Chapter 3

Processes

Part I
Processes & Threads*

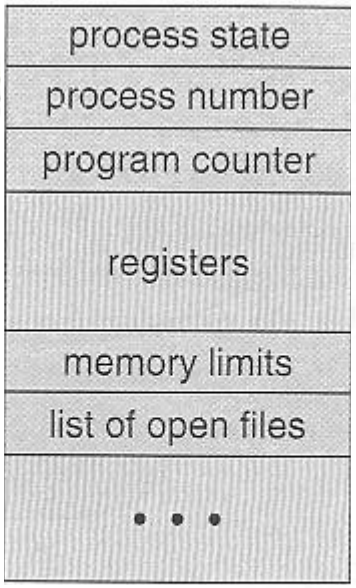
*Referred to slides by Dr. Sanjeev Setia at George Mason University

Process

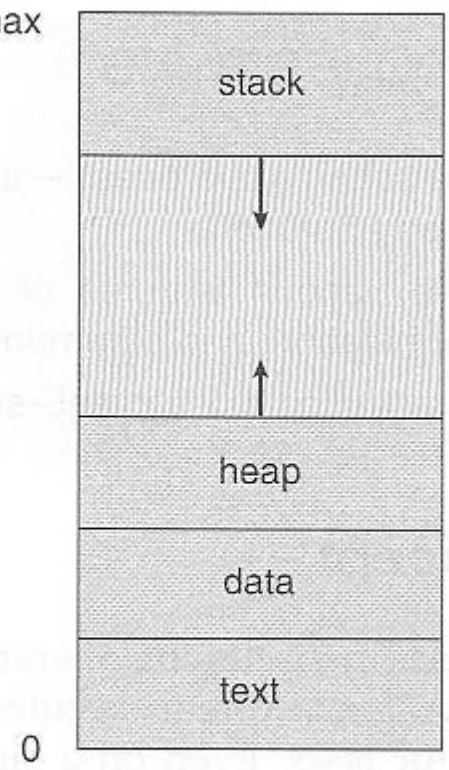
- A program in execution
- An instance of a program running on a computer
- The entity that can be assigned to and executed on a processor
- A unit of activity characterized by
  - the execution of a sequence of instructions
  - a current state
  - an associated set of system resources

Process Concept

- A process includes:
  - program counter
  - code segment
  - stack segment
  - data segment
- Process = Address Space + One thread of control



PCB



Process in Memory

Address Space

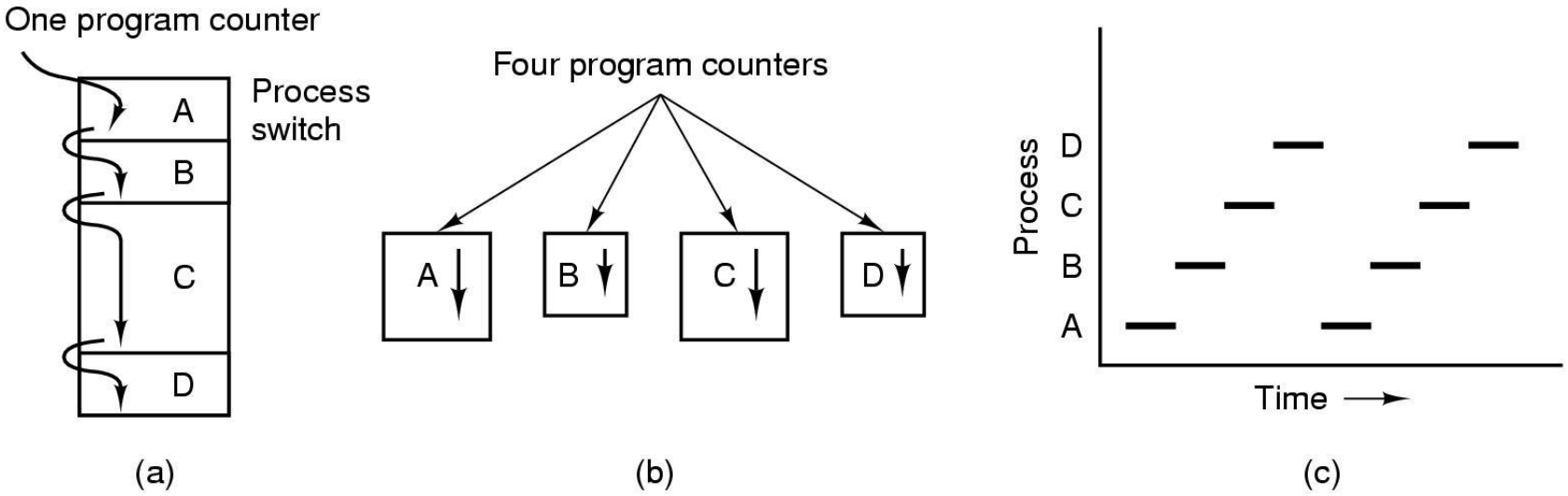
Multiprogramming

- *The interleaved execution of two or more computer programs by a single processor*

- An important technique that
  - enables a time-sharing system
  - allows the OS to overlap I/O and computation, creating an efficient system

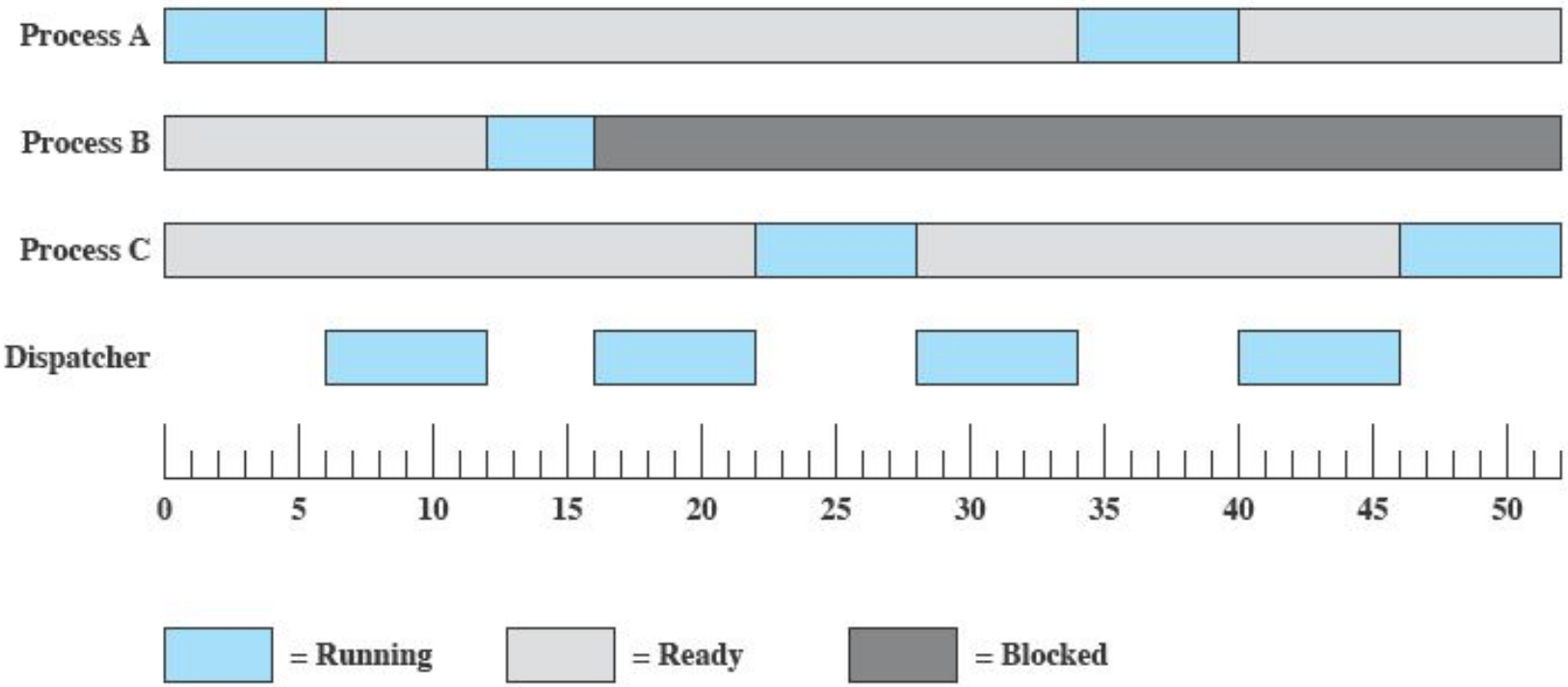
Processes

The Process Model



- Multiprogramming of four programs
- Conceptual model of 4 independent, sequential processes
- Only one program active at any instant

Multiprogramming



Cooperating Processes (I)

- Sequential programs consist of a single process
- Concurrent applications consist of multiple cooperating processes that execute concurrently
- Advantages
  - Can exploit multiple CPUs (hardware concurrency) for speeding up application
  - Application can benefit from software concurrency, e.g., web servers, window systems

Cooperating Processes (II)

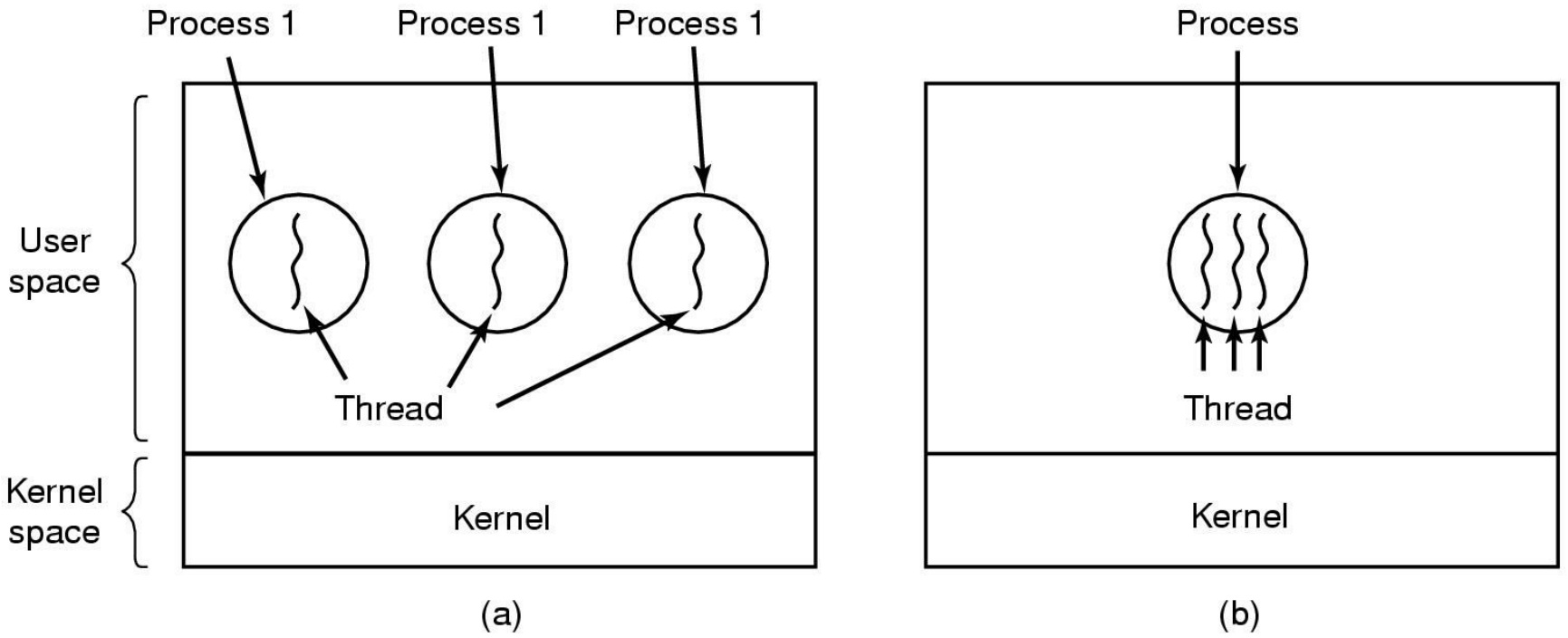
- Cooperating processes need to share information
- Since each process has its own address space, OS mechanisms are needed to let process exchange information
- Two paradigms for cooperating processes
  - Shared Memory
    - OS enables two independent processes to have a shared memory segment in their address spaces
  - Message-passing
    - OS provides mechanisms for processes to send and receive messages

Threads: Motivation

- Process created and managed by the OS kernel
  - Process creation expensive, e.g., fork system call
  - Context switching expensive
  - IPC requires kernel intervention expensive
  - Cooperating processes – no need for memory protection, i.e., separate address spaces

Threads

The Thread Model (1)



(a) Three processes each with one thread

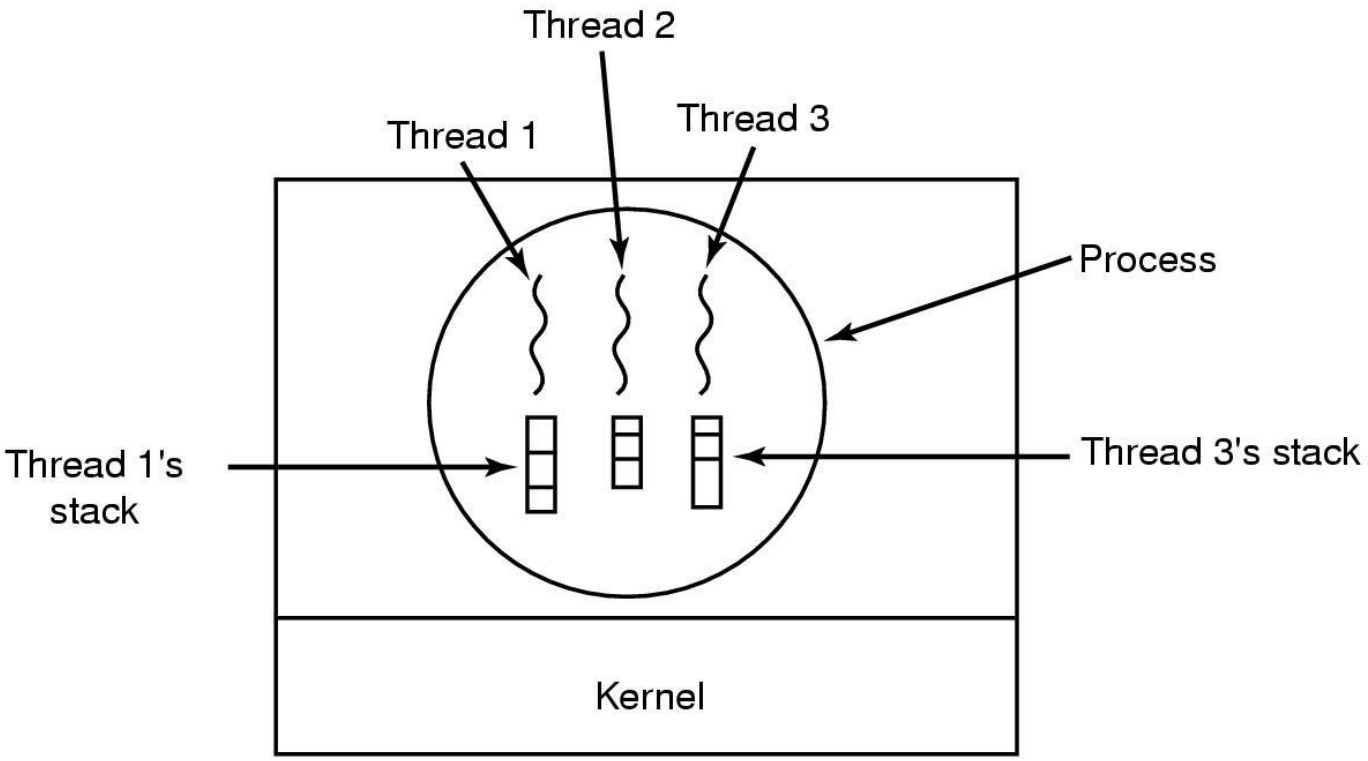
(b) One process with three threads

The Thread Model (2)

Per process items	Per thread items
Address space	Program counter
Global variables	Registers
Open files	Stack
Child processes	State
Pending alarms	
Signals and signal handlers	
Accounting information	

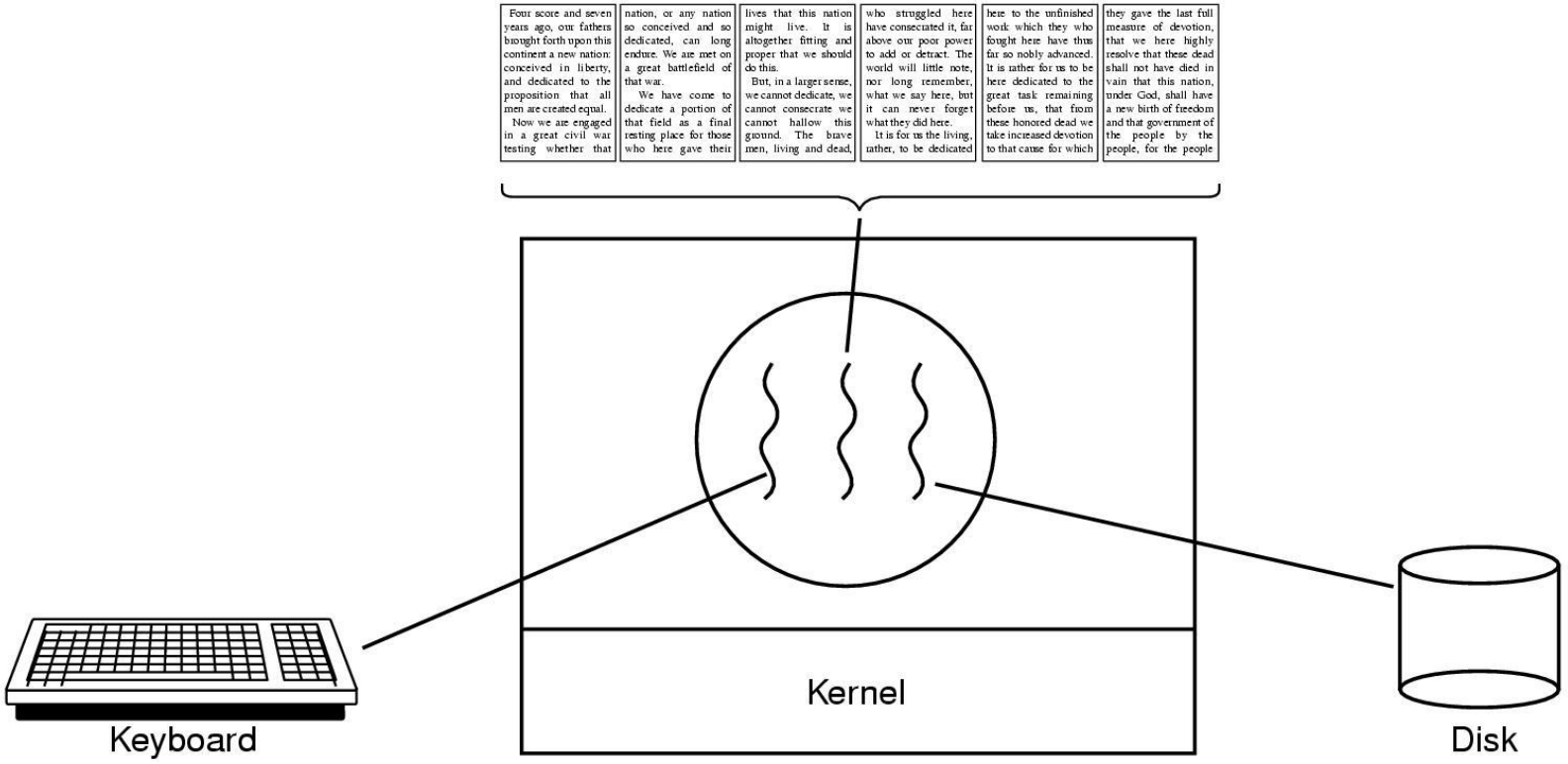
- Items shared by all threads in a process
- Items private to each thread

The Thread Model (3)



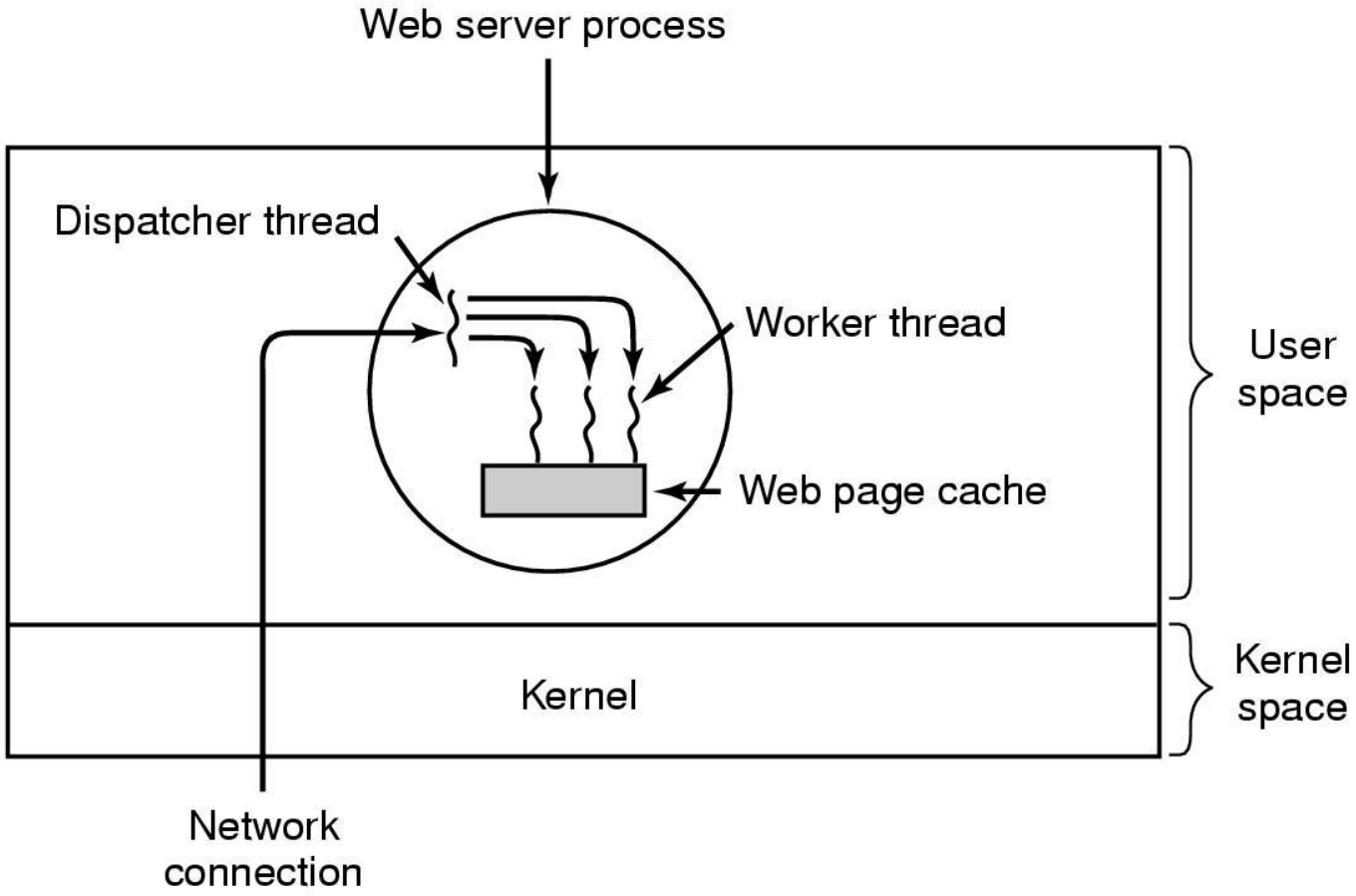
Each thread has its own stack

Thread Usage (1)



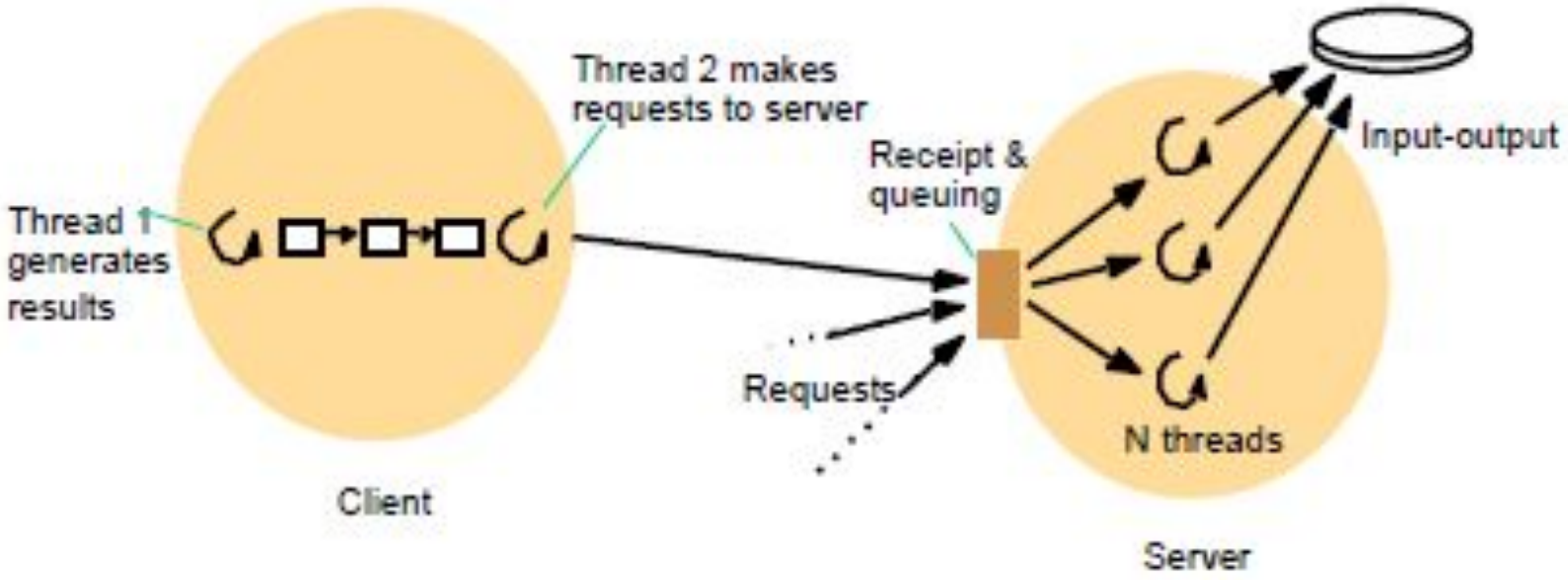
A word processor with three threads

Thread Usage (2)



A multithreaded Web server

Client and server with threads

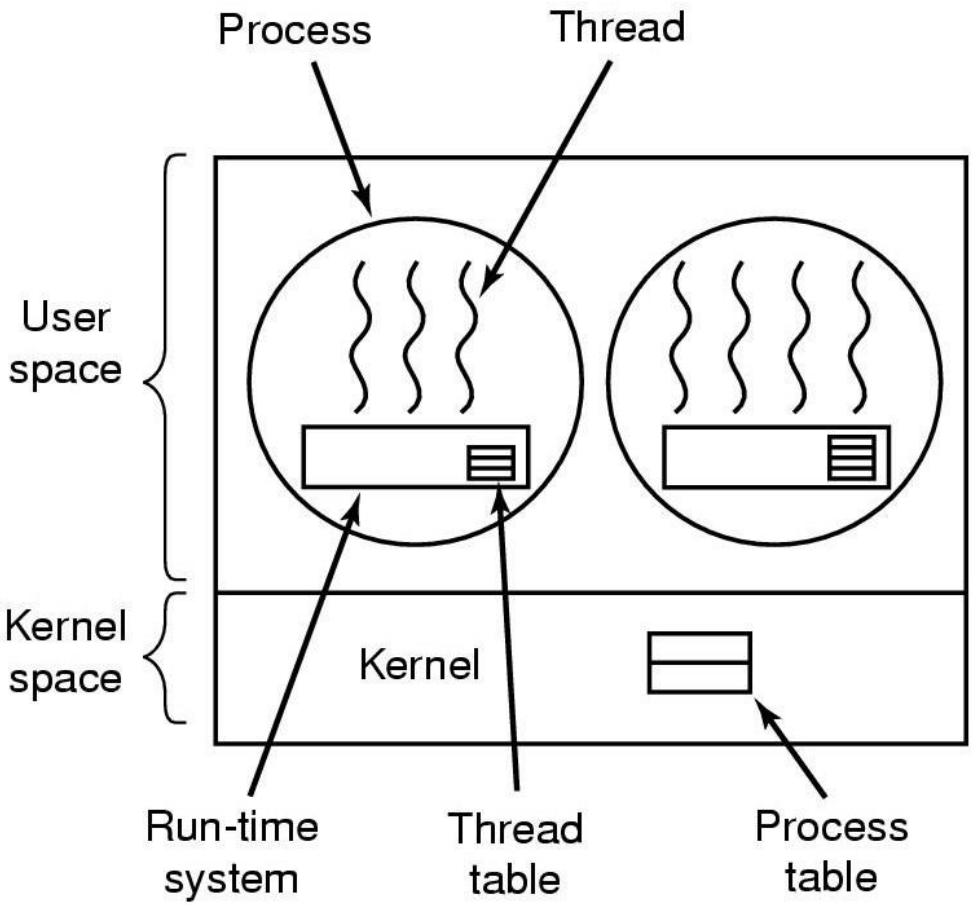


Thread Implementation - Packages

- Threads are provided as a package, including operations to create, destroy, and synchronize them

- A package can be implemented as:
  - User-level threads
  - Kernel threads

Implementing Threads in User Space



A user-level threads package

User-Level Threads

- Thread management done by user-level threads library

- Examples
  - POSIX *Pthreads*
  - Mach *C-threads*
  - Solaris *threads*
  - Java threads

User-Level Threads

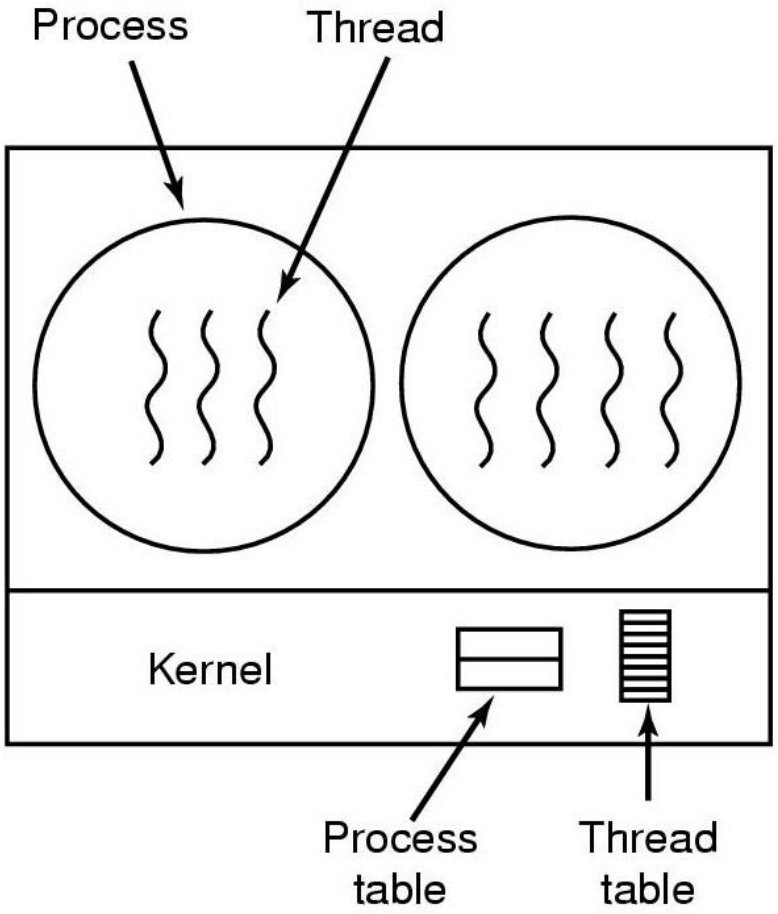
- Thread library entirely executed in user mode
- Cheap to manage threads
  - Create: setup a stack
  - Destroy: free up memory
- Context switch requires few instructions
  - Just save CPU registers
  - Done based on program logic
- A blocking system call blocks all peer threads

Kernel-Level Threads

- Kernel is aware of and schedules threads
- A blocking system call, will not block all peer threads

- Expensive to manage threads
- Expensive context switch
- Kernel Intervention

Implementing Threads in the Kernel



A threads package managed by the kernel

Kernel Threads

- Supported by the Kernel
- Examples: newer versions of
  - Windows
  - UNIX
  - Linux

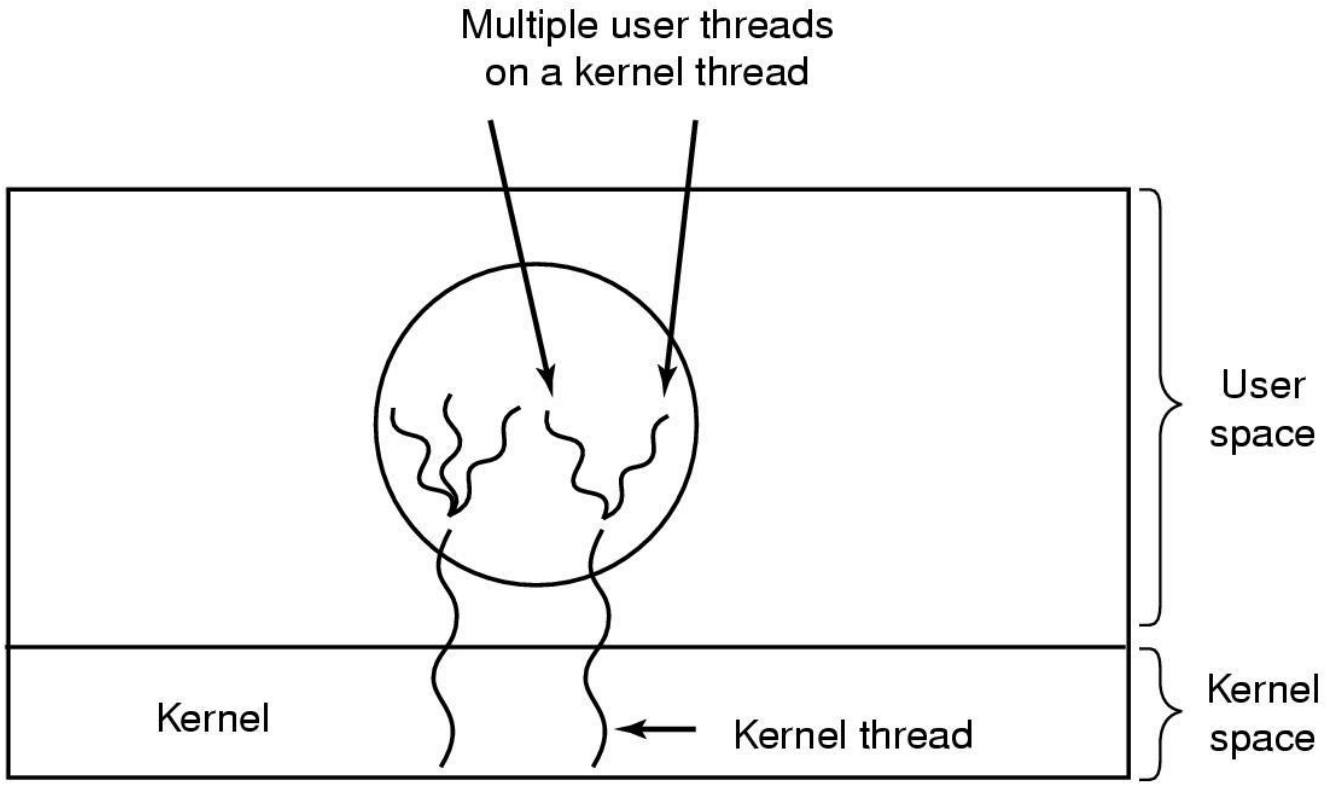
Linux Threads

- Linux refers to them as tasks rather than *threads*.
- Thread creation is done through clone() system call.
- Unlike fork(), clone() allows a child task to share the address space of the parent task (process)

Pthreads

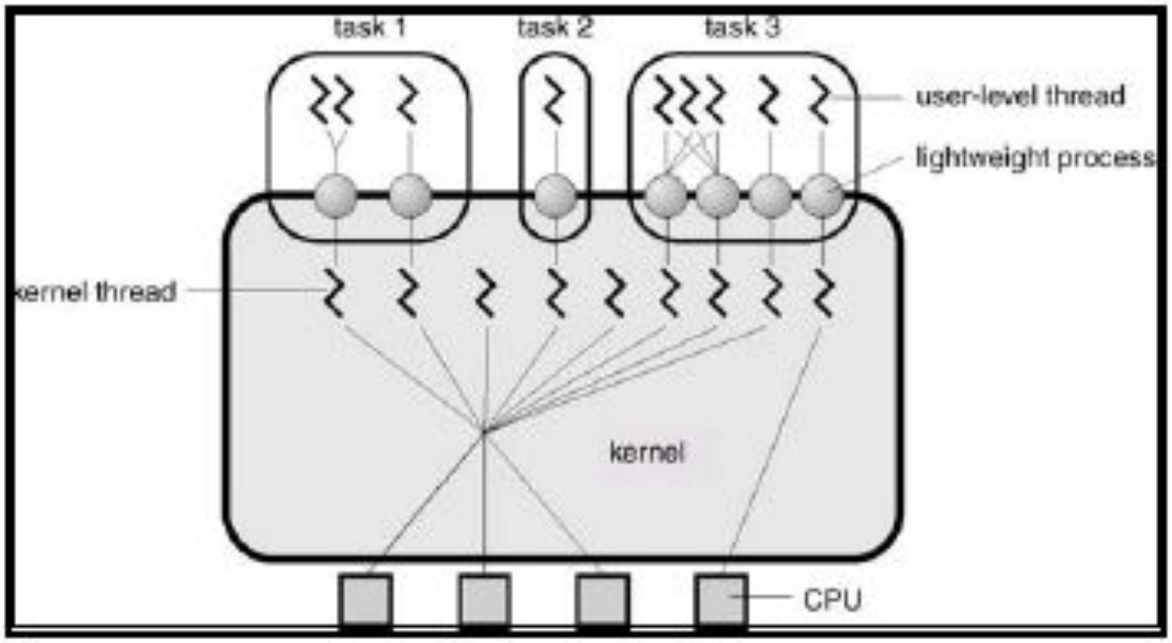
- A POSIX standard (IEEE 1003.1c) API for thread creation and synchronization.
- API specifies behavior of the thread library, implementation is up to development of the library.
- POSIX Pthreads - may be provided as either a user or kernel library, as an extension to the POSIX standard.
- Common in UNIX operating systems.

Hybrid Implementations



Multiplexing user-level threads onto kernel-level threads

Solaris Threads (LWP)



LWP Advantages

- Cheap user-level thread management
- A blocking system call will not suspend the whole process
- LWPs are transparent to the application
- LWPs can be easily mapped to different CPUs