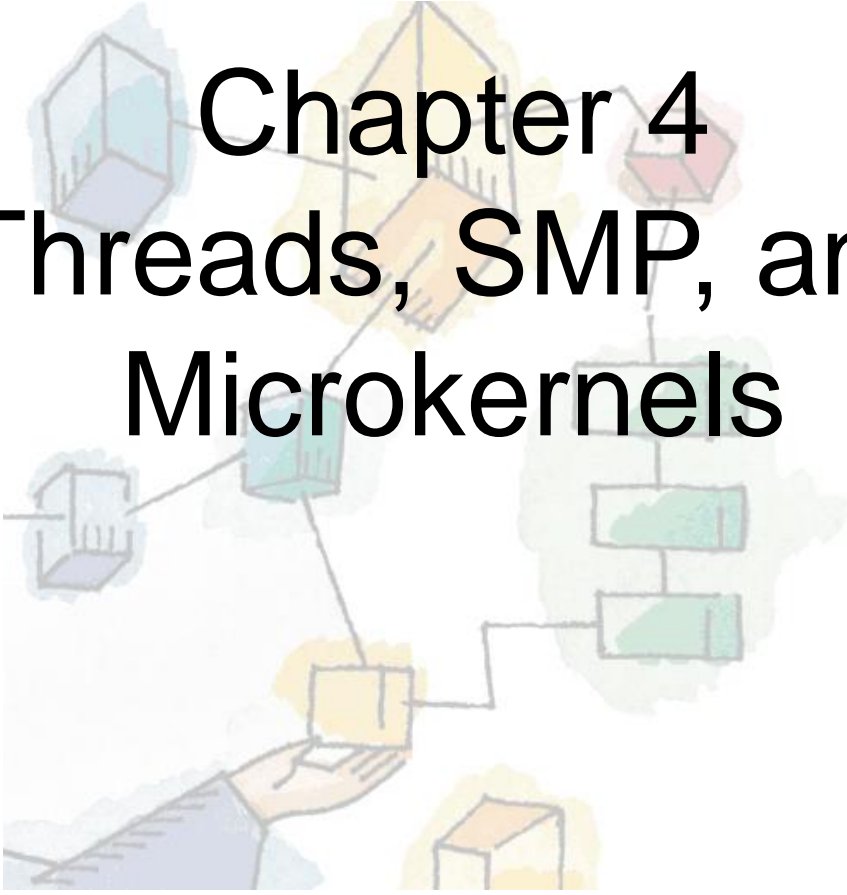


*Operating Systems:
Internals and Design Principles, 6/E*
William Stallings

Chapter 4 Threads, SMP, and Microkernels





Roadmap

→ Threads: Resource ownership and execution

- Symmetric multiprocessing (SMP).
- Microkernel
- Case Studies of threads and SMP:
 - Windows
 - Solaris
 - Linux





Processes and Threads

- Processes have two characteristics:
 - **Resource ownership** - process includes a virtual address space to hold the process image
 - **Scheduling/execution** - follows an execution path that may be interleaved with other processes
- These two characteristics are treated independently by the operating system

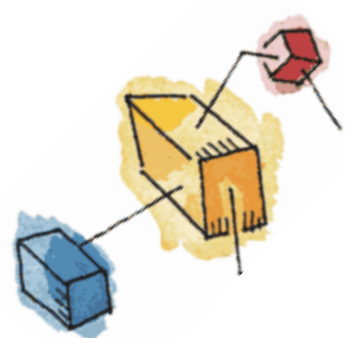




Processes and Threads

- The unit of dispatching is referred to as a ***thread*** or lightweight process
- The unit of resource ownership is referred to as a process or ***task***





Multithreading

- The ability of an OS to support multiple, concurrent paths of execution within a single process.

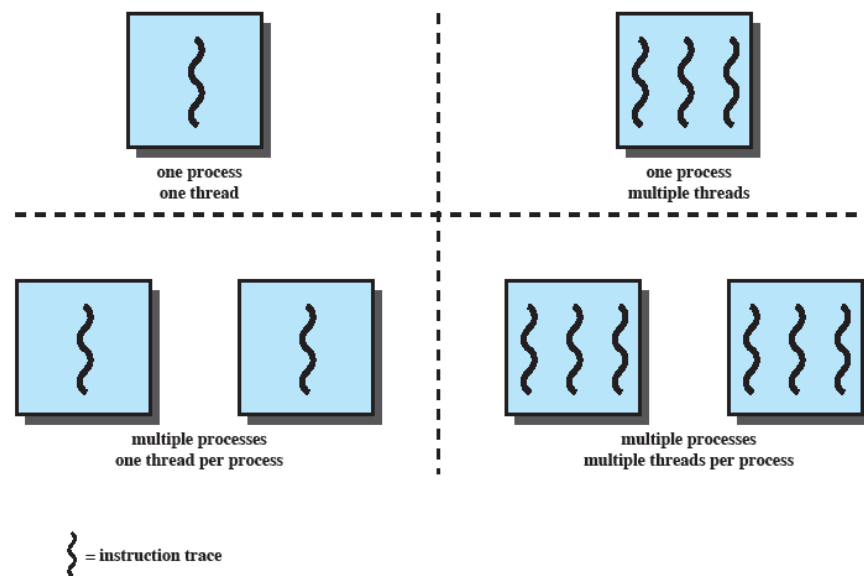


Figure 4.1 Threads and Processes [ANDE97]



Single Thread Approaches

- MS-DOS supports a single user process and a single thread.
- Some UNIX, support multiple user processes but only support one thread per process

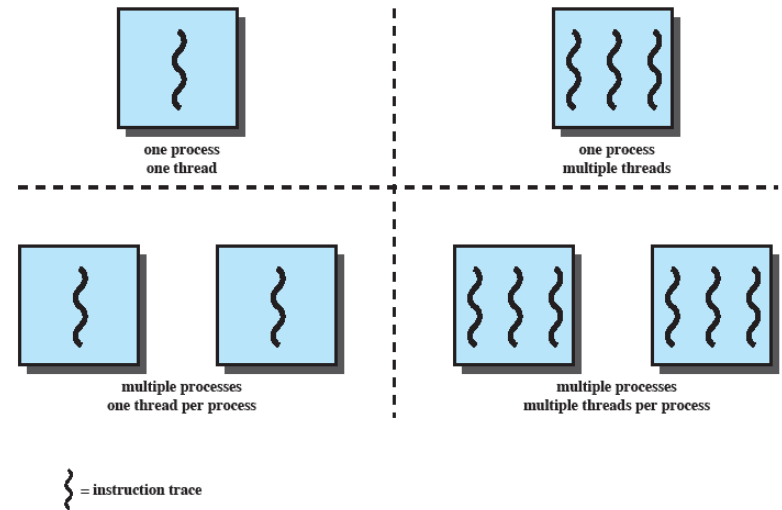


Figure 4.1 Threads and Processes [ANDE97]

Multithreading

- Java run-time environment is a single process with multiple threads
- Multiple processes **and** threads are found in Windows, Solaris, and many modern versions of UNIX

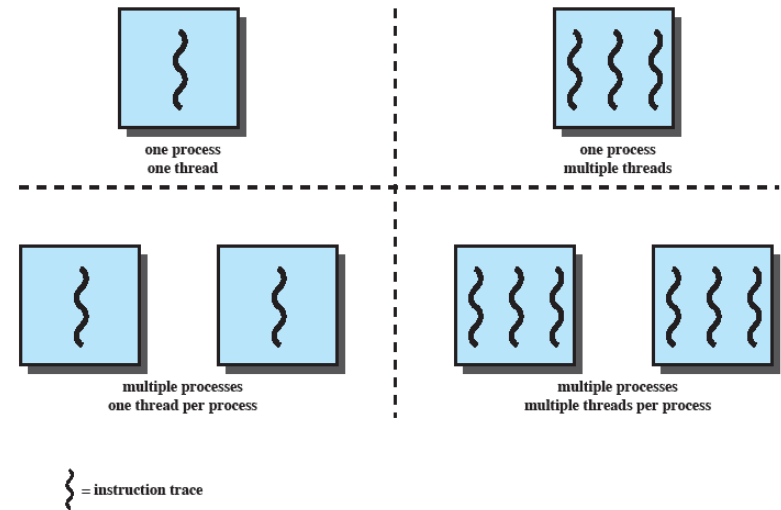
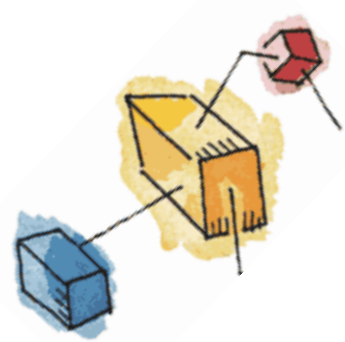


Figure 4.1 Threads and Processes [ANDE97]

Processes

- A virtual address space which holds the process image
- Protected access to
 - Processors,
 - Other processes,
 - Files,
 - I/O resources



An illustration in the top-left corner shows a large yellow box representing a process. Inside the box, there are several smaller boxes representing threads. One thread is highlighted in red, and another in blue. Lines connect these threads to the main process box.

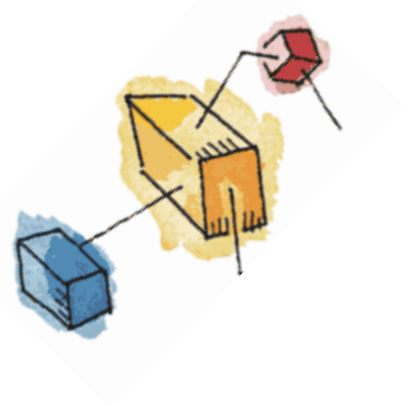
One or More Threads in Process

- Each thread has
 - An execution state (running, ready, etc.)
 - Saved thread context when not running
 - An execution stack
 - Some per-thread static storage for local variables
 - Access to the memory and resources of its process (all threads of a process share this)



One view...

- *One way to view a thread is as an independent program counter operating within a process.*





Threads vs. processes

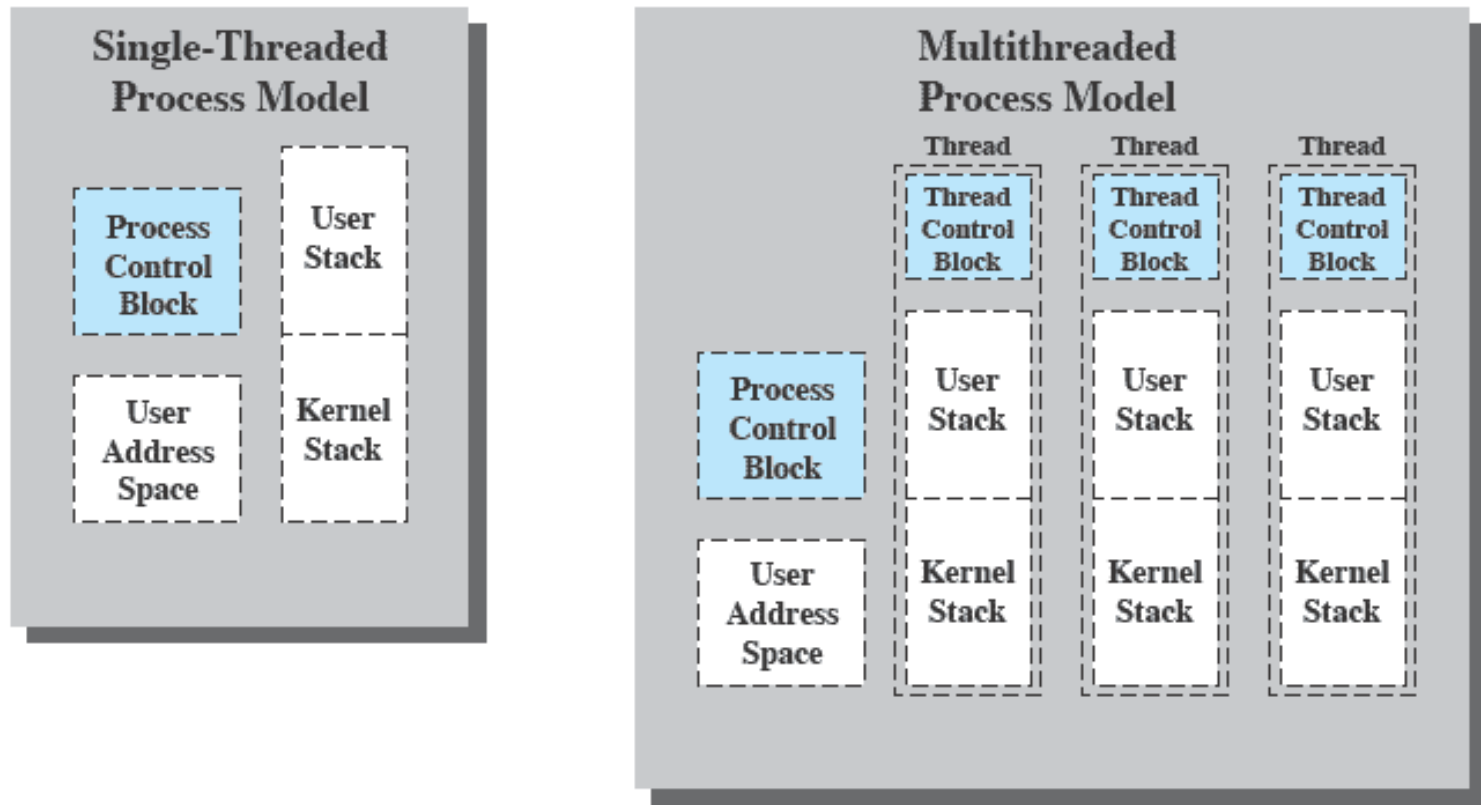


Figure 4.2 Single Threaded and Multithreaded Process Models

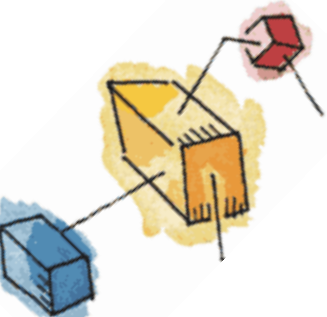




Benefits of Threads

- 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 processes
- Threads can communicate with each other
 - without invoking the kernel





Thread use in a Single-User System

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





Threads

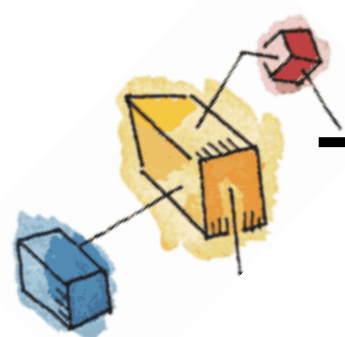
- Several actions that affect all of the threads in a process
 - The OS must manage these at the process level.
- Examples:
 - Suspending a process involves suspending all threads of the process
 - Termination of a process, terminates all threads within the process



Activities similar to Processes

- Threads have execution states and may synchronize with one another.
 - Similar to processes
- We look at these two aspects of thread functionality in turn.
 - States
 - Synchronisation





Thread Execution States

- States associated with a change in thread state
 - Spawn (another thread)
 - Block
 - Issue: will blocking a thread block other, or *all*, threads
 - Unblock
 - Finish (thread)
 - Deallocate register context and stacks

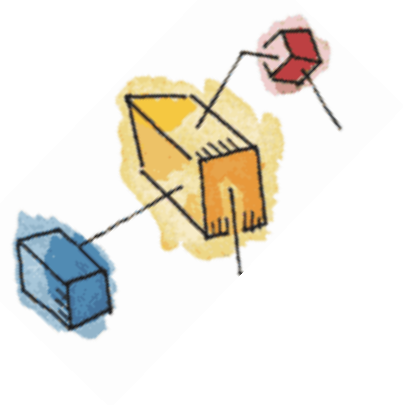




Example: Remote Procedure Call

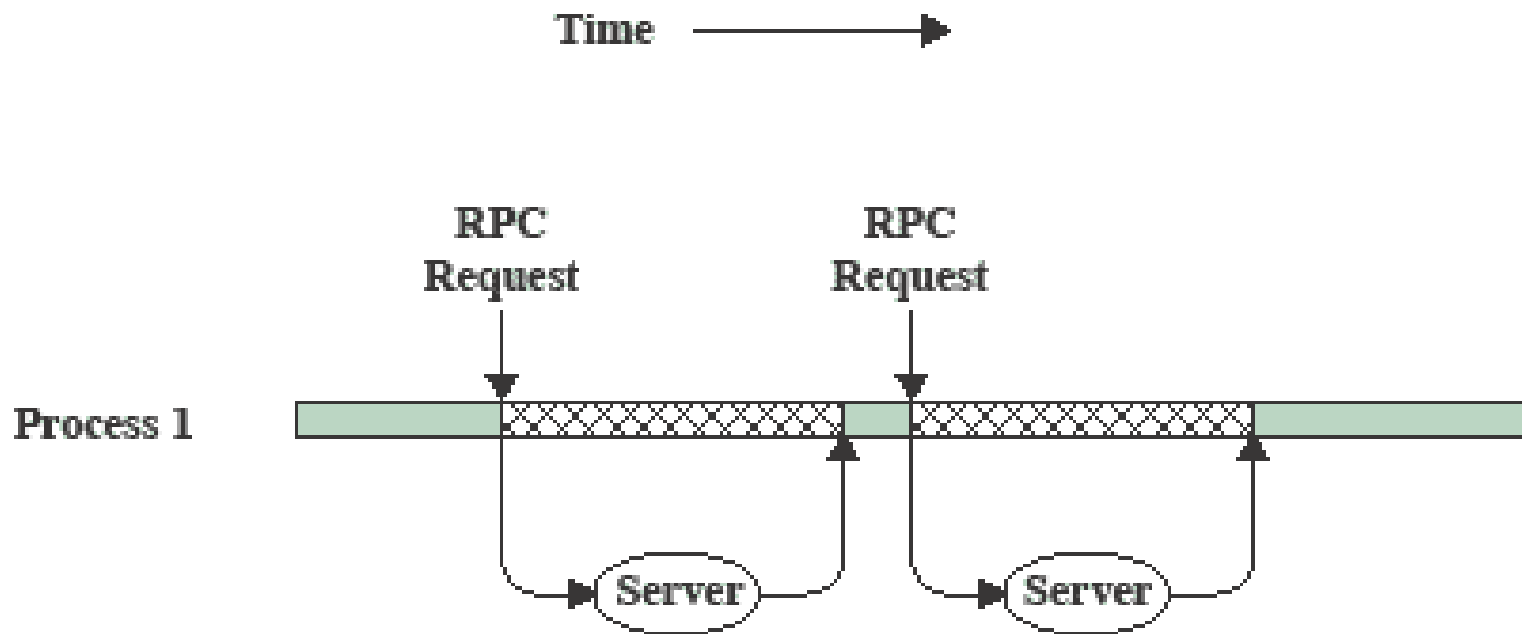
- Consider:
 - A program that performs two remote procedure calls (RPCs)
 - to two different hosts
 - to obtain a combined result.





RPC

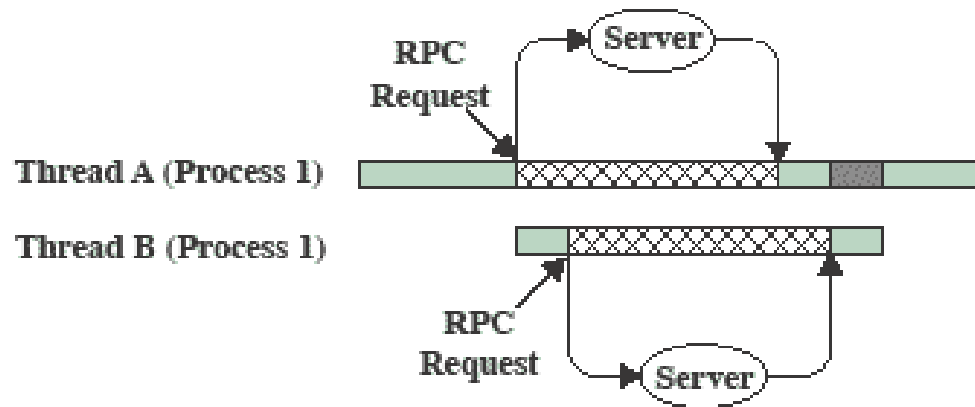
Using Single Thread






(a) RPC Using Single Thread



RPC Using One Thread per Server



(b) RPC Using One Thread per Server (on a uniprocessor)

-  Blocked, waiting for response to RPC
-  Blocked, waiting for processor, which is in use by Thread B
-  Running

Multithreading on a Uniprocessor

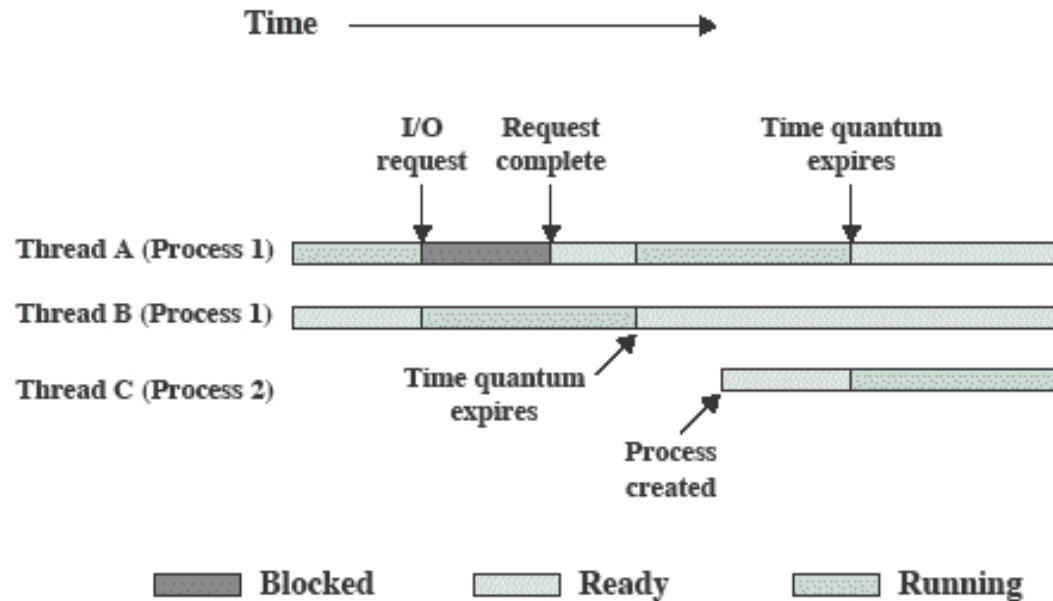


Figure 4.4 Multithreading Example on a Uniprocessor

Adobe PageMaker

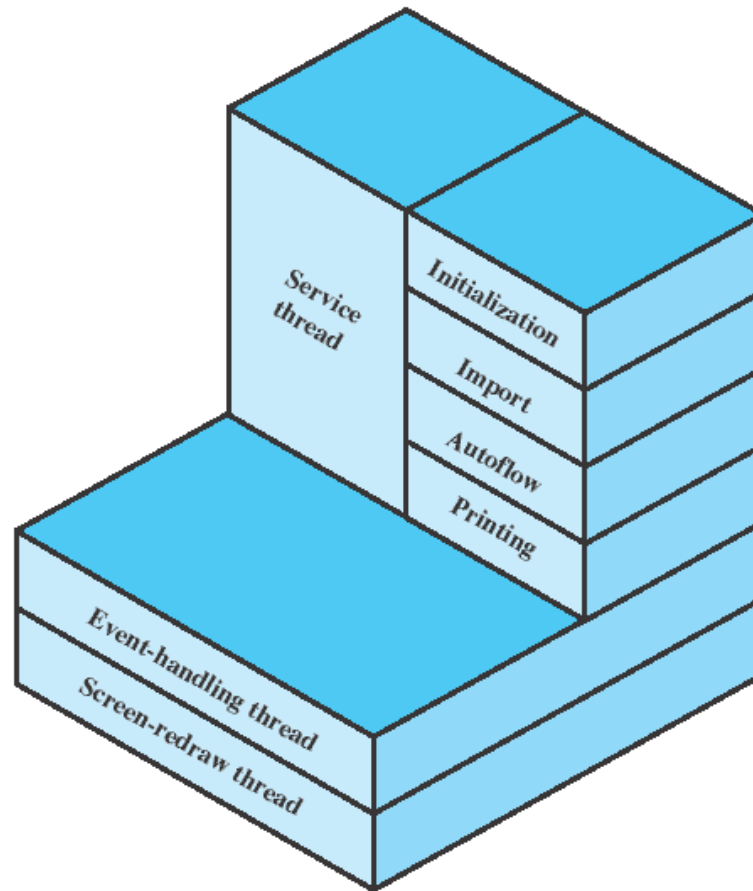
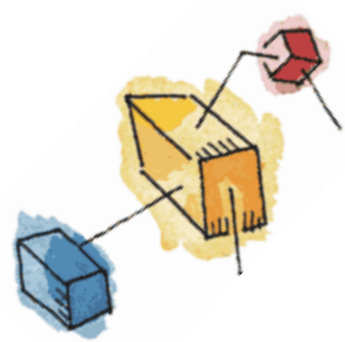


Figure 4.5 Thread Structure for Adobe PageMaker



Categories of Thread Implementation

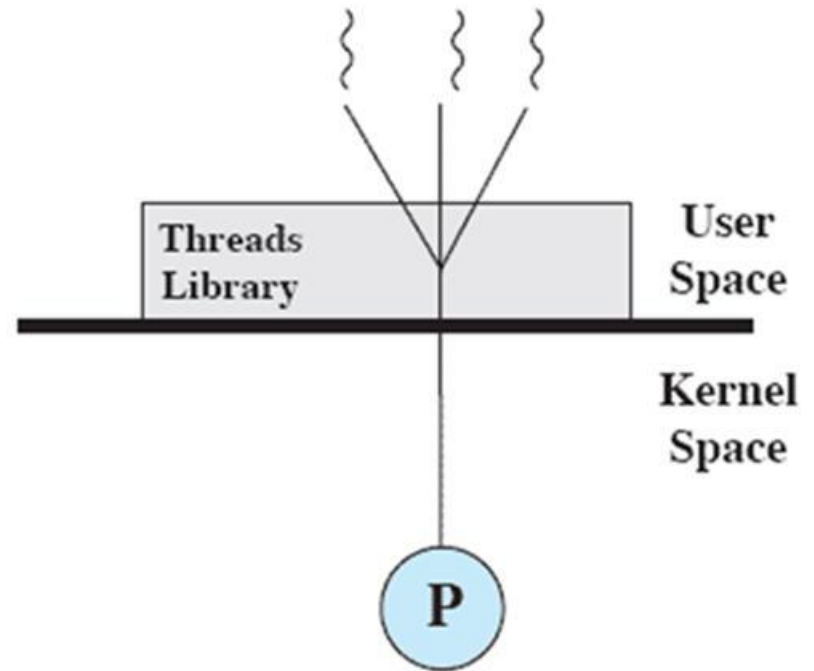
- User Level Thread (ULT)
- Kernel level Thread (KLT) also called:
 - kernel-supported threads
 - lightweight processes.





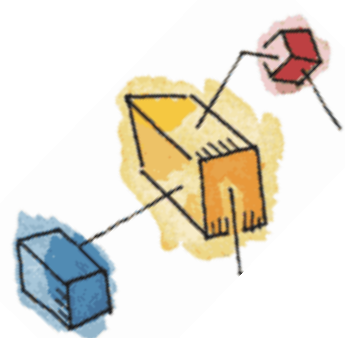
User-Level Threads

- All thread management is done by the application
- The kernel is not aware of the existence of threads

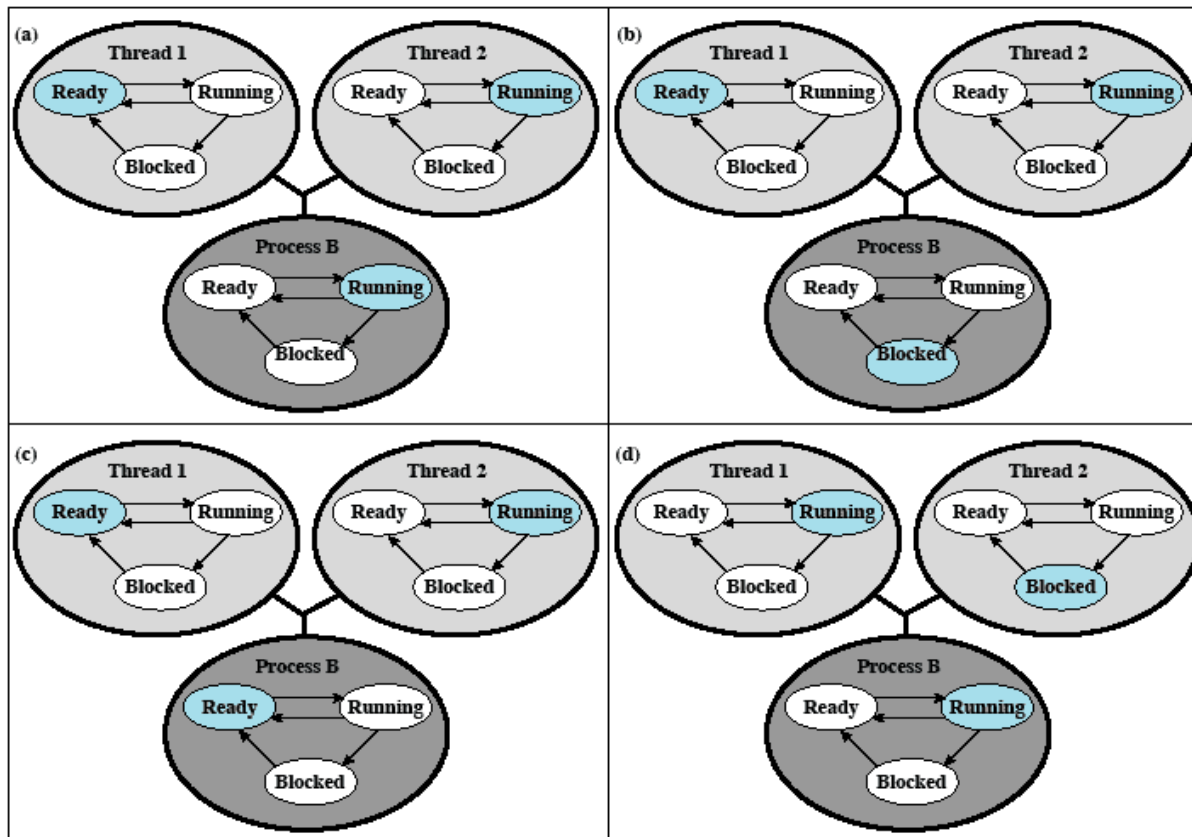


(a) Pure user-level





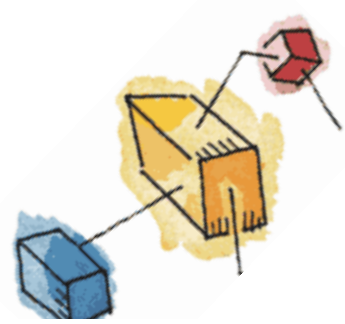
Relationships between ULT Thread and Process States



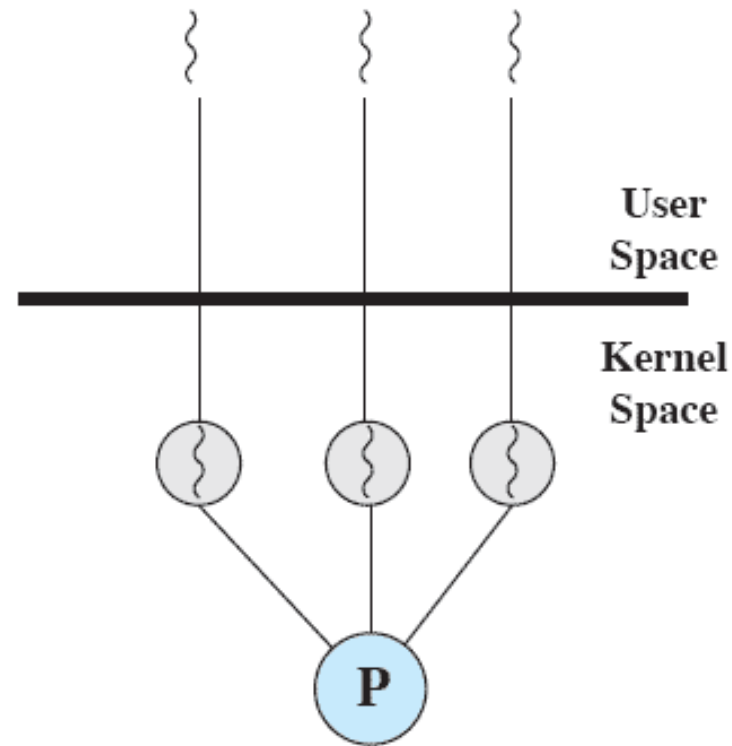
Colored state
is current state

Figure 4.7 Examples of the Relationships Between User-Level Thread States and Process States





Kernel-Level Threads



(b) Pure kernel-level

- Kernel maintains context information for the process and the threads
 - No thread management done by application
- Scheduling is done on a thread basis
- Windows is an example of this approach





Advantages of KLT

- The kernel can simultaneously schedule multiple threads from the same process on multiple processors.
- If one thread in a process is blocked, the kernel can schedule another thread of the same process.
- Kernel routines themselves can be multithreaded.





Disadvantage of KLT

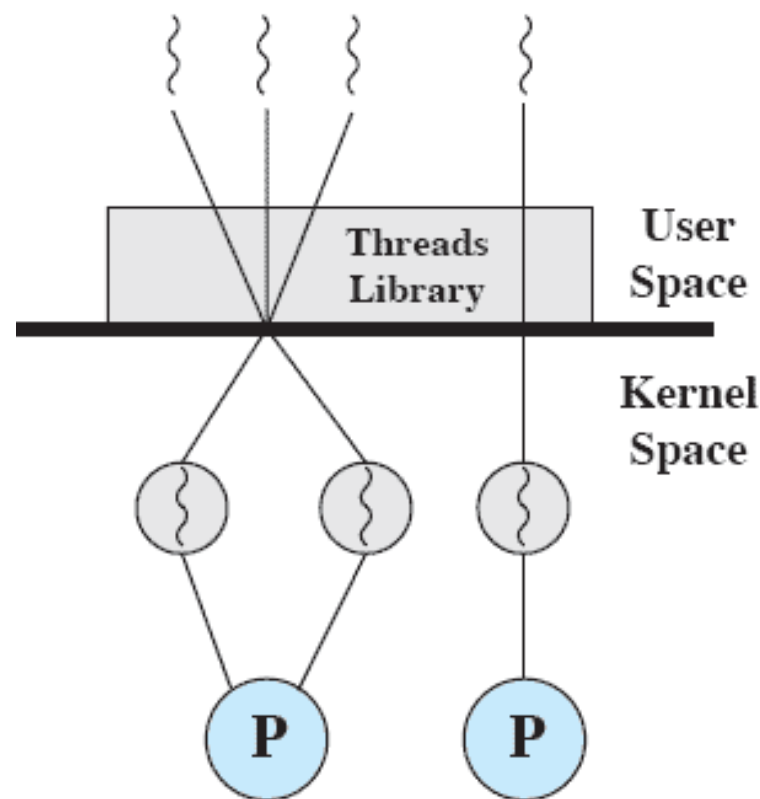
- The transfer of control from one thread to another within the same process requires a mode switch to the kernel





Combined Approaches

- Thread creation done in the user space
- Bulk of scheduling and synchronization of threads by the application
- Example is Solaris



(c) Combined






Relationship Between Thread and Processes

Table 4.2 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





Roadmap

- Threads: Resource ownership and execution

→ Symmetric multiprocessing (SMP).

- Microkernel
- Case Studies of threads and SMP:
 - Windows
 - Solaris
 - Linux





Traditional View

- Traditionally, the computer has been viewed as a sequential machine.
 - A processor executes instructions one at a time in sequence
 - Each instruction is a sequence of operations
- Two popular approaches to providing parallelism
 - Symmetric MultiProcessors (SMPs)
 - Clusters (ch 16)





Categories of Computer Systems

- Single Instruction Single Data (SISD) stream
 - Single processor executes a single instruction stream to operate on data stored in a single memory
- Single Instruction Multiple Data (SIMD) stream
 - Each instruction is executed on a different set of data by the different processors





Categories of Computer Systems

- Multiple Instruction Single Data (MISD) stream (Never implemented)
 - A sequence of data is transmitted to a set of processors, each of execute a different instruction sequence
- Multiple Instruction Multiple Data (MIMD)
 - A set of processors simultaneously execute different instruction sequences on different data sets



Parallel Processor Architectures

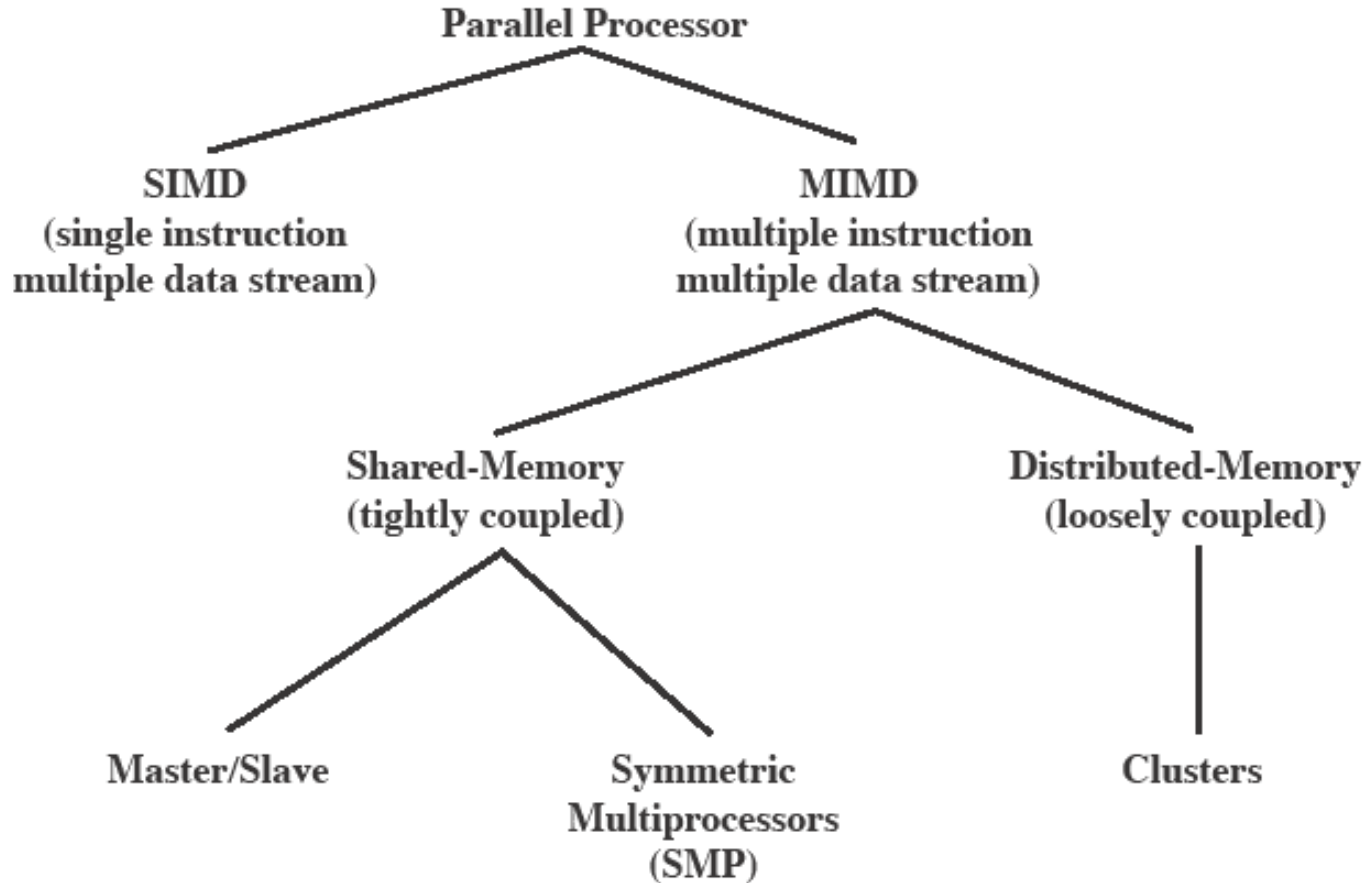
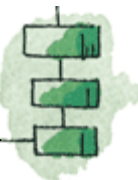
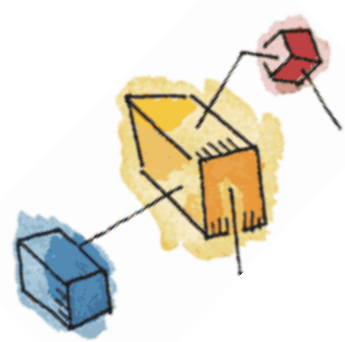


Figure 4.8 Parallel Processor Architectures

Symmetric Multiprocessing

- Kernel can execute on any processor
 - Allowing portions of the kernel to execute in parallel
- Typically each processor does self-scheduling from the pool of available process or threads



Typical SMP Organization

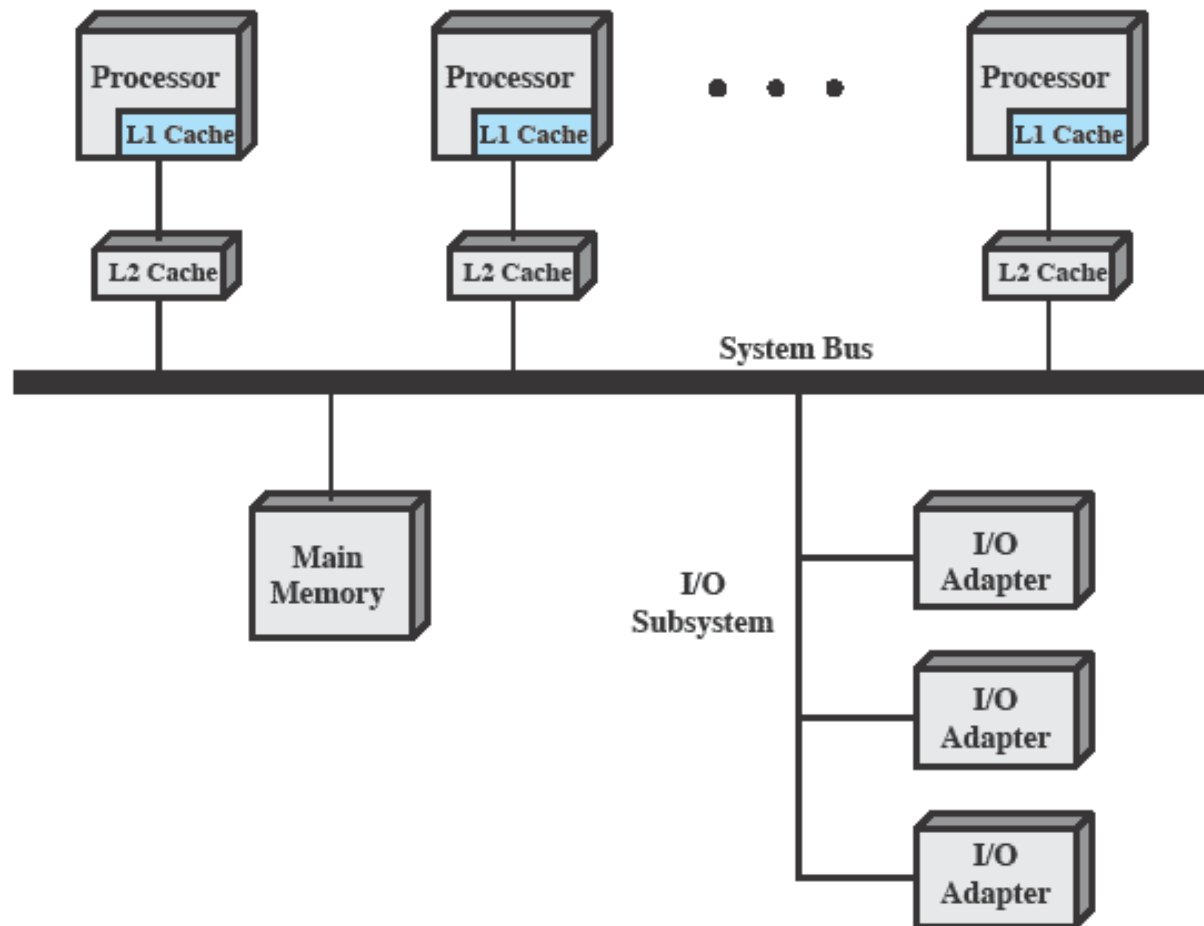
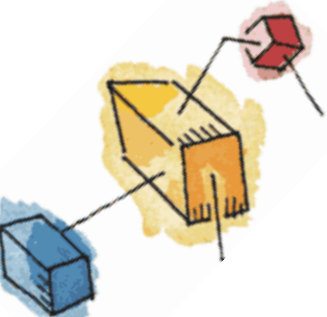


Figure 4.9 Symmetric Multiprocessor Organization



Multiprocessor OS Design Considerations

- The key design issues include
 - Simultaneous concurrent processes or threads
 - Scheduling
 - Synchronization
 - Memory Management
 - Reliability and Fault Tolerance



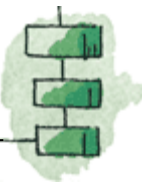


Roadmap

- Threads: Resource ownership and execution
- Symmetric multiprocessing (SMP).

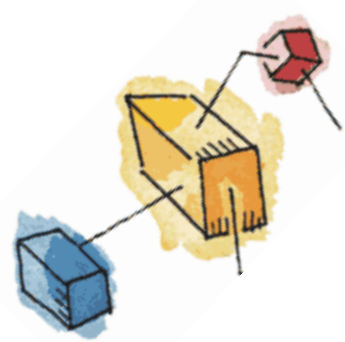
→ Microkernel

- Case Studies of threads and SMP:
 - Windows
 - Solaris
 - Linux

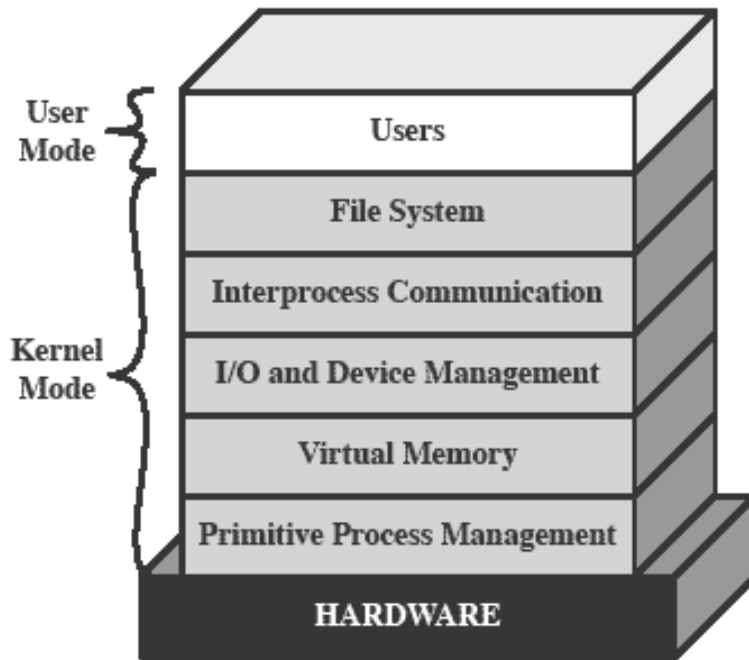


Microkernel

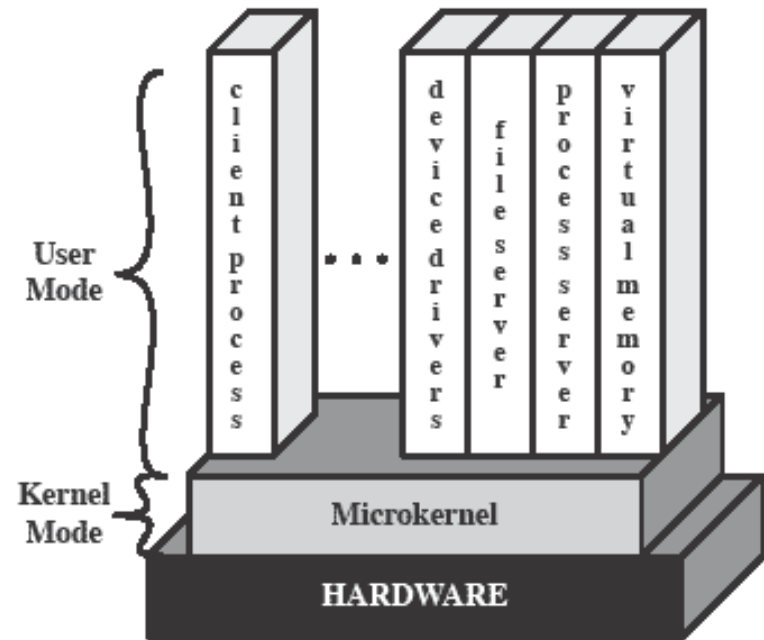
- A microkernel is a small OS core that provides the foundation for modular extensions.
- Big question is how small must a kernel be to qualify as a microkernel
 - *Must* drivers be in user space?
- In theory, this approach provides a high degree of flexibility and modularity.



Kernel Architecture



(a) Layered kernel



(b) Microkernel

Figure 4.10 Kernel Architecture



Microkernel Design: Memory Management

- Low-level memory management - Mapping each virtual page to a physical page frame
 - Most memory management tasks occur in user space

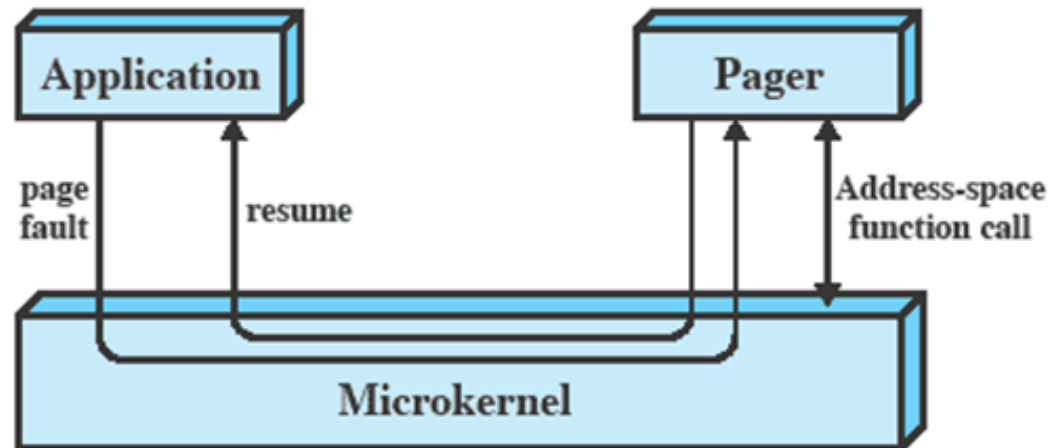
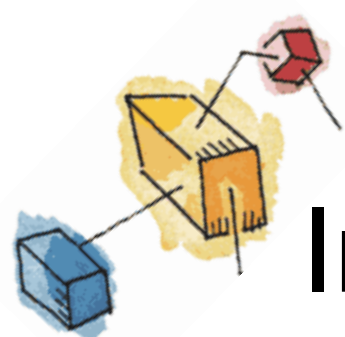


Figure 4.11 Page Fault Processing

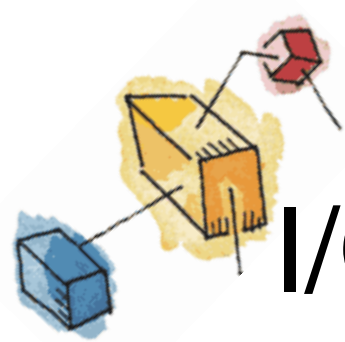




Microkernel Design: Interprocess Communication

- Communication between processes or threads in a microkernel OS is via messages.
- A message includes:
 - A header that identifies the sending and receiving process and
 - A body that contains direct data, a pointer to a block of data, or some control information about the process.

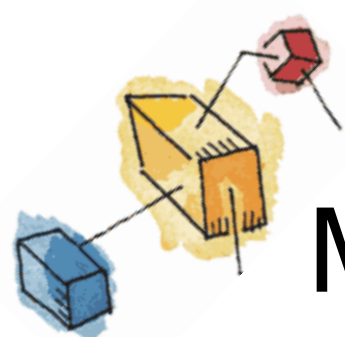




Microkernel Design: I/O and interrupt management

- Within a microkernel it is possible to handle hardware interrupts as messages and to include I/O ports in address spaces.
 - a particular user-level process is assigned to the interrupt and the kernel maintains the mapping.





Benefits of a Microkernel Organization

- Uniform interfaces on requests made by a process.
- Extensibility
- Flexibility
- Portability
- Reliability
- Distributed System Support
- Object Oriented Operating Systems





Roadmap

- Threads: Resource ownership and execution
- Symmetric multiprocessing (SMP).
- Microkernel

→ Case Studies of threads and SMP:

- Windows
- Solaris
- Linux





Different Approaches to Processes

- Differences between different OS's support of processes include
 - How processes are named
 - Whether threads are provided
 - How processes are represented
 - How process resources are protected
 - What mechanisms are used for inter-process communication and synchronization
 - How processes are related to each other





Windows Processes

- Processes and services provided by the Windows Kernel are relatively simple and general purpose
 - Implemented as objects
 - An executable process may contain one or more threads
 - Both processes and thread objects have built-in synchronization capabilities



Relationship between Process and Resources

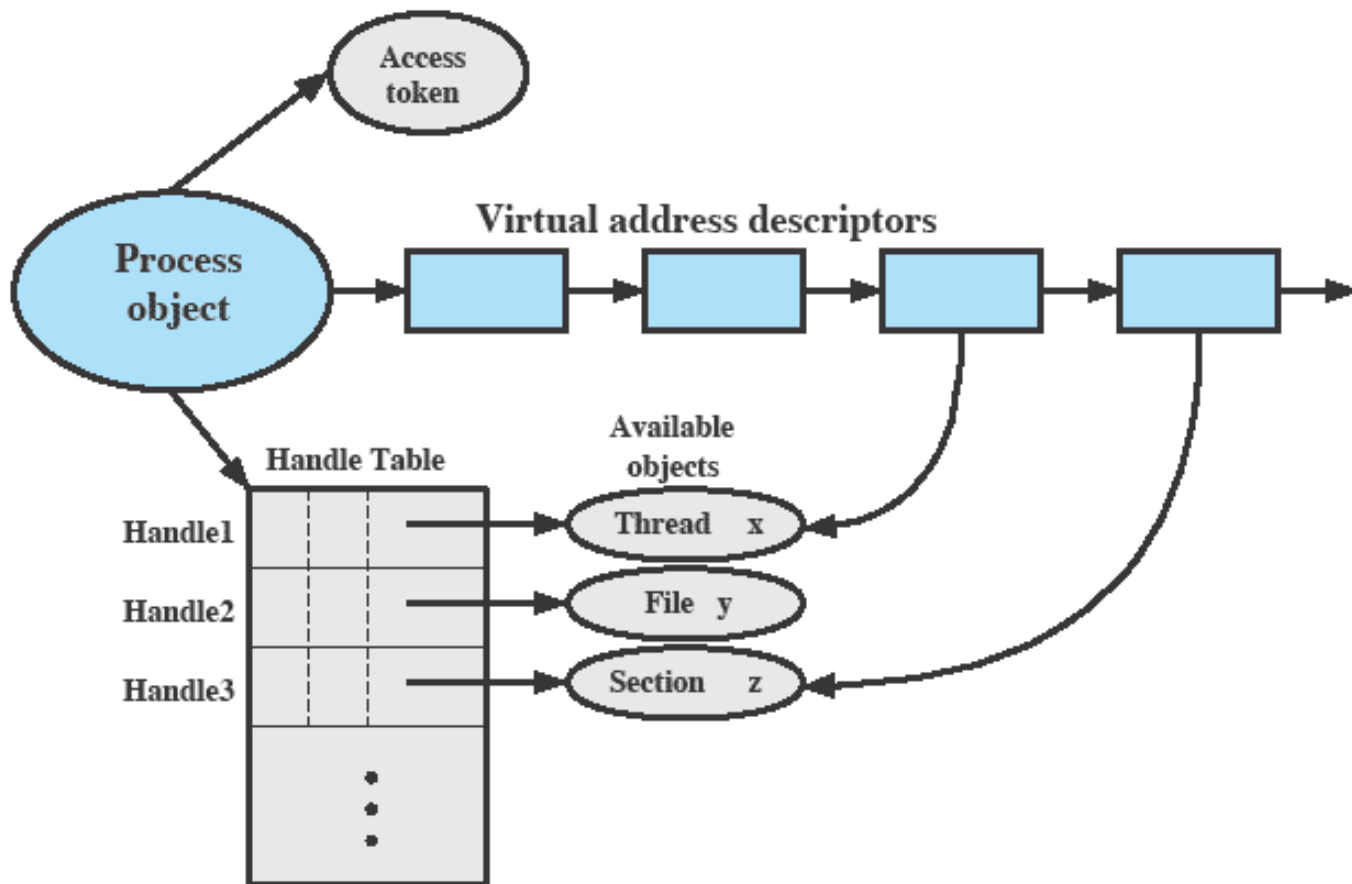


Figure 4.12 A Windows Process and Its Resources



Windows Process Object

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

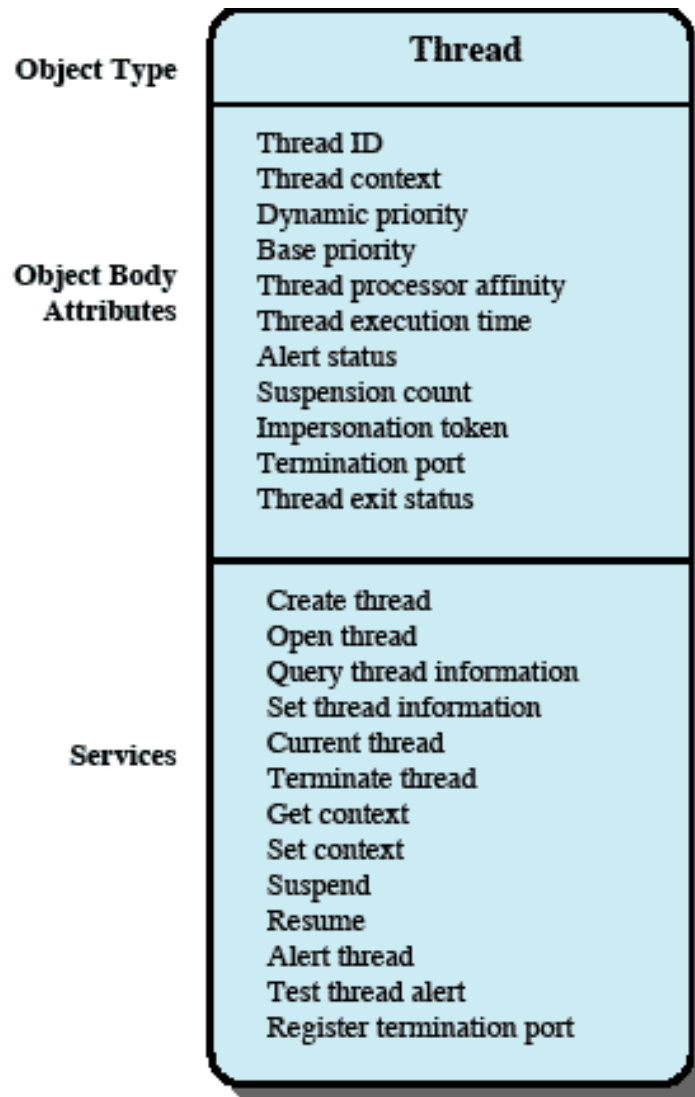
Create process
Open process
Query process information
Set process information
Current process
Terminate process

(a) Process object






Windows Thread Object



(b) Thread object



Thread States

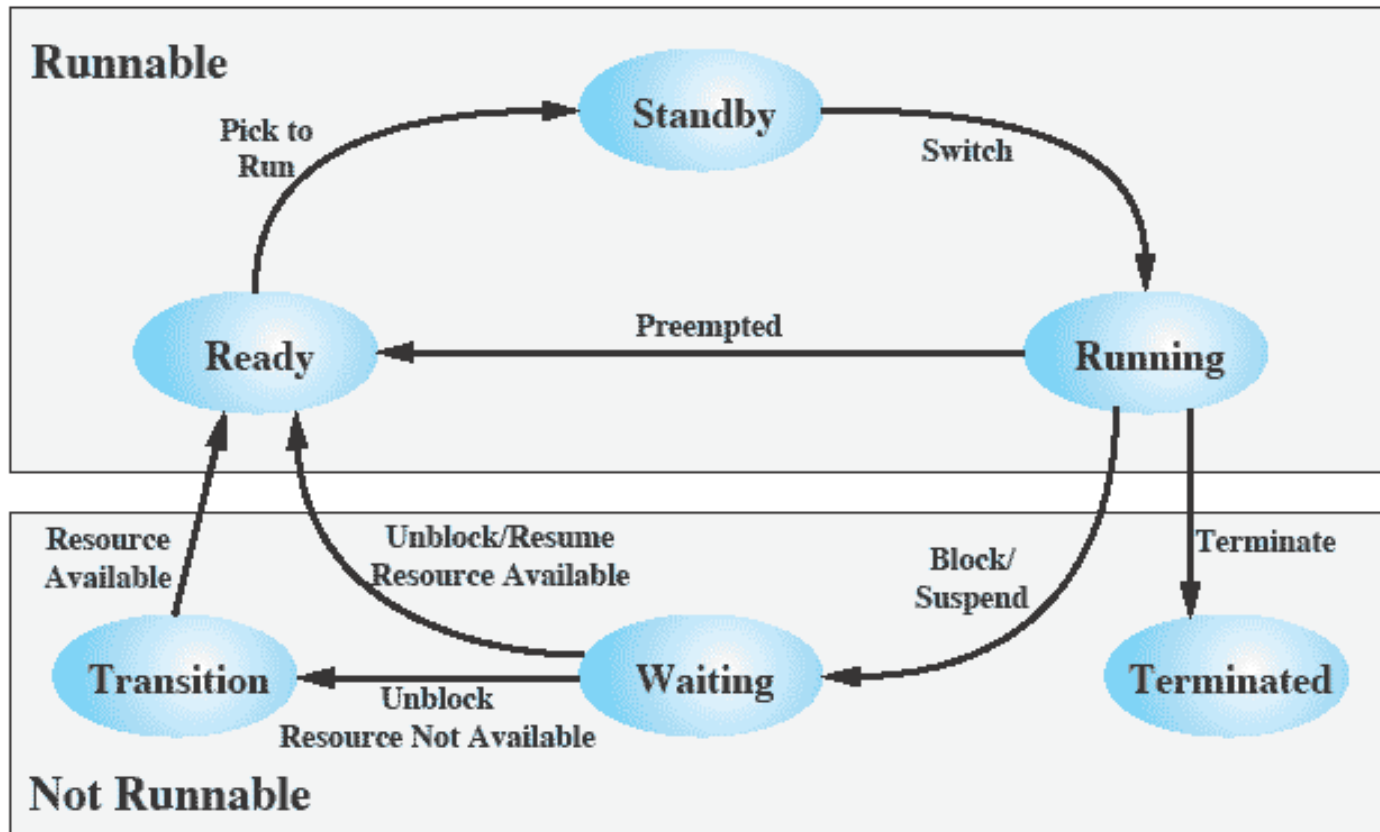


Figure 4.14 Windows Thread States



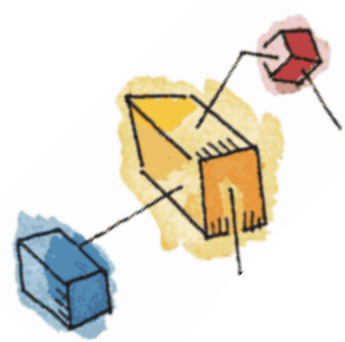
Windows SMP Support

- **Threads can run on any processor**
 - But an application can restrict affinity
- **Soft Affinity**
 - The dispatcher tries to assign a ready thread to the same processor it last ran on.
 - This helps reuse data still in that processor's memory caches from the previous execution of the thread.
- **Hard Affinity**
 - An application restricts threads to certain processor



Solaris

- Solaris implements multilevel thread support designed to provide flexibility in exploiting processor resources.
- Processes include the user's address space, stack, and process control block





Solaris Process

- Solaris makes use of four separate thread-related concepts:
 - Process: includes the user's address space, stack, and process control block.
 - User-level threads: a user-created unit of execution within a process.
 - Lightweight processes: a mapping between ULTs and kernel threads.
 - Kernel threads



Relationship between Processes and Threads

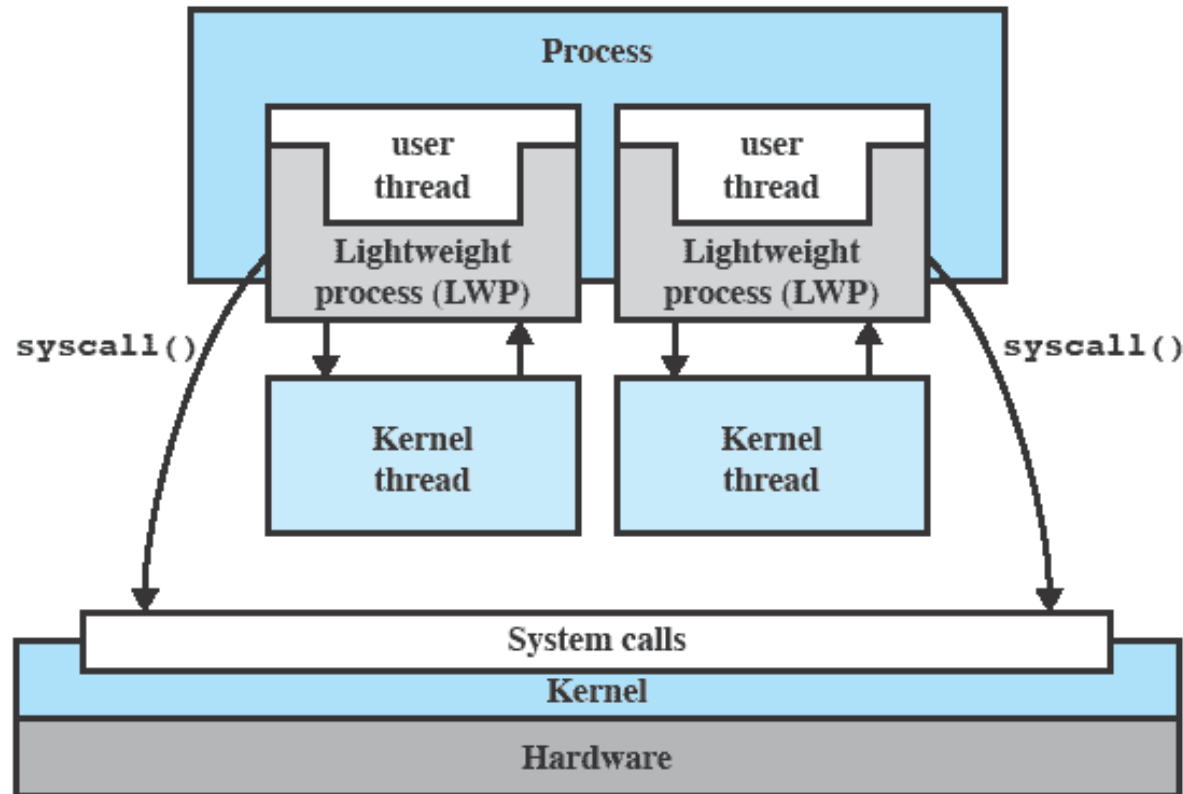
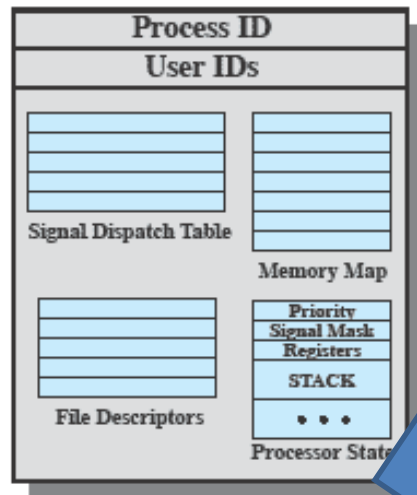


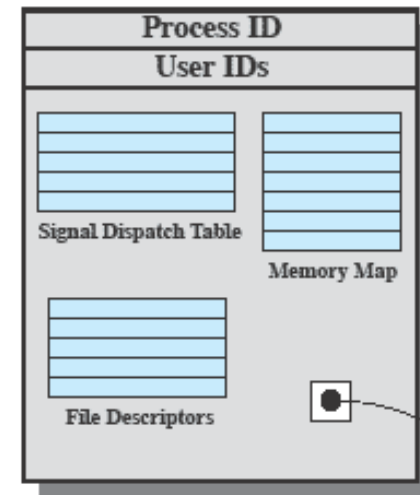
Figure 4.15 Processes and Threads in Solaris [MCDO07]

Traditional Unix vs Solaris

UNIX Process Structure



Solaris Process Structure



Solaris replaces the processor state block with a list of LWPs

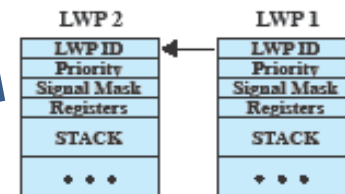


Figure 4.16 Process Structure in Traditional UNIX and Solaris [LEWI96]



LWP Data Structure

- An LWP identifier
- The priority of this LWP
- A signal mask
- Saved values of user-level registers
- The kernel stack for this LWP
- Resource usage and profiling data
- Pointer to the corresponding kernel thread
- Pointer to the process structure



Solaris Thread States

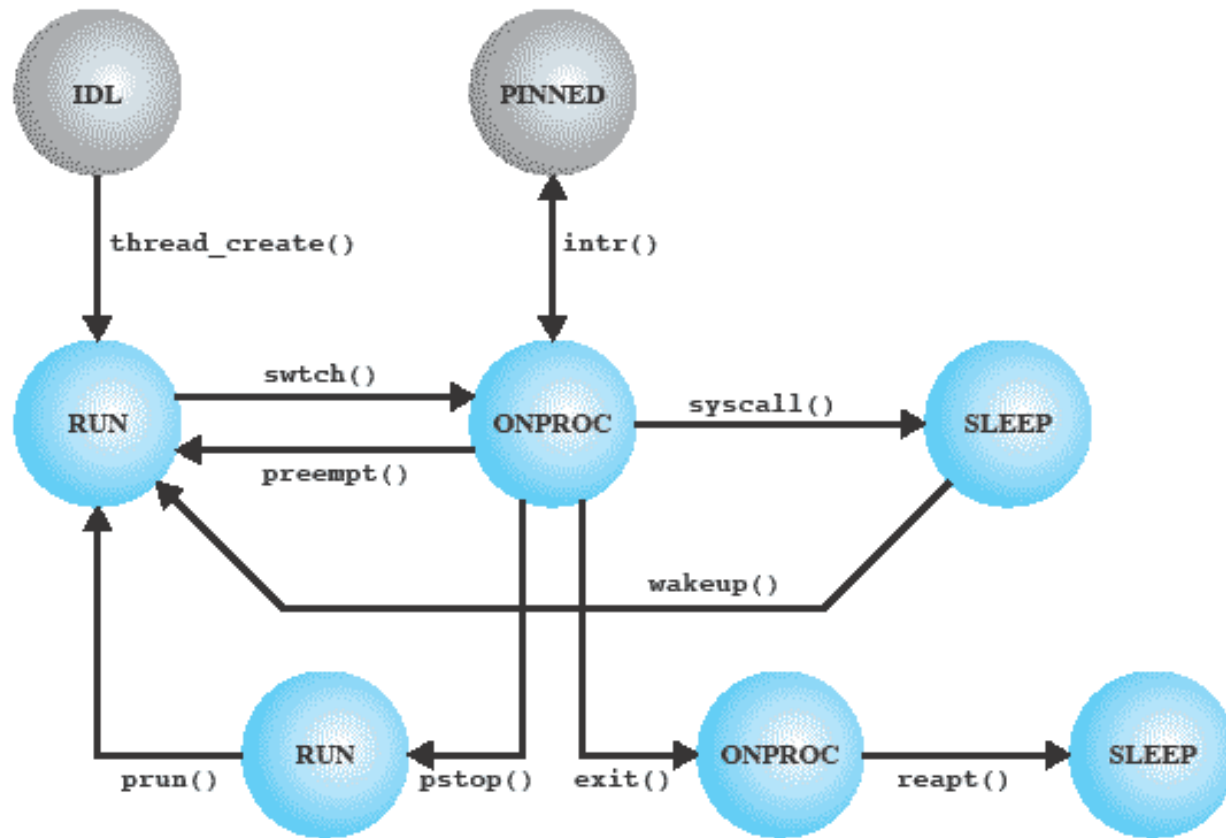
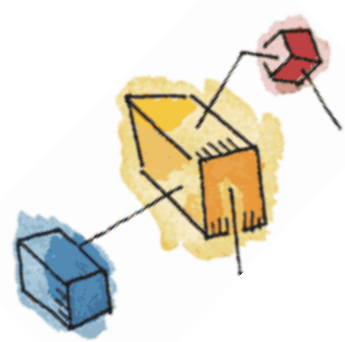


Figure 4.17 Solaris Thread States [MCDO07]

Linux Tasks

- A process, or task, in Linux is represented by a `task_struct` data structure
- This contains a number of categories including:
 - State
 - Scheduling information
 - Identifiers
 - Interprocess communication
 - And others



Linux

Process/Thread Model

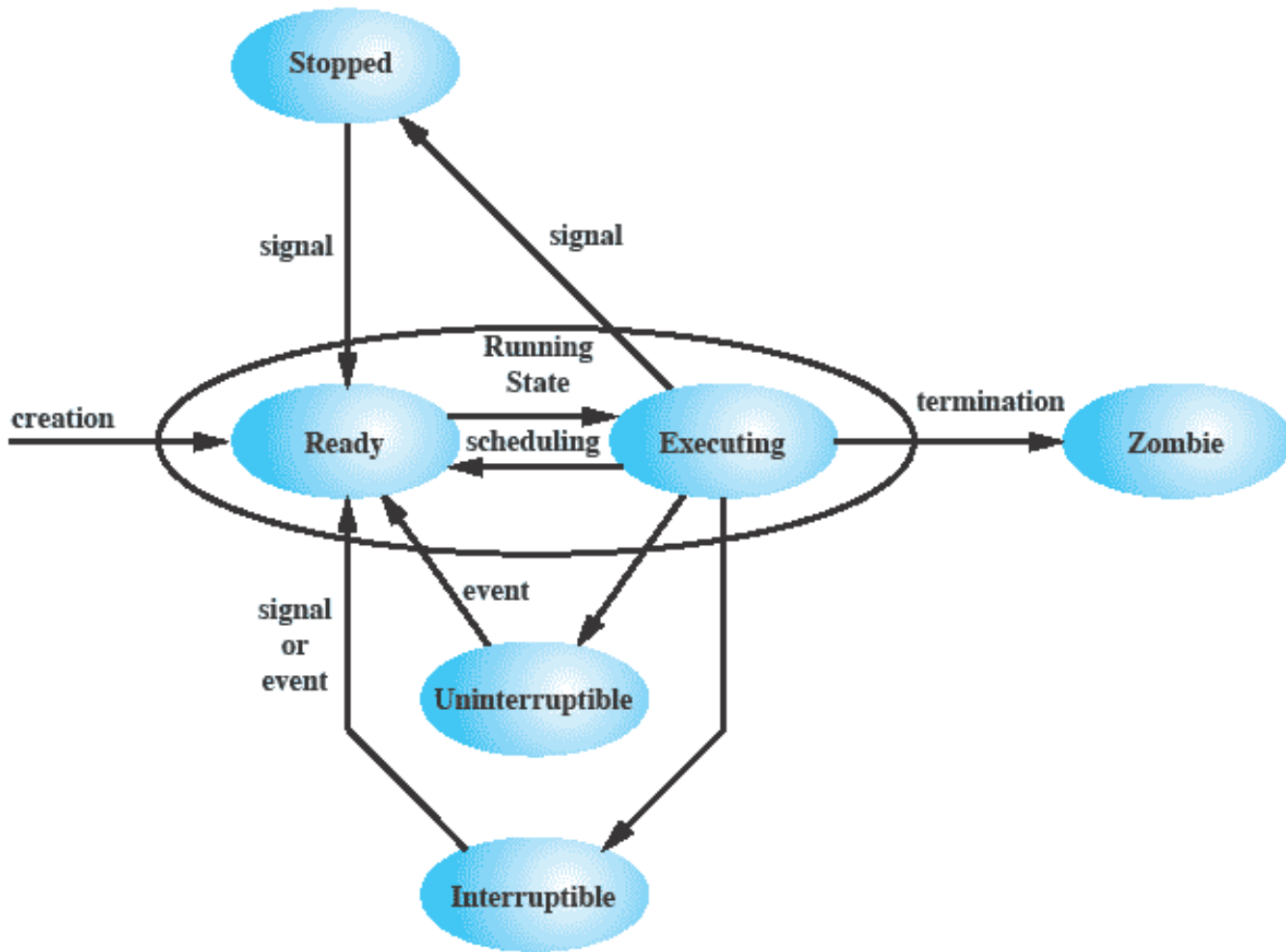


Figure 4.18 Linux Process/Thread Model