Operating System

Chapter 4. Threads

Lynn Choi

School of Electrical Engineering

Process Characteristics

- **Resource ownership**
  - Includes a virtual address space (process image)
  - Ownership of resources including main memory, I/O devices, and files
    - OS performs protection to prevent unwanted interferences among processes with respect to resources
- **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

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

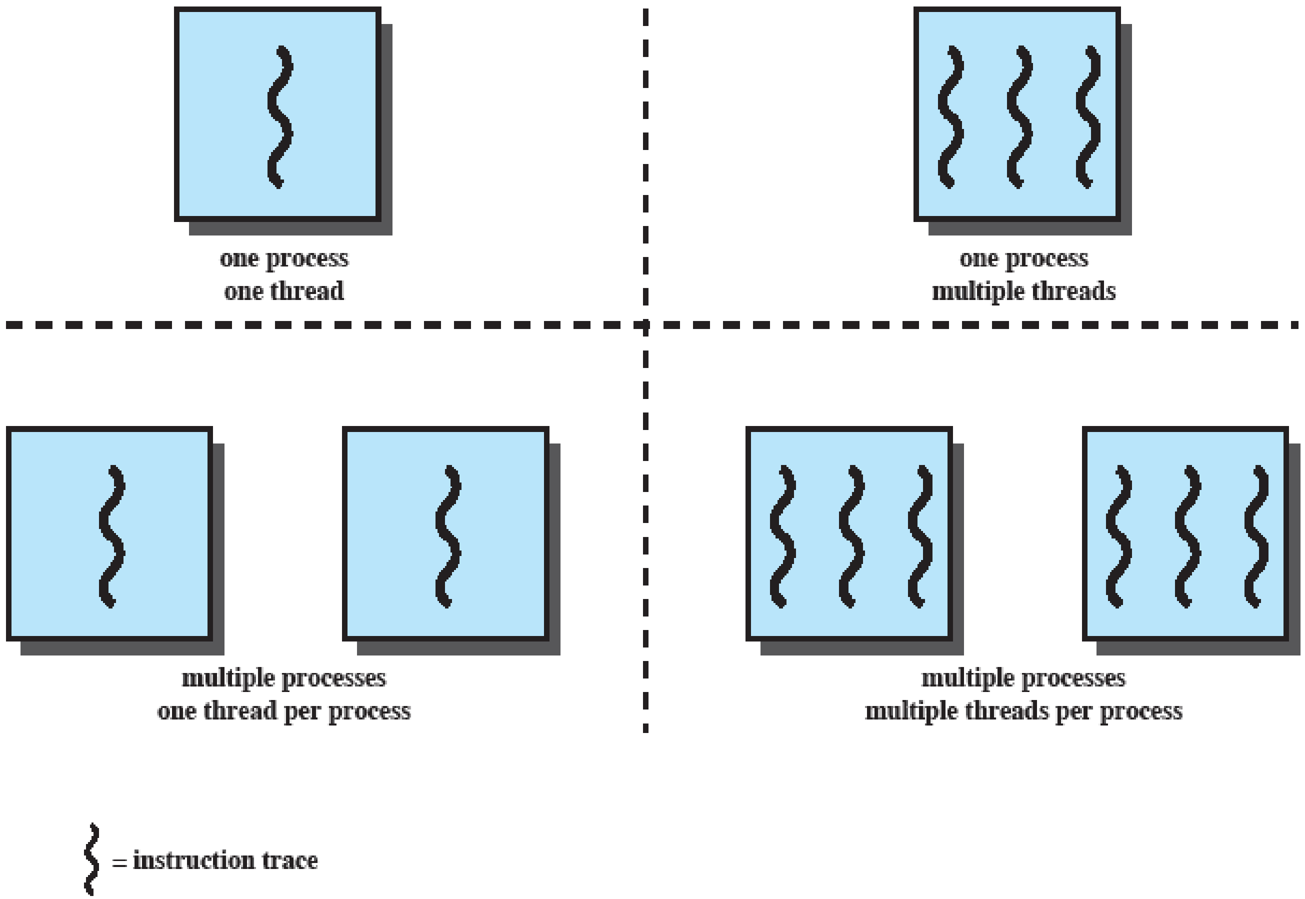


Figure 4.1 Threads and Processes [ANDE97]

Source: Pearson

Multithreaded Process Model

- **Process has**
  - Virtual address space (process image on memory)
  - Protected access to files, and I/O devices
- **Each thread within a process has**
  - Thread control block
    - 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

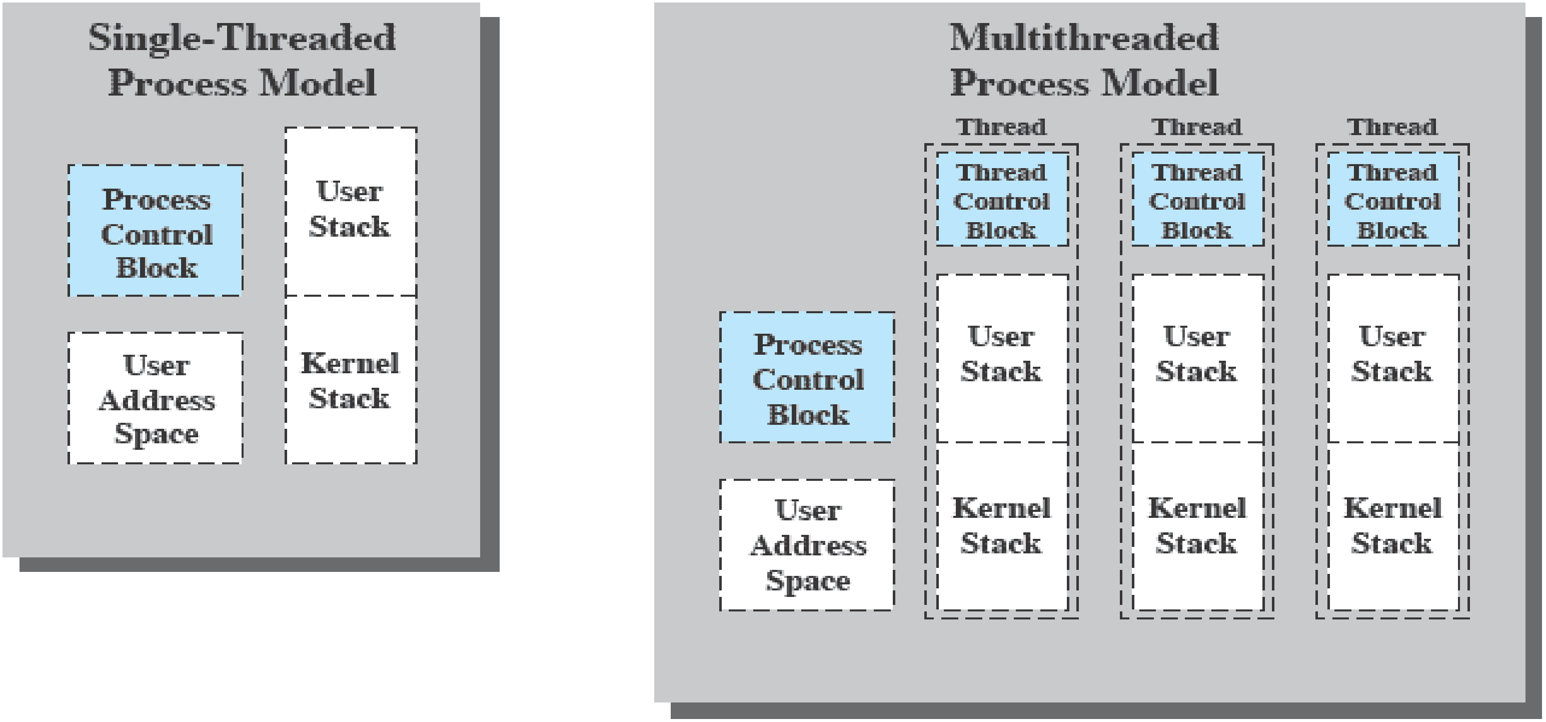


Figure 4.2 Single Threaded and Multithreaded Process Models

Source: Pearson

Multithreading

- **Benefits of threads**
  - Less time to create a new thread in an existing process
    - Thread creation is 10 times faster than process creation (Mach developers)
  - 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 execute 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

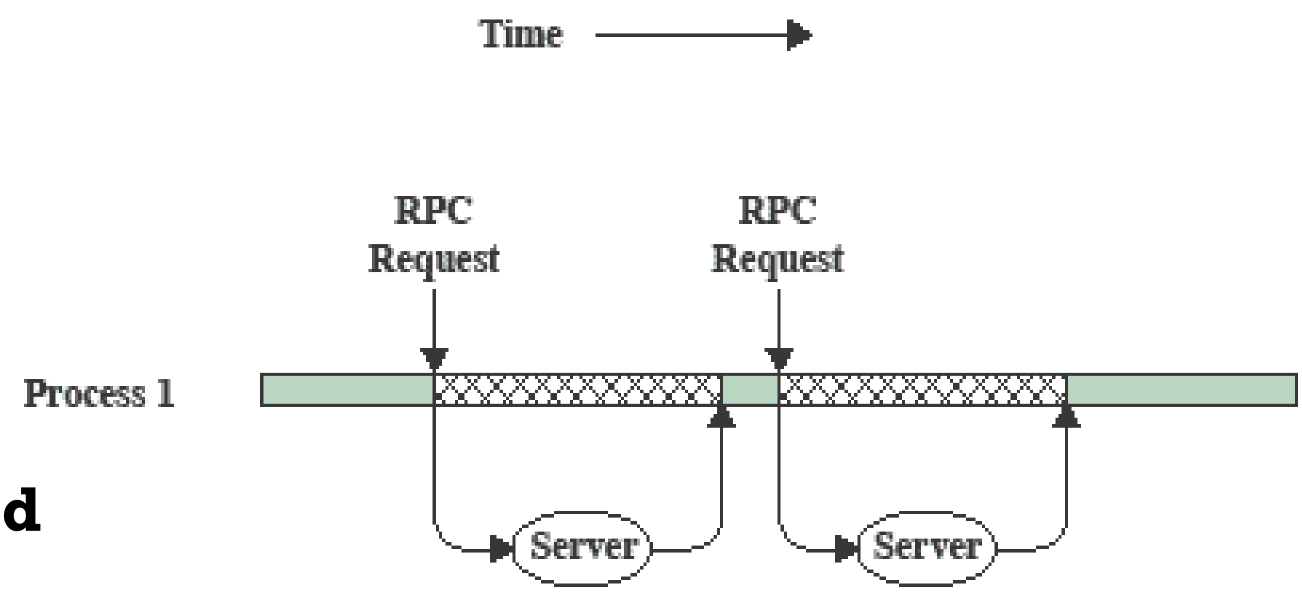
- **Other examples in a single-user system**
  - Foreground and background jobs
    - In a spreadsheet program, one thread can display menus and read user input while another thread executes user commands and update the spreadsheet
      - Increase the perceived speed of the application by prompting 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

Thread State

- **Thread State**
  - Ready, Run, Blocked
  - Suspended: do 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
  - 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



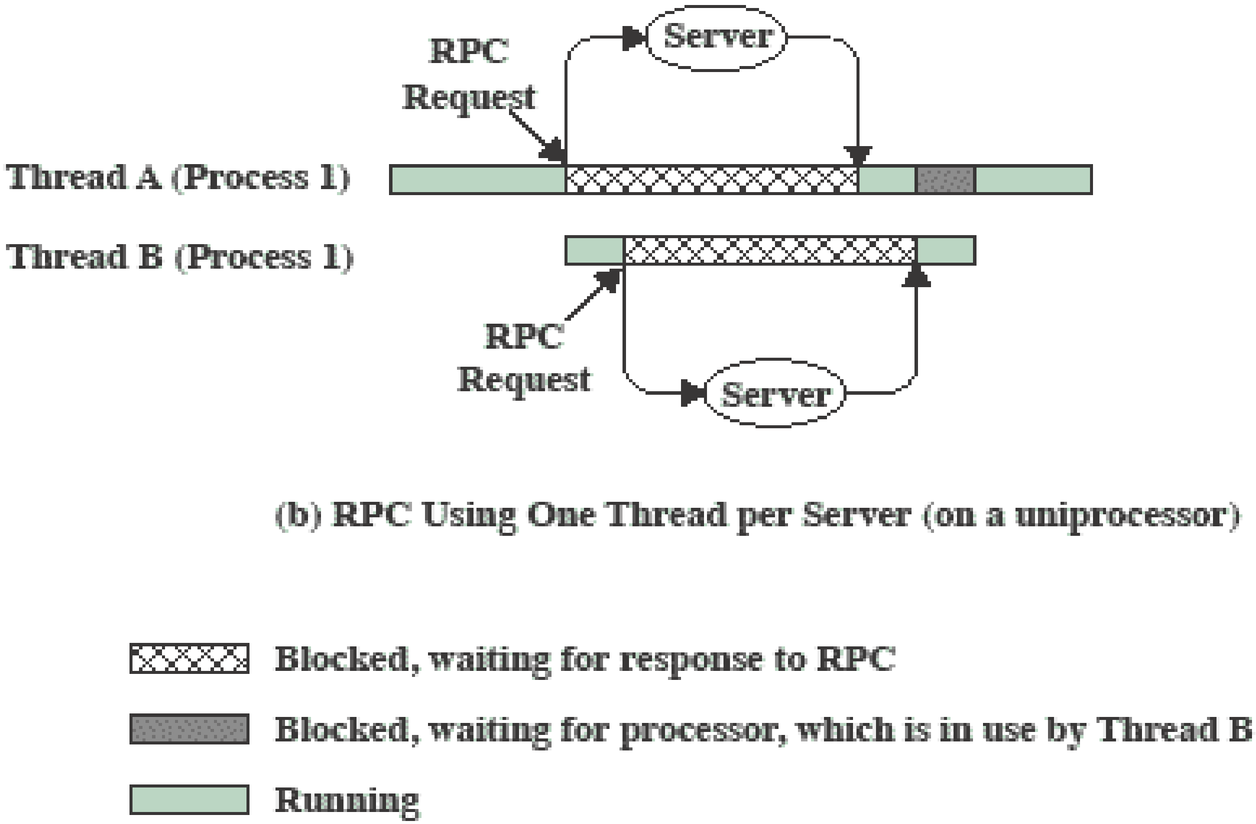
(a) RPC Using Single Thread

Source: Pearson

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

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)

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

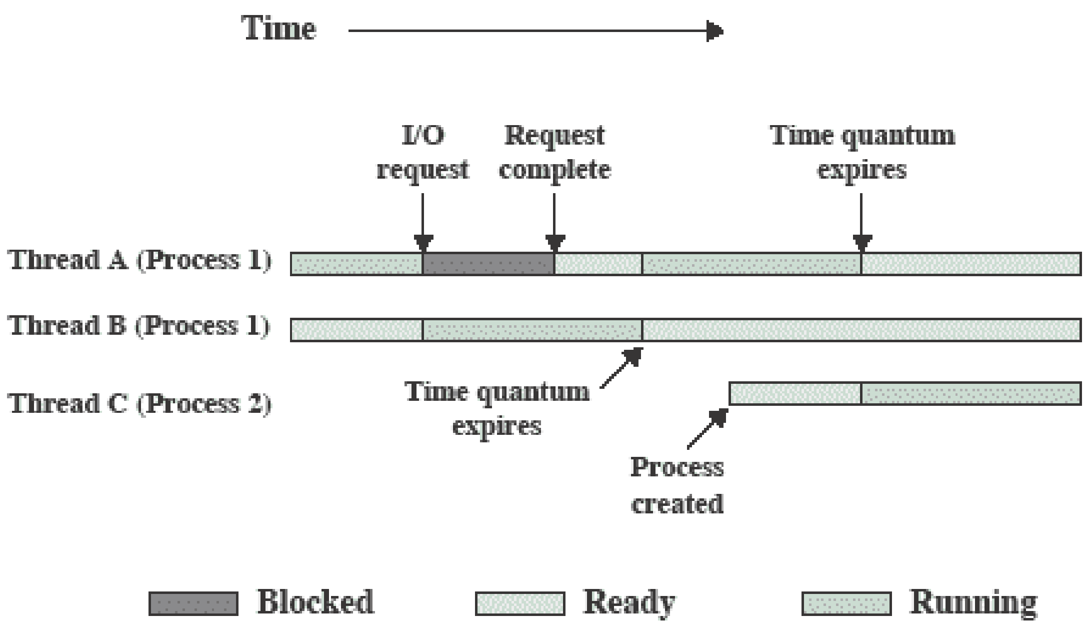
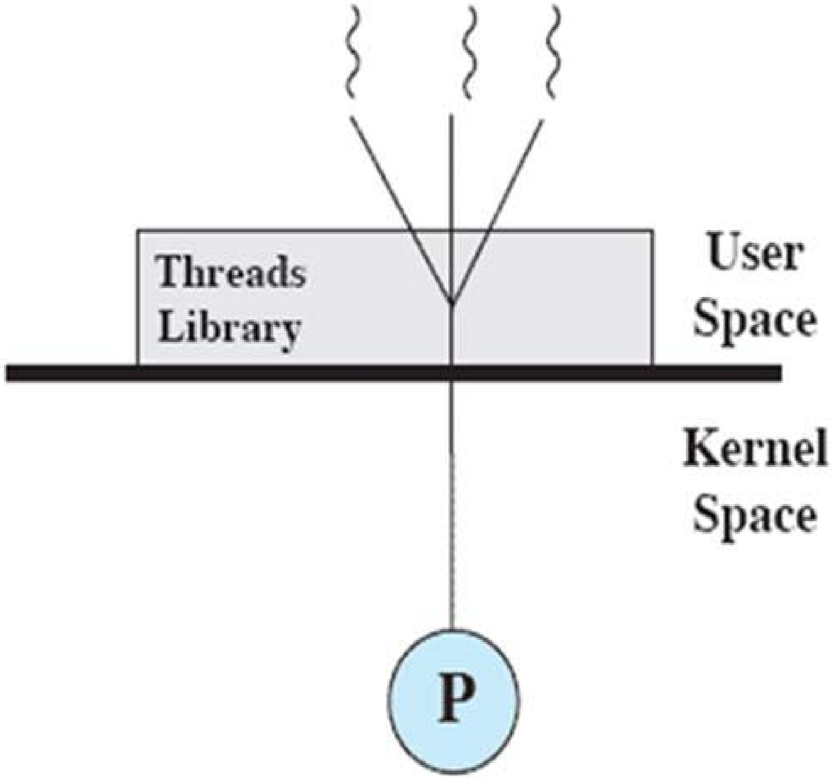


Figure 4.4 Multithreading Example on a Uniprocessor

Source: Pearson

User-Level Threads (ULTs)

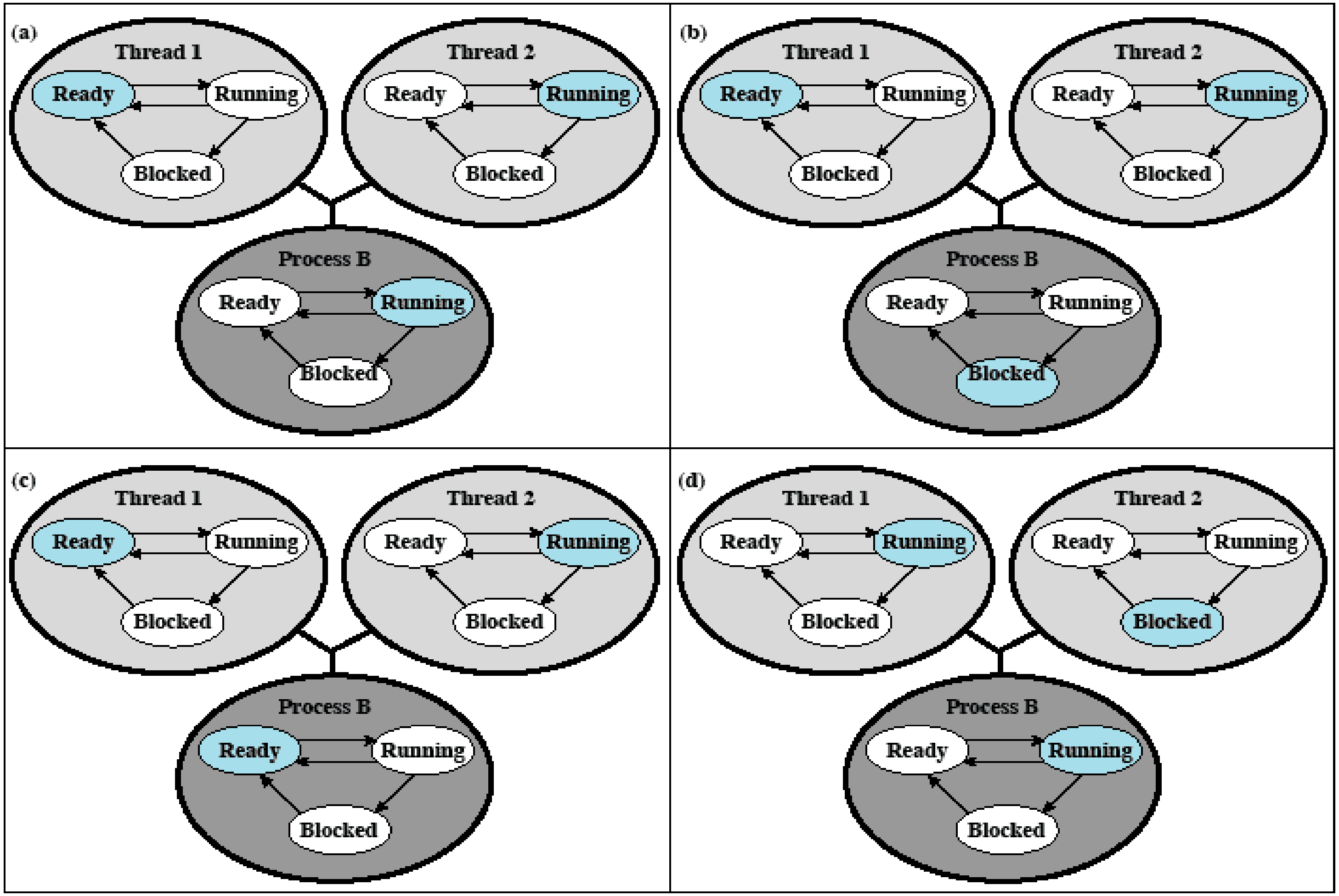
- **All thread management is done by the application**
  - The threads library contains code for creating and destroying threads, scheduling thread execution, saving and restoring thread contexts, and passing messages between threads
- **The kernel is not aware of the existence of threads**



(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

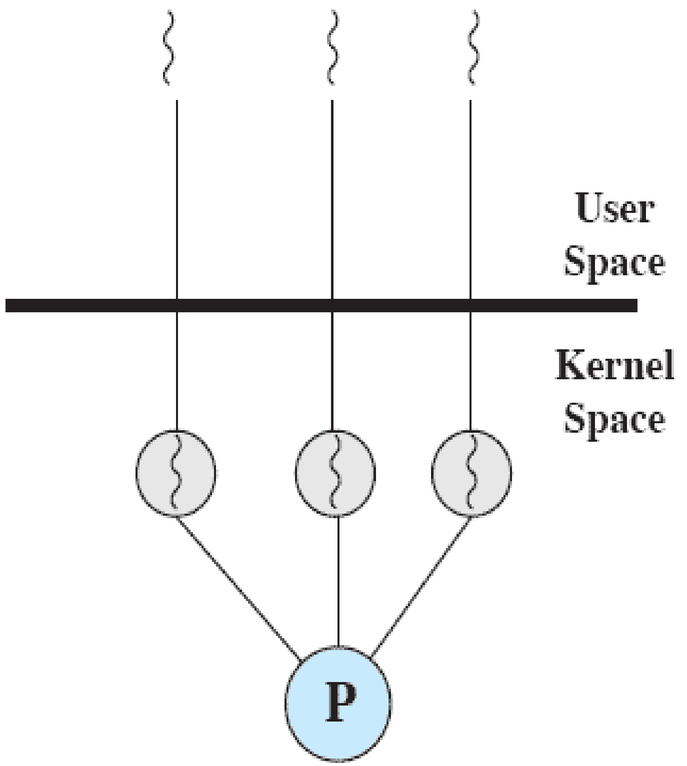
User-Level Threads

- **Advantages**
  - Thread switching does not require kernel mode privileges (faster switching)
  - Scheduling algorithm can be tailored to the application without disturbing OS scheduler
  - 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



(b) Pure kernel-level

Source: Pearson

- **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

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

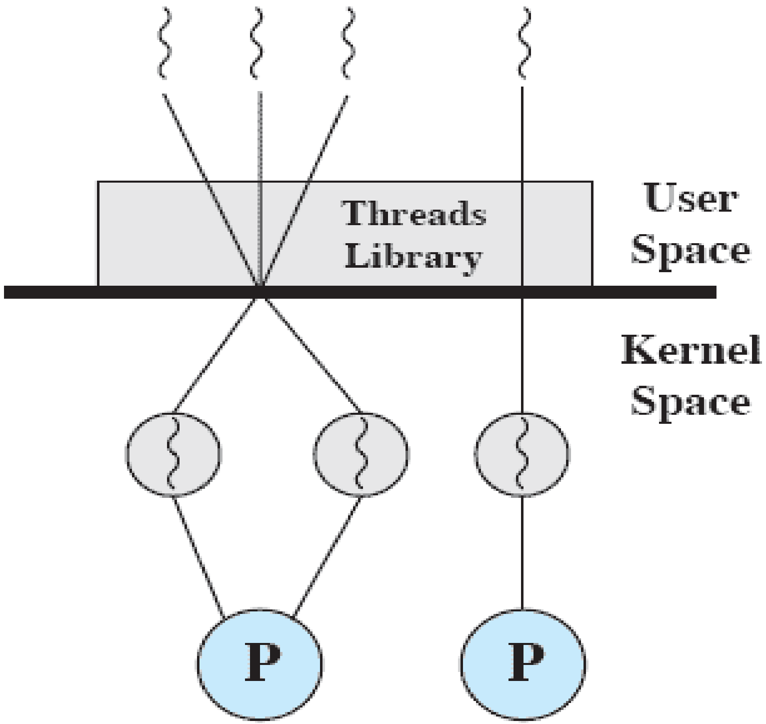
Table 4.1 Thread and Process Operation Latencies (μs)

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

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.
- **Example: Solaris**



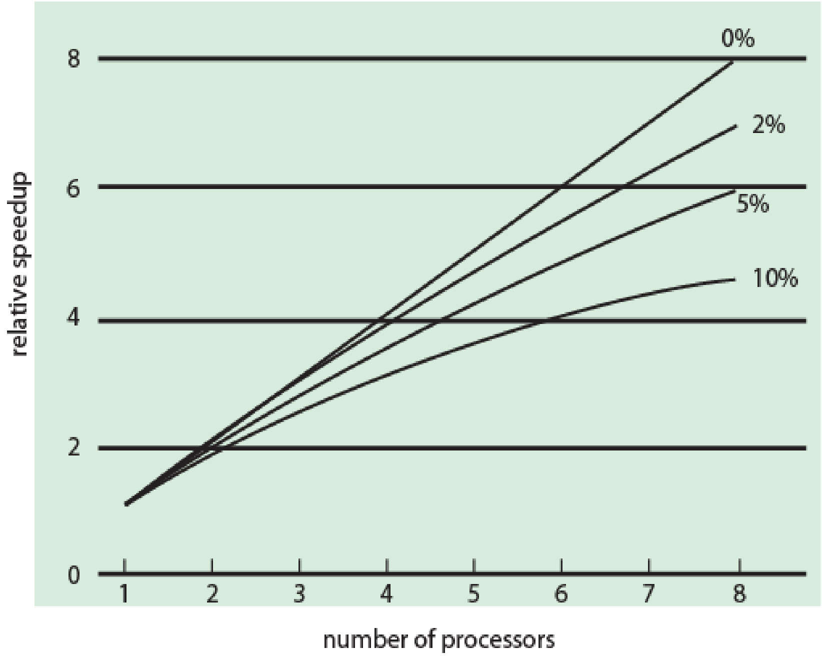
(c) Combined

Source: Pearson

Performance Impact of Multicores

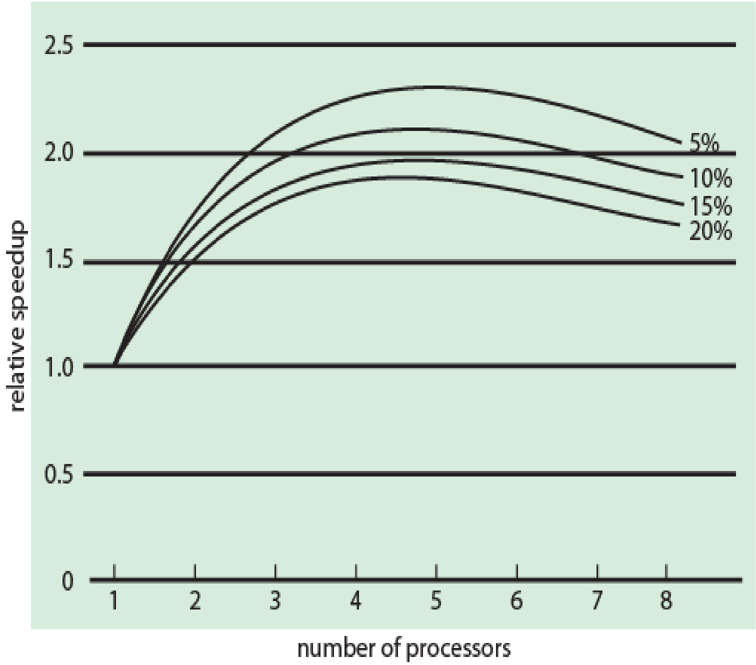
❑ Amdahl's law

- 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

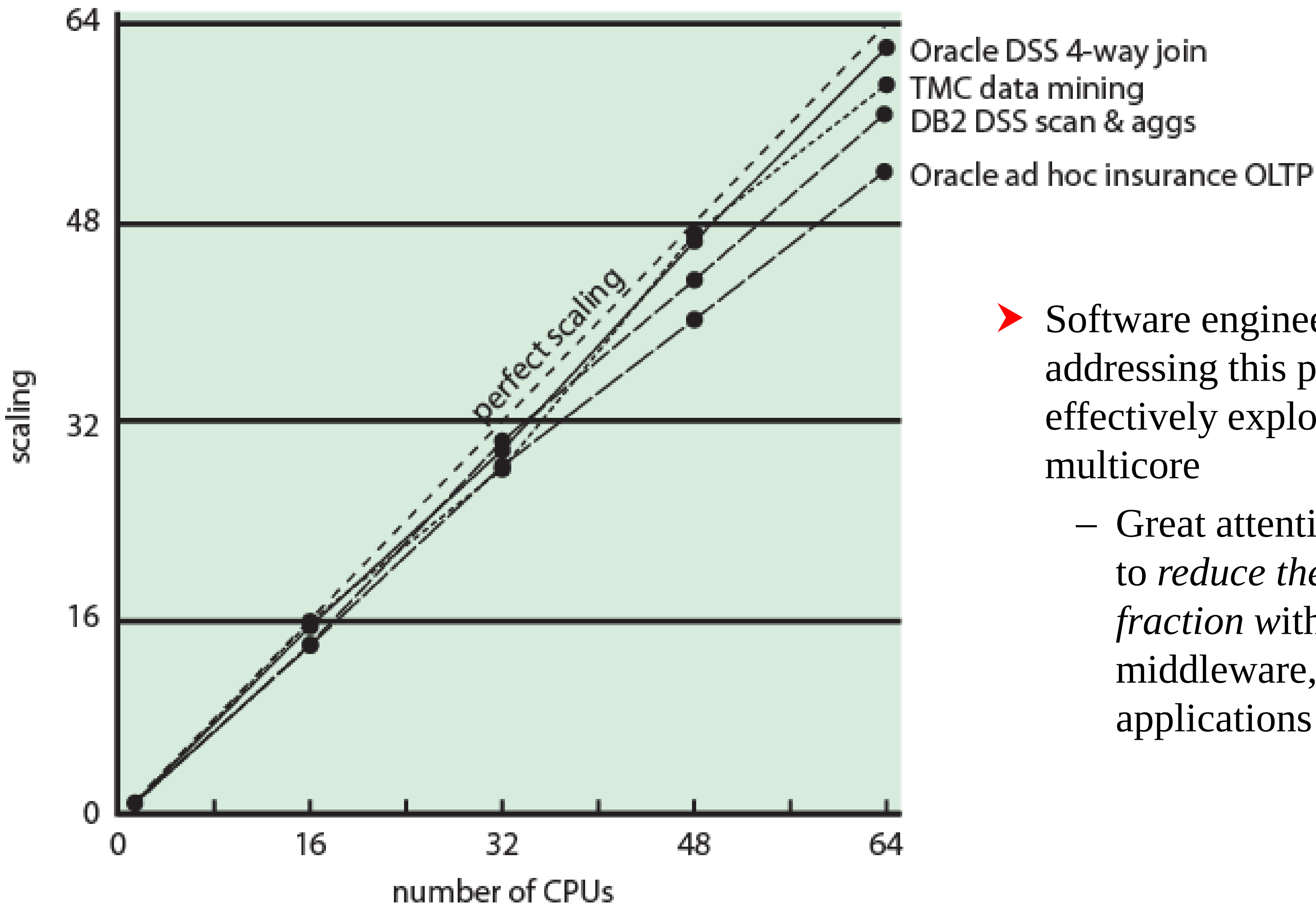
Source: Pearson



(b) Speedup with overheads

- 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



Source: Pearson

- Software engineers have been addressing this problem to effectively exploit the multicore
  - Great attention was paid to *reduce the serial fraction* within HW, OS, middleware, and DB applications

Figure 4.8 Scaling of Database Workloads on Multiple Processors

Applications for Multicores

- **Multithreaded native applications**
  - Characterized by having a small number of highly threaded processes
  - Lotus Domino, Siebel CRM
- **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
- **Multi-instance applications**
  - Can achieve speedup by running multiple instances of the same application in parallel

Solaris

- **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
  - 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

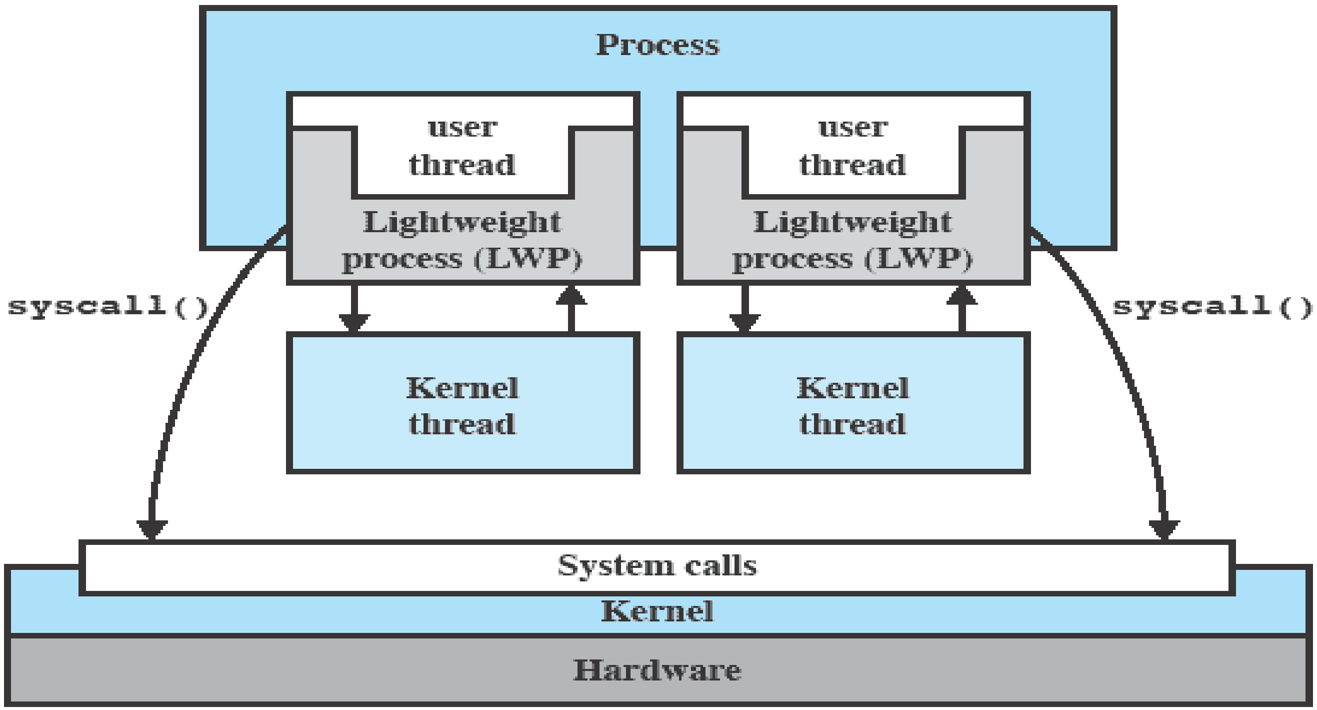


Figure 4.15 Processes and Threads in Solaris [MCDO07]

Source: Pearson

Traditional Unix vs Solaris

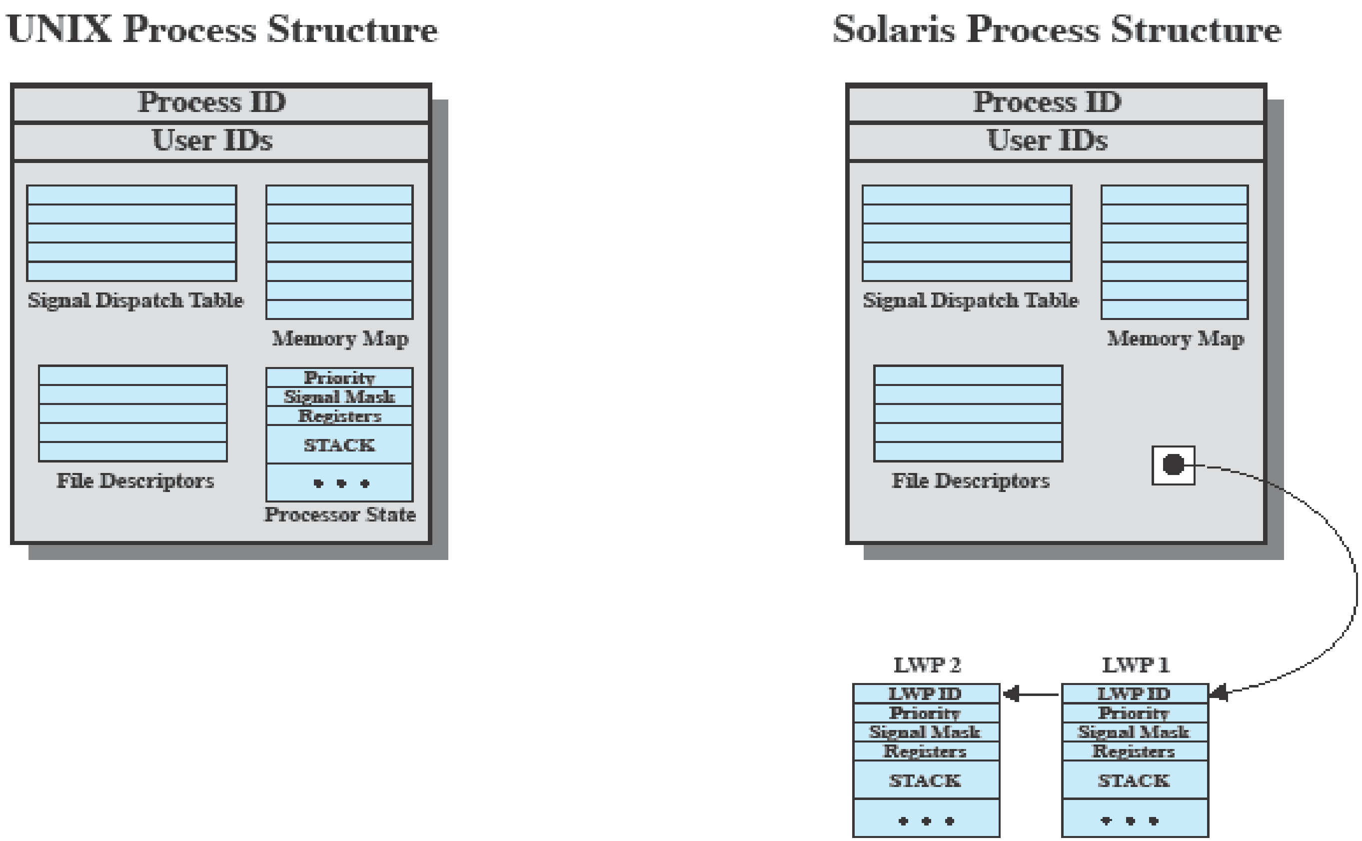


Figure 4.16 Process Structure in Traditional UNIX and Solaris [LEWI96]

Source: Pearson

Solaris Thread States

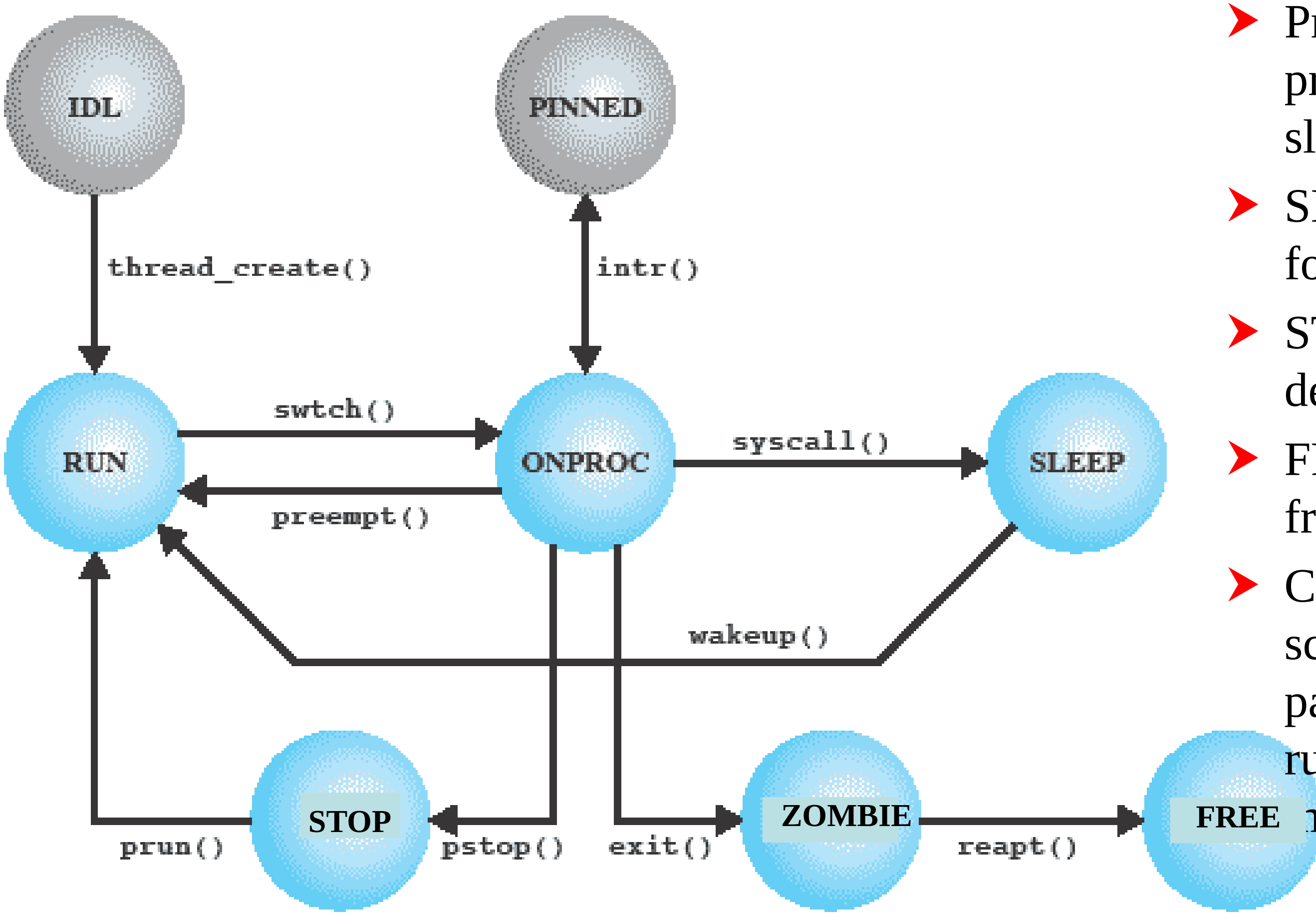


Figure 4.17 Solaris Thread States [MCDO07]

Source: Pearson

- 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
- CPU pinning (or affinity scheduling) fixes a thread to a particular CPU to efficiently run the thread (no cache misses)
  - PINNED thread cannot move to another processor until it is UNPINNED

Homework 3

- 4.1
- 4.3
- 4.5
- 4.7
- 4.10
- Read Chapter 5