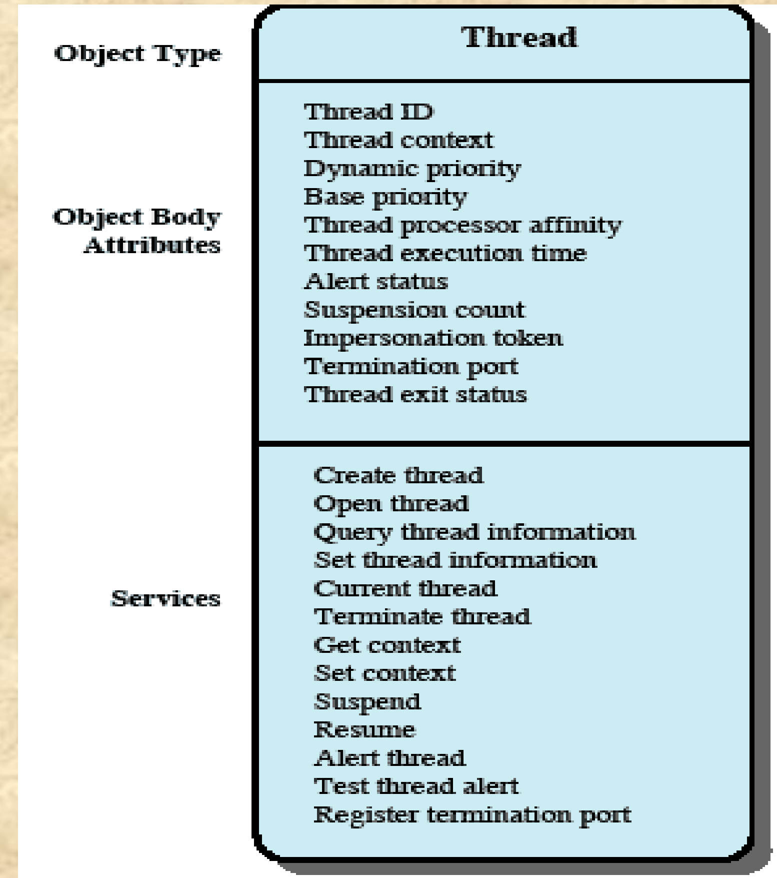
Windows makes use of two types of process-related objects:

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

Windows Process and Thread Objects



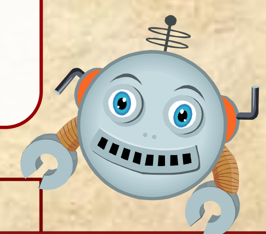
(a) Process object



(b) Thread object

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

Thread States

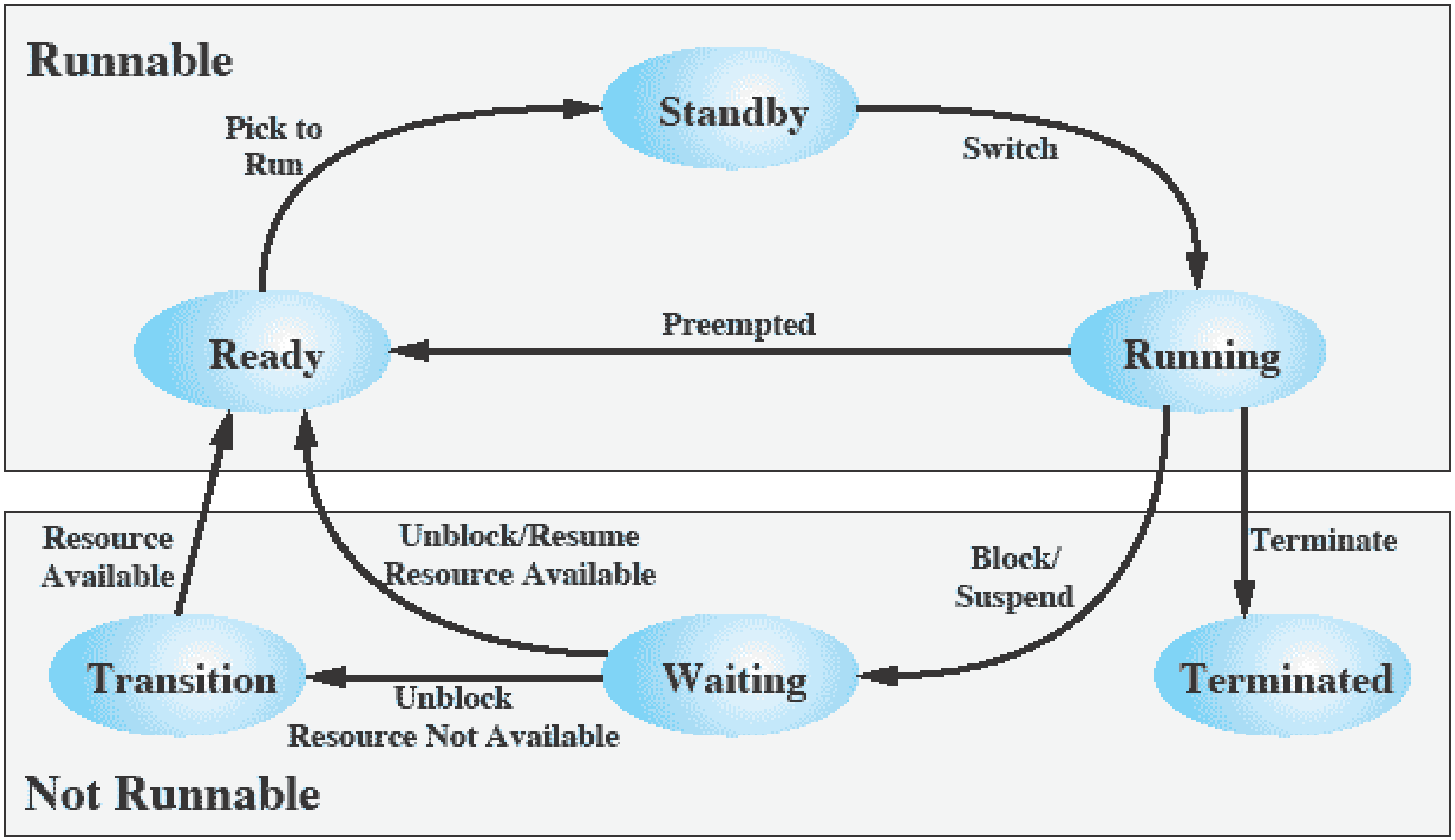
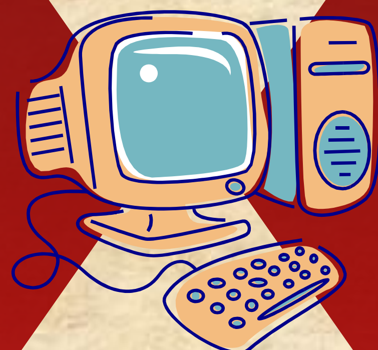


Figure 4.12 Windows Thread States

Linux Tasks

A process, or task, in Linux is represented by a task_struct data structure



This structure contains information in a number of categories

Linux Process/Thread Model

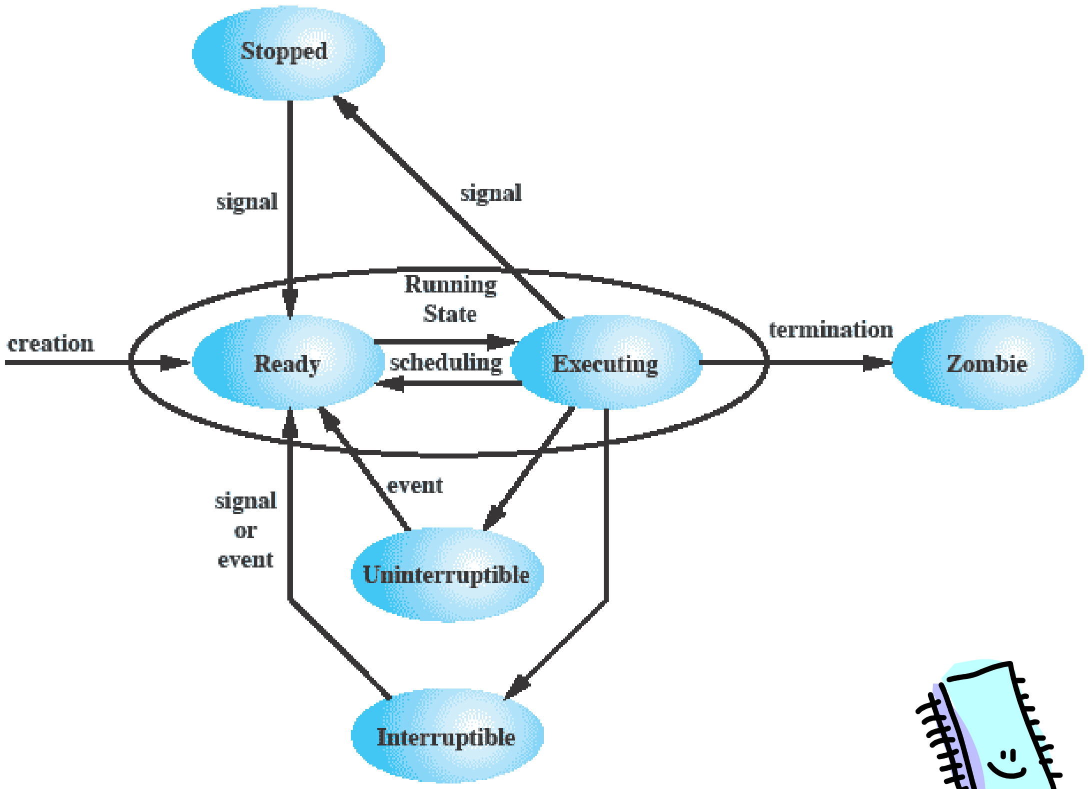


Figure 4.16 Linux Process/Thread Model

Linux Threads

Linux does not recognize a distinction between threads and processes

User-level threads are mapped into kernel-level processes

A new process is created by copying the attributes of the current process

The new process can be *cloned* so that it shares resources

The clone() call creates separate stack spaces for each process