

Operating System

Chapter 4. Threads



Lynn Choi

School of Electrical Engineering



高麗大學校

Computer System Laboratory

Motivation



- ❑ So far, we assumed that a process is running with a single thread of control
- ❑ However, most SW applications that run on modern computers are multithreaded
 - An application is typically implemented as a separate process with several threads of control.
 - A web browser might have one thread display portal news site while another thread shows a baseball broadcast
 - A word processor may have a thread for displaying graphics, another thread for responding to keystrokes from the user, and a third thread for performing grammar checking in the background
 - A web server may create a separate thread that can handle each individual request from clients and resume listening for additional requests
 - If the web server run as a single-threaded process, it would be able to service only one client at a time and a client may have to wait for a long time
 - Process creation was in common use before threads become popular. However, it is time consuming and resource intensive. If the new process will perform the same tasks as the existing process, why incur all that overhead?
- ❑ Virtually all modern operating systems provide features enabling a process to contain multiple threads of control

Process Characteristics



❑ Resource ownership

- A process may be allocated control or ownership of resources including main memory, I/O devices, and files
 - OS performs protection to prevent unwanted interferences among processes with respect to resources
- A process includes a virtual address space (process image)

❑ Scheduling unit

- Process is the entity that is scheduled and dispatched by OS
 - Has an execution state (Ready, Run) and schedule priority
 - The execution path (trace) may be interleaved with those of other processes

❑ Two characteristics are independent

- The scheduling unit can be treated independently by OS
 - In OS that supports threads, the scheduling unit is usually 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
 - Process is the unit of resource allocation and protection
 - Thread is the unit of dispatching with the following state
 - ▼ Thread execution state (Ready, Run)
 - ▼ Thread context
 - ▼ Thread execution stack

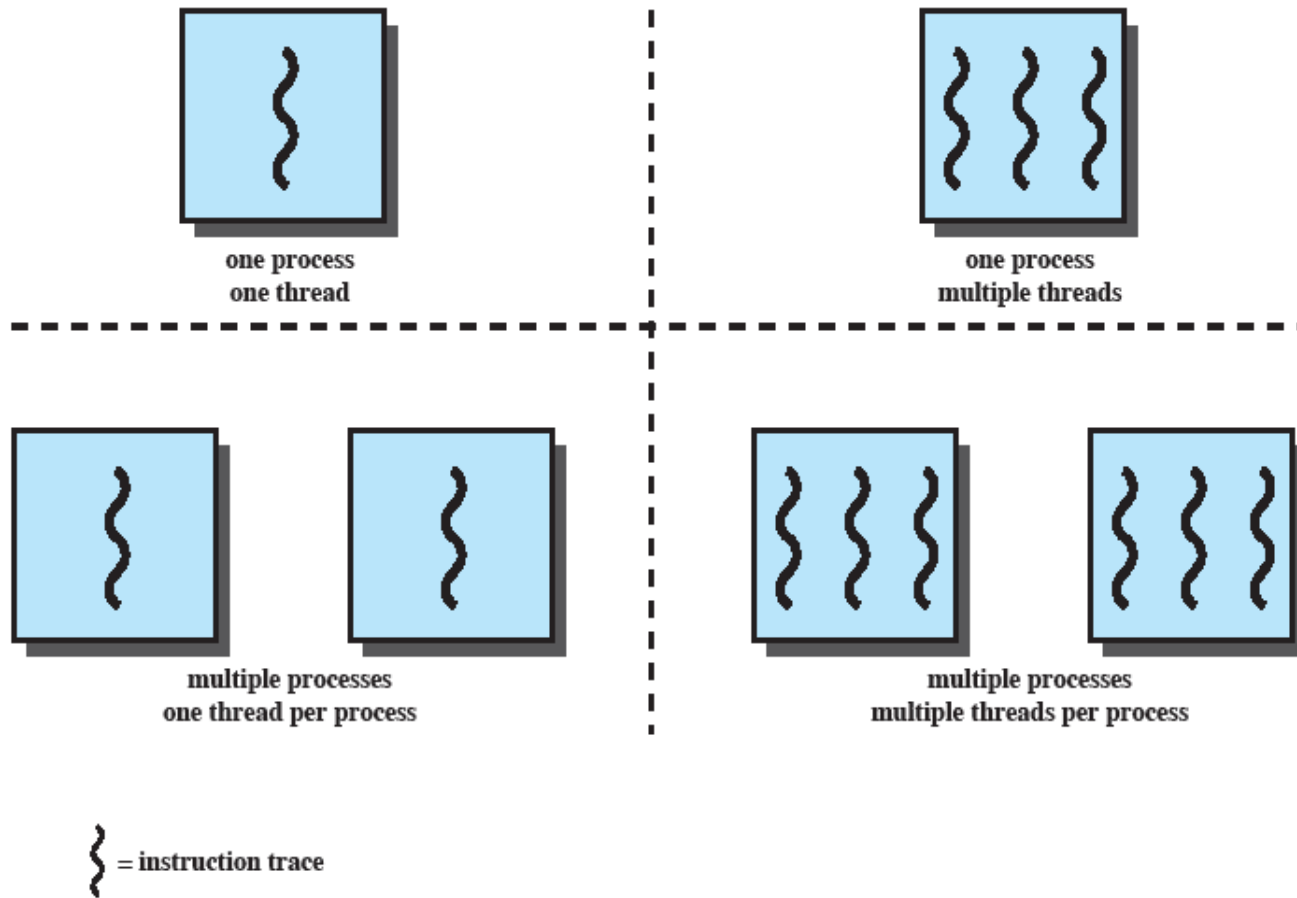
❑ Single-threaded approach

- Traditional approach of a single thread of execution per process
- No concept of thread
 - Examples: MS-DOS, old UNIX

❑ Multi-threaded approach

- One process with multiple threads of execution
 - Example: Java run-time environment
- Multiple processes with each of which supports multiple threads
 - Examples: Windows, Solaris, modern UNIX

Single-threaded vs. Multithreaded Approaches



Source: Pearson

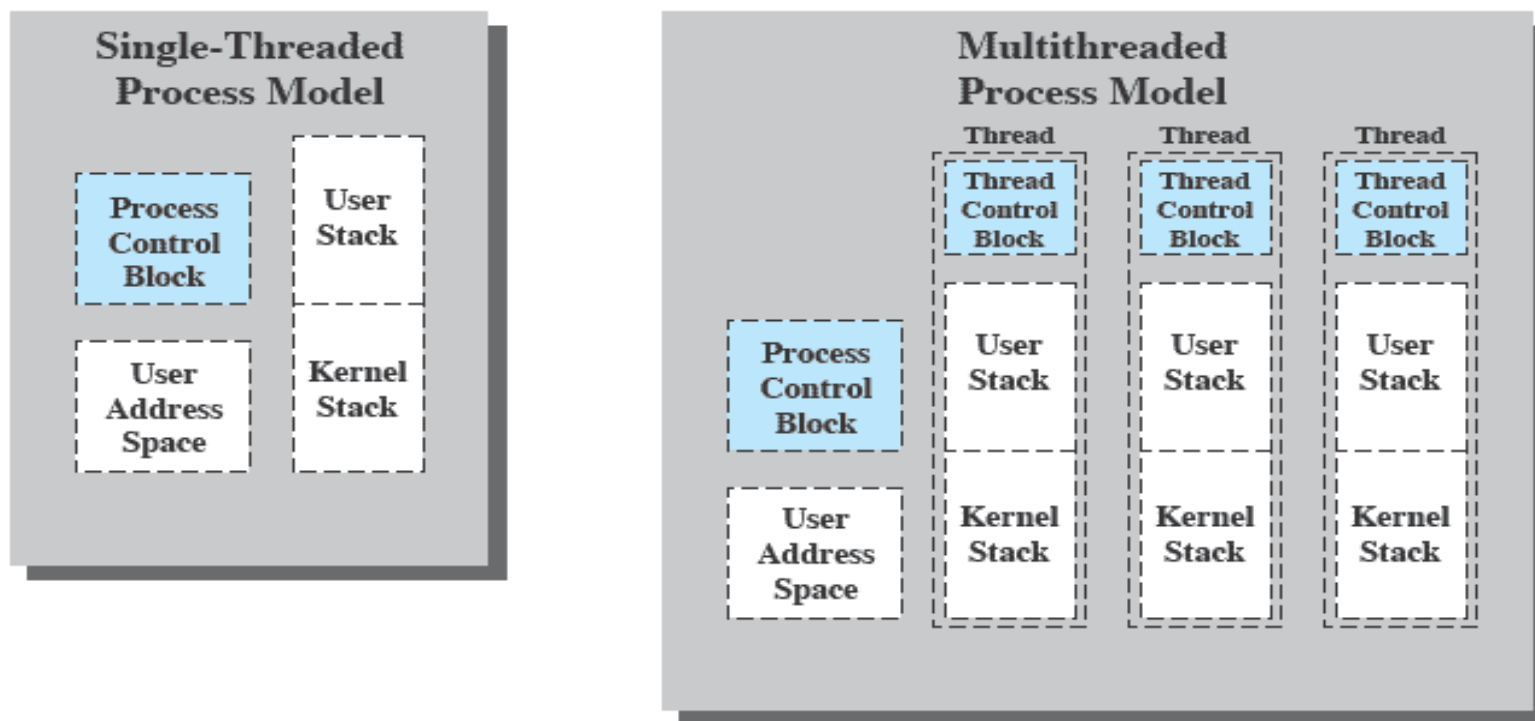
Figure 4.1 Threads and Processes [ANDE97]

Multithreaded Process Model



- ❑ **A process is a unit of resource allocation & protection, and has**
 - Virtual address space (process image on memory)
 - Protected access to processors, files, and I/O devices
- ❑ **Each thread within a process has**
 - Thread control block
 - Thread context: register values (PC, stack pointers)
 - Thread state, priority, and other thread-related state information
 - Execution stack (user stack, kernel stack)
- ❑ **All the threads of a process**
 - Share the same address space and share the resources of that process
 - When one thread alters the data item in memory, other threads see the results when they access the item.
 - If one thread opens a file with read privileges, other threads can also read from that file.

Threads vs. Processes



Source: Pearson

Figure 4.2 Single Threaded and Multithreaded Process Models

Multithreading



❑ Advantages of threads compared to process

- Less time to create a new thread in an existing process
 - Thread creation is 10 times faster than process creation in UNIX
- Less time to terminate a thread
 - You don't have to release I/O devices or memory
- Less time to switch between two threads
- Less time to communicate between two threads
 - Communication between processes require the kernel intervention to provide protection and communication (signal)
 - Threads can communicate without kernel through shared memory

❑ In OS with multithreading, scheduling and execution state is maintained at the thread-level, however some actions affect all the threads in the process

- Suspending (swapping) a process involves suspending all the threads of the process
- Termination of a process involves terminating all the threads with the process

Multithreaded Applications



□ File server

- A new thread can be spawned for each new file request
 - Since a server handles many requests, many threads will be created/destroyed
- On a multiprocessor environment, multiple threads within the same process can run simultaneously on different processors
- Faster to use threads to share files and coordinate their actions through shared memory
 - Processes/threads in a file server must share file data and coordinate actions

Multithreaded Applications



❑ Other examples in a single-user system

- Foreground and background jobs
 - In a spreadsheet program, one thread can display menus and read user input while another thread executes user commands and update the spreadsheet
 - ▼ Increase the perceived speed of the application by prompting for the next command before the previous command is complete
- Asynchronous processing
 - In a word processor, a separate thread can perform periodic backup from RAM buffer to disk
 - ▼ No need for fancy code in the main program to provide for time checks or to coordinate I/O
- Batch processing
 - One thread may process a batch job while another is reading the next batch
 - ▼ Even though one thread may be blocked for I/O, another thread may be executing

Benefits of Multithreading



❑ Faster Response

- A process can continue running even if part of it is blocked or takes a long time

❑ Parallel Processing

- The benefits of multithreading is even greater in a multicore system since threads can run in parallel on different processor cores

❑ Efficient sharing of memory and resources

- Threads can share code and data since they are in the same address space
- Threads can communicate through shared-memory rather than kernel-supported signals
- Threads can also share resources of the process to which they belong

❑ Economy

- Thread creation, switching, termination, and communication cost much less than process counterparts



❑ Thread State

- Ready, Run, Blocked
- Suspended: does not make sense since it is process-level state

❑ Thread operations that affects the state

- Spawn
 - When a new process is spawned, a thread for that process is also spawned
 - A thread may spawn another thread within the same process
 - The new thread is provided with its own register context and stack space and placed on the ready queue
- Block
 - When a thread needs to wait for an event, it will block (save its PC and registers)
 - The processor may switch to another ready thread in the same or different process
- Unblock
 - When the event occurs, the thread moves to the ready queue
- Finish
 - When a thread completes, the register context and stacks are deallocated

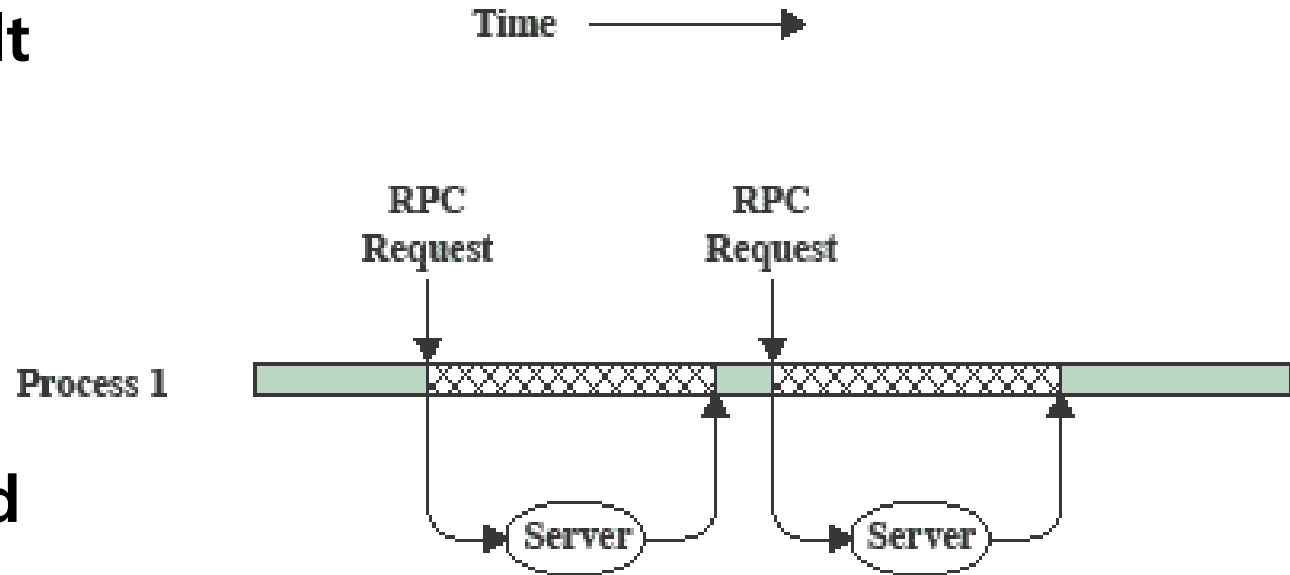
RPC Using Single Thread



- ❑ 2 RPCs to 2 hosts to obtain a combined result

- ❑ Single-threaded program

- Each RPC has to wait for a response from each server sequentially



(a) RPC Using Single Thread

Source: Pearson

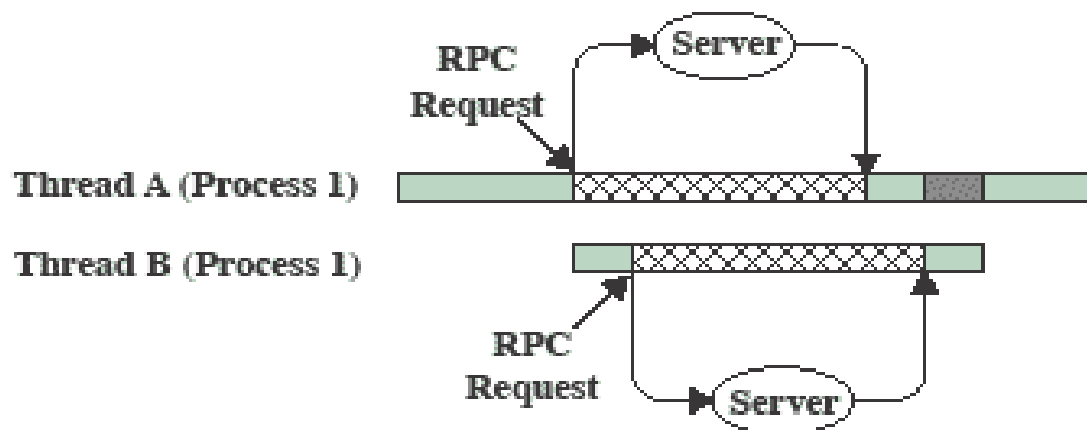
RPC Using One Thread per Server




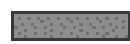

❑ Multi-threaded program

➤ Each RPC request must be generated sequentially

➤ Each request wait concurrently for the two replies



(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

Source: Pearson

Interleaving of Multiple Threads Within Multiple Processes



- ❑ 3 threads of 2 processes are interleaved on a processor
- ❑ Thread switching occurs when the current thread is blocked or its time slice expires

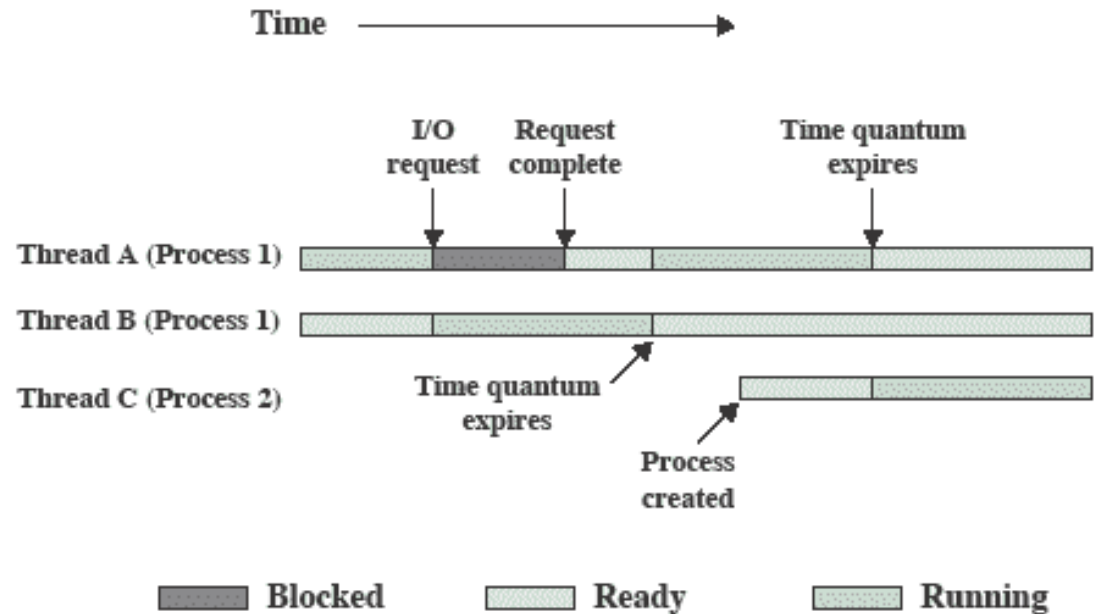


Figure 4.4 Multithreading Example on a Uniprocessor

Source: Pearson

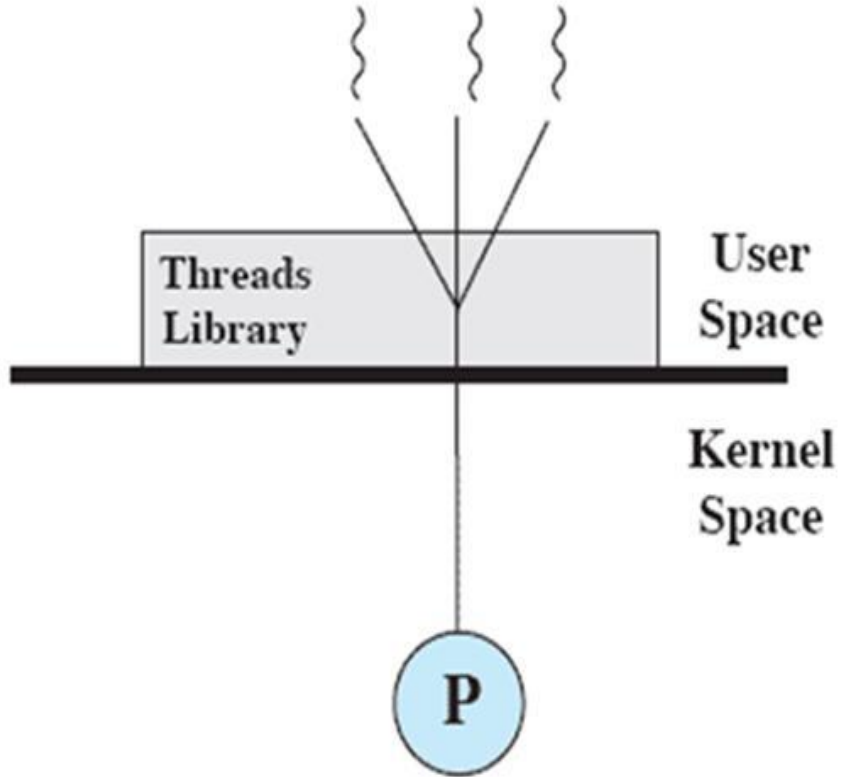
User-Level Threads (ULTs)



- ❑ **All thread management is done by the application**

- The threads library contains code for creating and destroying threads, scheduling thread execution, saving and restoring thread contexts, and passing messages between threads

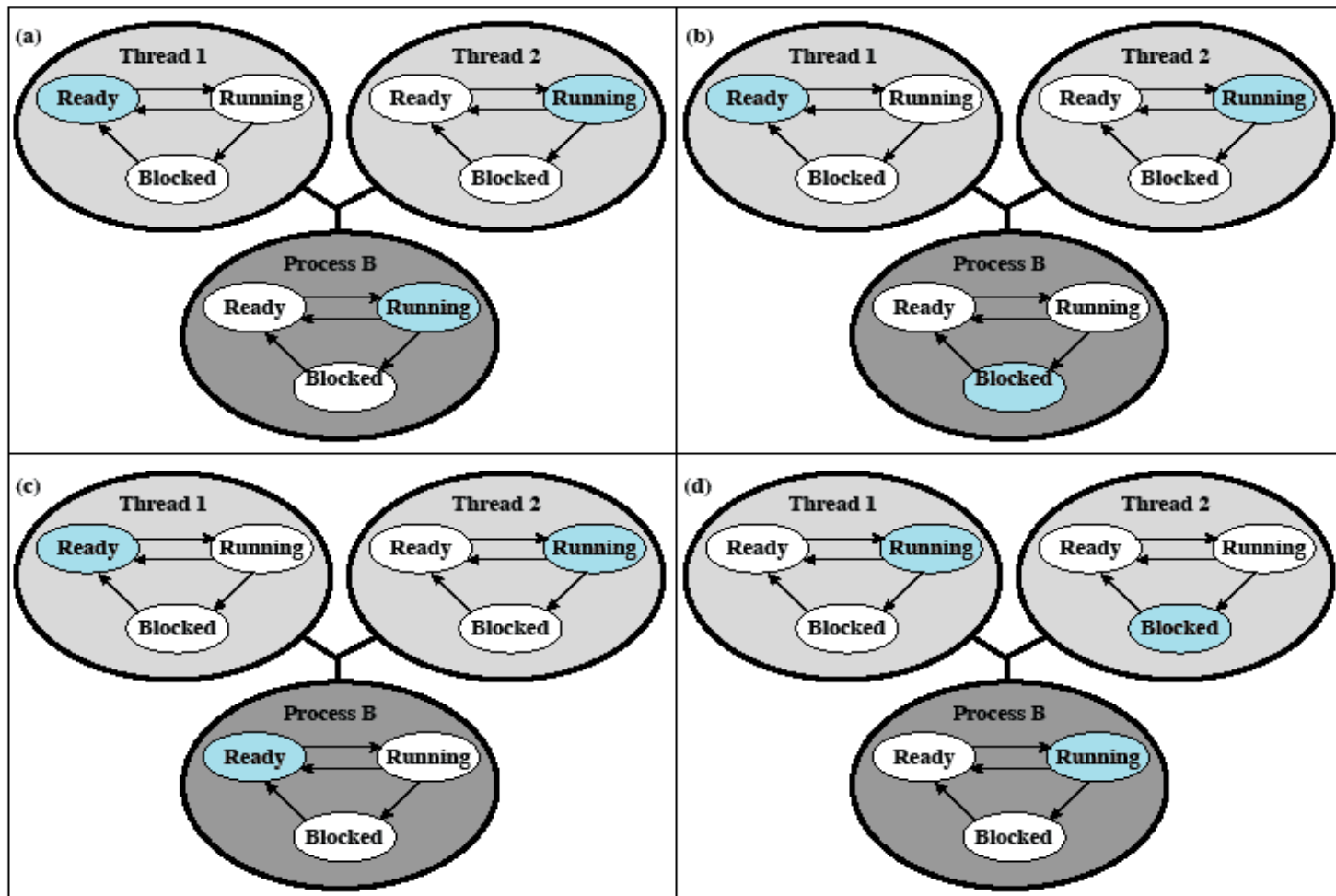
- ❑ **The kernel is not aware of the existence of threads**



(a) Pure user-level

Source: Pearson

ULT States and Process States



Colored state
is current state

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

Source: Pearson

User-Level Threads



□ Advantages

- Thread switching does not require kernel mode privileges (faster switching)
- Scheduling algorithm can be tailored to the application without disturbing OS scheduler
 - One application may benefit most from a simple round robin scheduling while another might benefit from a priority-based scheduling
- ULTs can run on any OS. No changes are required to the underlying kernel

□ Disadvantages

- In a typical OS, many system calls are blocked
 - As a result, when a ULT executes a system call, not only the thread is blocked, but also all the other threads within the process are blocked.
- A multithreaded application cannot take advantage of multiprocessing
 - A kernel assigns one process to only one processor. Therefore, only a single thread can execute at a time

Kernel-Level Threads (KLTs)

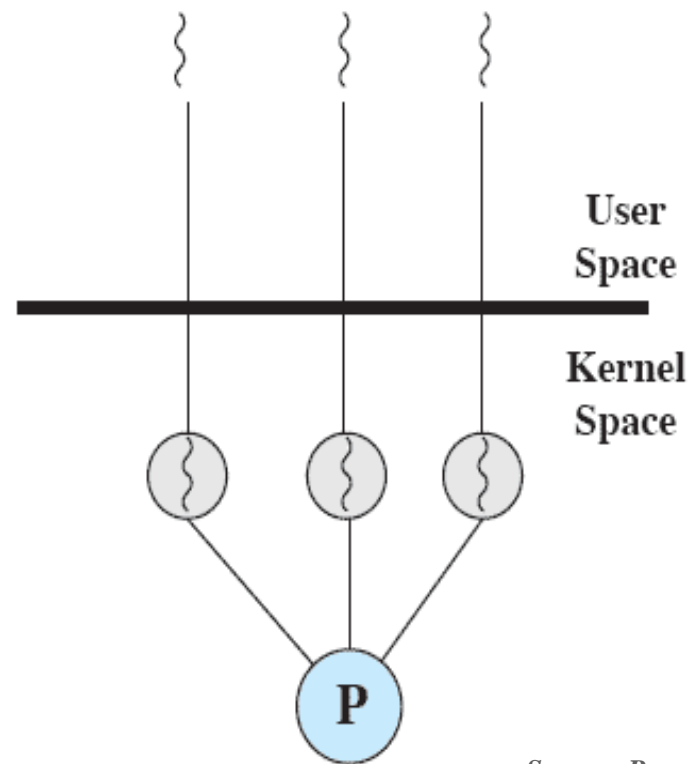


❑ Thread management is done by the kernel

- No thread management is done by the application
 - Simply an API to the kernel thread facility
 - Example: Windows

❑ Advantages

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



Source: Pearson

(b) Pure kernel-level

Disadvantage of KLTs



❑ Disadvantages

- Thread switching within the same process requires a mode switch to the kernel
- More than an order of magnitude difference between ULTs and KLTs and similarly between KLTs and processes

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 (μ s)

Source: Pearson

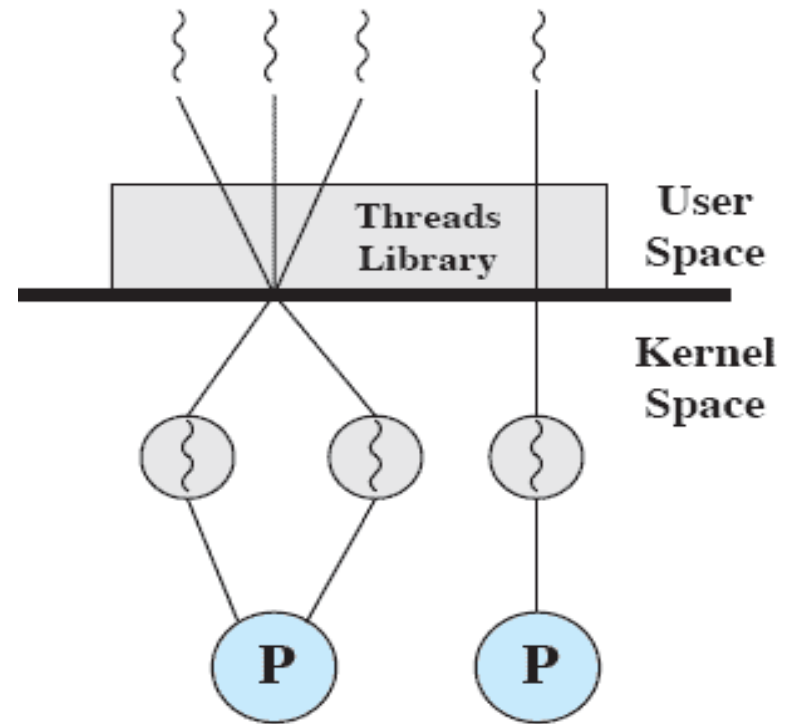
- Null Fork: create a new process/thread that invokes null procedure
- Signal Wait: signal a waiting process/thread and wait on a condition (overhead of synchronizing two processes/threads)

Combined Approach



❑ Thread creation is done completely in the user space

- Multiple ULTs from a single application are mapped onto the same or smaller number of KLTs
 - To achieve the best overall results, the programmer adjust the number of KLTs for a particular application
- Multiple threads within the same process can run in parallel on multiple processors
 - A blocking system call need not block the entire process
- If properly designed, can combine the advantages of both ULT and KLT approach while minimizing the disadvantages.



(c) Combined

Source: Pearson

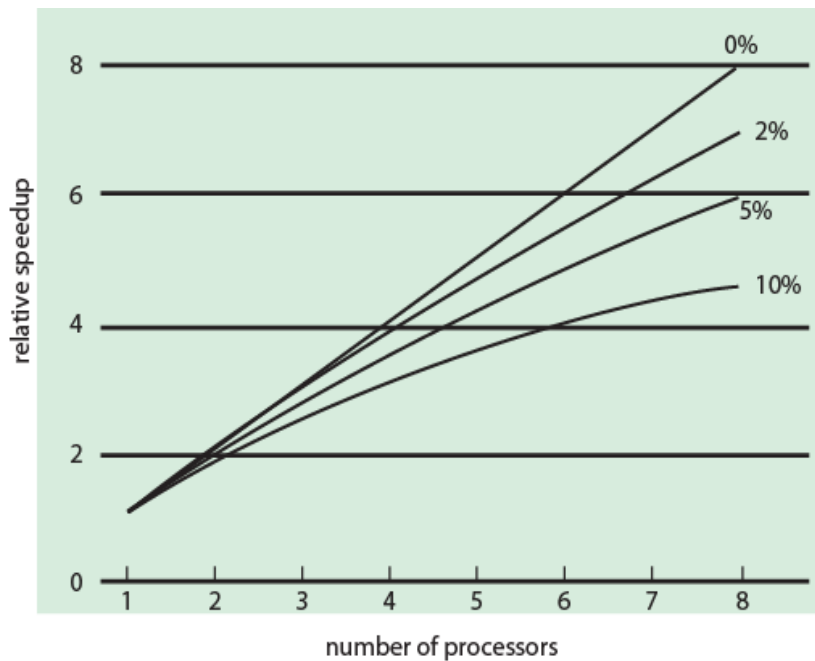
❑ Example: Solaris

Performance Impact of Multicores

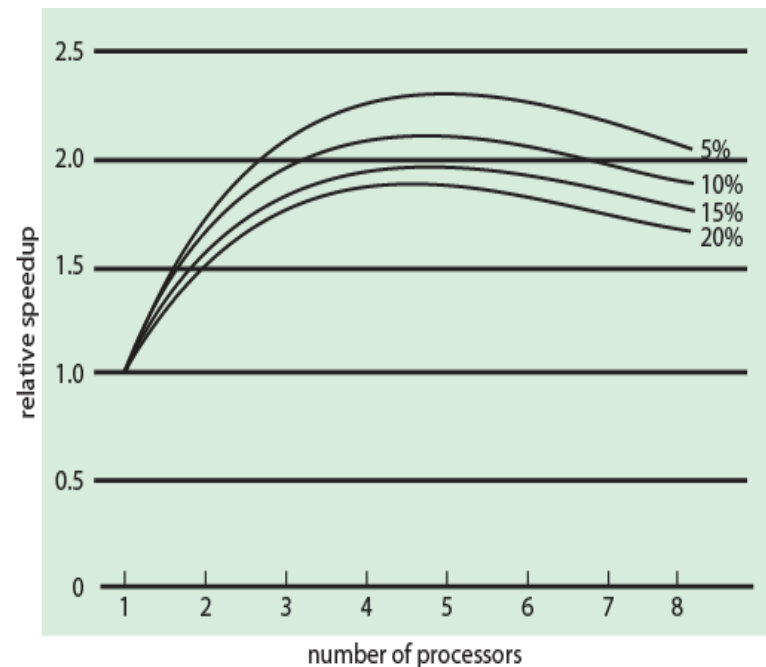


□ Amdahl's law

- Speedup = time to execute a program on a single processor /
time to execute the program on N processors
= $1 / ((1 - f) + f / N)$ where $(1 - f)$ is an inherently serial fraction



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



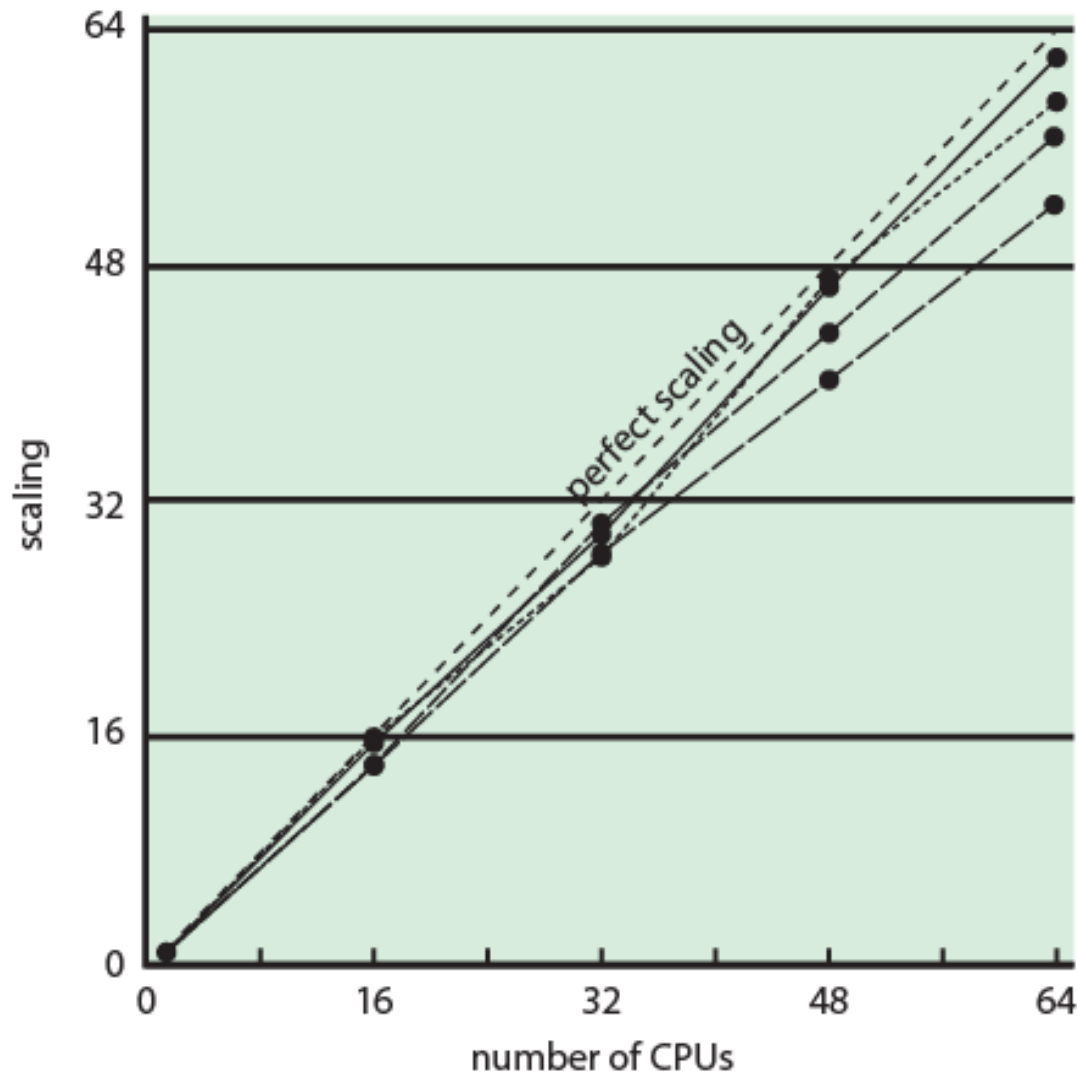
(b) Speedup with overheads

Source: Pearson

- If 10% is inherently serial ($f = 0.9$)
 - Speedup on 8 processors is only 4.7

- In real systems, overheads come from
 - Communication, workload distribution, and cache coherence

Database Workloads on Multicores



- Software engineers have been addressing this problem to effectively exploit the multicore
- There are numerous applications that can effectively exploit multicores
 - Database workloads
 - Great attention was paid to *reduce the serial fraction* within HW, OS, middleware, and DB applications
 - Servers typically handle numerous independent transactions in parallel

Source: Pearson

Applications for Multicores



❑ Multithreaded native applications

- Characterized by having a small number of highly threaded processes
 - IBM (Lotus) Domino, Oracle (Siebel) CRM (Customer Relationship Manager)

❑ Multiprocess applications

- Characterized by the presence of many single-threaded processes
 - Oracle database, SAP

❑ Java applications

- Java language facilitate multithreaded applications
- Java Virtual Machine is also a multithreaded process that provides scheduling and memory management for Java applications
 - Sun's Java Application Server, IBM's Websphere, all applications based on J2EE (Java 2 Enterprise Edition) application server

❑ Multi-instance applications

- Can achieve speedup by running multiple instances of the same application in parallel



❑ Solaris provides four thread-related objects

➤ Process

- Normal UNIX process
- Includes user's address space, stack, and process control block

➤ User-level thread (ULT)

- Implemented by a threads library at the application-level
- Invisible to the OS

➤ Lightweight process (LWP)

- Can be viewed as a mapping between ULTs and kernel threads
- Each LWP maps to one kernel thread
- LWPs are scheduled by the kernel independently and may execute in parallel on multiprocessors

➤ Kernel thread

- These are fundamental entities that can be scheduled and dispatched to run on any processors
- There are kernel threads that are not associated with LWPs
 - The use of kernel threads to implement system functions reduces the overhead of switching within the kernel (from a process switch to a thread switch)

Processes and Threads in Solaris

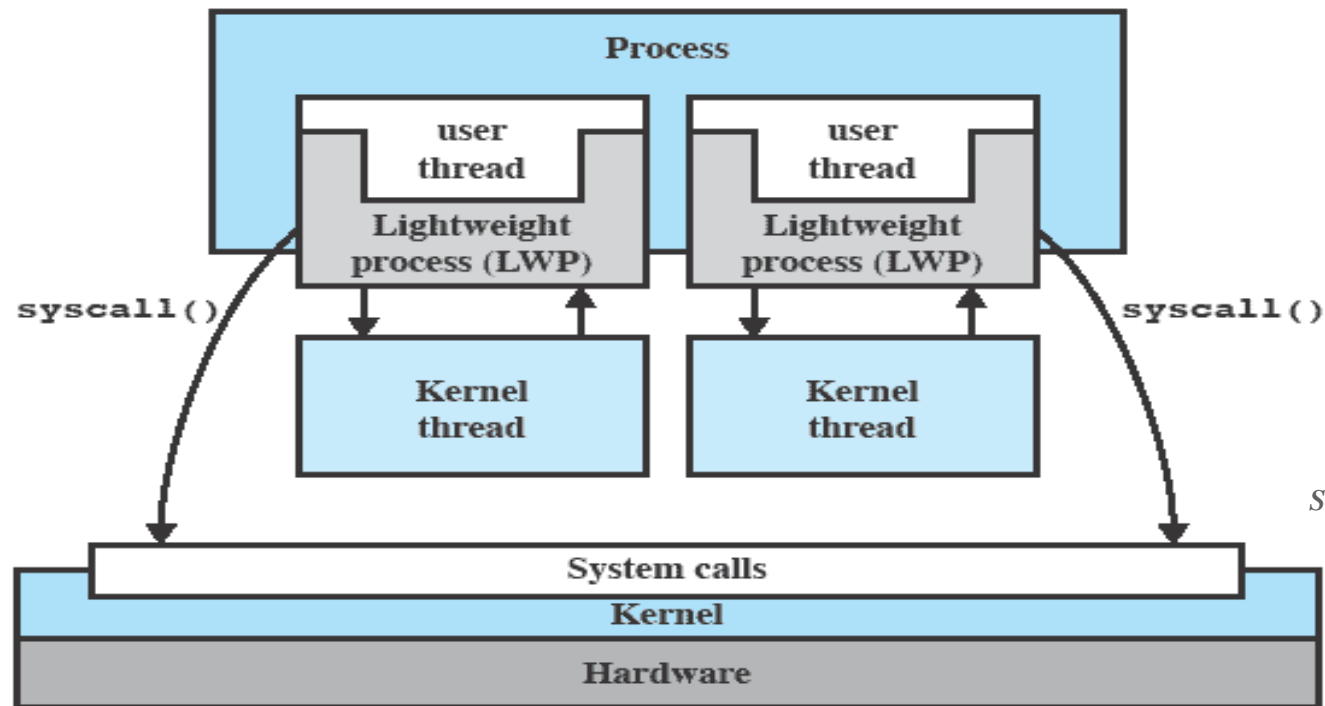


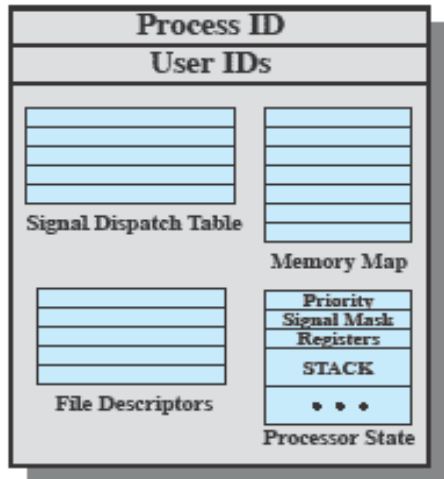
Figure 4.15 Processes and Threads in Solaris [MCDO07]

- A process may consist of a single ULT bound to a single LWP
 - A single thread of execution, corresponding to a traditional UNIX process
- A process may contain multiple threads, each bound to a single LWP, which in turn is bound to a single kernel thread
 - When an application requires concurrency

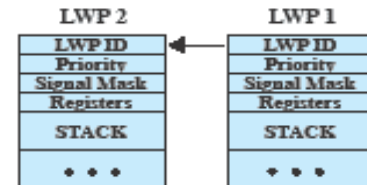
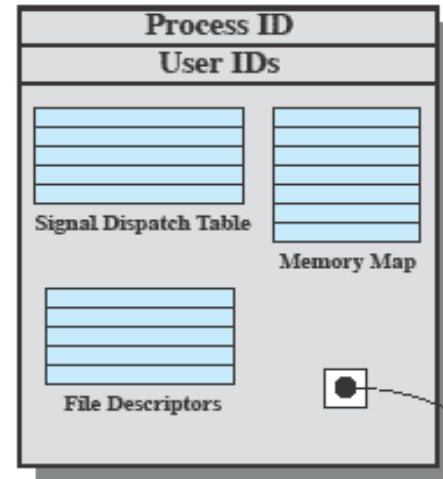
Traditional Unix vs Solaris



UNIX Process Structure



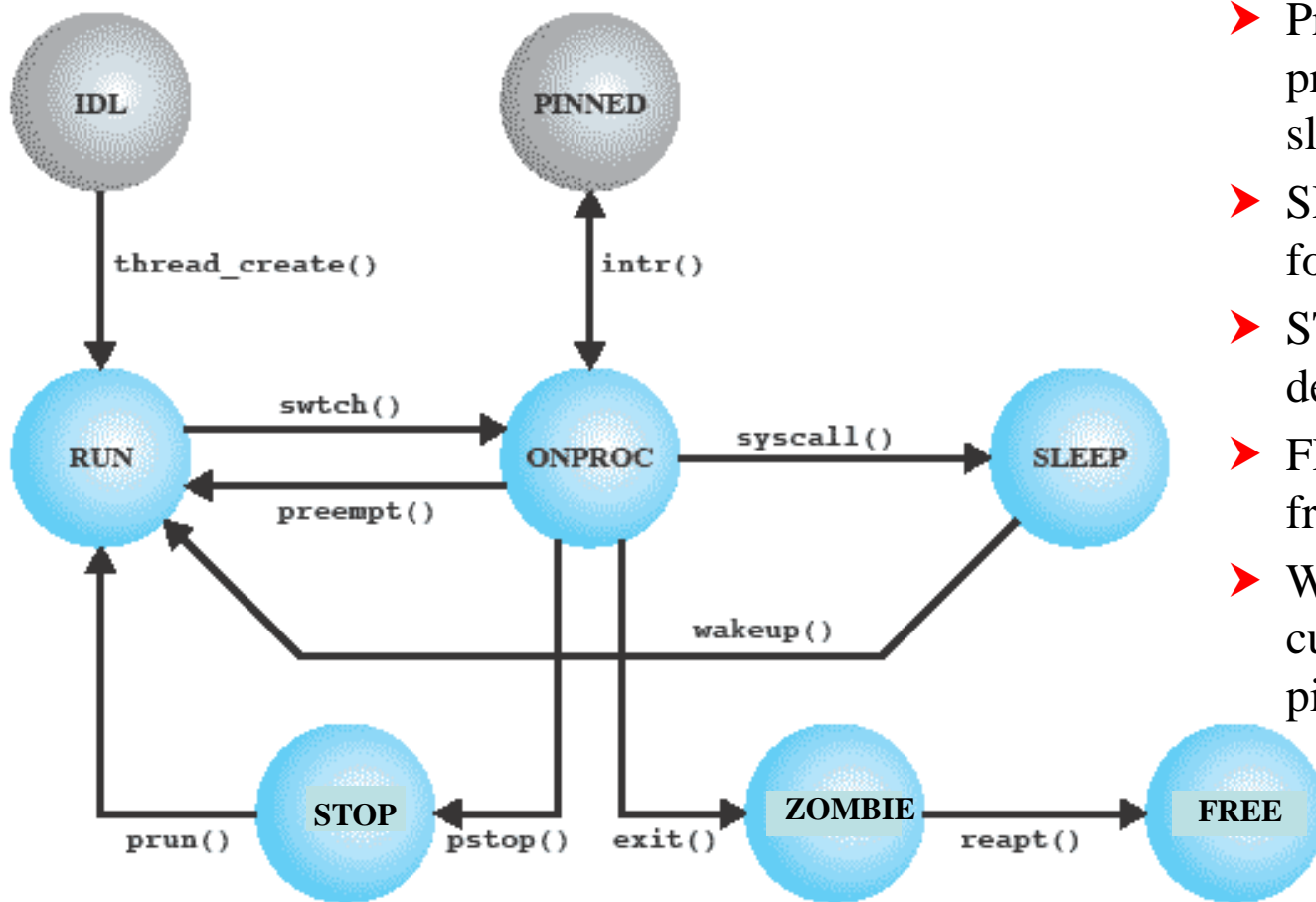
Solaris Process Structure



Source: Pearson

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

Solaris Thread States



Source: Pearson

Figure 4.17 Solaris Thread States [MCDO07]

- Preemption by a higher priority thread or due to time slice
- SLEEP means blocked to wait for an event
- STOP might be done for debugging purpose
- FREE is awaiting removal from OS thread data structure
- When an interrupt occurs, the currently running thread is pinned.
 - A PINNED thread cannot move to another processor
 - It is simply suspended until the interrupt is processed

Homework 3



- 4.1
- 4.3
- 4.7
- 4.9
- 4.11
- Read Chapter 5