LOVELY PROFESSIONAL UNIVERSITY

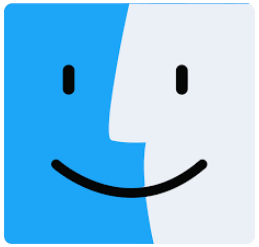
CSE316
OPERATING SYSTEMS

Lecture #0

The kick start session

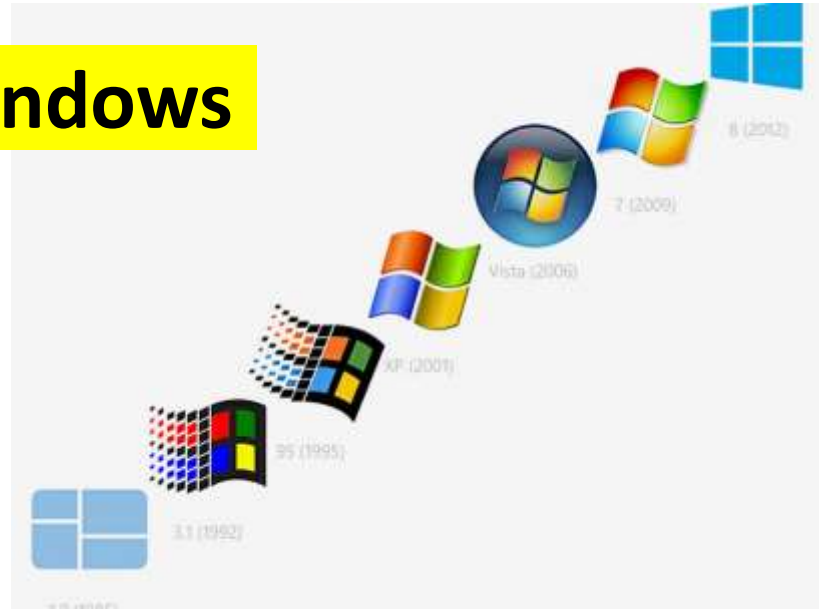
Operating Systems

Linux



MAC OS

Windows



Android

Course details

- LTP – 3 0 0 [Three lectures/week]
- Credit – 3

Assessment/Evaluation Scheme

- Attendance: 5%
- CA: 25%
- MTT: 20%
- ETT: 50%

Complete evaluation criteria for the course

CA3: 30 marks- Test (Simulation Based Assignment)

To assess the students' logic building and programming skills. Scenario-based problems on operating system concepts will be given to code in C language. The students will code and submit the code online with in stipulated time.

NOTE: The CA3 will be allocated in 3rd Week and student will submit it in 10th Week.

CA1: 30 marks- Test (Online Quiz of 30 Questions)

To assess the students' understanding on Introduction to Operating System, Process Management, CPU Scheduling, and Process Synchronization

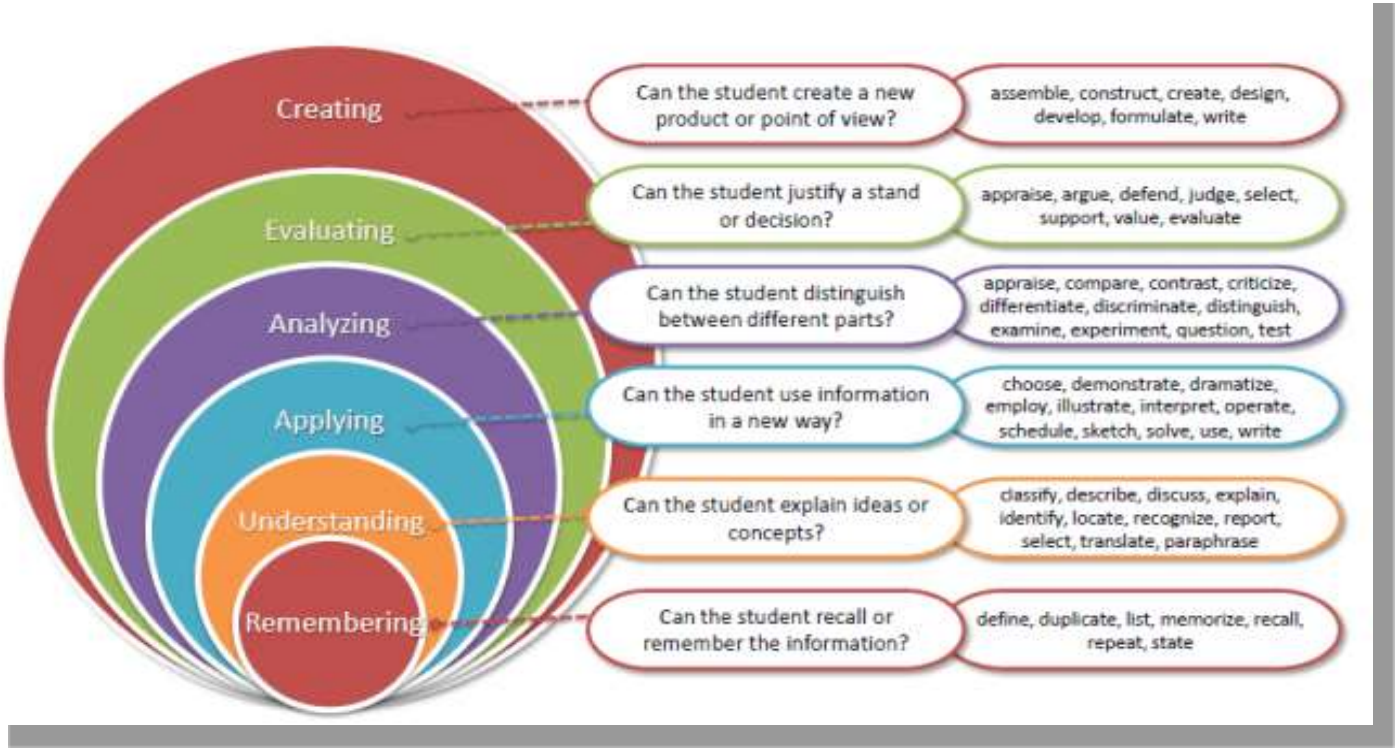
NOTE: The CA2 will be allocated in 5th Week and conducted in 6th Week.

CA2: 30 marks- Test (Class Test-Student will write the answers on test sheets)

To assess the students' understanding on Threads, Deadlock, Protection and Security, and Memory Management.

NOTE: The CA will be allocated in 11th Week and conducted in 12th Week.

Revised Bloom's Taxonomy



Course outcomes

Through this course students should be able to:
CO1 :: understand the role, functionality and layering of the system software components.

CO2 :: use system calls for managing processes, memory and the file system.

CO3 :: Analyze important algorithms eg process scheduling and memory management algorithms.

CO4 :: use and outline the various security measures that ensure threat free operation of a system.

CO5 :: apply various operations on processes, threads and analyze methods to synchronize their execution.

CO6 :: simulate inter-process communication techniques like message passing and shared memory.

Program Outcomes

PO1

Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2

Problem analysis::Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3

Design/development of solutions::Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4

Conduct investigations of complex problems::Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

Program Outcomes

PO5

Modern tool usage::Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO6

The engineer and society::Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7

Environment and sustainability::Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8

Ethics::Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9

Individual and team work::Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

Program Outcomes

PO10

Communication::Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11

Project management and finance::Demonstrate knowledge and understanding of the engineering, management principles and apply the same to one's own work, as a member or a leader in a team, manage projects efficiently in respective disciplines and multidisciplinary environments after consideration of economic and financial factors.

PO12

Life-long learning::Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PO13

Competitive Skills::Ability to compete in national and international technical events and building the competitive spirit alongwith having a good digital footprint.

Course contents

Unit I

Introduction to Operating System : Operating System Meaning, Supervisor & User Mode, review of computer organization, introduction to popular operating systems like UNIX, Windows, etc., OS structure, system calls, functions of OS, evolution of OS

Process Management : PCB, Operations on Processes, Co-operating and Independent Processes, Inter-Process Communication, Process states, Operations on processes, Process management in UNIX, Process concept, Life cycle, Process and threads

Course contents

Unit II

CPU Scheduling : Types of Scheduling, Scheduling Algorithms, Scheduling criteria, CPU scheduler - preemptive and non preemptive, Dispatcher, First come first serve, Shortest job first, Round robin, Priority, Multi level feedback queue, multiprocessor scheduling, real time scheduling, thread scheduling

Course contents

Unit III

Process Synchronization : Critical Section Problem, Semaphores, Concurrent processes, Cooperating processes, Precedence graph, Hierarchy of processes, Monitors, Dining Philosopher Problem, Reader-writer Problem, Producer consumer problem, classical two process and n-process solutions, hardware primitives for synchronization

Threads : Overview, Multithreading Models, scheduler activations, examples of threaded programs

Course contents

Unit IV

Deadlock : Deadlock Characterization, Handling of deadlocks- Deadlock Prevention, Deadlock Avoidance & Detection, Deadlock Recovery, Starvation

Protection and Security : Need for Security, Security Vulnerability like Buffer overflow, Trapdoors, Backdoors, cache poisoning etc, Authentication-Password based Authentication, Password Maintenance & Secure Communication, Application Security - Virus, Program Threats, Goals of protection, Principles of protection, Domain of protection, Access matrix, implementation of access matrix, System and network threats, Examples of attacks

Course contents

Unit V

Memory Management : Logical & Physical Address Space, Swapping, Contiguous Memory allocation, Paging, Segmentation, Page replacement algorithms, Segmentation - simple, multi-level and with paging, Page interrupt fault, Fragmentation - internal and external, Schemes - Paging - simple and multi level, Overlays - swapping, Virtual memory concept, Demand paging

Course contents

Unit VI

File Management : File Concepts, Access methods, Directory Structure, File System Mounting and Sharing, Protection, Allocation methods, Free-Space Management, Directory Implementation

Device management : Dedicated, shared and virtual devices, Serial access and direct access devices, Disk scheduling methods, Direct Access Storage Devices – Channels and Control Units

Inter process communication : Introduction to IPC (Inter process communication) Methods, Pipes - popen and pclose functions, Co-processes, Shared memory, Stream pipes, FIFOs, Message queues, Passing File descriptors, Semaphores

Text & Reference Books

Text Books:

1. OPERATING SYSTEM CONCEPTS by ABRAHAM SILBERSCHATZ, PETER B. GALVIN, GERG GAGNE, WILEY

References:

1. DESIGN OF THE UNIX OPERATING SYSTEM by MAURICE J. BACH, Pearson Education India
2. REAL-TIME SYSTEMS by JANE W. S. LIU, Pearson Education India

Importance of the course

- Many of the recruiters (like TCS, CTS, HCL, IBM, Tech Mahindra, Amdocs etc.) ask operating systems in Technical Quiz and Technical Interview round.
- This course is very important for GATE aspirants, it's having good weightage in GATE.

Importance of the course

- https://nptel.ac.in/courses/106105214

- https://onlinecourses.nptel.ac.in/noc20_cs04/preview

Next Class: Introduction to Operating System

Chapter 1: Introduction

(OS Structure, Modes and Services)

What is an Operating System?

- A program that acts as an intermediate/ interface between a user of a computer and the computer hardware.
- Resource allocator
- Control Program

- Operating system goals:
  - Execute user programs and make solving user problems easier
  - Make the computer system convenient to use
  - Use the computer hardware in an efficient manner

Computer System Structure

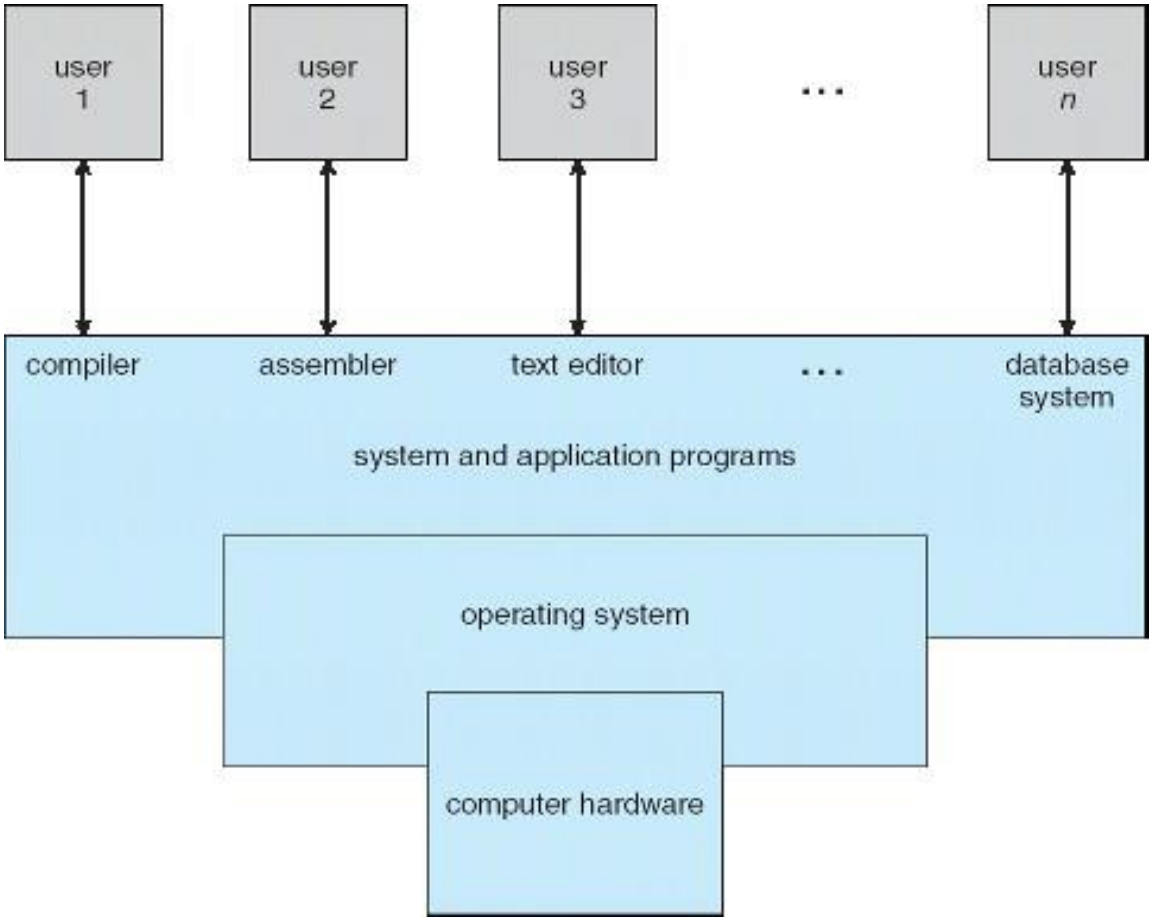
- Computer system can be divided into four components
  - **1. Hardware** – provides basic computing resources CPU, memory, I/O devices
  - **2. Operating system**

    Controls and coordinates use of hardware among various applications and users
  - **3. Application programs** – define the ways in which the system resources are used to solve the computing problems of the users

    Word processors, compilers, web browsers, database systems, video games
  - **4. Users**

    People, machines, other computers

Four Components of a Computer System



Operating System Definition

To understand more fully the OS role, we explore OS from 2 view points.:

1. User view: In single user, it should be easy to use.

In other cases, where users access the same user through different terminals, More emphasize is on resource allocation and utilization.

2. System View:

- OS is a **resource allocator**
  - Manages all resources
  - Decides between conflicting requests for efficient and fair resource use
- OS is a **control program**
  - Controls execution of programs to prevent errors and improper use of the computer

Who controls the execution of programs to prevent errors and improper use of computer?

a) Resource allocator

b) Control Program

c) Hardware

d)None of the above

(b)

Operating-System Operations

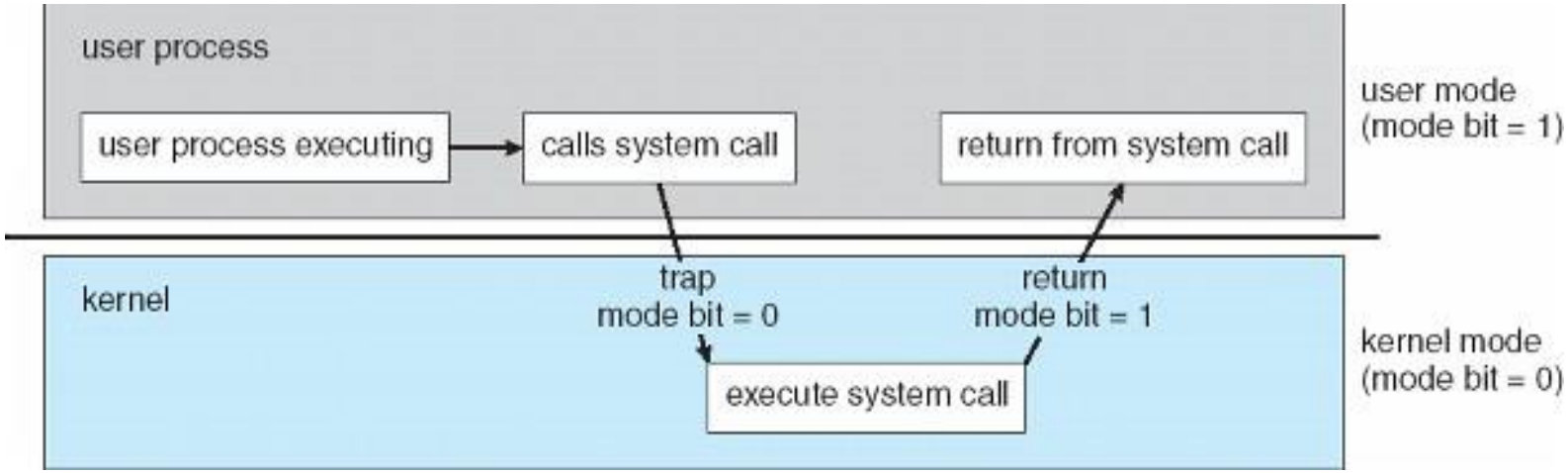
- Modern OS's are Interrupt driven.
- Program or software send generate events by using system calls. Error or request by a software creates **exception** or **trap**
  - Division by zero, request for operating system service

- **Dual-mode operation** allows OS to protect itself and other system components
  - **User mode** and **kernel mode**

  - **Mode bit** provided by hardware

    Provides ability to distinguish when system is running user code or kernel code

Transition from User to Kernel Mode



The operating system switches from user mode to kernel mode so the mode bit will change from?

a) 0 to 1

b) 1 to 0

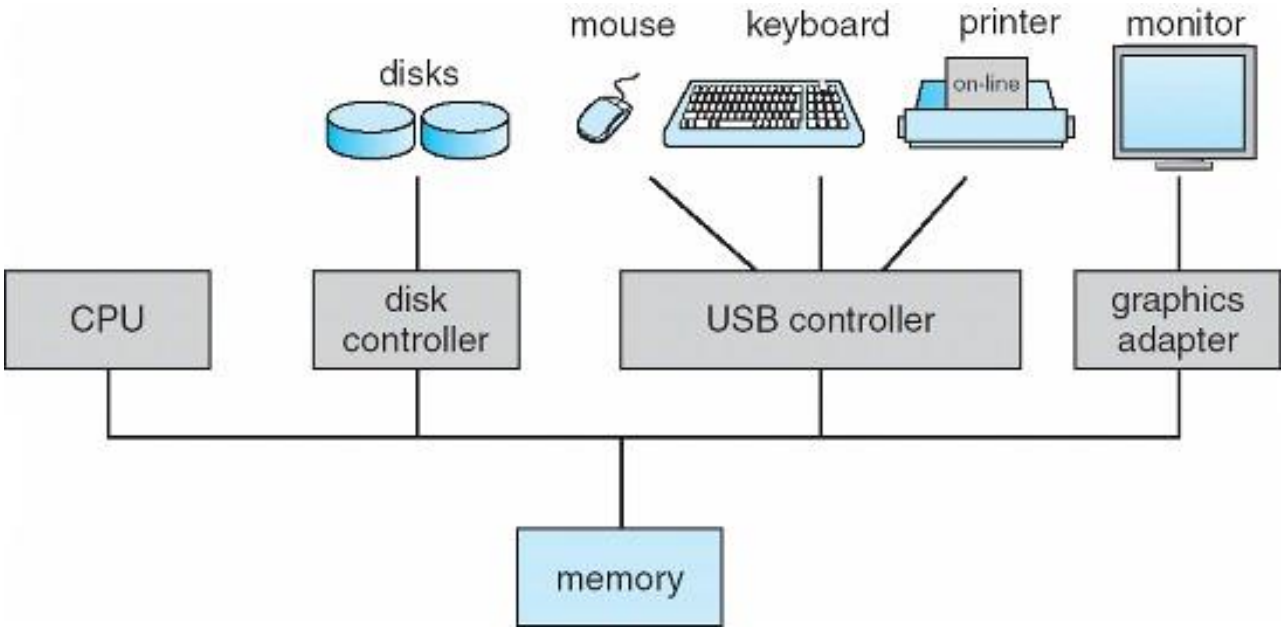
c) Remain constant

d)None

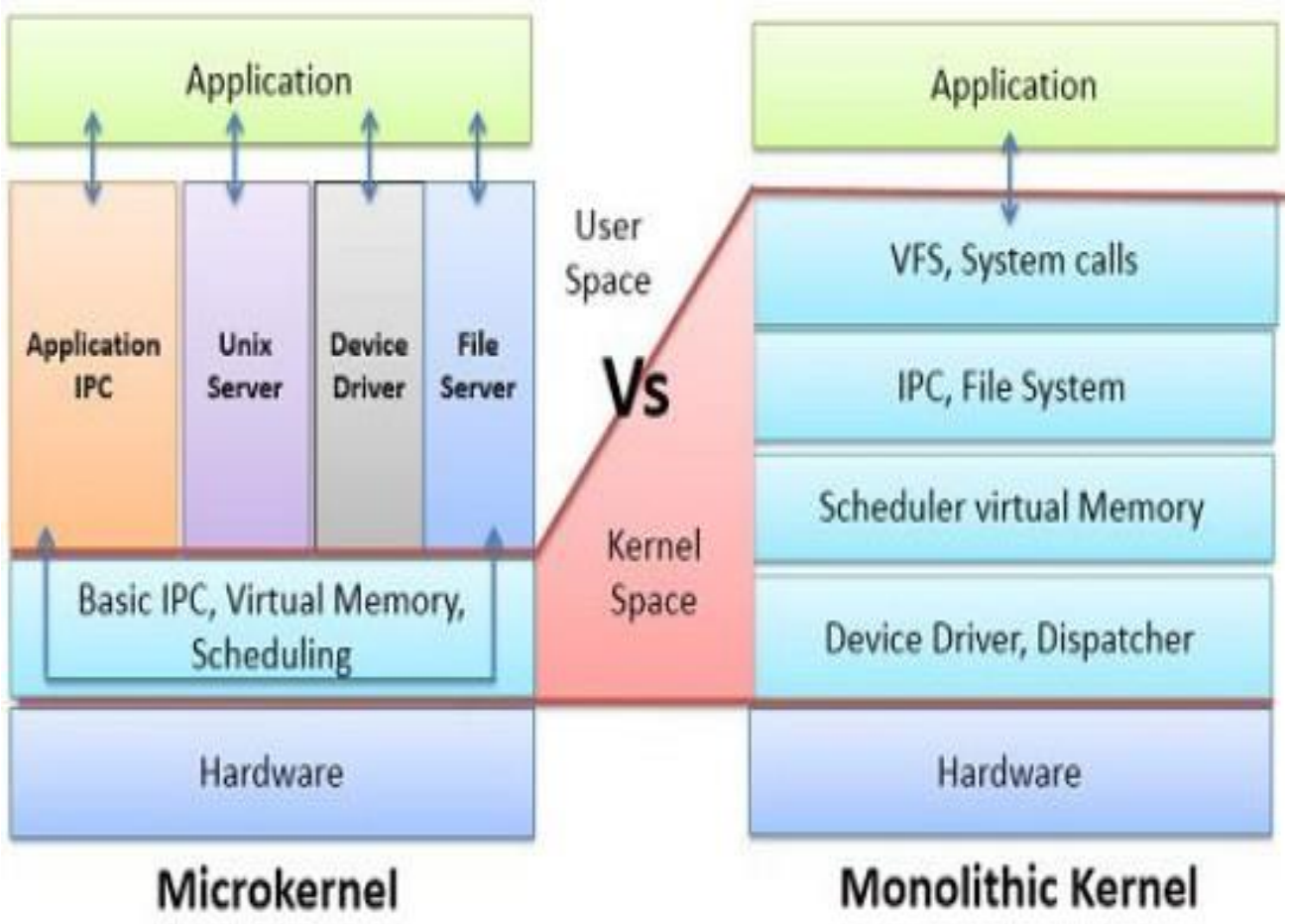
(b)

Computer System Organization

- Computer-system operation
  - One or more CPUs, device controllers connect through common bus providing access to shared memory
  - Concurrent execution of CPUs and devices competing for memory cycles



Types of kernel



Key Differences Between Microkernel and Monolithic Kernel

The basic point on which microkernel and monolithic kernel is distinguished is that **microkernel** implement user services and kernel services in **different address spaces** and **monolithic kernel** implement both user services and kernel services under **same address space**

The size of microkernel is **small** as only kernel services reside in the kernel address space. However, the size of monolithic kernel is comparatively **larger** than microkernel because both kernel services and user services reside in the same address space.

The execution of monolithic kernel is **faster** as the communication between application and hardware is established using the **system call**. On the other hands, the execution of microkernel is **slow** as the communication between application and hardware of the system is established through **message passing**

It is easy to extend microkernel because new service is to be added in user address space that is isolated from kernel space, so the kernel does not require to be modified. Opposite is the case with monolithic kernel if a new service is to be added in monolithic kernel then entire kernel needs to be modified.

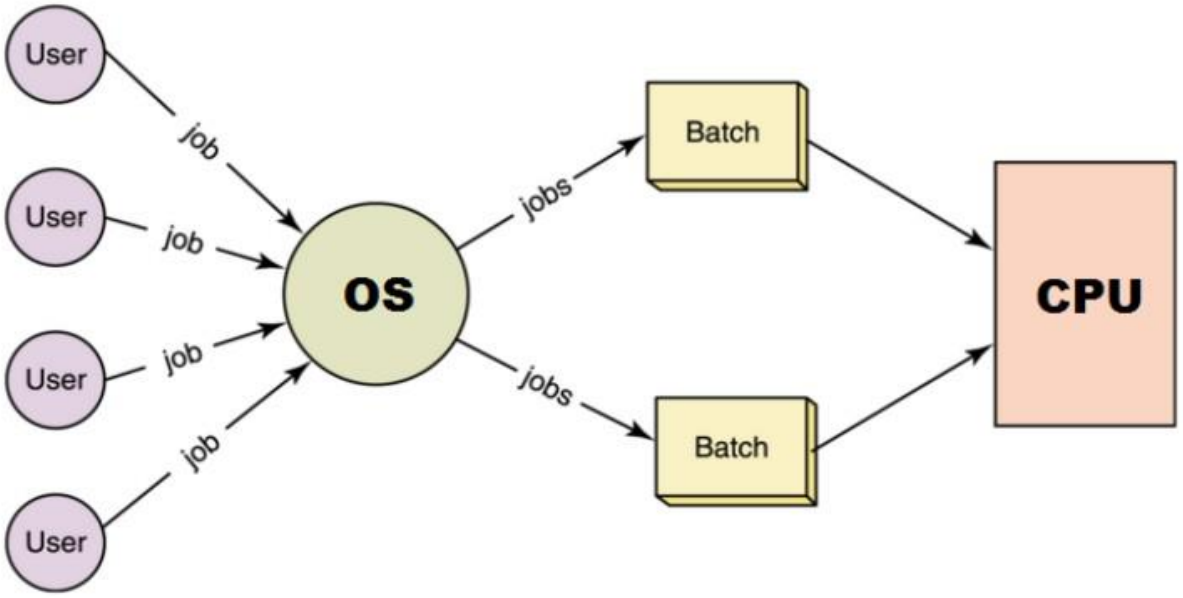
Microkernel is more **secure** than monolithic kernel as if a service fails in microkernel the operating system remain unaffected. On the other hands, if a service fails in monolithic kernel entire system fails.

Monolithic kernel designing requires **less code**, which further leads to fewer bugs. On the other hands, microkernel designing needs more code which further leads to more bugs.

TYPES OF OS

Batch Systems

“Batch operating system. The users of a batch operating system do not interact with the computer directly. Each user prepares his job on an off-line device like punch cards and submits it to the computer operator. To speed up processing, jobs with similar needs are batched together and run as a group” *



*[https://www.tutorialspoint.com/operating_system/os_types.htm]

TYPES OF OS

Batch Systems

- Early computers were Physically enormous machines run from a console
- The common input devices were card readers and tape drives.
- The common output devices were line printers, tape drives, and card punches.
- The user did not interact directly with the computer systems.
- User prepare a job -which consisted of the program, and submitted it to the computer operator.
- after minutes, hours, or days, the output appeared.
- To speed up processing, operators batched jobs with similar needs together and ran them through the computer as a group.

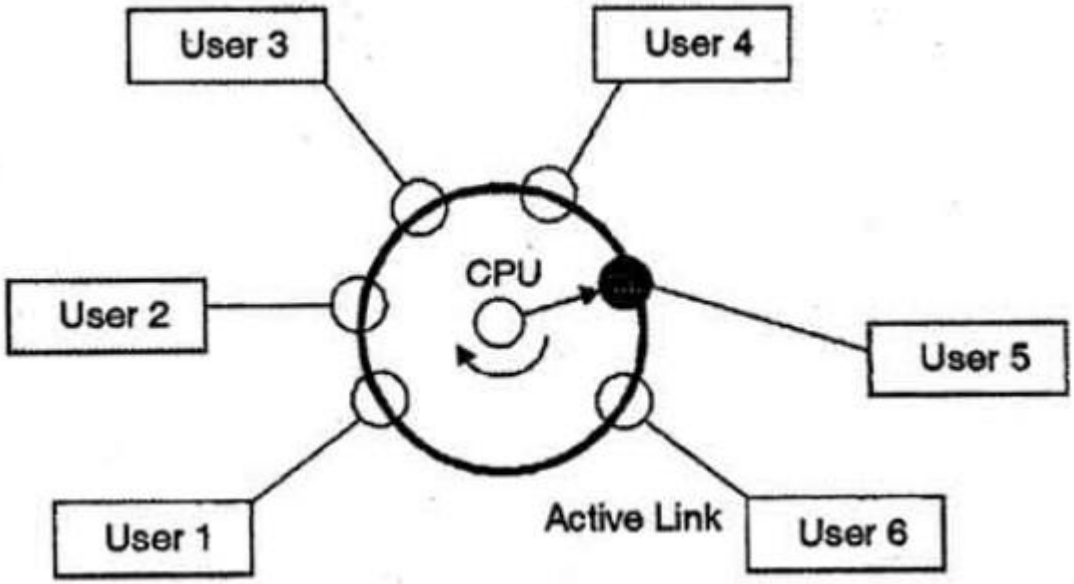
Multiprogrammed OS

- Needed for efficiency
- Single user cannot keep CPU and I/O devices busy at all times
- Multiprogramming organizes jobs (code and data) so CPU always has one to execute
- A subset of total jobs in system is kept in memory
- One job selected and run via **job scheduling**
- When it has to wait (for I/O for example), OS switches to another job

Timesharing OS

- **Timesharing (multitasking)** is logical extension in which CPU switches jobs so frequently that users can interact with each job while it is running, creating **interactive** computing
  - Each user has at least one program executing in memory □**process**
  - If several jobs ready to run at the same time □ **CPU scheduling**
  - If processes don’t fit in memory, **swapping** moves them in and out to run
  - **Virtual memory** allows execution of processes not completely in memory

Timesharing OS

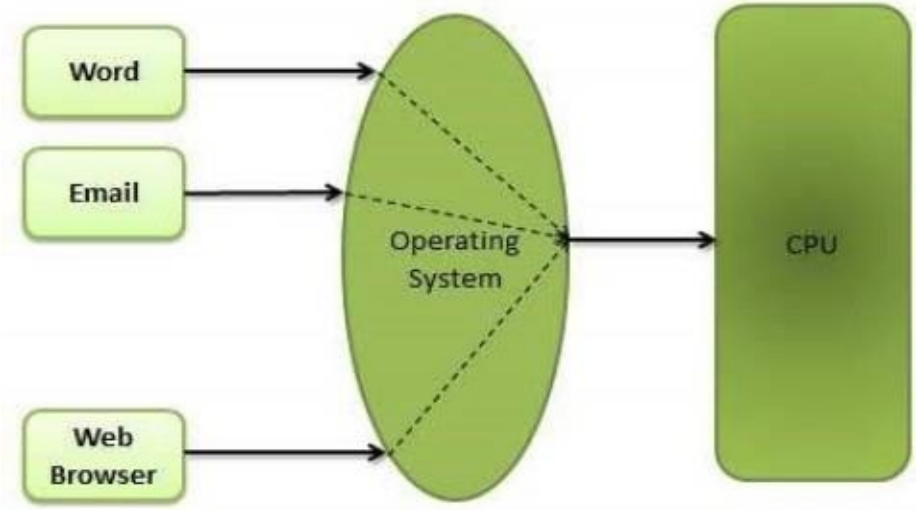


Multitasking Systems

- Types of Multitasking:

1. **Preemptive:** the operating system parcels CPU *time slices* to each program.
2. **Cooperative:** each program can control the CPU for as long as it needs it.

   If a program is not using the CPU, however, it can allow another program to use it temporarily.

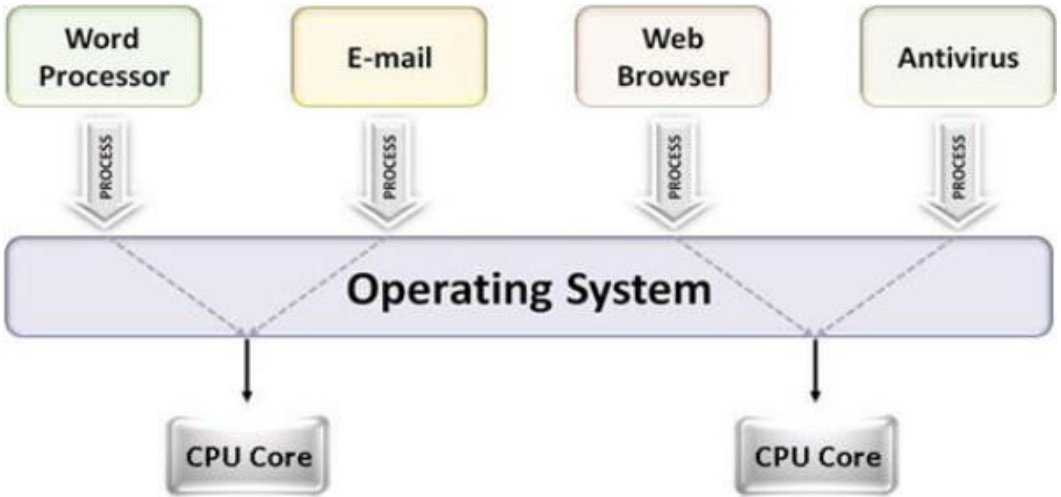


Multiprocessing OS

- Multi-processor systems; that is, they **have multiple CPU.**

Exp: dual core processor has 2 process cycles

- **Also known as parallel systems** or **tightly coupled systems**
- **Such systems have more than one processor** in close communication, sharing the computer bus, the clock, and sometimes memory and peripheral devices.



Distributed Systems

- A network is a communication path between two or more systems.
- Each system over the network keeps copy of the data, and this leads to Reliability (Because if one system crashes , data is not lost).

- CLIENT SERVER SYSTEMS
- PEER TO PEER SYSTEMS

Real Time Systems

- **Time bound systems**
- Real time systems are of 2 types:

- **1. Soft Real time Systems:** Process should complete in specific time but May have some delay (Positive delay) and will nor harm the system.
- **Exp: Session expires but can be re-logged in.**

- **2. Hard Real Time Systems:** Each process is assigned a specific time instance, and Process must complete in that time otherwise system will crash.

Real Time Embedded Systems

- is a computing environment that **reacts to input within a specific time period.**
- Time Driven
- Task specific
- Exp: Microwave, Washing Machine…

Q. In which type of operating system users do not interact directly with the computer system?

a) Multiprogramming operating systems

b) Multiprocessing operating systems

c) Batch operating systems

d)Distributed operating systems

(c)

Q. What is the objective of multiprogramming operating systems?

a) Maximize CPU utilization

b) Switch the CPU among processes

c) Achieve multitasking

d)None of the above

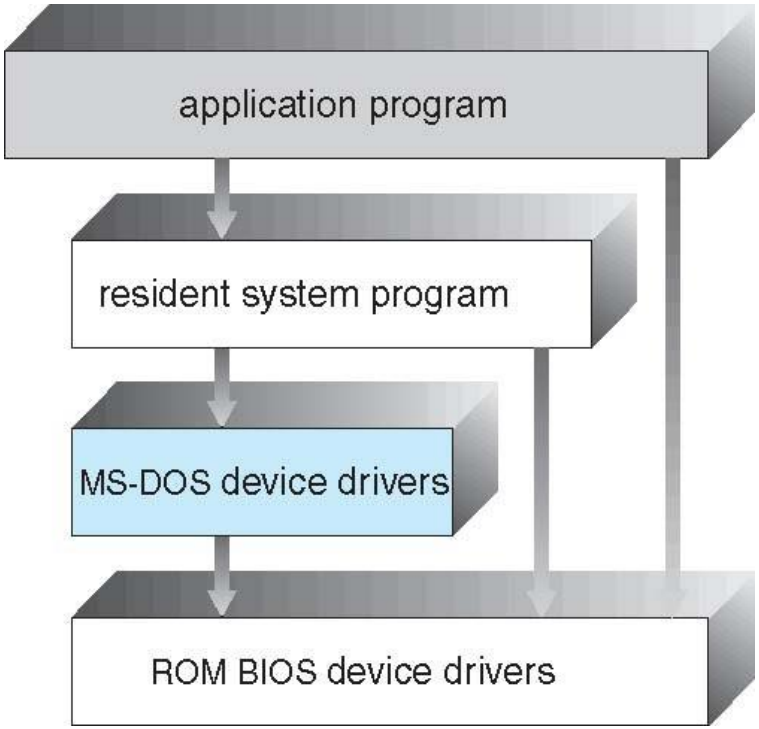
(a)

Operating System Structure

- General-purpose OS is very large program
- Various ways to structure ones
  - Simple structure – MS-DOS
  - More complex -- UNIX
  - Layered – an abstraction
  - Microkernel -Mac

Simple Structure -- MS-DOS

- MS-DOS – written to provide the most functionality in the least space
  - Not divided into modules



Non Simple Structure -- UNIX

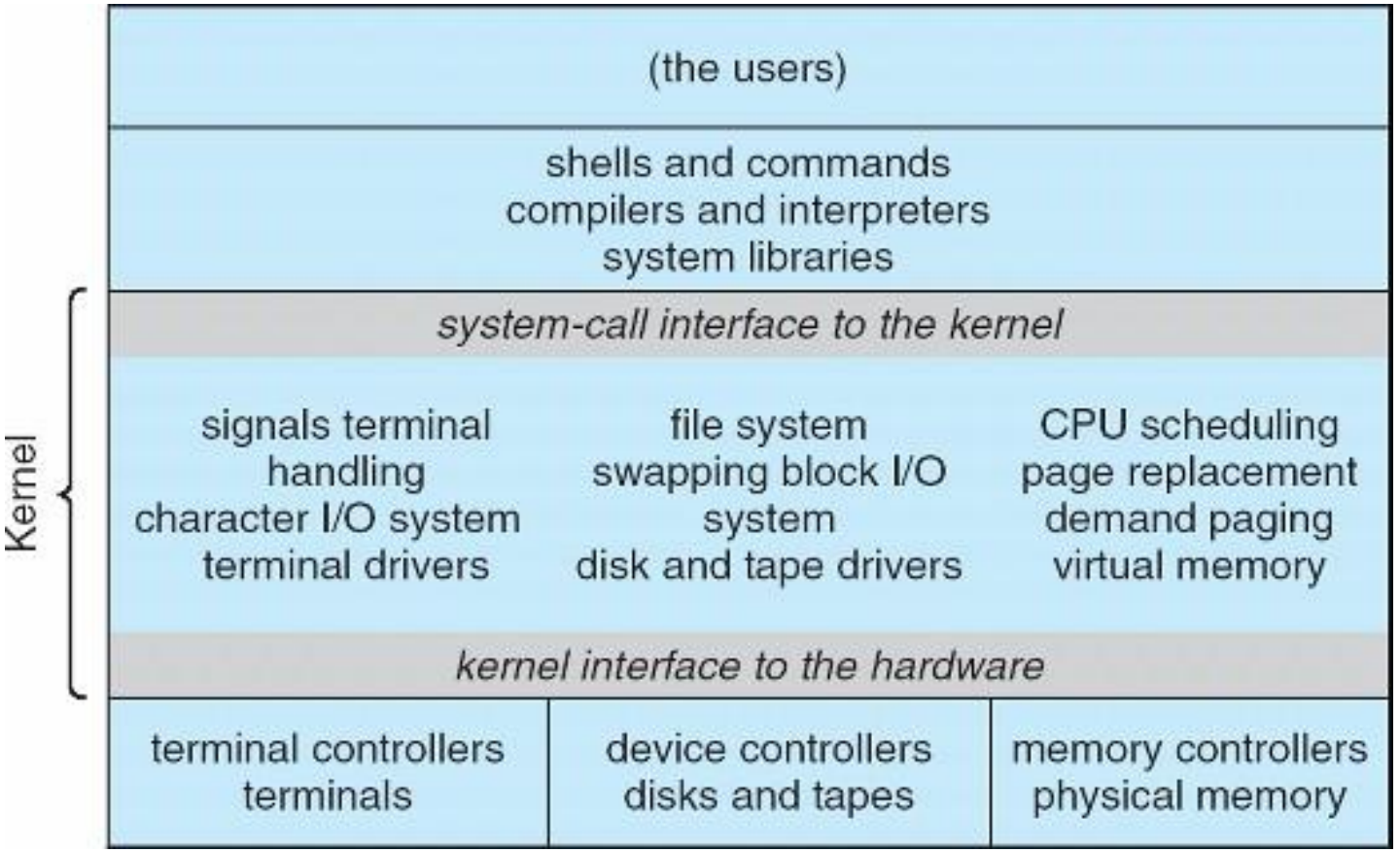
UNIX – limited by hardware functionality

The UNIX OS consists of two separable parts:

- Systems programs
- The kernel

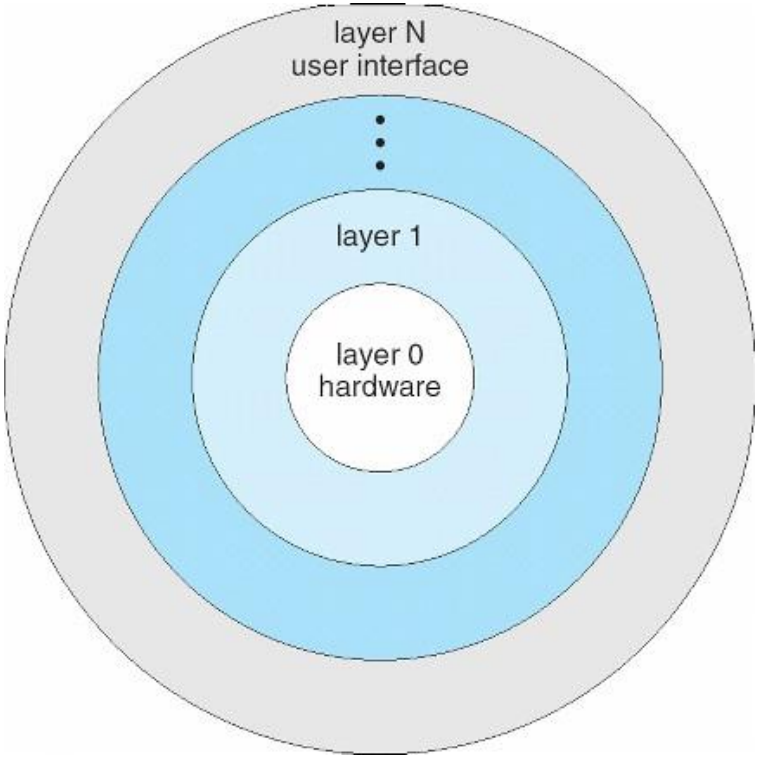
Traditional UNIX System Structure

Beyond simple but not fully layered



Layered Approach

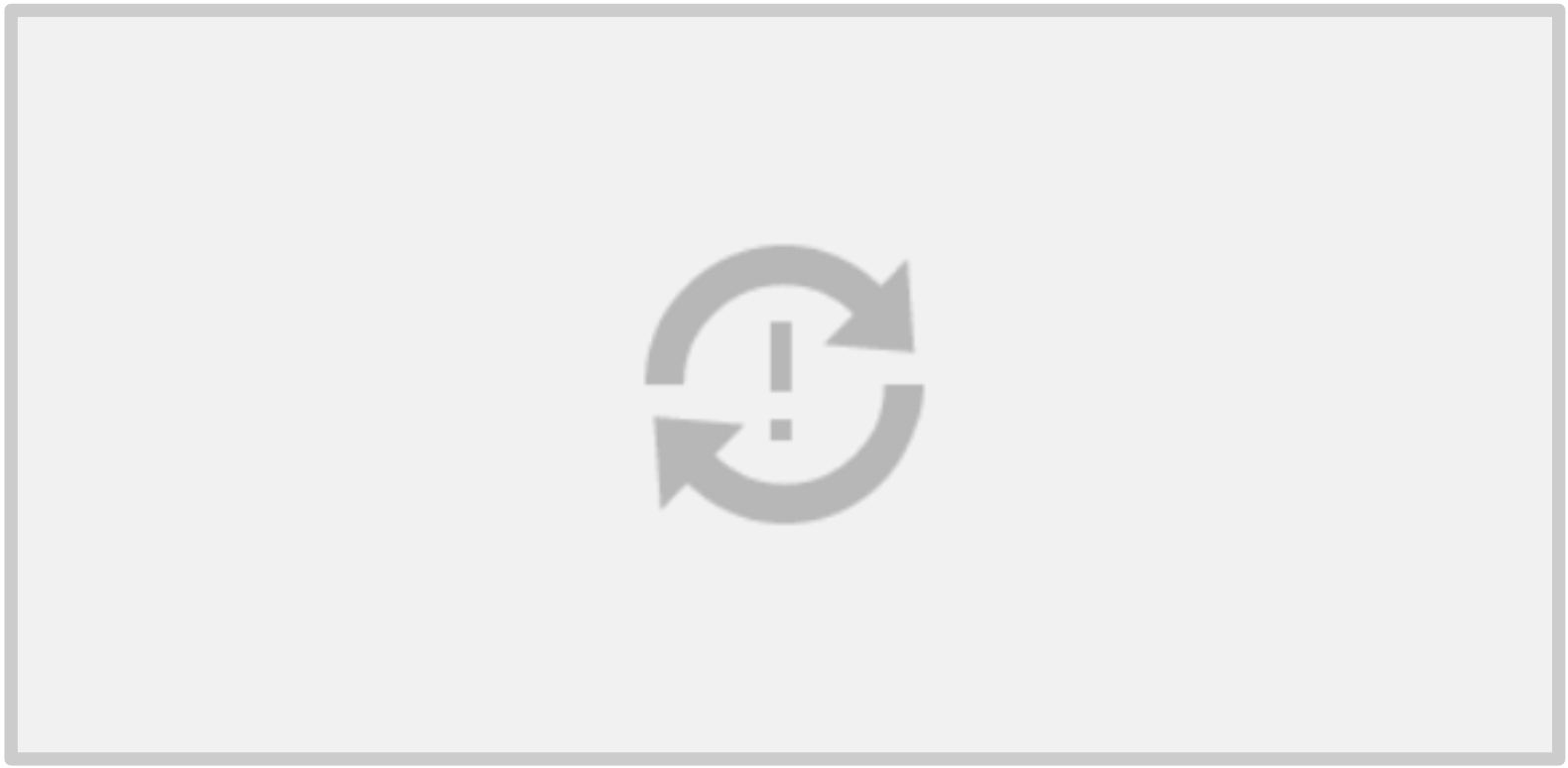
- The operating system is divided into a number of layers (levels)
- Each layer is built on top of lower layers.
- The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface.
- With modularity, layers are selected, each layer uses functions (operations) and services of only lower-level layers



Microkernel System Structure

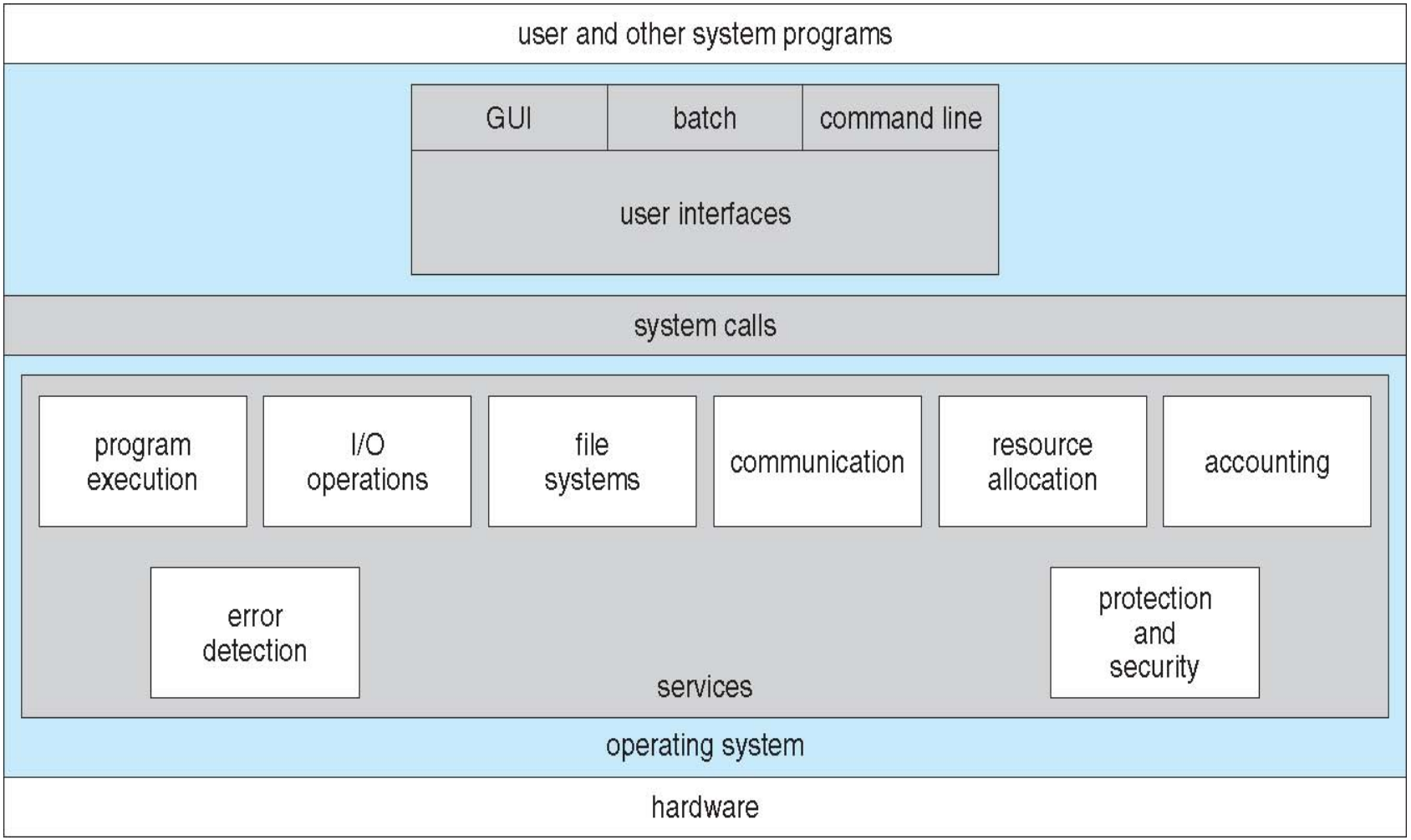
- Communication takes place between user modules using **message passing**
- Example of **microkernel: Mach**
  - Mac OS X kernel (**Darwin**) partly based on Mach
- Benefits:
  - Easier to extend a microkernel
  - Easier to port the operating system to new architectures
  - More reliable (less code is running in kernel mode)
  - More secure
- Disadvantage:
  - Performance overhead of user space to kernel space communication

Microkernel System Structure



Operating System Services

- An operating system provides an **environment for the programs to run.**
- It provides certain services to programs



Operating System Services

- Operating-system services provides functions that are helpful to the user:

  - **User interface** - Almost all operating systems have a user interface (UI)

    Varies between Command-Line (CLI), Graphics User Interface (GUI).

  - **Program execution** - The system must be able **to load a program into memory and to run that program**, end execution, either normally or abnormally (indicating error)

Operating System Services

- **I/O operations** - A running program may require I/O, which may involve a file or an I/O device.

- **File-system manipulation** - read and write files and directories, create and delete them, search them, list file Information, permission management.

Operating System Services

- **Error detection – OS needs to be constantly aware of possible errors**

  May occur in the CPU and memory hardware, in I/O devices, in user program

  For each type of error**, OS should take the appropriate action** to ensure correct and consistent computing

  Debugging facilities can greatly enhance the user's and programmer's abilities to efficiently use the system.

Operating System Services

- **Communications** – Processes may exchange information, on the same computer or between computers over a network

  Communications may be via shared memory or through message passing (packets moved by the OS)

- **Resource allocation –** OS must ensure allocation of resources to all programs running.

  Many types of resources - such as **CPU cycle time, main memory, and file storage**, **I/O devices**

Operating System Services

- **Accounting -** To keep track of which users use how much and what kinds of

computer resources.

- **Protection and Security -**

Protection involves **ensuring that all access to system resources is controlled**

Security of the system from outsiders requires **user authentication**, extends to defending external I/O devices from invalid access attempts

Any Query ???

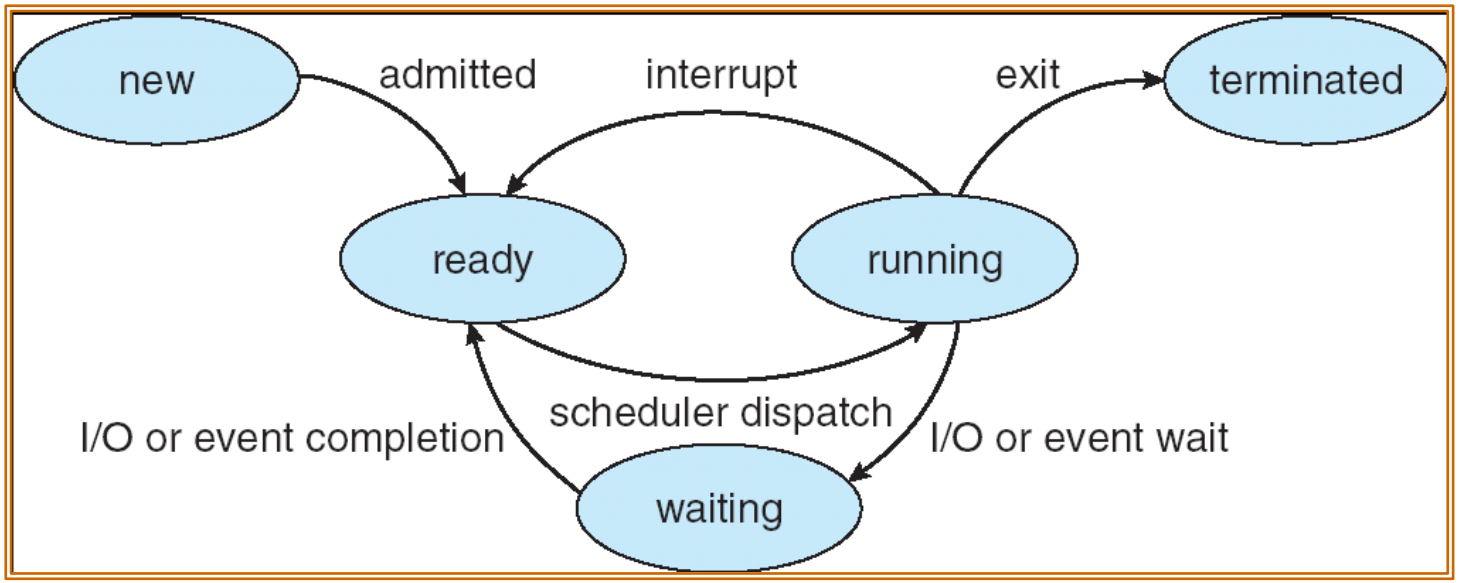
Process Concept

- An operating system executes a variety of programs:
  - Batch system – jobs
  - Time-shared systems – user programs or tasks
- Textbook uses the terms *job* and *process* almost interchangeably
- Process – a program in execution; process execution must progress in sequential fashion
- A process includes:
  - program counter
  - stack
  - data section

Process State

- As a process executes, it changes *state*
  - **new**: The process is being created
  - **running**: Instructions are being executed
  - **waiting**: The process is waiting for some event to occur
  - **ready**: The process is waiting to be assigned to a processor
  - **terminated**: The process has finished execution

Diagram of Process State

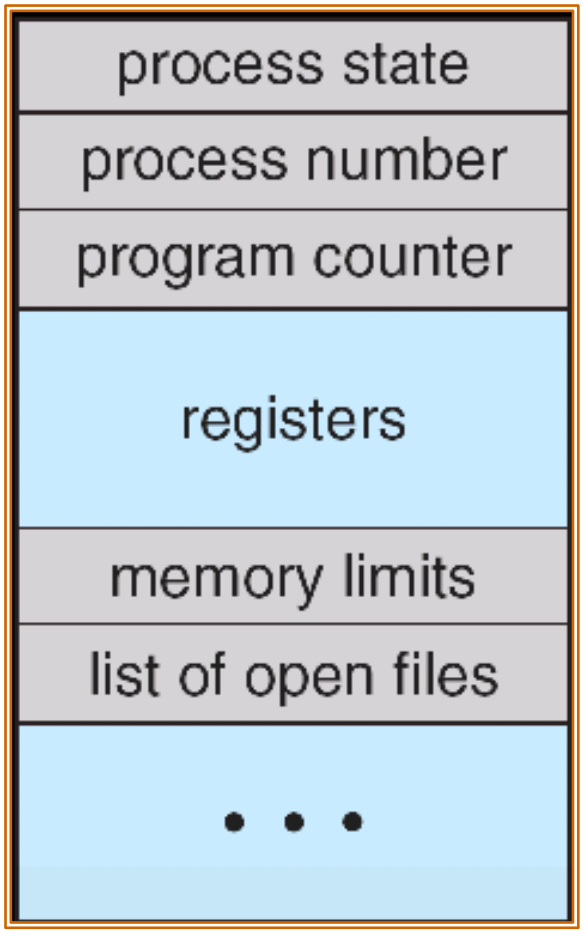


Process Control Block (PCB)

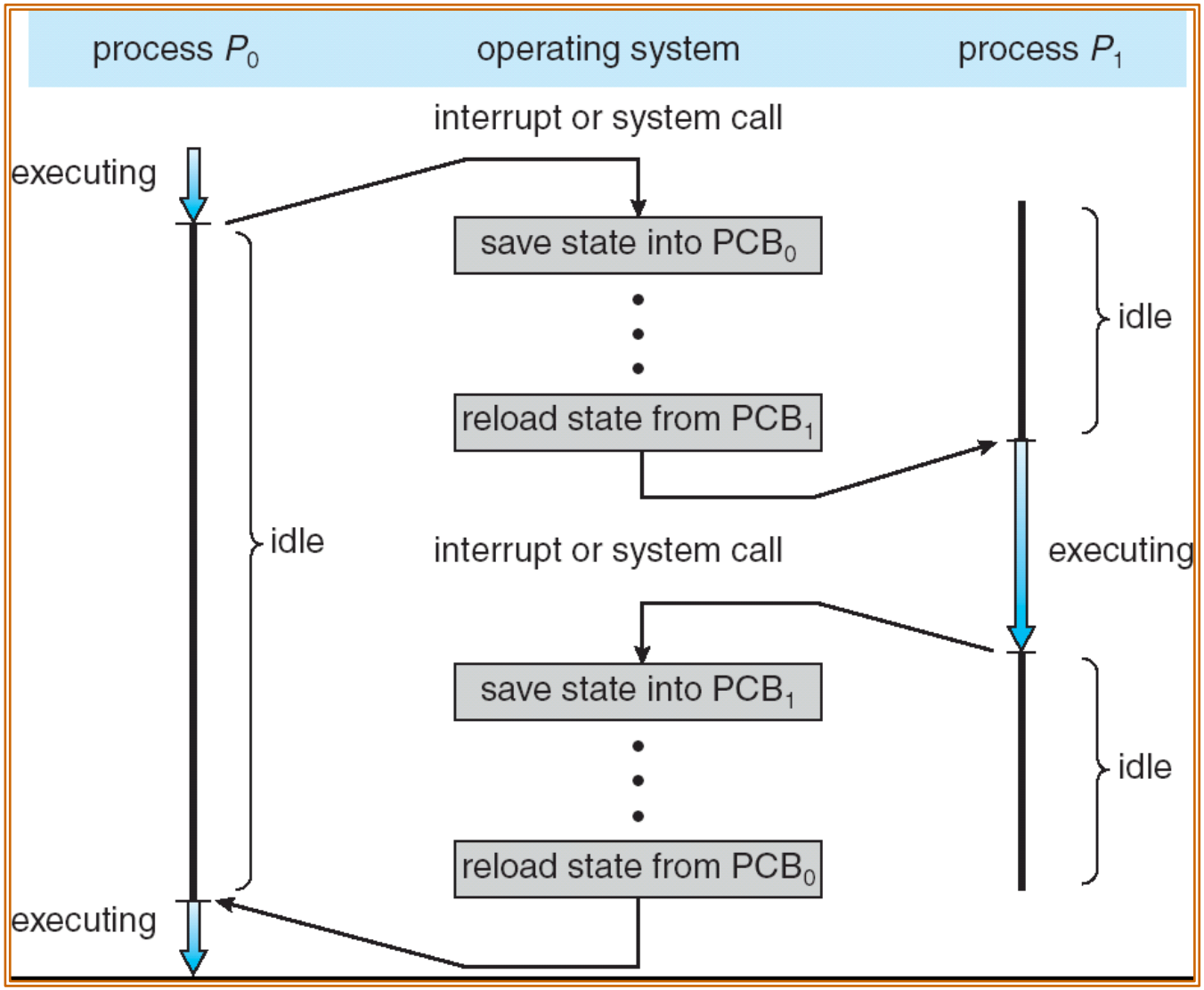
Information associated with each process

- Process state
- Program counter
- CPU registers
- CPU scheduling information
- Memory-management information
- Accounting information
- I/O status information

Process Control Block (PCB)



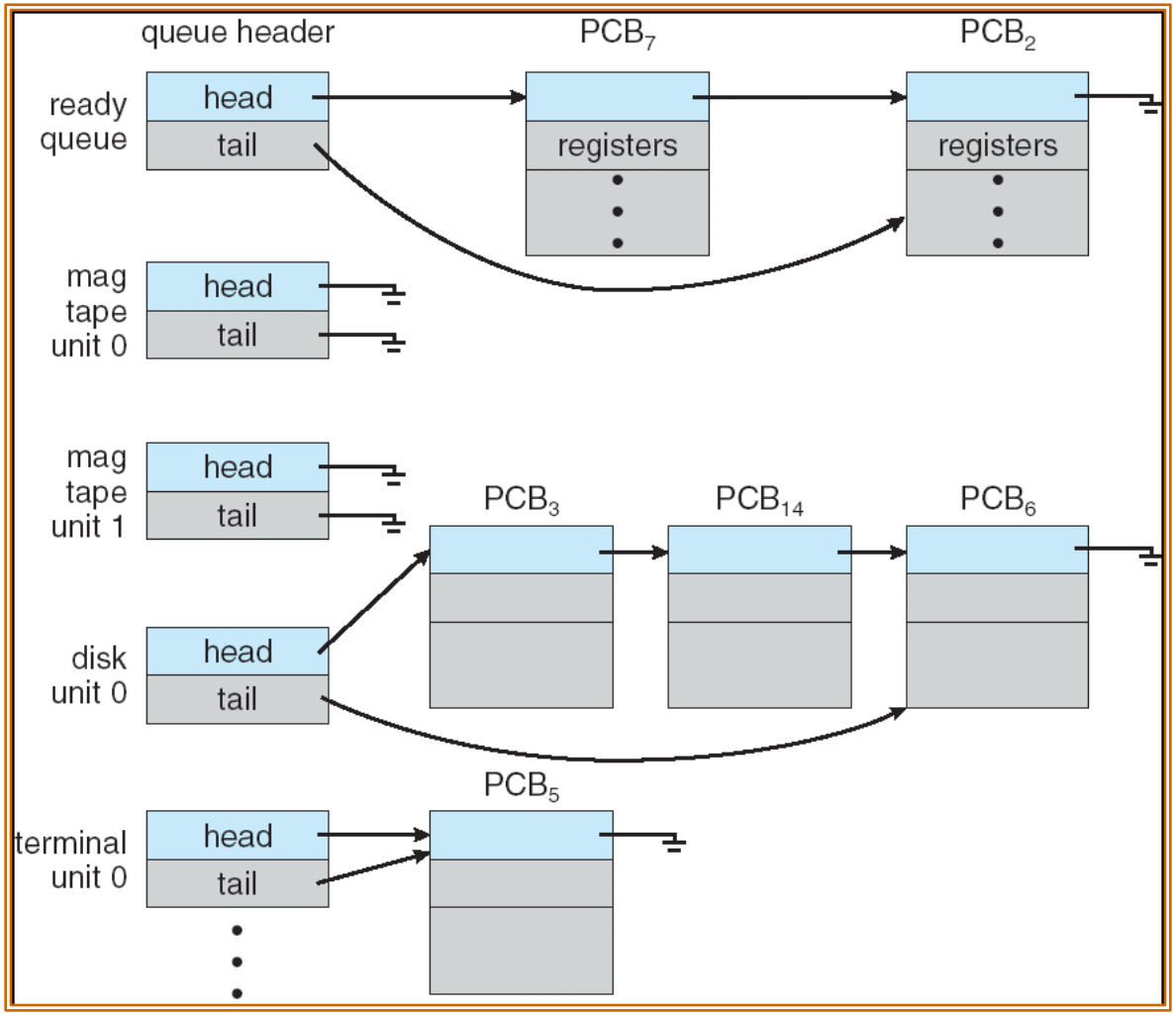
CPU Switch From Process to Process



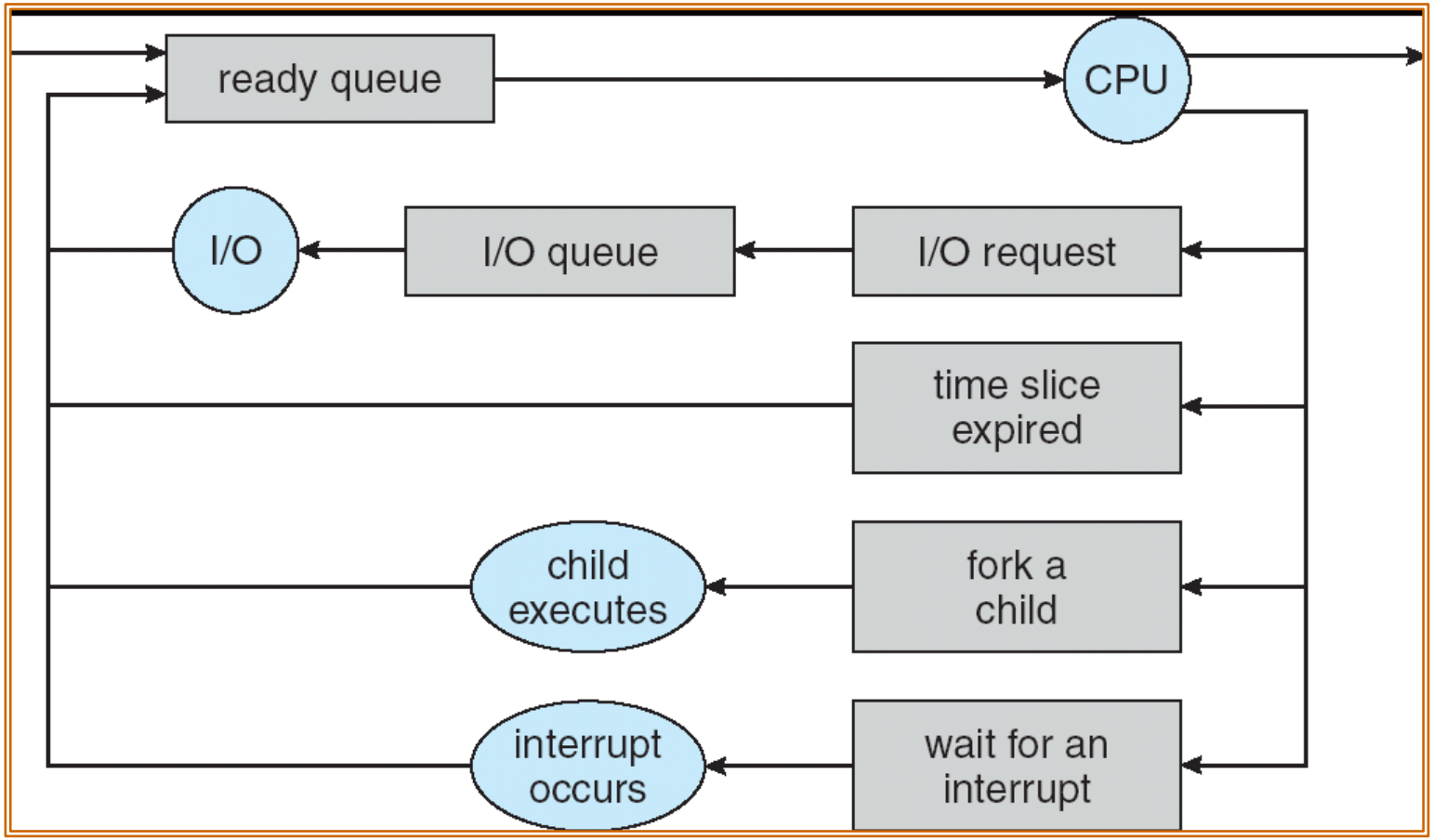
Process Scheduling Queues

- *Job queue* – set of all processes in the system
- *Ready queue* – set of all processes ready and waiting to execute
- *Device queues* – set of processes waiting for an I/O device
- Process migration between the various queues

Ready Queue And Various I/O Device Queues



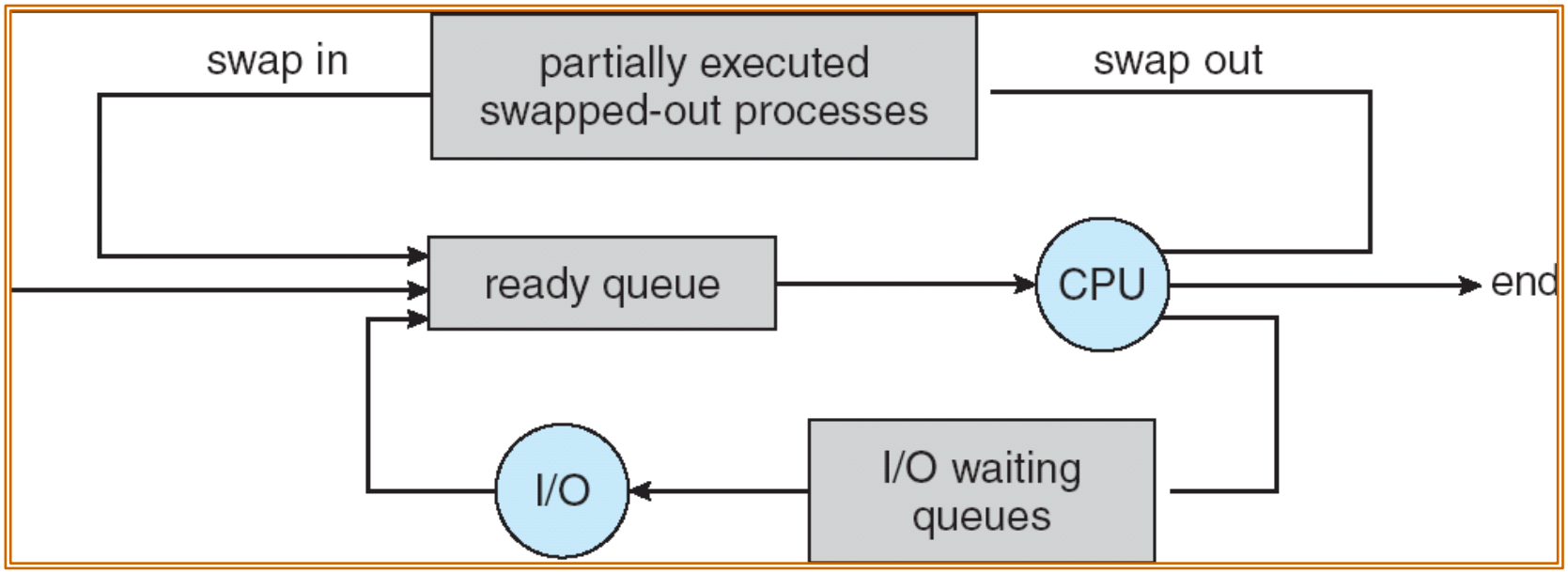
Representation of Process Scheduling



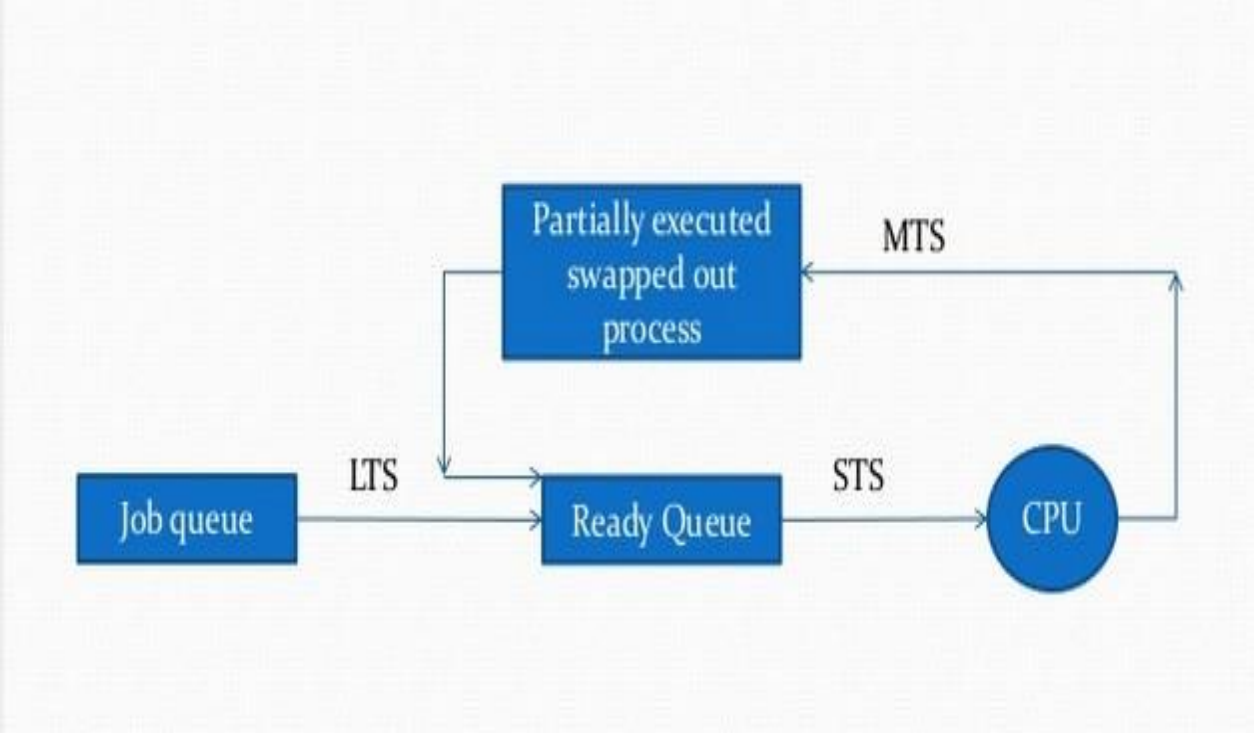
Schedulers

- *Long-term scheduler* (or job scheduler) – selects which processes should be brought into the ready queue
- *Short-term scheduler* (or CPU scheduler) – selects which process should be executed next.

Addition of Medium Term Scheduling



Scheduling


Partially executed swapped out process
MTS
LTS
Job queue
Ready Queue
STS
CPU

Schedulers (Cont.)

- Short-term scheduler is invoked very frequently (milliseconds) ⇒ (must be fast)
- Long-term scheduler is invoked very infrequently (seconds, minutes) ⇒ (may be slow)
- The long-term scheduler controls the *degree of multiprogramming*
- Processes can be described as either:
  - I/O-*bound process* – spends more time doing I/O than computations, many short CPU bursts
  - *CPU-bound process* – spends more time doing computations; few very long CPU bursts

Context Switch

- When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process
- Context-switch time is overhead; the system does no useful work while switching
- Dependent on hardware support

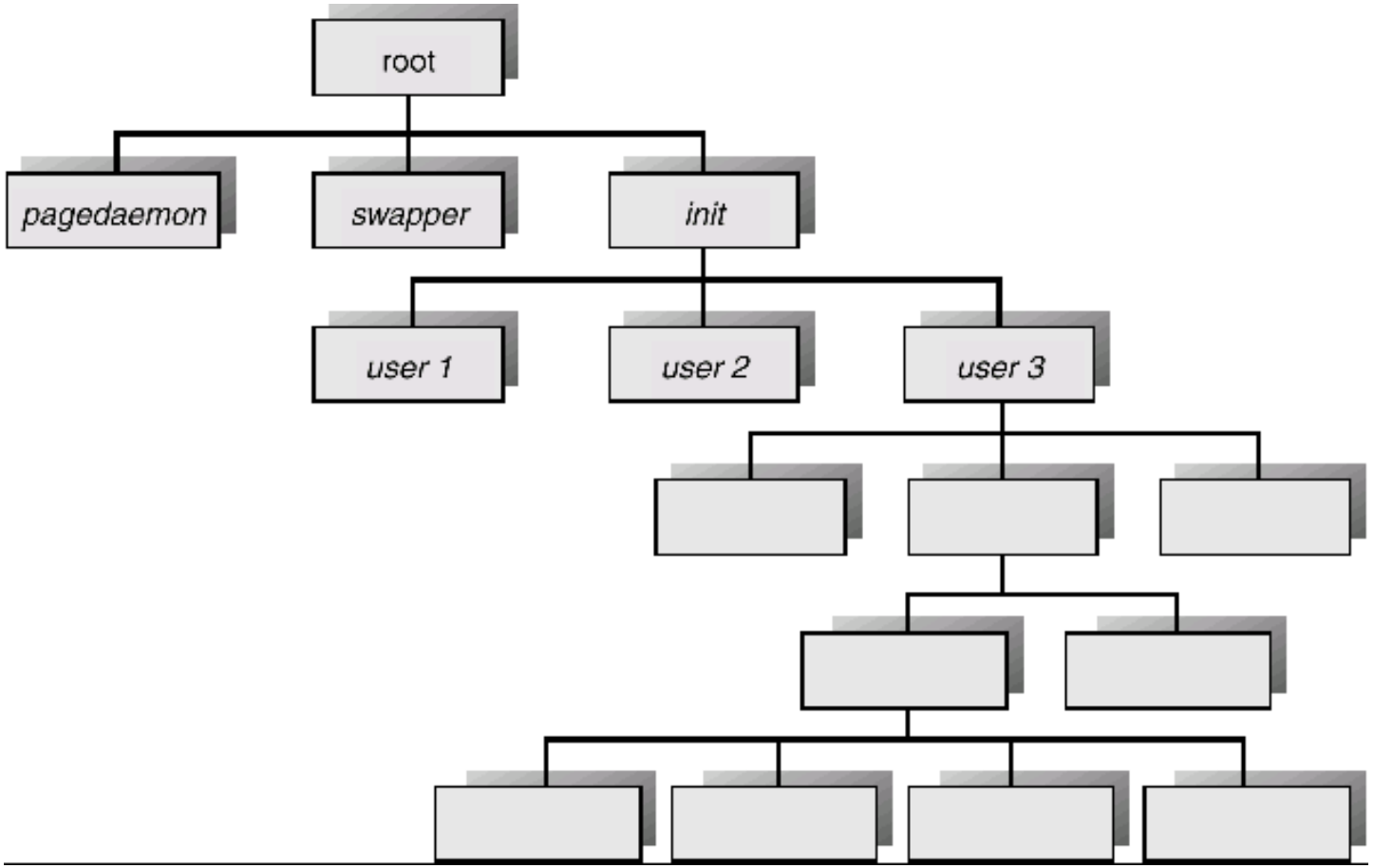
Process Creation

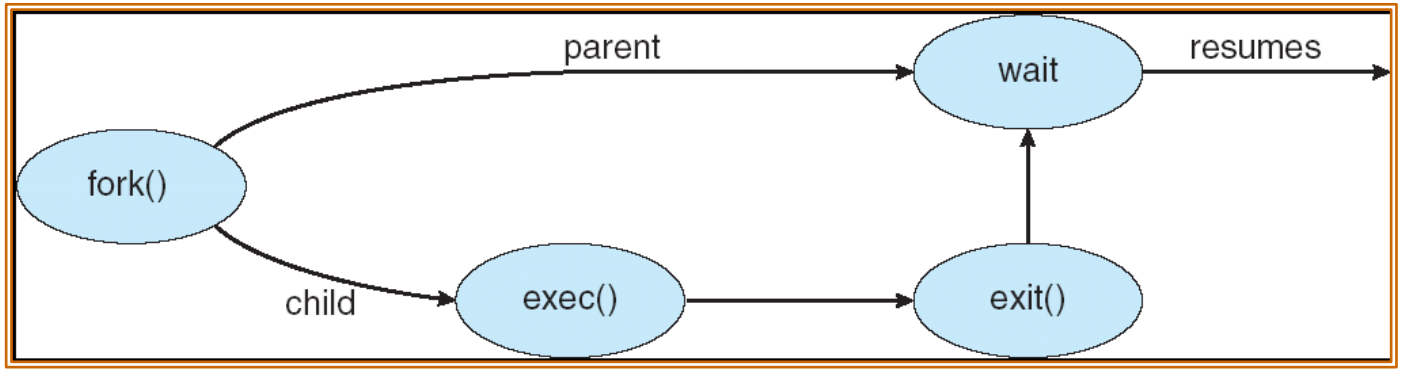
- Parent process create children processes, which, in turn create other processes, forming a tree of processes
- Resource sharing
  - Parent and children share all resources
  - Children share subset of parent's resources
  - Parent and child share no resources
- Execution
  - Parent and children execute concurrently
  - Parent waits until children terminate

Process Creation (Cont.)

- Address space
  - Child duplicate of parent
  - Child has a program loaded into it
- UNIX examples
  - **fork** system call creates new process
  - **exec** system call used after a **fork** to replace the process' memory space with a new program

A Tree of Processes On A Typical UNIX System




parent
wait
resumes
fork()
child
exec()
exit()

Process Termination

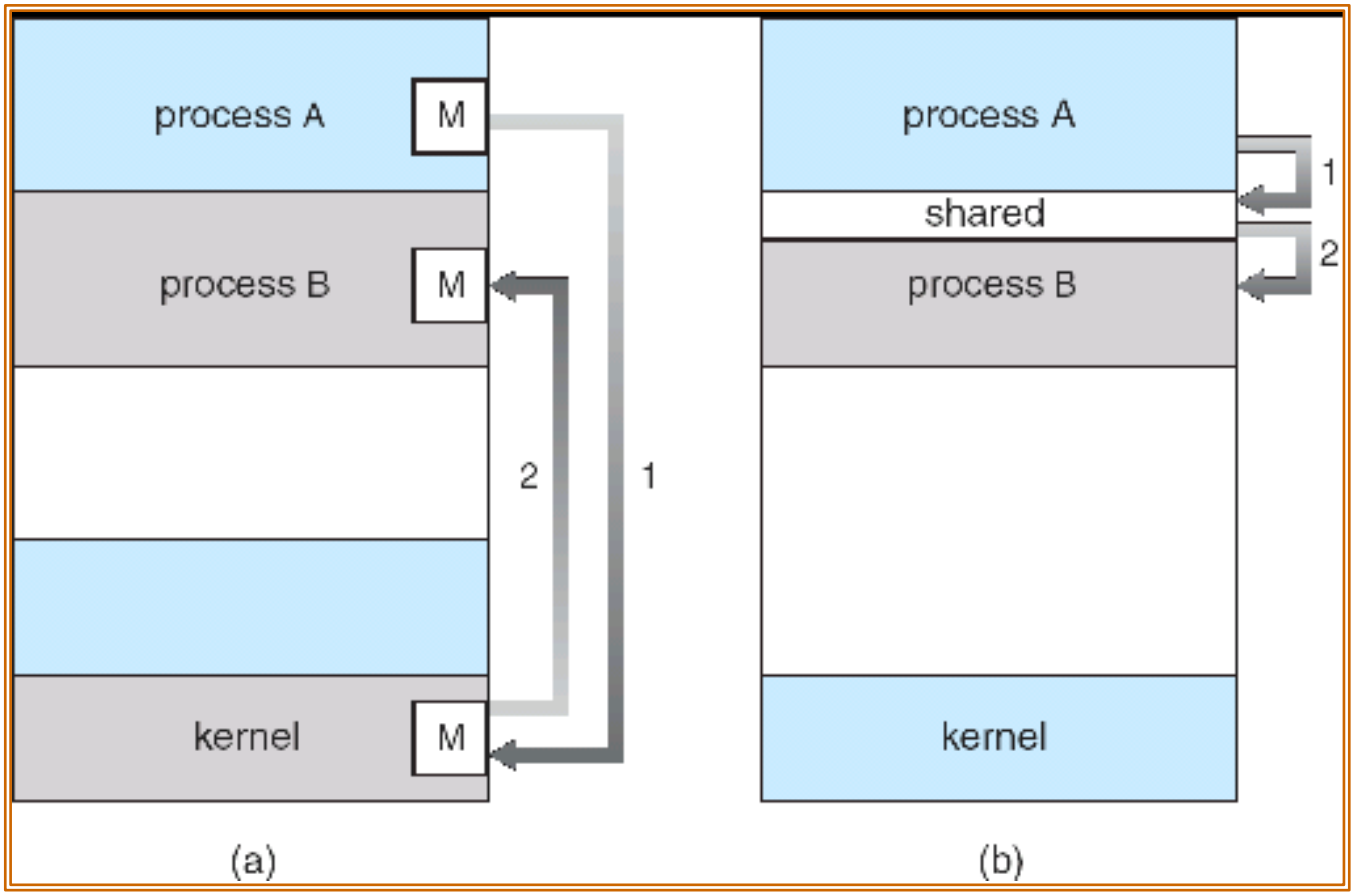
- Process executes last statement and asks the operating system to terminate it (**exit**)
  - Output data from child to parent (via **wait**)
  - Process' resources are deallocated by operating system
- Parent may terminate execution of children processes (**abort**)
  - Child has exceeded allocated resources
  - Task assigned to child is no longer required
  - If parent is exiting
    - Some operating system do not allow child to continue if its parent terminates
      - All children terminated - *cascading termination*

Types of Processes

- *Independent* process cannot affect or be affected by the execution of another process
- *Cooperating* process can affect or be affected by the execution of another process
- Reasons for providing an environment that allows process cooperation
  - Information sharing
  - Computation speed-up
  - Modularity
  - Convenience

Producer-Consumer Problem

- Paradigm for cooperating processes, *producer* process produces information that is consumed by a *consumer* process
  - *unbounded-buffer* places no practical limit on the size of the buffer. Consumer waits
  - *bounded-buffer* assumes that there is a fixed buffer size. Producer waits.


process A
M
process B
M
2
1
kernel
M
(a)
process A
1
shared
2
process B
kernel
(b)

Interprocess Communication (IPC)

- Mechanism for processes to communicate and to synchronize their actions
- Message system – processes communicate with each other by passing of messages.
- IPC facility provides two operations:
  - **send**(*message*)
  - **receive**(*message*)

  Messages can be of 2 types: Fixed length, variable length.
- If *P* and *Q* wish to communicate, they need to:
  - establish a *communication link* between them
  - exchange messages via send/receive
- Implementation of communication link
  - physical (e.g., shared memory, hardware bus)
  - logical (e.g., logical properties)

- Several methods for logically implementing a link and the send/receive operations. For example:
  - Direct or indirect communication
  - Symmetric or asymmetric communication

Implementation Questions

- How are links established?
- Can a link be associated with more than two processes?
- How many links can there be between every pair of communicating processes?
- What is the capacity of a link?
- Is the size of a message that the link can accommodate fixed or variable?
- Is a link unidirectional or bi-directional?

Direct Communication

- Processes must name each other explicitly:
  - **send** (*P, message*) – send a message to process P
  - **receive**(*Q, message*) – receive a message from process Q
- Properties of communication link
  - Links are established automatically
  - A link is associated with exactly one pair of communicating processes
  - Between each pair there exists exactly one link
  - The link may be unidirectional, but is usually bi-directional

- A variant of this scheme employs asymmetry in addressing. Only the sender names the recipient; the recipient is not required to name the sender.

- receive (id, message) - Receive a message from any process; the variable id is set to the name of the process with which communication has taken place.

- `send(P, message)`—Send a message to process P.

Indirect Communication

- Messages are directed and received from mailboxes (also referred to as ports)
  - Each mailbox has a unique id
  - Processes can communicate only if they share a mailbox
- Properties of communication link
  - Link established only if processes share a common mailbox
  - A link may be associated with many processes
  - Each pair of processes may share several communication links
  - Link may be unidirectional or bi-directional

Indirect Communication

- Operations
  - create a new mailbox
  - send and receive messages through mailbox
  - destroy a mailbox
- Primitives are defined as:

  send(*A, message*) – send a message to mailbox A

  receive(*A, message*) – receive a message from mailbox A

Indirect Communication

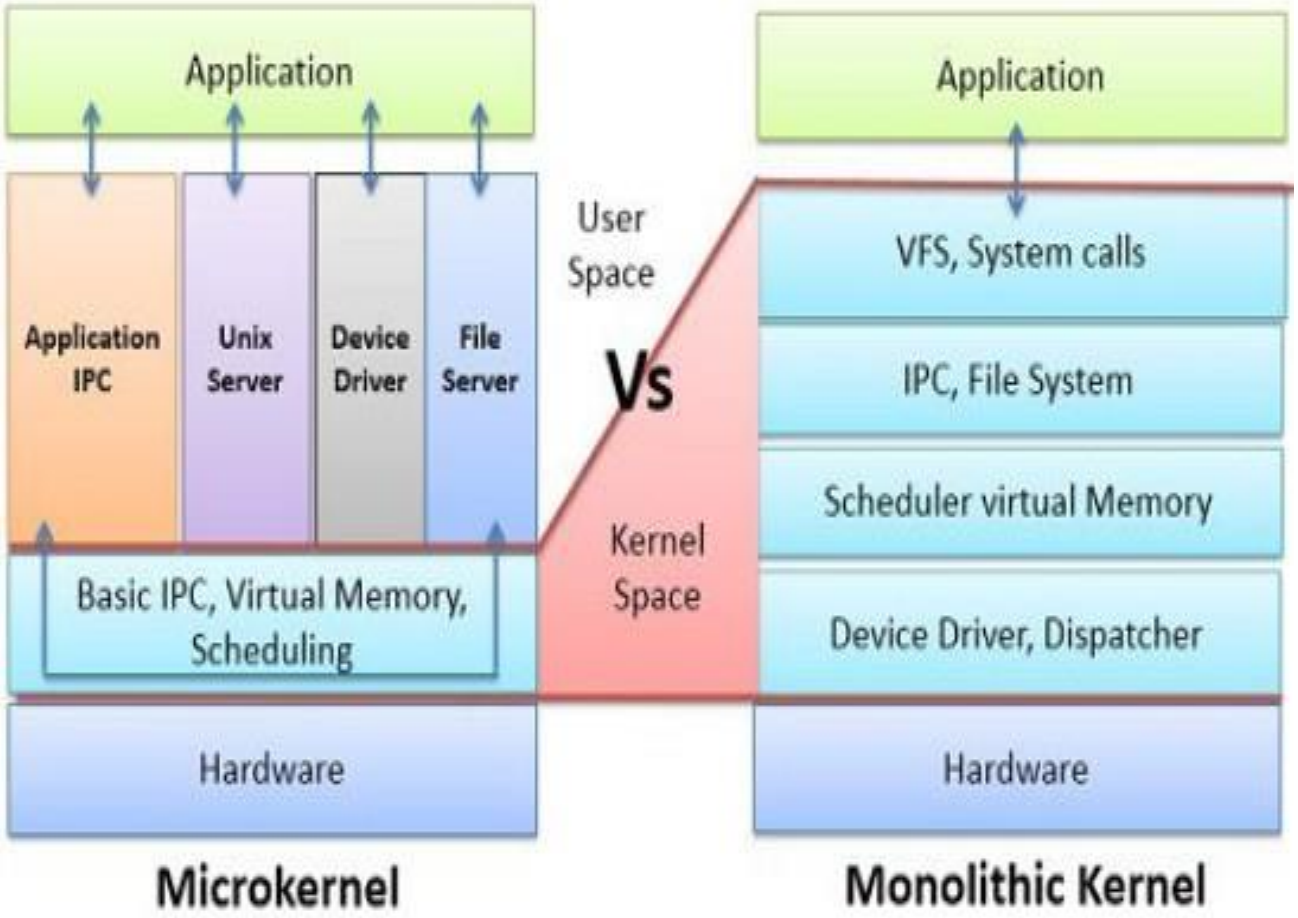
- Mailbox sharing
  - P_1, P_2, and P_3 share mailbox A
  - P_1, sends; P_2 and P_3 receive
  - Who gets the message?
- Solutions
  - Allow a link to be associated with at most two processes
  - Allow only one process at a time to execute a receive operation
  - Allow the system to select arbitrarily the receiver. Sender is notified who the receiver was.

Synchronization

- Message passing may be either blocking or non-blocking
- **Blocking** is considered **synchronous**
  - **Blocking send** has the sender block until the message is received
  - **Blocking receive** has the receiver block until a message is available
- **Non-blocking** is considered **asynchronous**
  - **Non-blocking** send as the sender sends the message and continue
  - **Non-blocking** receive has the receiver receive a valid message or null

Buffering

- Queue of messages attached to the link; implemented in one of three ways
  1. Zero capacity – 0 messages
     Sender must wait for receiver (rendezvous)
  2. Bounded capacity – finite length of *n* messages
     Sender must wait if link full
  3. Unbounded capacity – infinite length
     Sender never waits


Application
Application IPC
Unix Server
Device Driver
File Server
Basic IPC, Virtual Memory, Scheduling
Hardware
Microkernel
User Space
Vs
Kernel Space
Application
VFS, System calls
IPC, File System
Scheduler virtual Memory
Device Driver, Dispatcher
Hardware
Monolithic Kernel

Key Differences Between Microkernel and Monolithic Kernel

- The basic point on which microkernel and monolithic kernel is distinguished is that microkernel implement user services and kernel services in different address spaces and monolithic kernel implement both user services and kernel services under same address space
- The size of microkernel is small as only kernel services reside in the kernel address space. However, the size of monolithic kernel is comparatively larger than microkernel because both kernel services and user services reside in the same address space.

- **The execution of monolithic kernel is faster as the communication between application and hardware is established using the system call. On the other hands, the execution of microkernel is slow as the communication between application and hardware of the system is established through message passing**
- **It is easy to extend microkernel because new service is to be added in user address space that is isolated from kernel space, so the kernel does not require to be modified. Opposite is the case with monolithic kernel if a new service is to be added in monolithic**

- **Microkernel is more secure than monolithic kernel as if a service fails in microkernel the operating system remain unaffected. On the other hands, if a service fails in monolithic kernel entire system fails.**
- **Monolithic kernel designing requires less code, which further leads to fewer bugs. On the other hands, microkernel designing needs more code which further leads to more bugs.**