Operating Systems: Internals and Design Principles

Chapter 4
Threads

Eighth Edition

By William Stallings

4.1 Processes and Threads

Resource Ownership

Process includes a virtual address space to hold the process image

- the OS performs a protection function 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

Processes and Threads

- The two characteristics are independent and could be treated independently by the OS
- 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

Single Threaded Approaches

- A single thread of execution per process, in which the concept of a thread is not recognized, is referred to as a single-threaded approach
- MS-DOS is an example

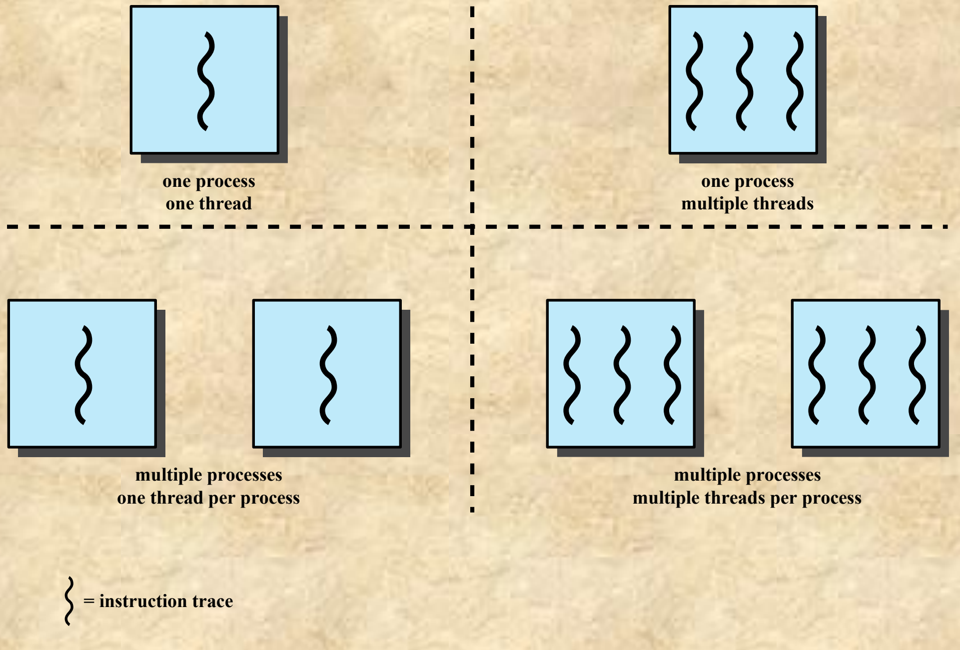


Figure 4.1 Threads and Processes

Multithreaded Approaches

- The right half of Figure 4.1 depicts multithreaded approaches
- A Java run-time environment is an example of a system of one process with multiple threads
- Windows, Solaris, and many modern versions of UNIX support the use of multiple processes and multiple threads

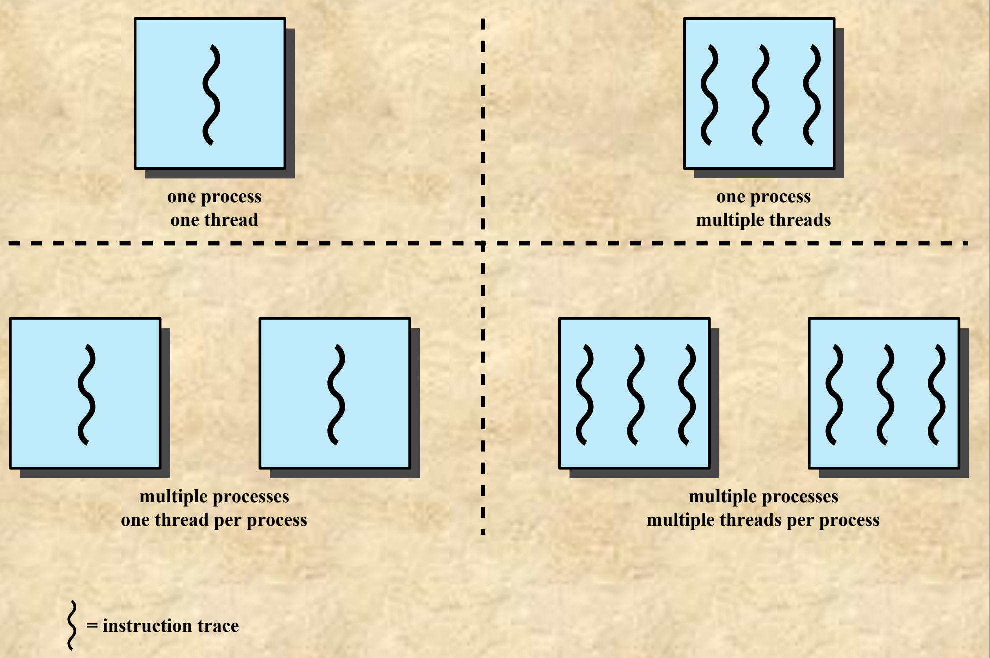
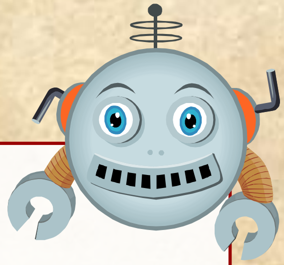


Figure 4.1 Threads and Processes

Processes

- The unit of resource allocation and a unit of protection
- Associated with processes are
  - A virtual address space that holds the process image
  - Protected access to:
    - processors
    - other processes (for IPC)
    - 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
- 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)

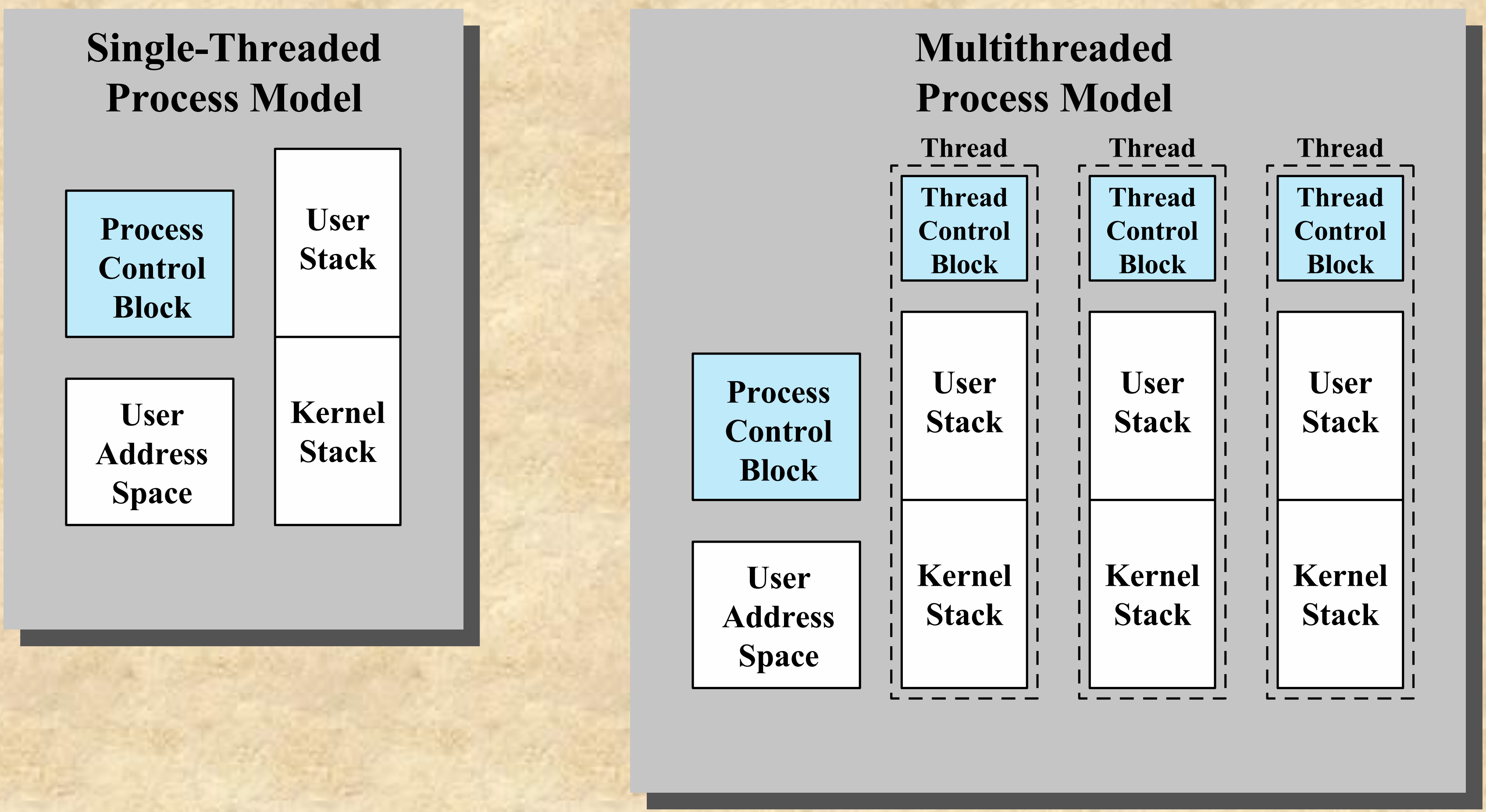


Figure 4.2 Single Threaded and Multithreaded Process Models

Benefits of Threads

- Takes less time to create a new thread than a process
- 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 threads 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
- Block
- Unblock
- Finish

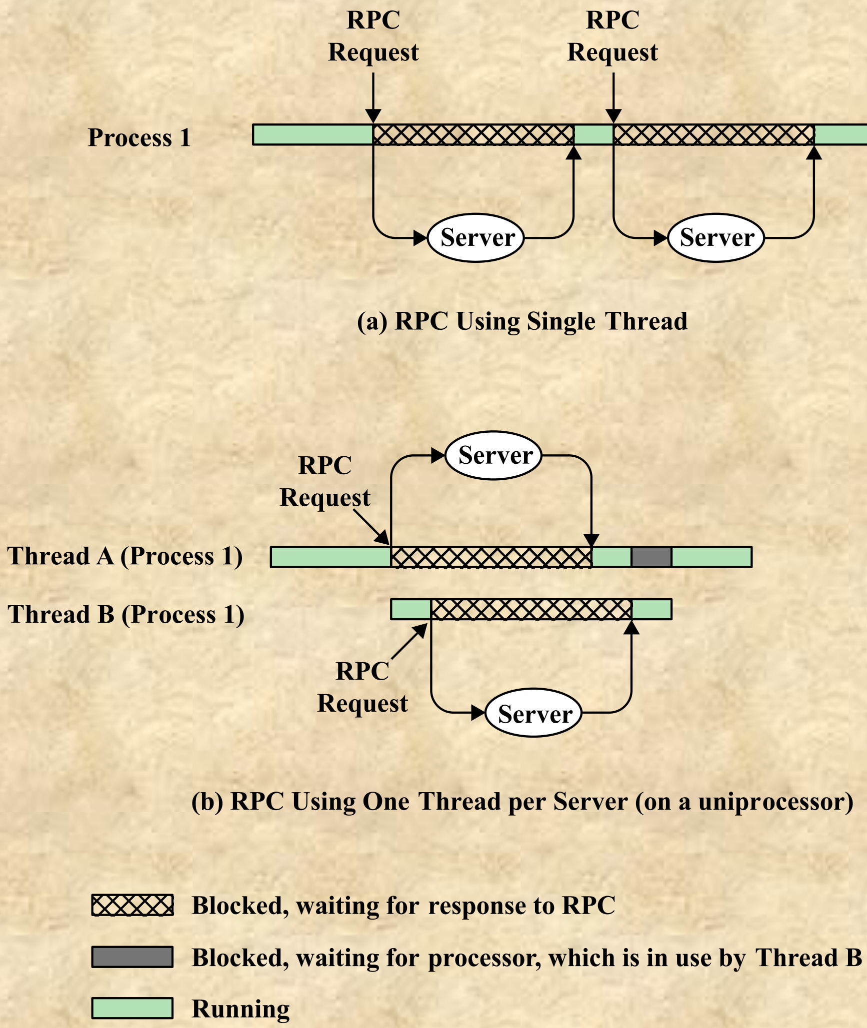


Figure 4.3 Remote Procedure Call (RPC) Using Threads

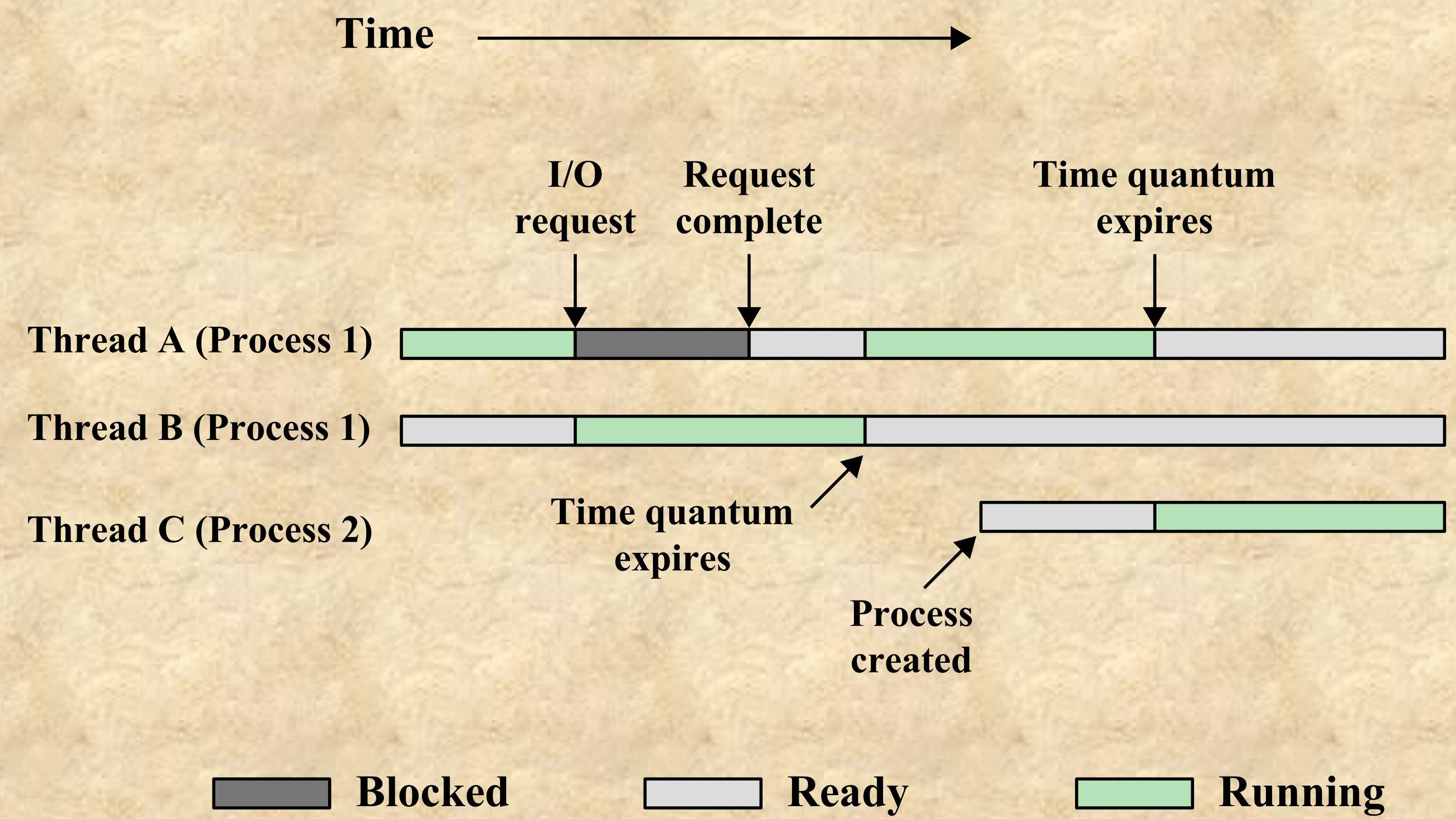


Figure 4.4 Multithreading Example on a Uniprocessor

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

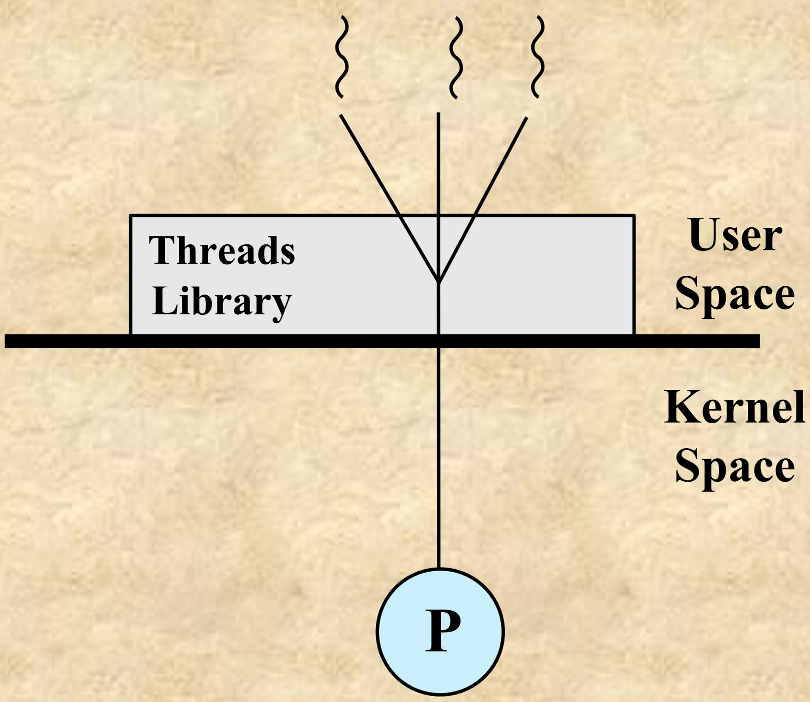
4.2 Types of Threads

User Level Thread (ULT)

Kernel level Thread (KLT)

User-Level Threads (ULTs)

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



(a) Pure user-level

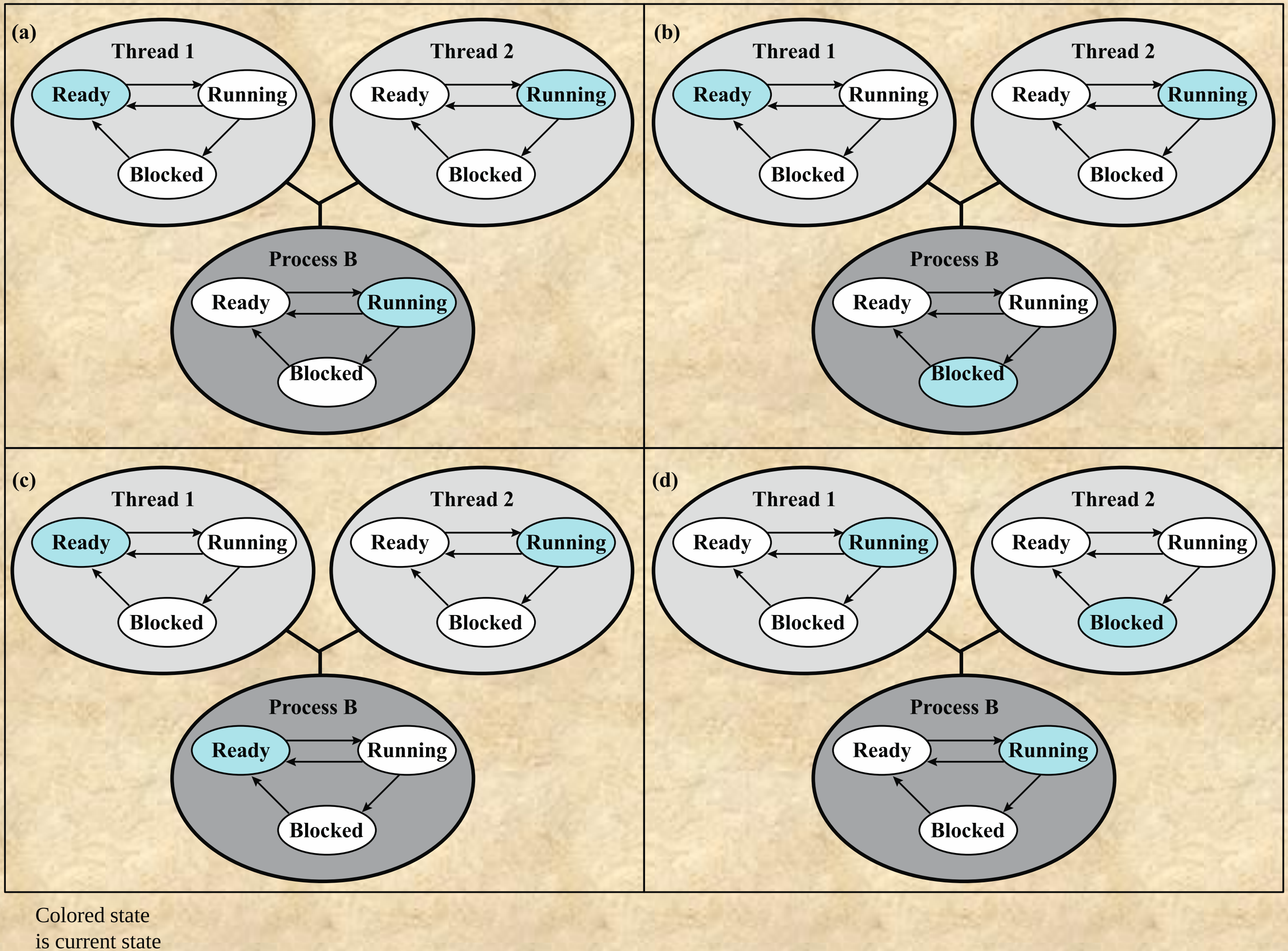
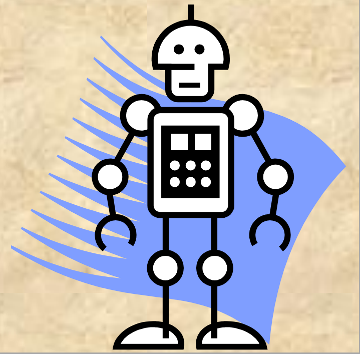


Figure 4.6 Examples of the Relationships Between User-Level Thread States and Process States

Advantages of ULTs

- Thread switching does not require kernel mode privileges
- Scheduling can be application specific
- ULTs can run on any OS



Disadvantages of ULTs

- In a typical OS, many system calls are blocking
  - as a result, when a ULT executes a system call, not only is that thread blocked, but all of the threads within the process are blocked
- In a pure ULT strategy, a multithreaded application cannot take advantage of multiprocessing

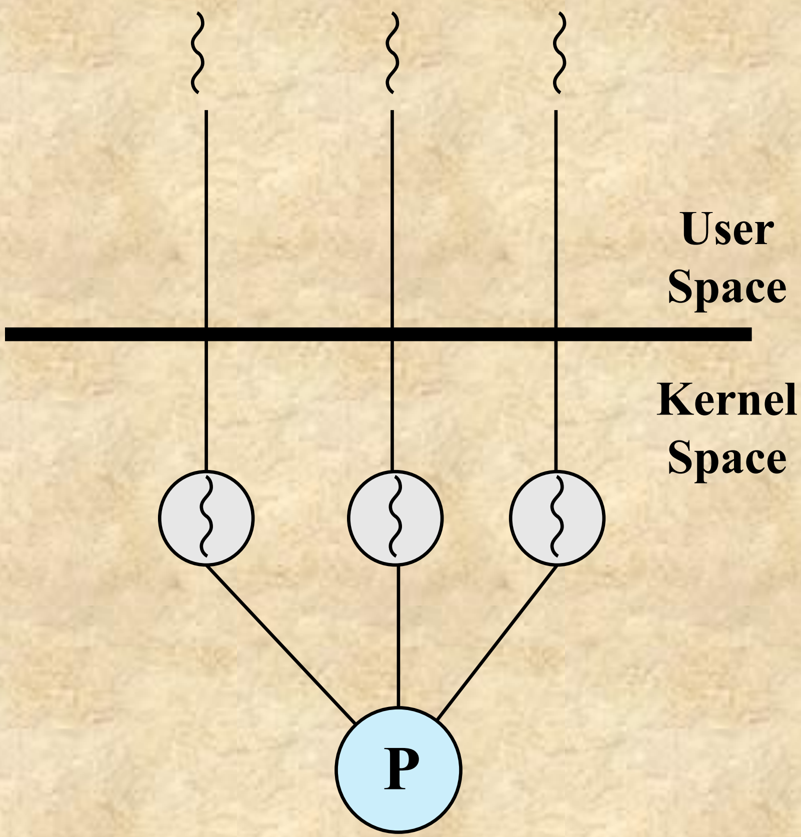
Overcoming ULT Disadvantages

Jacketing

- converts a blocking system call into a non-blocking system call

Writing an application as multiple processes rather than multiple threads

Kernel-Level Threads (KLTs)

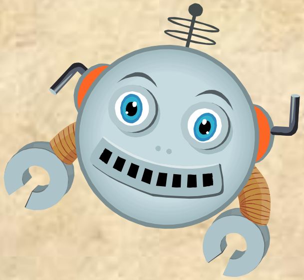


(b) Pure kernel-level

- Thread management is done by the kernel
  - no thread management is done by the application
  - Windows is an example of this approach

Advantages of KLTs

- 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 can be multithreaded



Disadvantage of KLTs

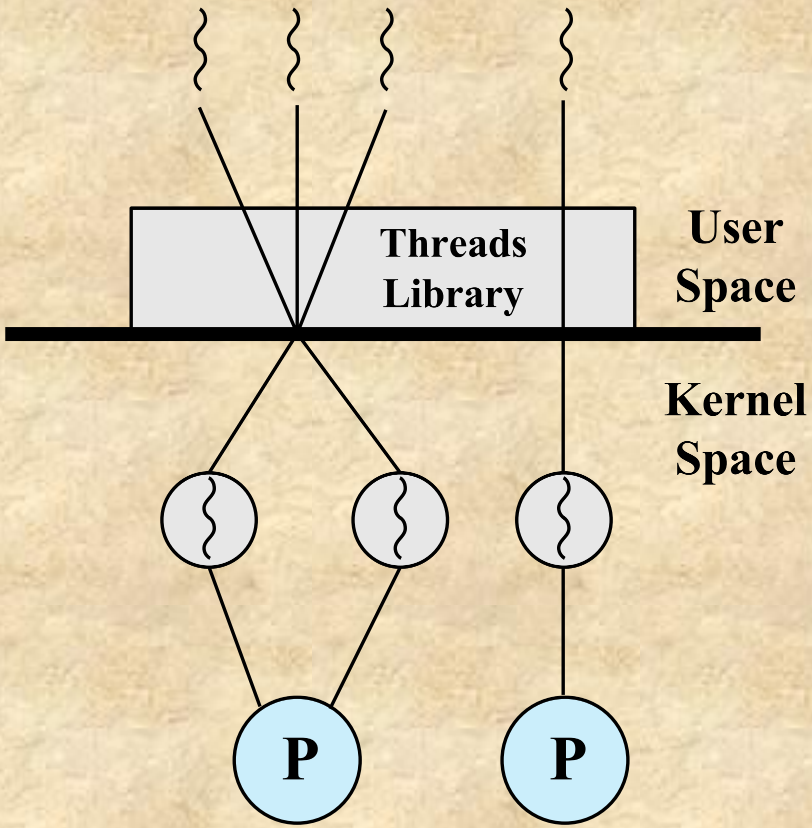
- The transfer of control from one thread to another within the same process requires a mode switch to the kernel

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

Table 4.1
Thread and Process Operation Latencies (μs)

Combined Approaches

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



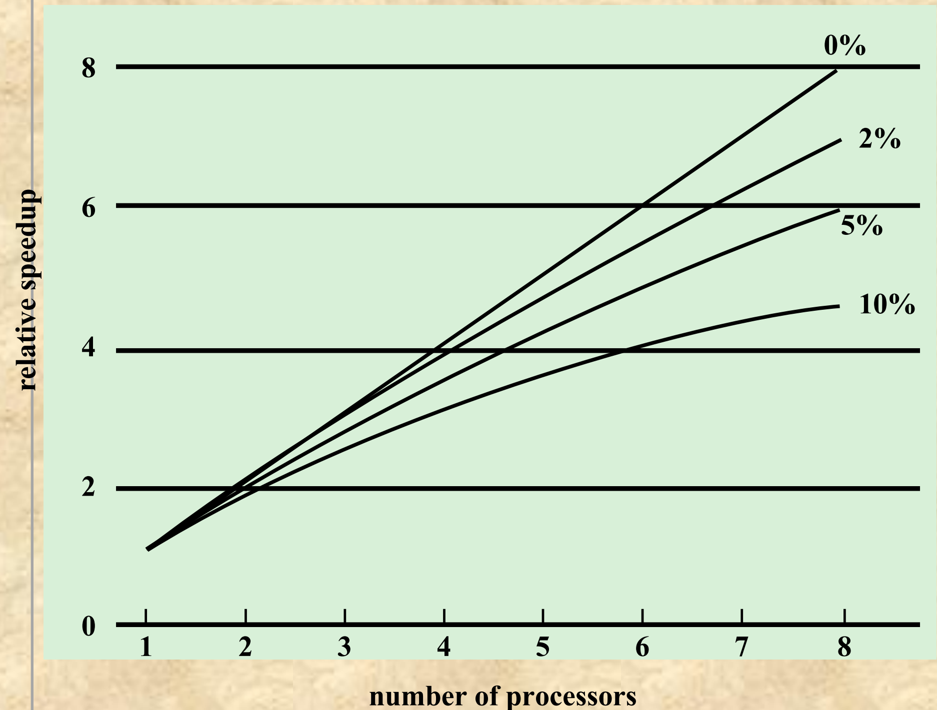
(c) Combined

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

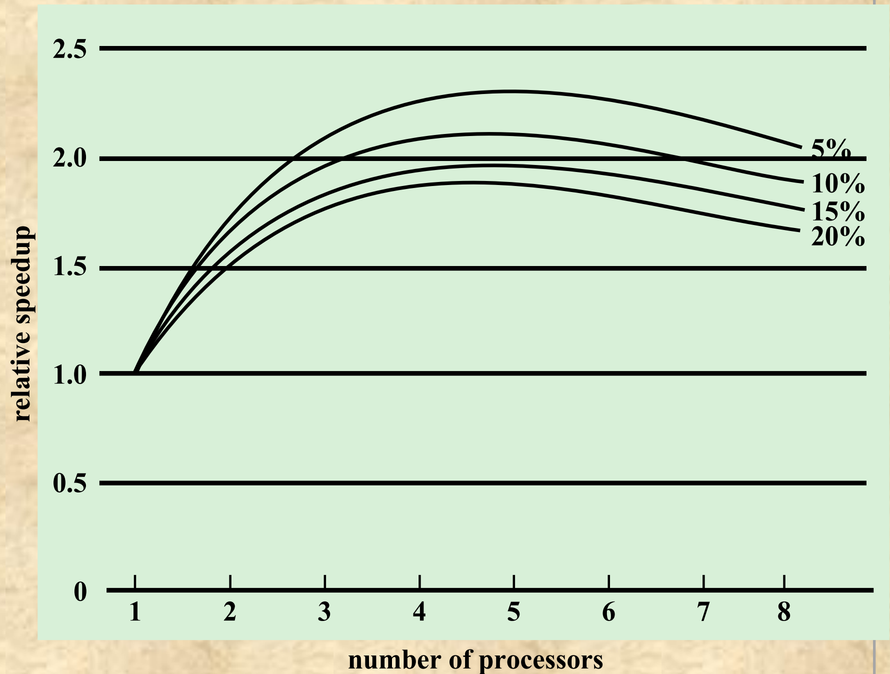
Table 4.2
Relationship between Threads and Processes

4.3 Multicore & Multithreading

- Speedup = $\frac{\textit{time to execute program on a single processor}}{\textit{time to execute program on N parallel processors}} = \frac{1}{(1-f)+\frac{f}{N}}$



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



(b) Speedup with overheads

Figure 4.7 Performance Effect of Multiple Cores

Applications That Benefit

- Multithreaded native applications
  - characterized by having a small number of highly threaded processes
- Multiprocess applications
  - characterized by the presence of many single-threaded processes
- Java applications
- Multiinstance applications
  - multiple instances of the application in parallel

4.4 Windows 8 Process and Thread Management

- An **application** consists of one or more processes
- Each **process** provides the resources needed to execute a program
- A **thread** is the entity within a process that can be scheduled for execution
- A **job object** allows groups of processes to be managed as a unit
- A **thread pool** is a collection of worker threads that efficiently execute asynchronous callbacks on behalf of the application
- A **fiber** is a unit of execution that must be manually scheduled by the application
- **User-mode scheduling (UMS)** is a lightweight mechanism that applications can use to schedule their own threads

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

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. Normally, these are connected to environment subsystem and debugger processes, respectively.
Exit status	The reason for a process's termination.

Table 4.3

Windows Process Object Attributes

(Table is on page 175 in textbook)

Table 4.4

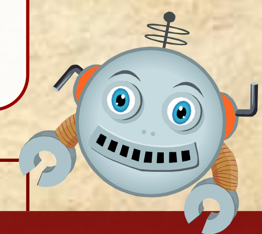
Windows Thread Object Attributes

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 a waiting thread may 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.

(Table is on page 175 in textbook)

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

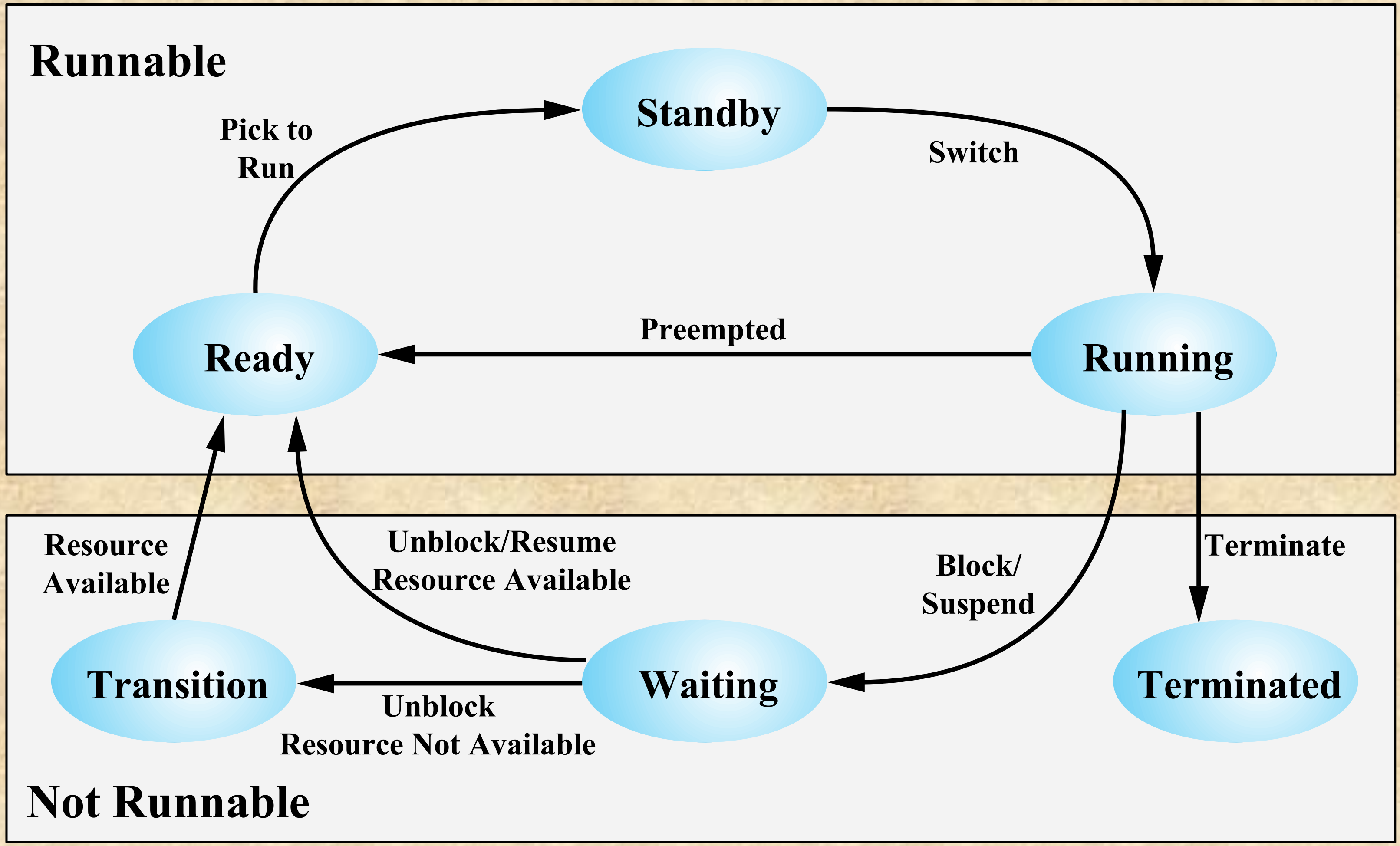


Figure 4.11 Windows Thread States

4.5 Solaris Process

- makes use of four thread-related concepts:

Process	• includes the user's address space, stack, and process control block
User-level Threads	• is a user-created unit of execution within a process • invisible to the OS
Lightweight Processes (LWP)	• can be viewed as a mapping between ULTs and kernel threads • support ULT and map to one kernel thread
Kernel Threads	• fundamental entities that can be scheduled and dispatched to run on one of the system processors

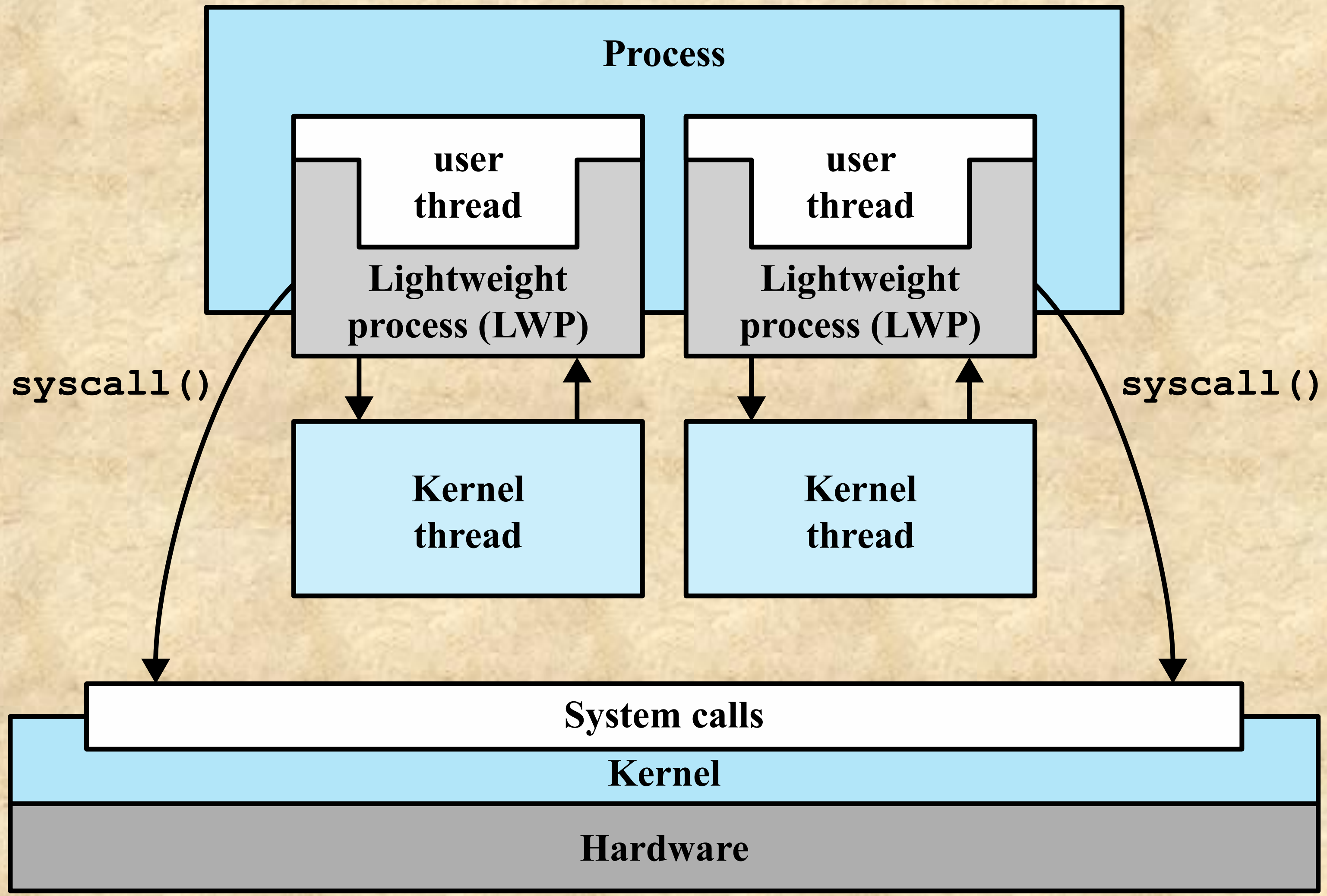


Figure 4.12 Processes and Threads in Solaris

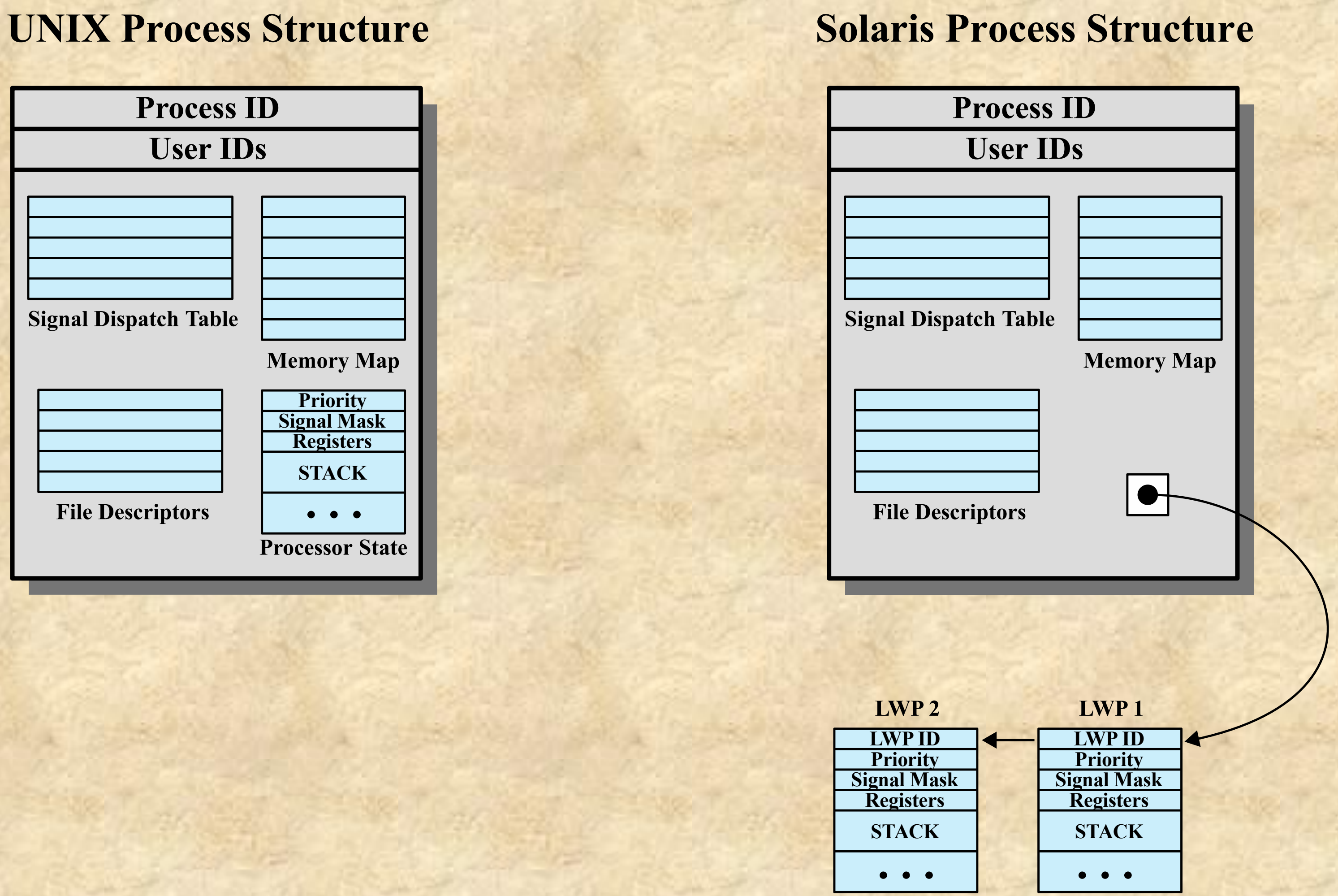


Figure 4.13 Process Structure in Traditional UNIX and Solaris [LEWI96]

A Lightweight Process (LWP) Data Structure Includes:

- 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

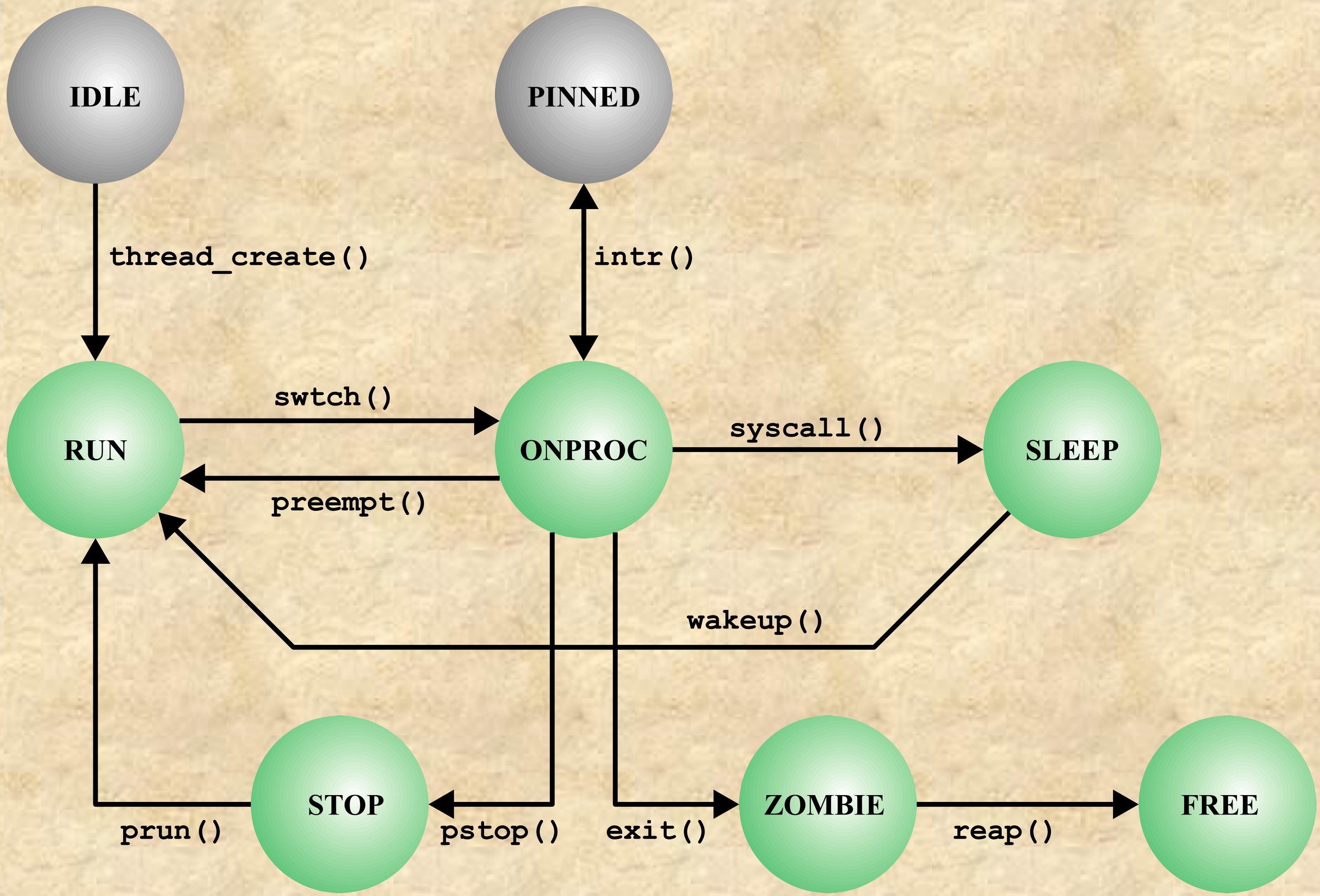
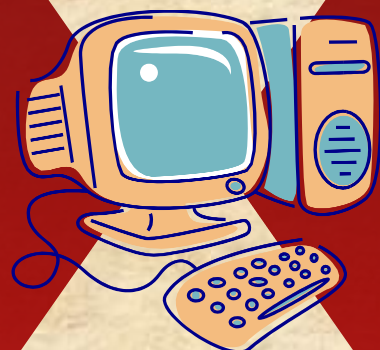


Figure 4.14 Solaris Thread States

4.6 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

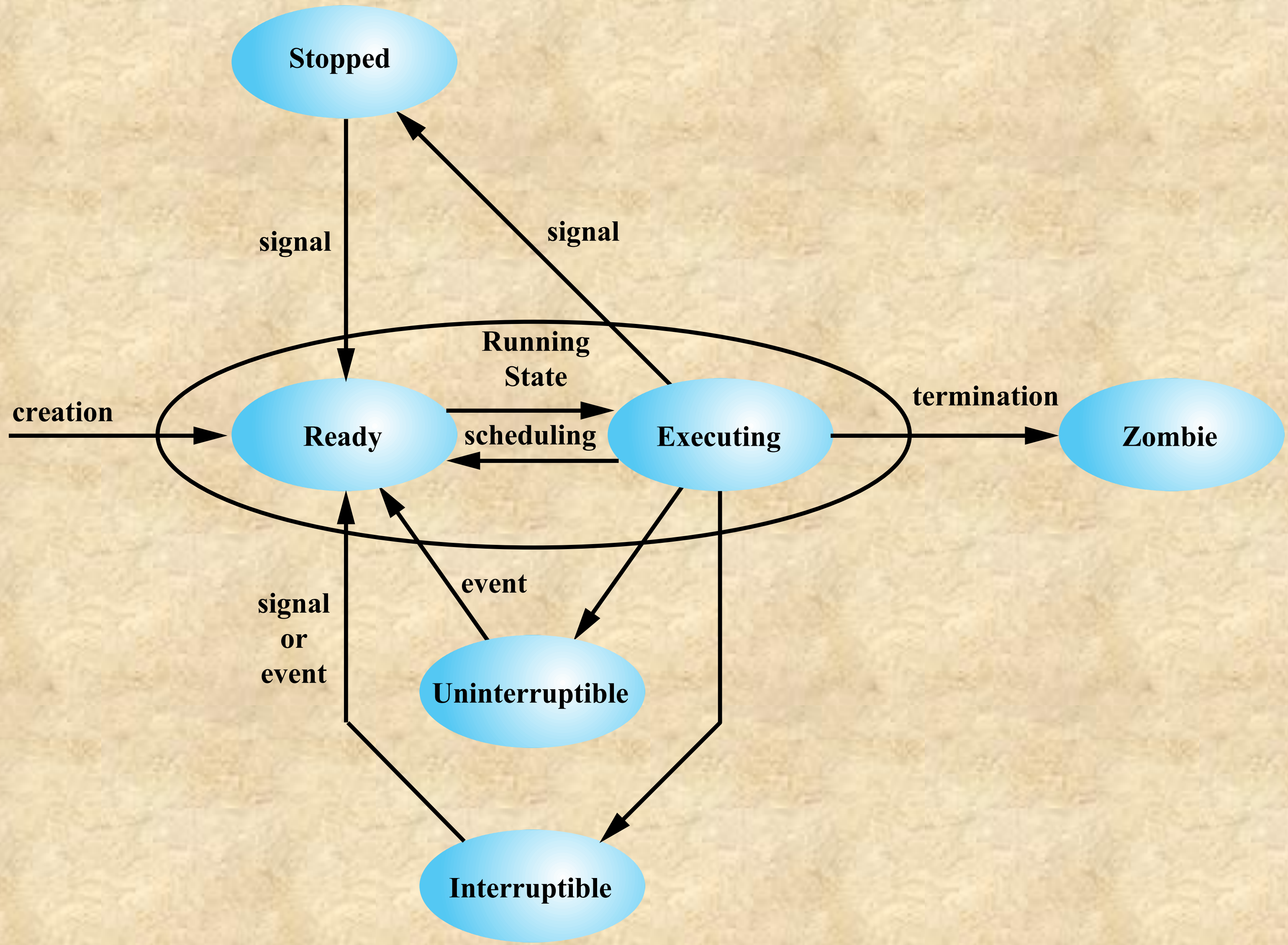


Figure 4.15 Linux Process/Thread Model

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 such as files and virtual memory

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

Summary

- Processes and threads
  - Multithreading
  - Thread functionality
- Types of threads
  - User level and kernel level threads
- Multicore and multithreading
- Windows 8 process and thread management
  - Changes in Windows 8
  - Windows process
  - Process and thread objects
  - Multithreading
  - Thread states
  - Support for OS subsystems
- Solaris thread and SMP management
  - Multithreaded architecture
  - Motivation
  - Process structure
  - Thread execution
  - Interrupts as threads
- Linux process and thread management
  - Tasks/threads/namespaces