Threads, SMP, and Microkernels

Chapter 4

Concepts about Processes so far

- 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
  → how?

Process

- Dispatching is referred to as a thread or lightweight process
- Resource of ownership is referred to as a process or task

Multithreading

- Operating system supports multiple threads of execution within a single process
- MS-DOS supports a single thread
- UNIX supports multiple user processes but only supports one thread per process
- Windows, Solaris, Linux, Mach, and OS/2 support multiple threads

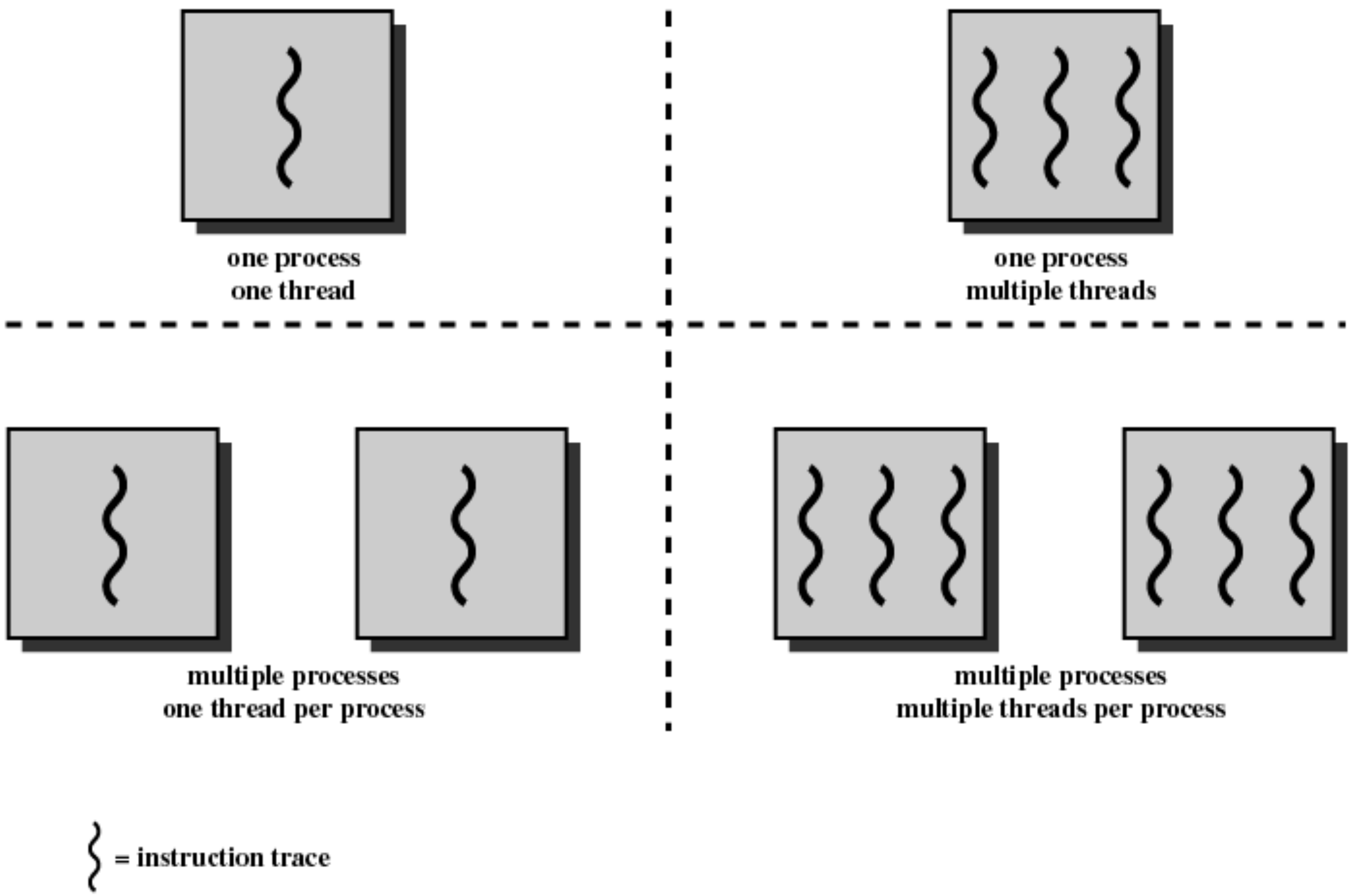


Figure 4.1 Threads and Processes [ANDE97]

Processes vs Threads

- Processes:
  - Have a virtual address space which holds the process image
  - Protected access to processors, other processes, files, and I/O resources

Processes vs Thread

- Threads:
  - An execution state (running, ready, etc.)
  - Saved thread context when not running
  - Has 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

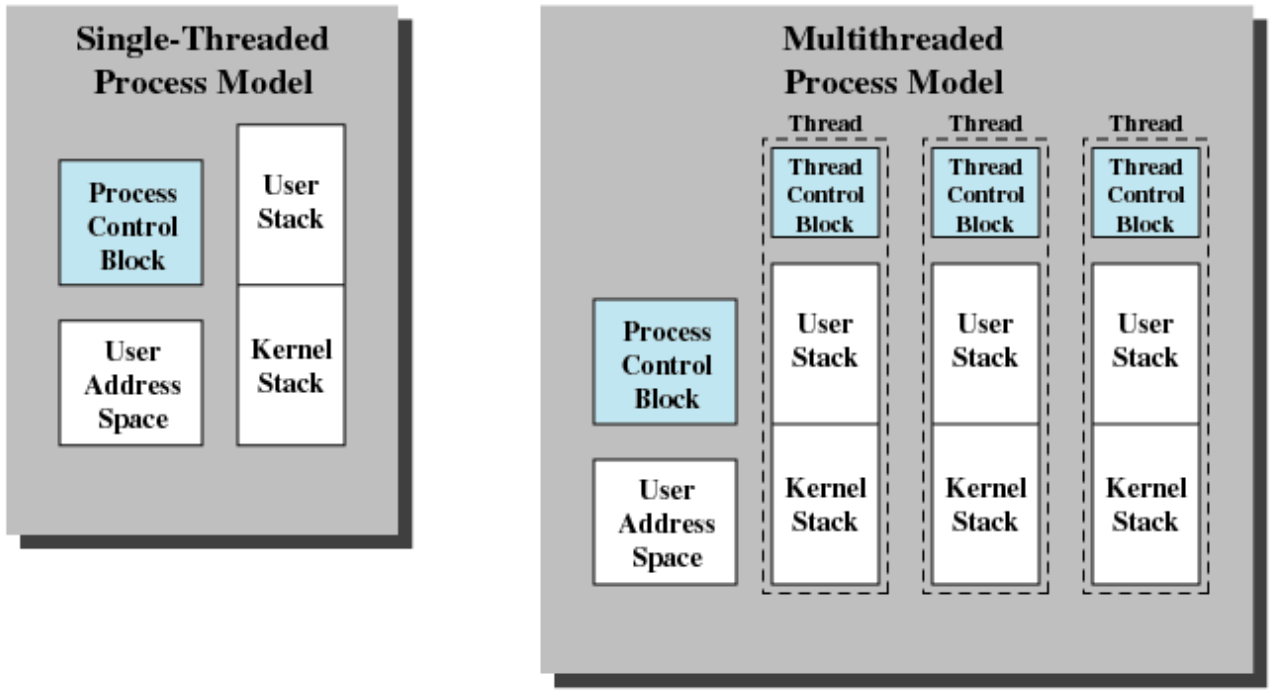


Figure 4.2 Single Threaded and Multithreaded Process Models

Benefits of Threads

- Takes less time to create a new thread than a process
  - Can skip resource allocation through the kernel
  - Factor 10 times faster
- Less time to terminate a thread than a process
  - Don't have to release resources through the kernel
- Less time to switch between two threads within the same process
- More efficient communication between multiple execution entities
  - Threads within the same process share memory

Uses of Threads in a Single-User Multiprocessing System

- Foreground to background work
  - T1 handles sampling, <u>T2 background checks</u>, T3 data processing
- Asynchronous processing
  - T1 computes everything, T2 passes through RS232
- Speed of execution
  - Multiple threads on multiple CPUs/cores; I/O blocking doesn't stop the app, only one thread (e.g., printing and Word 6)
- Modular program structure

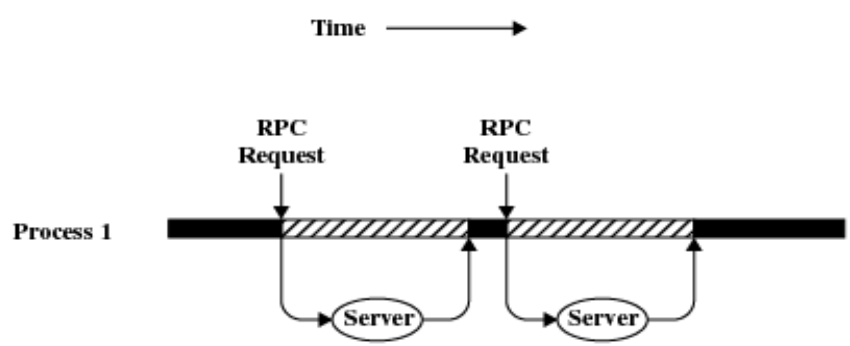
Threads

- Suspending a process involves suspending all threads of the process since all threads share the same address space
- Termination of a process, terminates all threads within the process

Thread States

- States associated with a change in thread state
  - Spawn
    - Spawn another thread
  - Block
  - Unblock
  - Finish
    - Deallocate register context and stacks

Remote Procedure Call Using Single Thread



(a) RPC Using Single Thread

Remote Procedure Call Using Threads

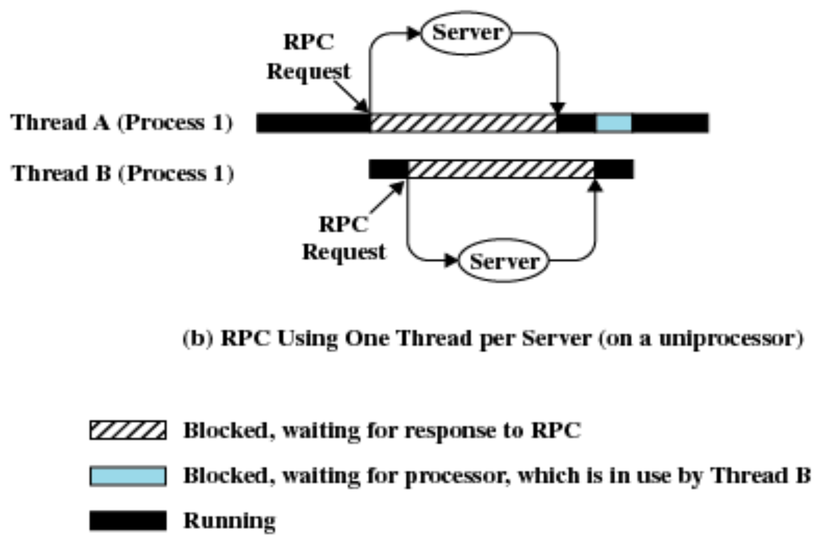


Figure 4.3 Remote Procedure Call (RPC) Using Threads

Assuming independence of the function calls!

Multithreading

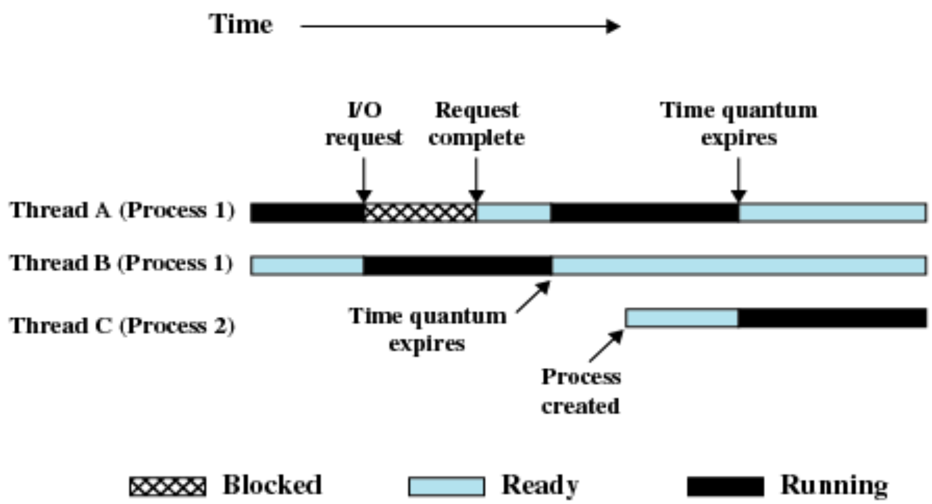


Figure 4.4 Multithreading Example on a Uniprocessor

Adobe PageMaker

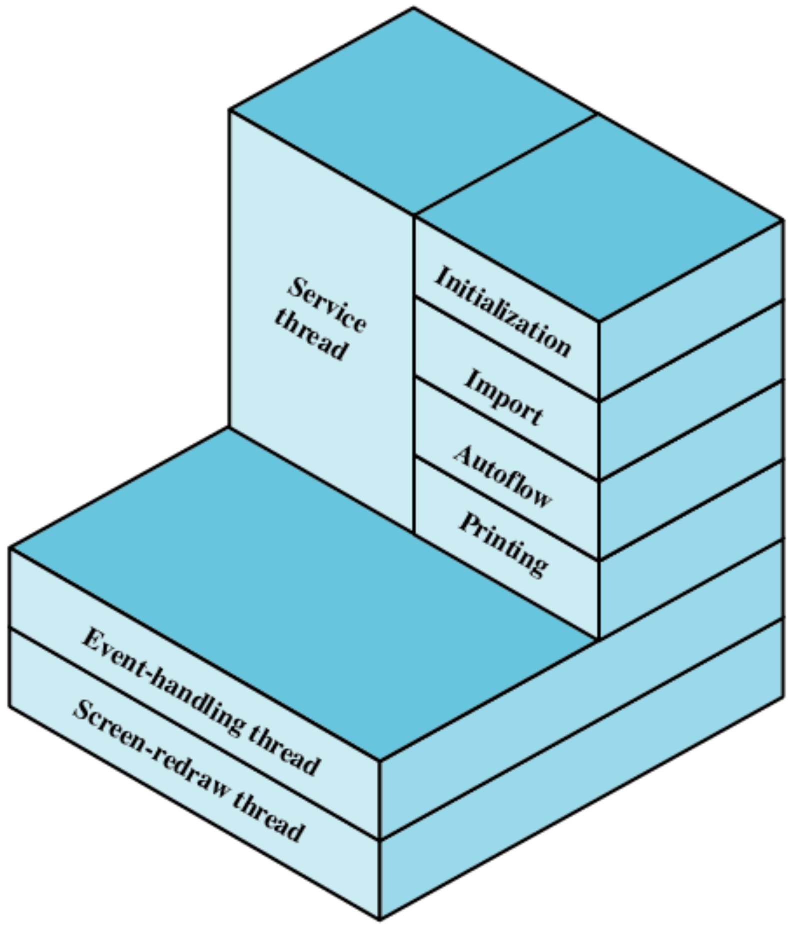
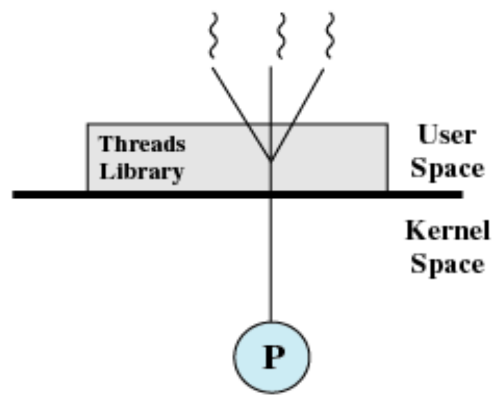


Figure 4.5 Thread Structure for Adobe PageMaker

User-Level Threads

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

User-Level Threads



- Many threads in the application.
- Application schedules execution.

- One process in the kernel.
- Kernel schedules processes.

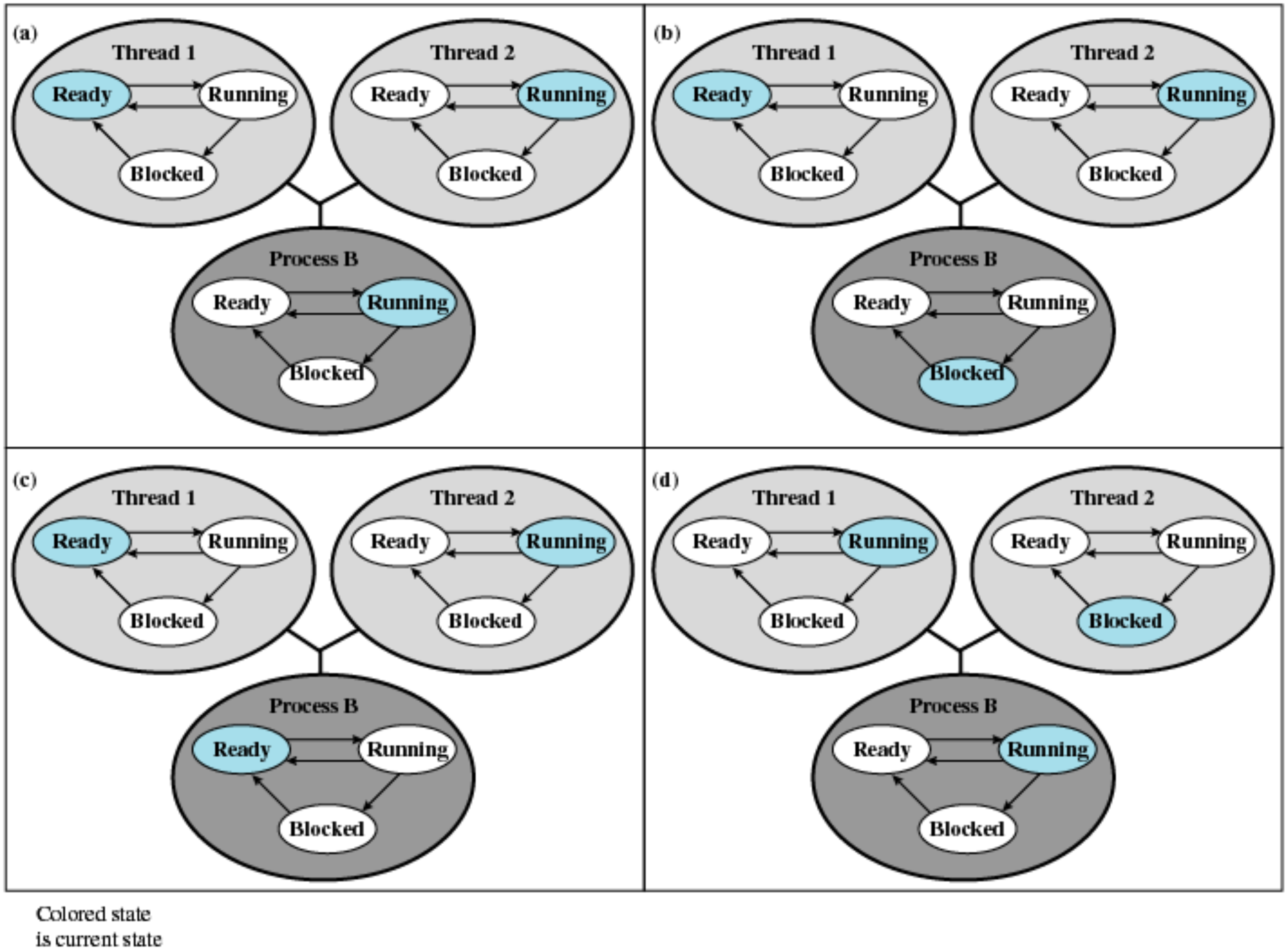


Figure 4.7 Examples of the Relationships Between User-Level Thread States and Process States

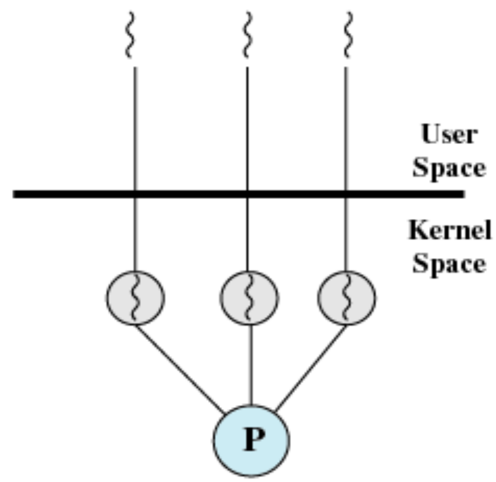
T2 calls a kernel I/O function; kernel blocks the process.

T2 waits for something other than kernel.

Kernel-Level Threads

- Windows is an example of this approach
- Kernel maintains context information for the process and the threads
- Scheduling is done on a thread basis

Kernel-Level Threads



(b) Pure kernel-level

- Application consists of sets of threads.
- Kernel knows processes & threads.
- Kernel schedules processes & threads.

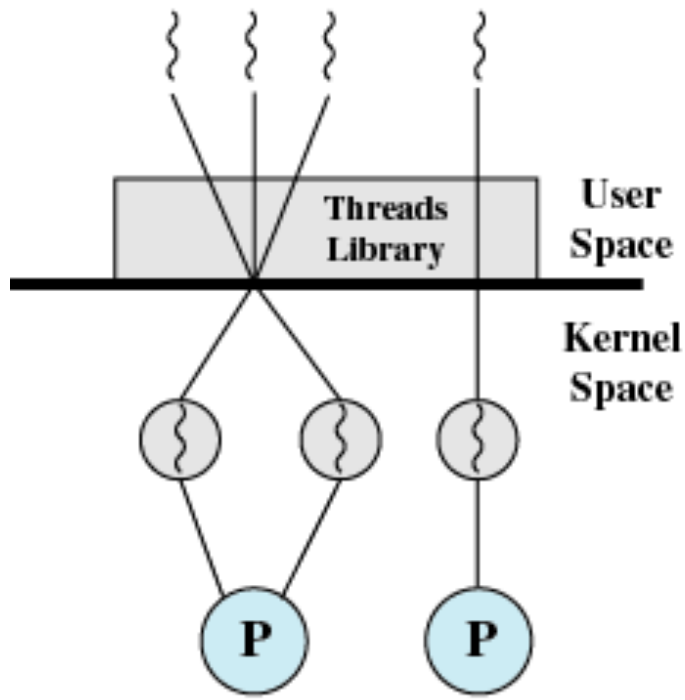
VAX Running UNIX-Like Operating System

Table 4.1 Thread and Process Operation Latencies (μs) [ANDE92]

Operation	User-Level Threads	Kernel-Level Threads	Processes
Null Fork	34	948	11,300
Signal Wait	37	441	1,840

Null Fork measures the system overhead of the fork() call.
Signal Wait measures the system overhead of the signal() call.

Combined Approaches



(c) Combined

- Applications consists of multiple threads.
- Threads can be grouped to kernel threads.

- Kernel knows processes & threads.
- Kernel schedules processes & threads.

Example: IO processes.

Relationship Between Threads and Processes

Table 4.2 Relationship Between Threads and Processes

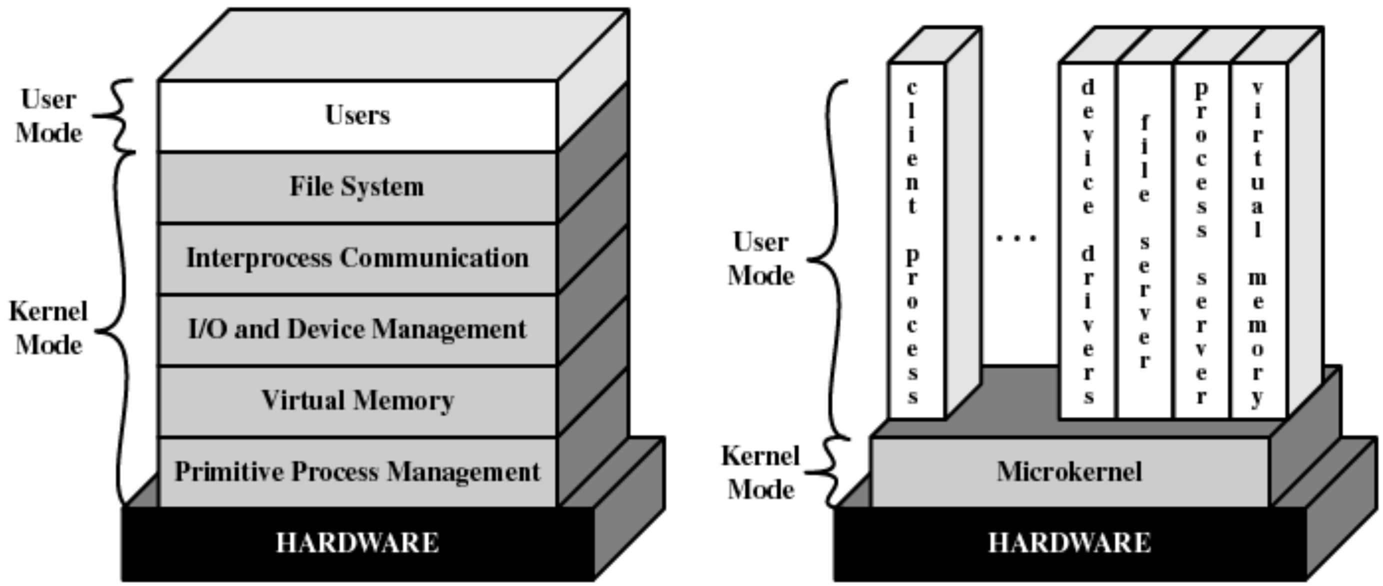
Threads:Processes	Description	Example Systems
1:1	Each thread of execution is a unique process with its own address space and resources.	Traditional UNIX implementations
M:1	A process defines an address space and dynamic resource ownership. Multiple threads may be created and executed within that process.	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

Microkernels

- Small operating system core
- Contains only essential core operating systems functions
- Many services traditionally included in the operating system are now external subsystems
  - Device drivers
  - File systems
  - Virtual memory manager
  - Windowing system
  - Security services

Instead of …

Use this …



(a) Layered kernel

(b) Microkernel

Figure 4.10 Kernel Architecture

Benefits of a Microkernel Organization

- Uniform interface on request made by a process
  - Don't distinguish between kernel-level and user-level services
  - All services are provided by means of message passing
- Extensibility
  - Allows the addition of new services
- Flexibility
  - New features added
  - Existing features can be subtracted

Benefits of a Microkernel Organization

- Portability
  - Changes needed to port the system to a new processor is changed in the microkernel - not in the other services
- Reliability
  - Modular design
  - Small microkernel can be rigorously tested

Benefits of Microkernel Organization

- Distributed system support
  - Message are sent without knowing what the target machine is
- Object-oriented operating system
  - Components are objects with clearly defined interfaces that can be interconnected to form software

Microkernel Design

- Low-level memory management
  - Mapping each virtual page to a physical page frame

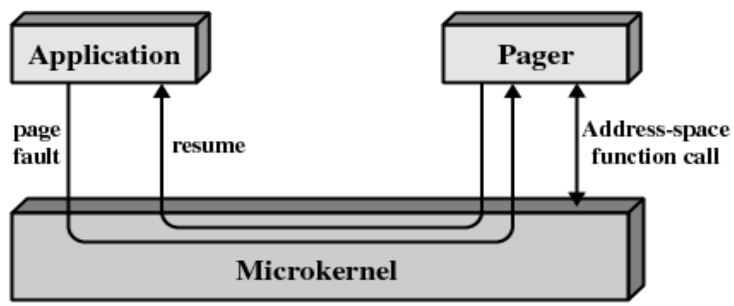


Figure 4.11 Page Fault Processing

Grant/map/flush calls

Microkernel Design

- Interprocess communication
  - Concepts: messages, ports, capabilities
  - Microkernel requires copying of messages; remapping pages may be faster
- I/O and interrupt management
  - Interrupts are messages sent to processes
  - Microkernel doesn't need to know anything about the IRQ handling function

Subsequent slides are for private study

Windows Processes

- Implemented as objects
- An executable process may contain one or more threads
- Both processes and thread objects have built-in synchronization capabilities

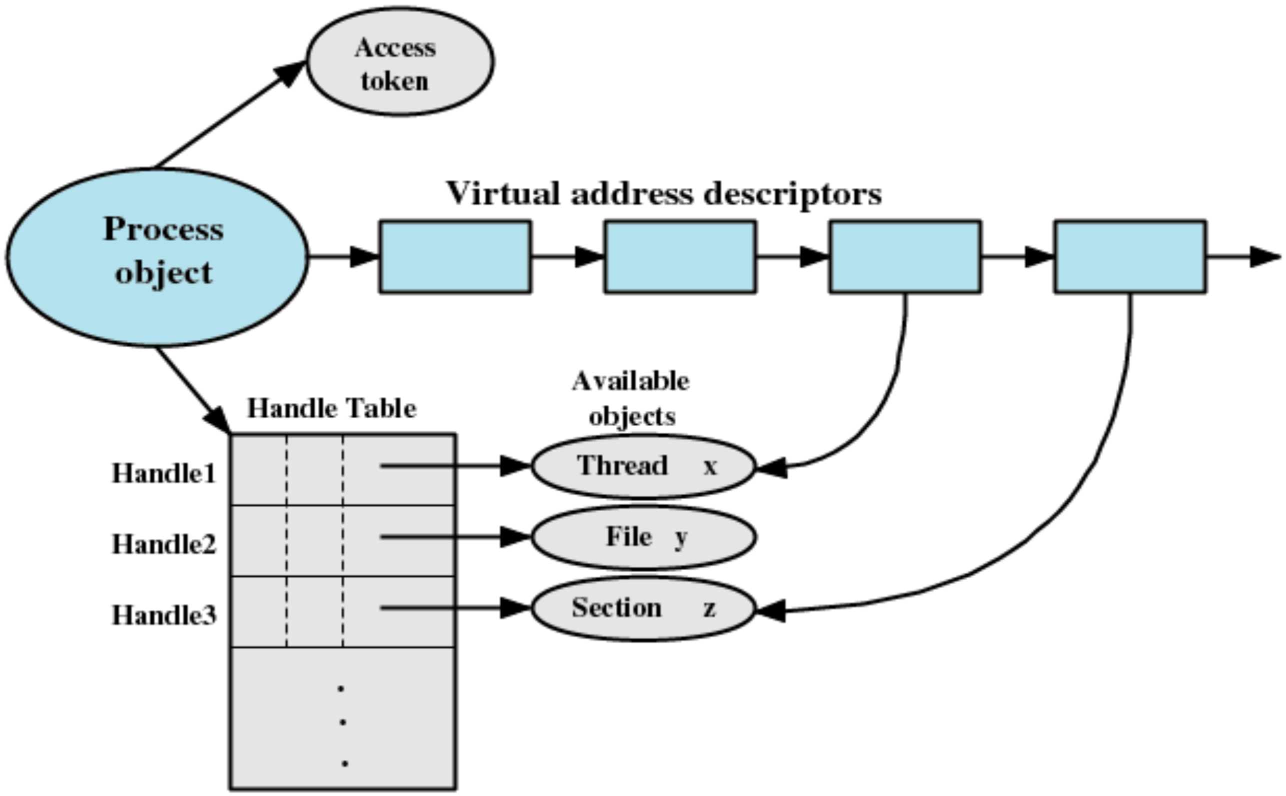
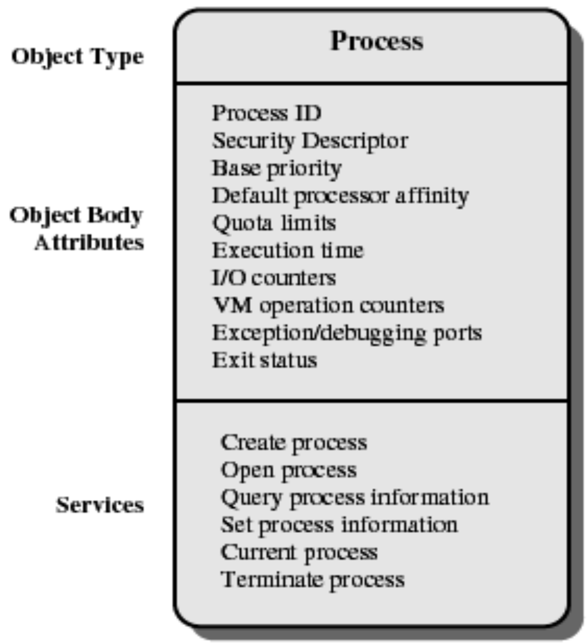


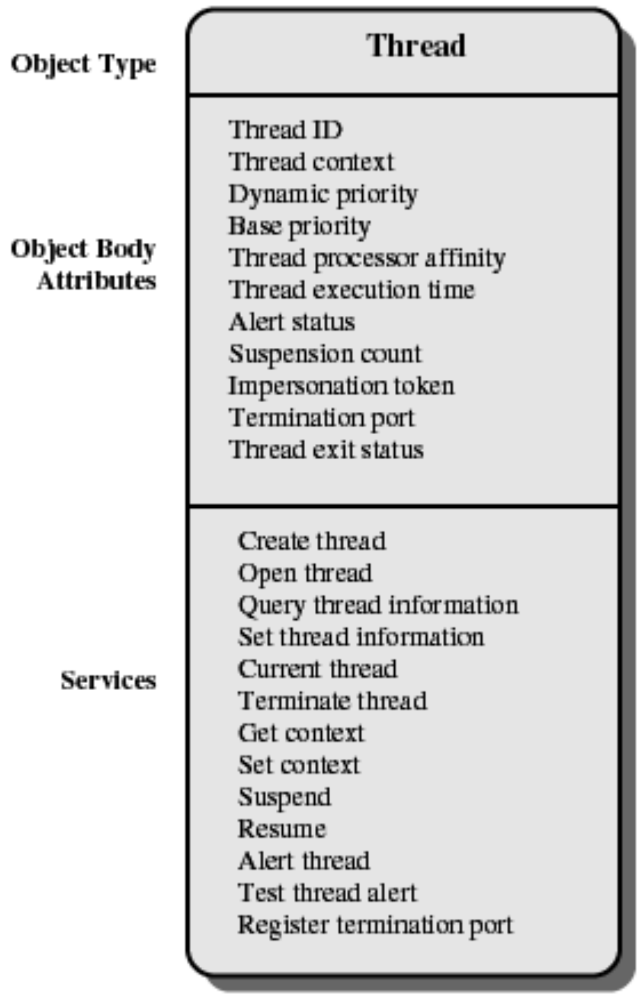
Figure 4.12 A Windows Process and Its Resources

Windows Process Object



(a) Process object

Windows Thread Object



(b) Thread object

Windows 2000 Thread States

- Ready
- Standby
- Running
- Waiting
- Transition
- Terminated

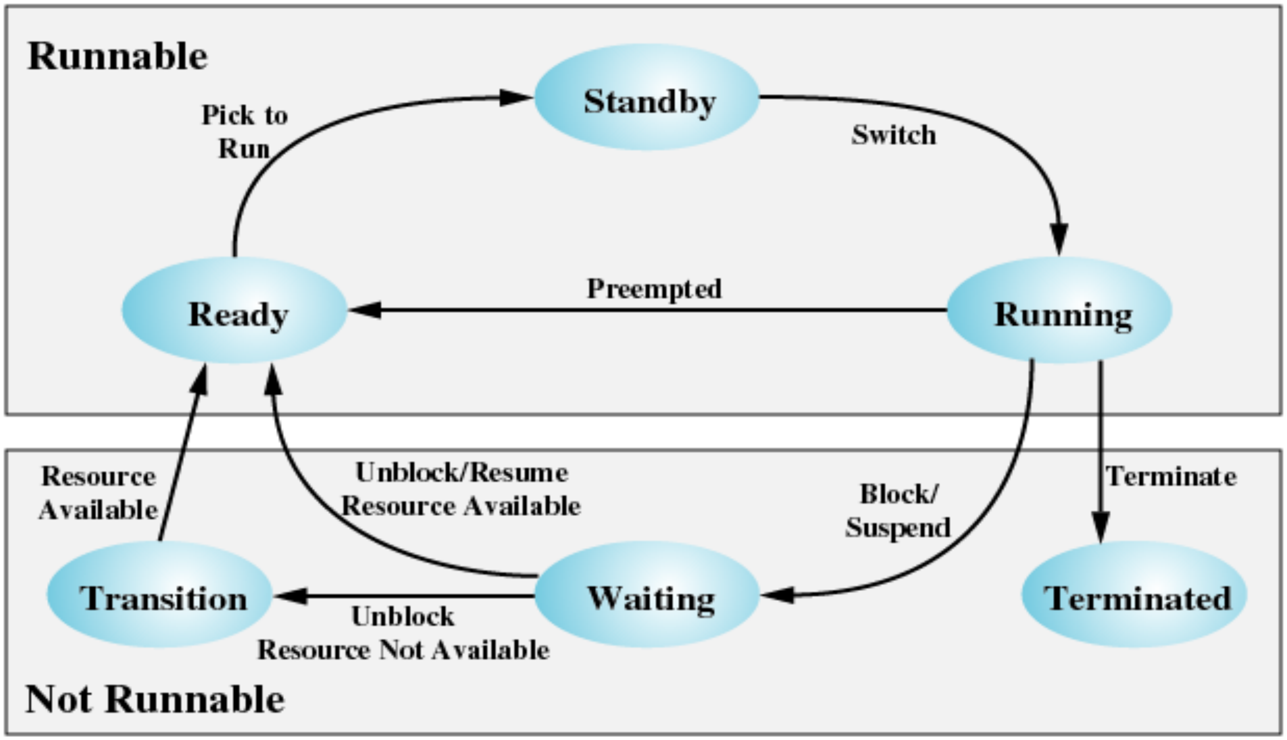


Figure 4.14 Windows Thread States

Solaris

- Process includes the user’s address space, stack, and process control block
- User-level threads
- Lightweight processes (LWP)
- Kernel threads

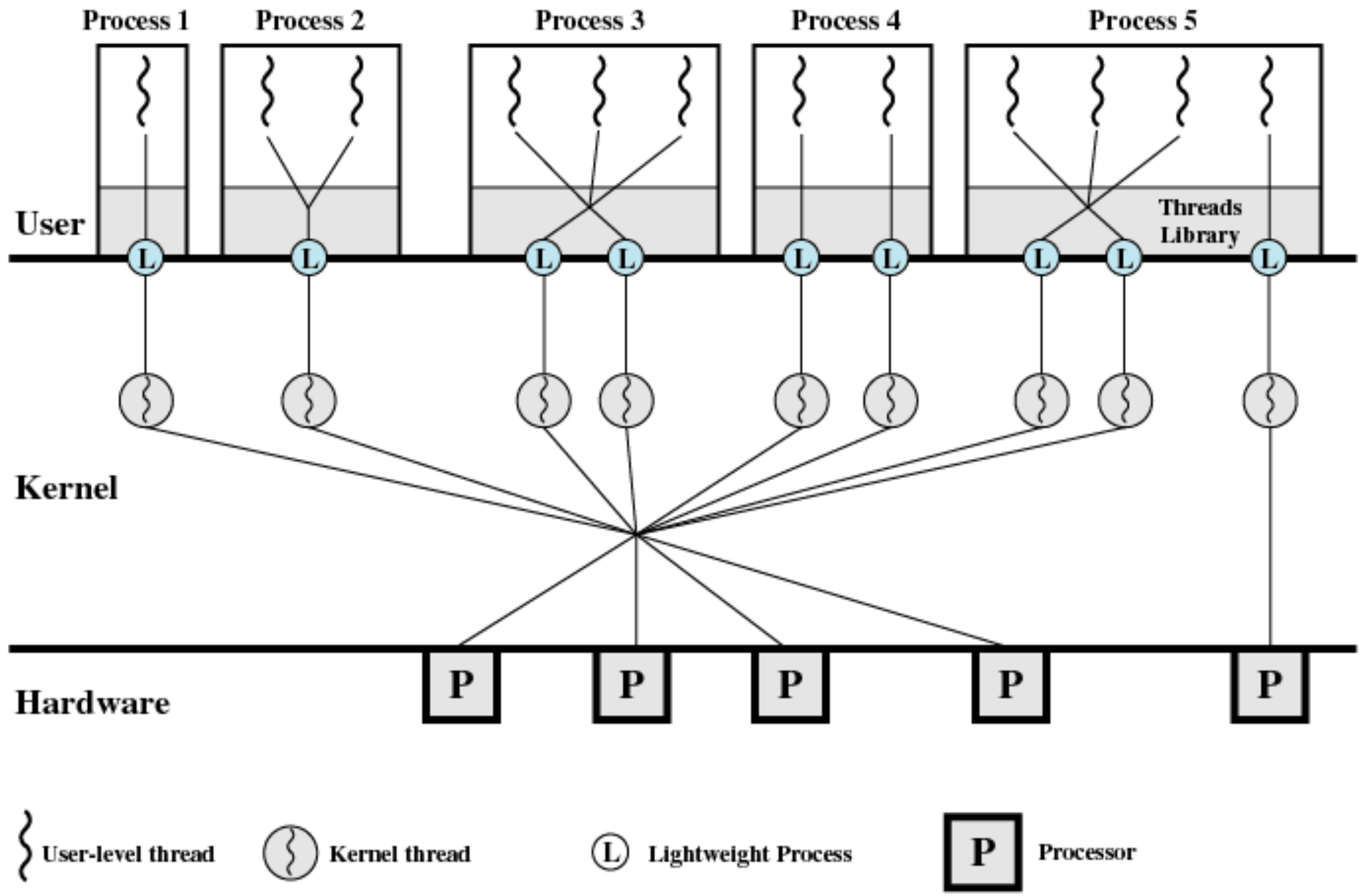


Figure 4.15 Solaris Multithreaded Architecture Example

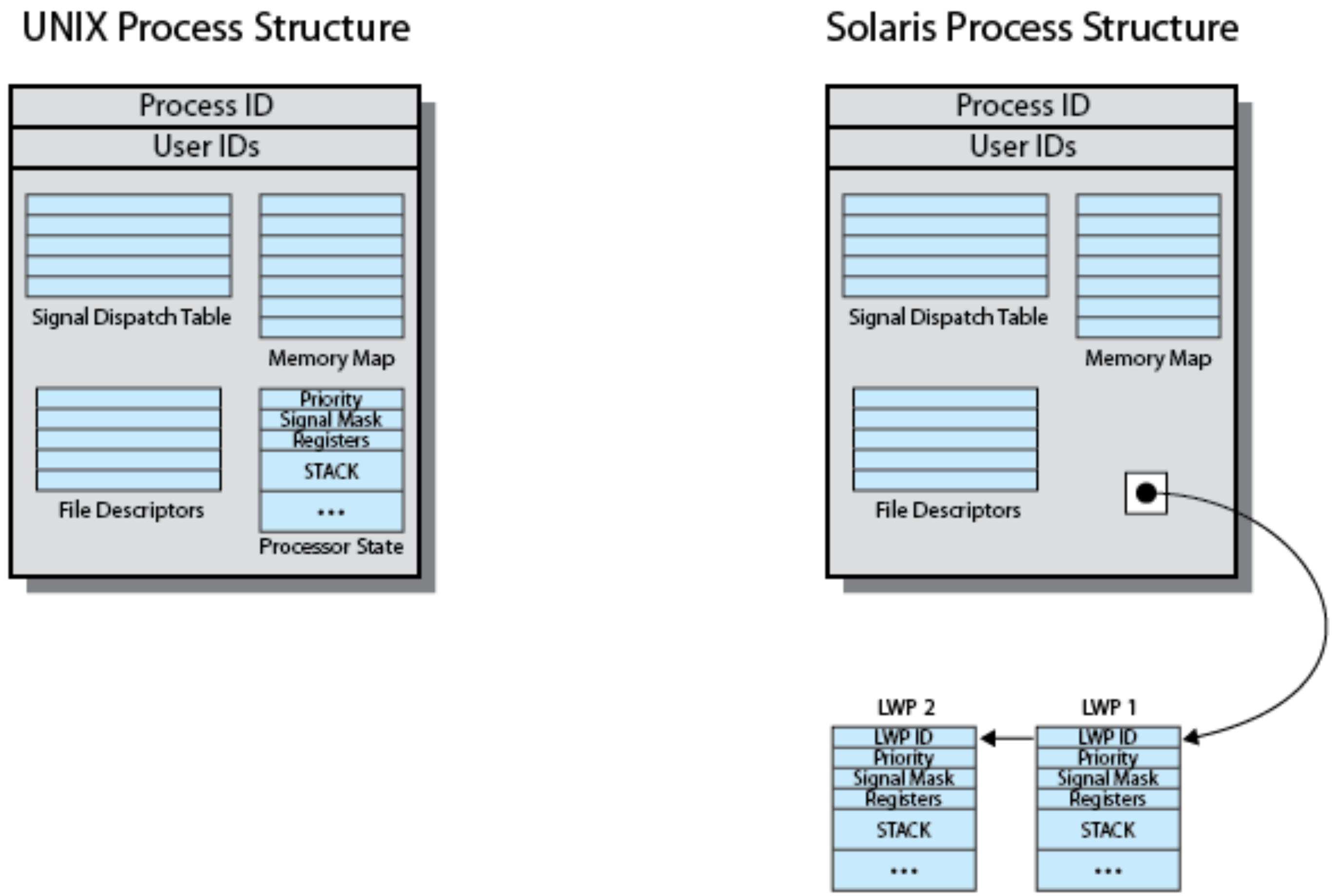


Figure 4.16 Process Structure in Traditional UNIX and Solaris [LEWI96]

Solaris Lightweight Data Structure

- Identifier
- Priority
- Signal mask
- Saved values of user-level registers
- Kernel stack
- Resource usage and profiling data
- Pointer to the corresponding kernel thread
- Pointer to the process structure

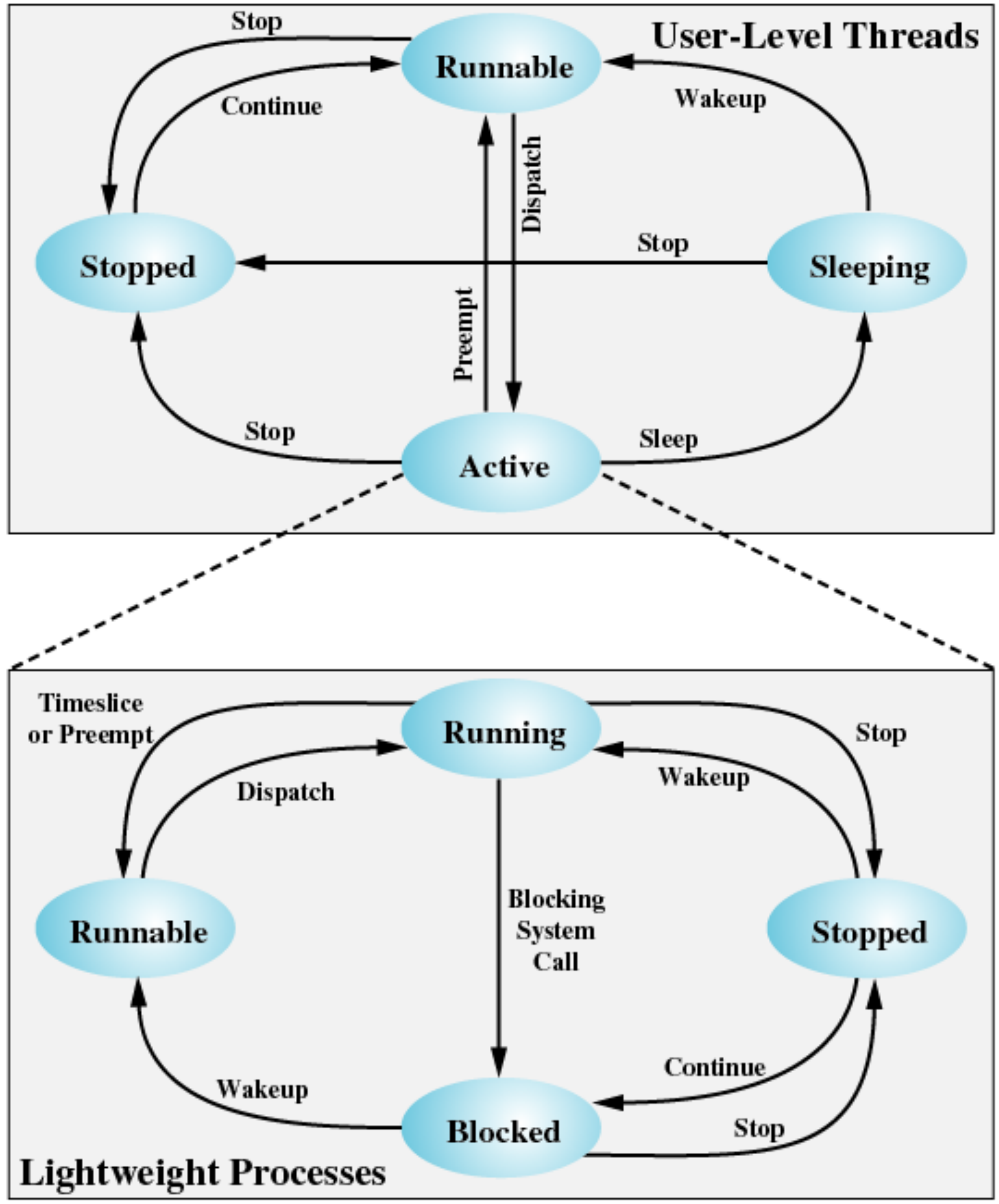


Figure 4.17 Solaris User-Level Thread and LWP States

Linux Task Data Structure

- State
- Scheduling information
- Identifiers
- Interprocess communication
- Links
- Times and timers
- File system
- Address space
- Processor-specific context

Linux States of a Process

- Running
- Interruptable
- Uninterruptable
- Stopped
- Zombie

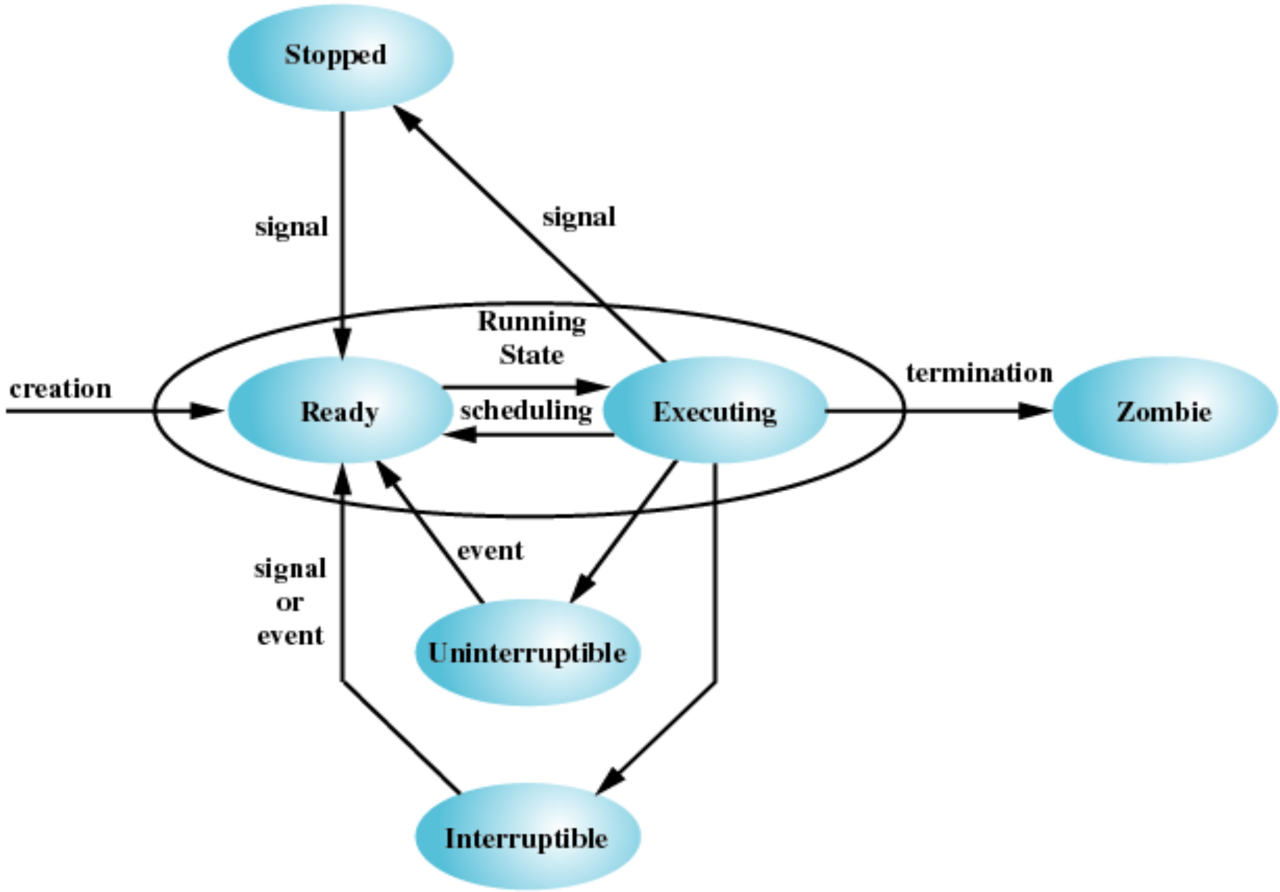


Figure 4.18 Linux Process/Thread Model