CS3203 : System Software

Books

Text Books

1. *Stalling William; “Operating Systems”, 6th Edition, Pearson Education.*
2. *Silberschatz A., Galvin P., GagneG.; “Operating System Concepts”, 9th Edition, JohnWiley and Sons.*
3. *D M Dhamdhere; "Systems Programming & Operating Systems"; Tata McGraw HillPublications, ISBN – 0074635794*
4. *John J Donovan; " Systems Programming " ; Tata Mc-Graw Hill edition , ISBN-13978-0-07-460482-3*

Books

Text Books

1. *Silberschatz A., Galvin P., Gagne G ;"Operating System Principles" 7th Edition John Wiley and Sons.*
2. *YashavantKanetkar; "Unix Shell Programming", 2 nd Edition, BPB Publications.*
3. *Forouzan B. A., Gilberg R. F.; "Unix And Shell Programming", 1 st Edition, Australia Thomson Brooks Cole.*
4. *Achyut S. Godbole ,AtulKahate; "Operating Systems", 3 rd Edition, McGraw Hill.*
5. *Robert Love, " Linux System Programming " ;O'Reilly, ISBN 978-0-596-00958-8*
6. *Mahesh Jadhav; " Easy Linux Device Driver "; HighTechEasy publishing, Second edition.*
7. *Ray Duncan; "Advanced MSDOS programming"; Microsoft press*

Course Project: OPERATING SYSTEMS

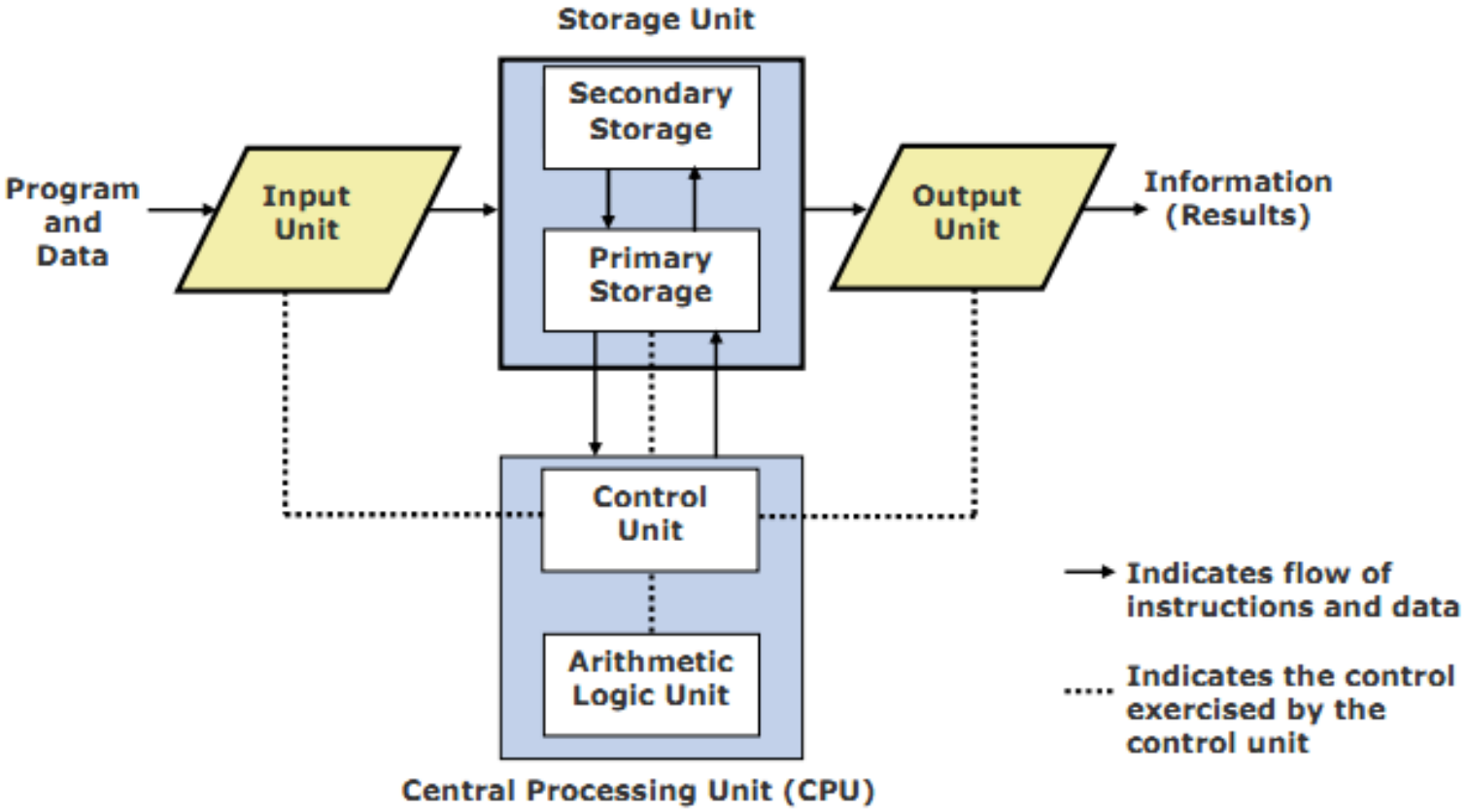
- Implementation of a multiprogramming operating system:

1. Stage I:
    i. CPU/ Machine Simulation
    ii. Supervisor Call through interrupt
2. Stage II:
    i. Paging
    ii. Error Handling
    iii. Interrupt Generation and Servicing
    iv. Process Data Structure
3. Stage III:
    i. Multiprogramming
    ii. Virtual Memory
    iii. Process Scheduling and Synchronization
    iv. Inter-Process Communication
    v. I/O Handling, Spooling and Buffering

Unit I

Introduction to Operating System

Block Diagram of a Computer (Logical)



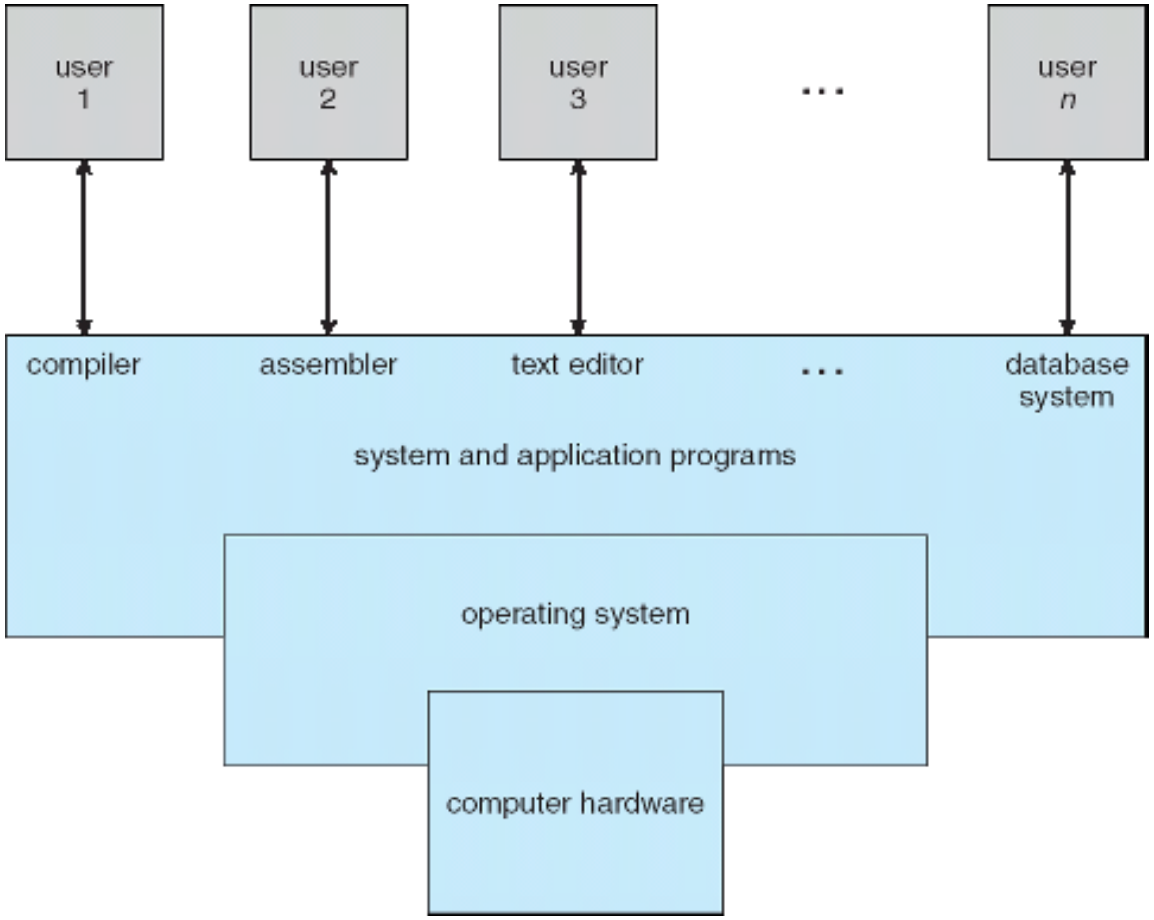
Operating System (OS)

- What do you do with just computer hardware?
  - If someone gives you a computer with no software whatsoever, how do you get it to do anything?
  - You write a program that runs on the hardware
- In the early days, that was the way it worked ...
  - You started with just the bare hardware
  - You wrote a program that did *everything*:
    - Including managing all aspects of the hardware
    - Including solving your particular problem
- Your program was all the computer did!

Without an OS

- Each program runs directly on the hardware
- Each program must do everything
- Each program needs to know the details of the hardware and how to use it
- If the hardware changes, the program must change as well
- The hardware supports only one program at a time - each user must wait until the previous program is done to “share” the hardware with other users.
- Writing programs is incredibly complex and expensive
- OS is like an extended machine to provide a better interface for convenience.

Four Components of a Computer System:



Computer System Structure

- Computer system can be divided into four components:
  - Hardware – provides basic computing resources
    - CPU, memory, I/O devices
  - Operating system
    - Controls and coordinates use of hardware among various applications and users
  - 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
  - Users
    - People, machines, other computers

Operating System (OS)

- Integrated set of programs that controls the resources of a computer system
- Provides its users with an interface which is more convenient to use than bare machine
- Directly sits on top of the hardware and controls it
- Manages computer hardware
- Provides a basis for application programs
- Acts as an intermediary between computer user and computer hardware

Introduction

- “An operating system is similar to a *government*. Like a government, it performs no useful function by itself. It simply provides an ***environment*** within which other programs can do useful work”

- “The fundamental goal of computer systems is to execute user programs and to make solving user problems easier”

Views of an Operating System

- Resource Allocator
  - Operating system manages or allocates *resources*
  - What resources?
    - CPU time, RAM Memory, disk & other I/O devices, interrupt numbers (IRQ's), files, network ports & sockets, and other software resources
  - Who uses the resources?
    - Resource users: One or more user processes & threads
- Control Program
  - Operation and control of hardware devices
  - Implements security and protection
  - Execution of user program to prevent errors and prevent improper use of the computer

Goals of an OS

- Execute user programs and make solving user problems **easier**
- Make the computer system **convenient** to use
- Use the computer resources in an **efficient** manner
- Ability to **evolve**
  - Can you think of reasons which make evolution necessary?
    - H/W changes, new services, fixes
- Different design goals for different OS
  - Mainframes, PC, Handheld

The Role of an OS

- A computer is a set of resources for the movement, storage, and processing of data.
- The OS is responsible for managing these resources.

Operating System as Software

- The OS functions in the same way as an ordinary computer software
- It is a program that is executed by the CPU
- The OS frequently relinquishes control and must depend on the processor to allow it to regain control.

OS as Resource Manager

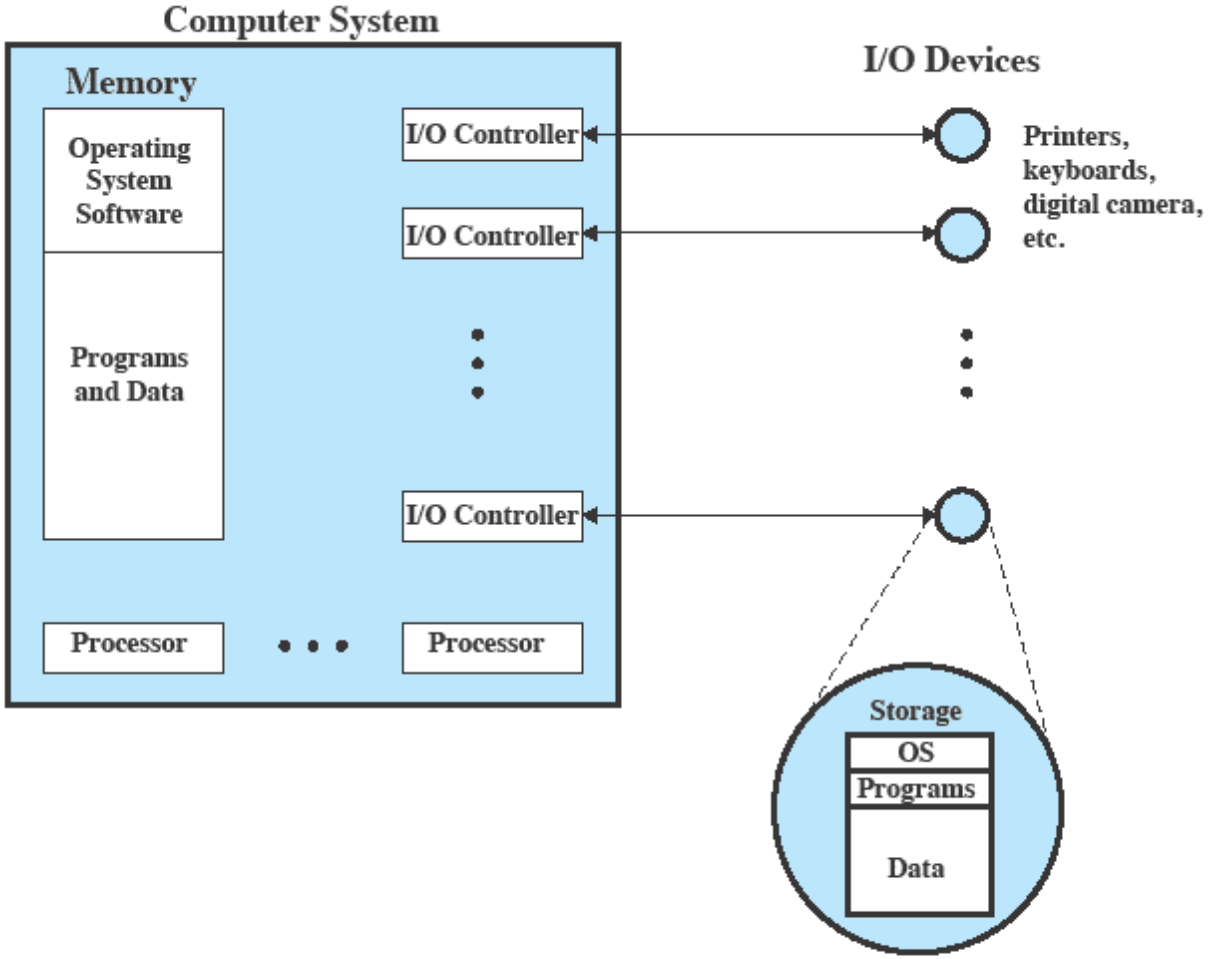


Figure 2.2 The Operating System as Resource Manager

Common Tasks by OS

Task	When performed
Maintain information for security	While registering new users
Construct a list of resources in the system	During booting
Verify identity of a user	At login time
Initiate execution of programs	At user commands
Maintain information for protection	When users specify or change protection information
Check protection information and perform resource allocation	At user/program request
Maintain current status of all resources	Continually during OS operation
Maintain current status of all programs and perform scheduling	Continually during OS operation

Basic function of OS:

- Process Management
- Main memory management
- Secondary storage management
- I/O Subsystem Management
- File Management

1. Process Management

- A process is a program in execution. It is a unit of work within the system. Program is a *passive entity*, process is an *active entity*.
- Process needs resources to accomplish its task
  - CPU, memory, I/O, files
  - Initialization data
- Process termination requires reclaim of any reusable resources
- Single-threaded process has one **program counter** specifying location of next instruction to execute
  - Process executes instructions sequentially, one at a time, until completion
- Multi-threaded process has one program counter per thread
- Typically system has many processes, some user, some operating system running concurrently on one or more CPUs
  - Concurrency by multiplexing the CPUs among the processes / threads

Process Management Activities

The operating system is responsible for the following activities in connection with process management:

- Creating and deleting both user and system processes
- Suspending and resuming processes
- Providing mechanisms for process synchronization
- Providing mechanisms for process communication
- Providing mechanisms for deadlock handling

2. Main Memory Management

- Main memory -array of words/bytes
- All data in memory before and after processing
- All instructions in memory in order to execute
- Memory management determines what is in memory when
  - Optimizing CPU utilization and computer response to users
- Memory management activities
  - Keeping track of which parts of memory are currently being used and by whom
  - Deciding which processes (or parts thereof) and data to move into and out of memory
  - Allocating and deallocating memory space as needed

3. Secondary Storage Management

- Main memory is small and volatile
- Secondary storage is large and nonvolatile

  - OS activities related to disk management include
    - Free space management
    - Storage allocation
    - Disk scheduling

4. I/O Subsystem management

- One purpose of OS is to hide peculiarities of hardware devices from the user
- I/O subsystem responsible for
  - Memory management of I/O including
    - ✔ buffering (storing data temporarily while it is being transferred),
    - ✔ caching (storing parts of data in faster storage for performance),
    - ✔ spooling (the overlapping of output of one job with input of other jobs)
  - General device-driver interface
  - Drivers for specific hardware devices

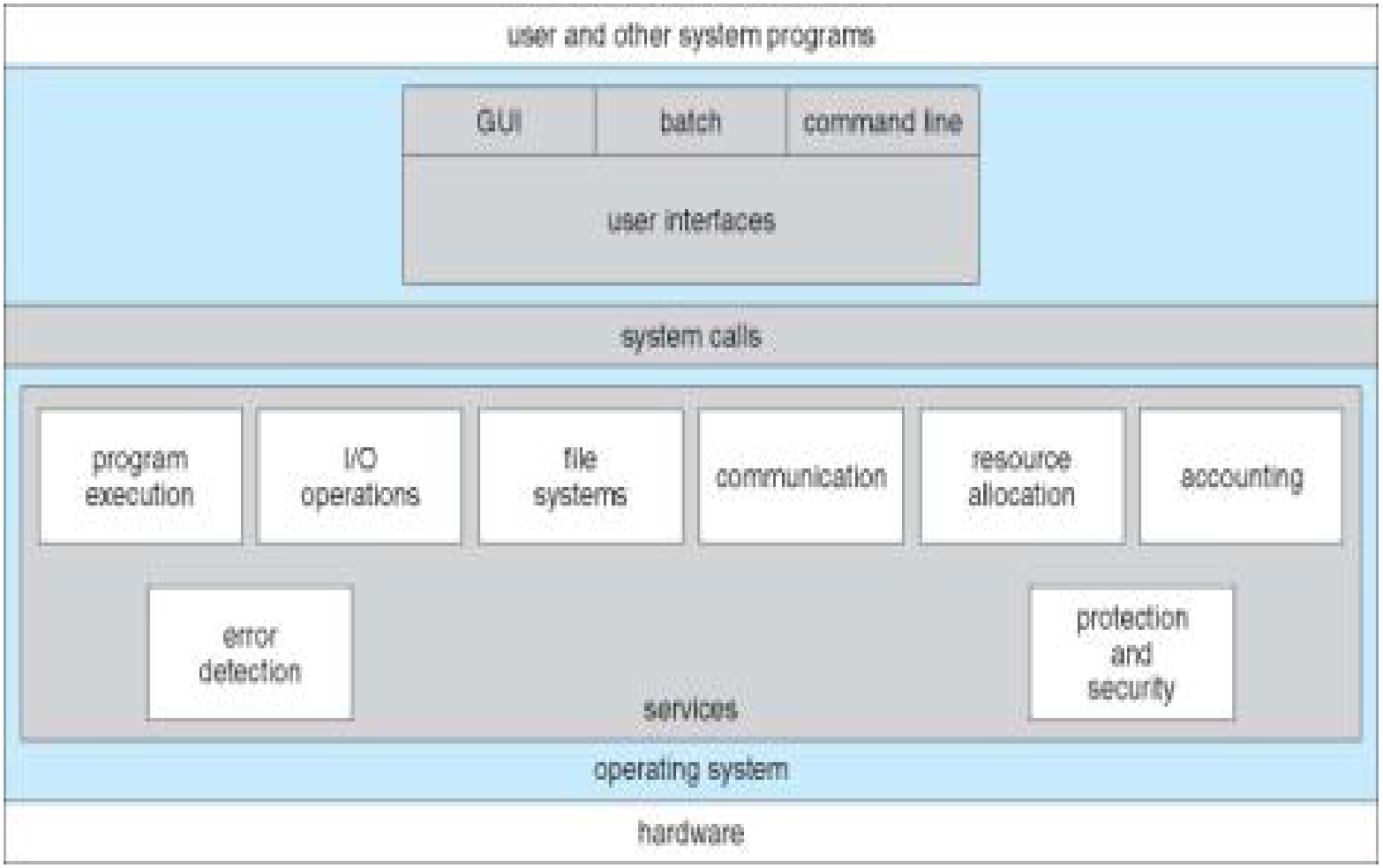
5. File Management

- OS provides uniform, logical view of information storage
- Abstracts physical properties to logical storage unit - **file**
- Each medium is controlled by device (i.e., disk drive, tape drive)
- Varying properties include access speed, capacity, data-transfer rate, access method (sequential or random)
- File-System management
- Files usually organized into directories

FS Management Activities

- Creating and deleting files
- Creating and deleting directories
- Supporting primitives for manipulating files and directories
- Mapping files onto secondary storage
- Backing up files on stable (non-volatile) storage media

OS Services



A view of OS services

Operating System Services

- Operating-system provides certain services to programs and users of that program.(For convenience of programmers)
  - User interface - Almost all operating systems have a user interface (UI)
    - Varies between Command-Line (CLI), Graphics User Interface (GUI), Batch
  - 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)
  - I/O operations - A running program may require I/O, which may involve a file or an I/O device
    - For protection & security IO done by OS not by user

Operating System Services (Cont)

- File-system manipulation - The file system is of particular interest. Obviously, programs need to read and write files and directories, create and delete them, search them, list file Information, permission management.

- Communications – Processes may exchange information, on the same computer or between computers over a network
  - Communications may be via shared memory(Tightly coupled system) or through message passing(Loosely coupled system) packets moved by the OS

Operating System Services (Cont)

- Error detection – OS needs to be constantly aware of possible errors
  - May occur in the CPU and memory hardware(memory error, power failure)
  - In I/O devices(connection failure in n/w)
  - In user program(access to illegal memory, arithmetic overflow)
  - 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 (Cont)

- Another set of OS functions exists for ensuring the efficient operation of the system itself via resource sharing
  - **Resource allocation -** When multiple users or multiple jobs running concurrently, resources must be allocated to each of them
    - Many types of resources - Some (such as CPU cycles, main memory, and file storage) may have special allocation code, others (such as I/O devices) may have general request and release code
  - **Accounting -** To keep track of which users use how much and what kinds of computer resources

Operating System Services (Cont)

- **Protection and security -** The owners of information stored in a multiuser or networked computer system may want to control use of that information, concurrent processes should not interfere with each other
  - **Protection** involves ensuring that all access to system resources is controlled
  - **Security** of the system from outsiders requires user authentication(password) , defending external I/O devices modem,NIC from invalid access attempts

User – OS Interface

- GUI
  - Graphical User Interface
  - Desktop, icons, mouse
- CLI
  - Command Line Interface
  - Allows direct command entry
  - Sometimes implemented in kernel, sometimes by systems program
  - Command interpreter
  - Windows: Command shell
  - UNIX/LINUX: Terminal – bash, ksh, csh: shells
  - Built-in commands vs external commands

Protection and Control

- Ensuring proper operation of the operating system, and programs
- Examples

  (a) only certain users can access certain files;

  (b) restricted access to operating system parts of RAM;

  (c) restrictions about which instructions can be executed by applications;

  (d) restricted access to RAM of other processes

Protection

- A main way that modern operating systems provide protection is through use of dual-mode hardware features
- User-mode & kernel mode
- Hardware boots in ***kernel mode***; switches to ***user-mode*** when running user processes
- System call or interrupt enables switch back to kernel mode

Protection modes

- User mode
  - limited access to RAM memory; just the memory of a single program running
  - limited access to machine instructions; ***illegal*** to execute I/O instructions, interrupt management instructions and some others
- kernel mode
  - access to all RAM memory
  - access to all machine instructions
- User vs. kernel mode implemented as a bit in CPU

So, if we have these protection modes, how does a process gain access to the OS?

How can a user process request an operation or function be carried out by the operating system?

System Calls

- User programs are not allowed to access system resources directly. They must ask the OS to do that for them.
- OS provides a set of functions that can be called by user programs to request for OS services. These functions are called “system calls”
- System calls run in kernel mode.
- They can be called by executing a special instruction (trap or software interrupt) which causes processor to switch to the kernel mode and jump to a previously defined location in the kernel.
- When the system call finishes, processor returns to the user program and runs in user mode.

Dual Mode Operation

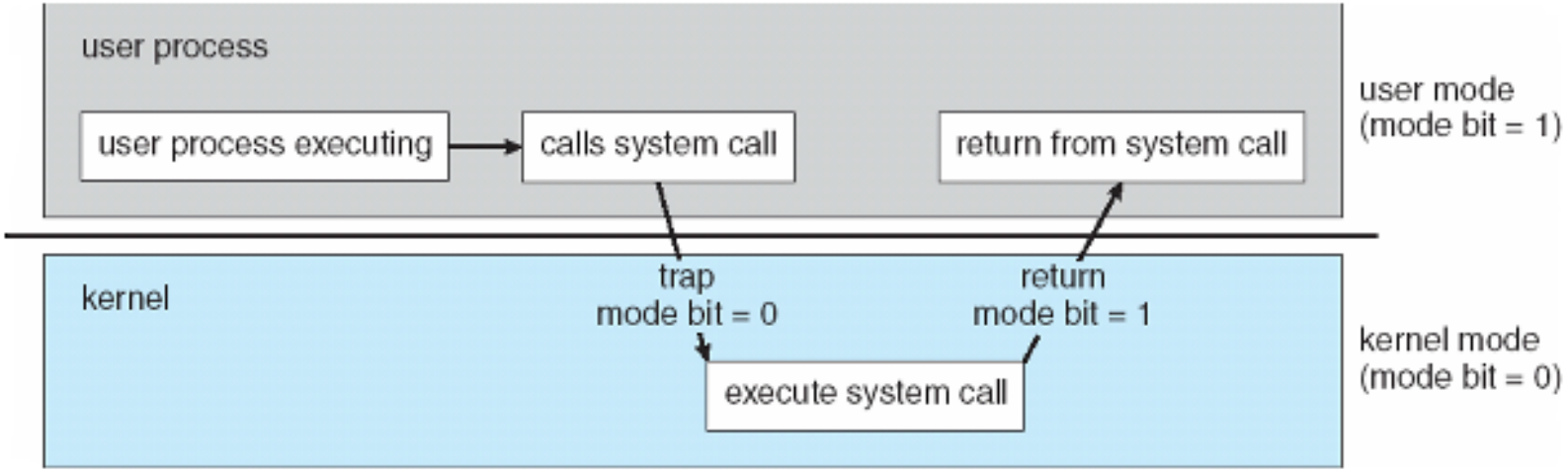


Figure : Transition from User to Kernel Mode

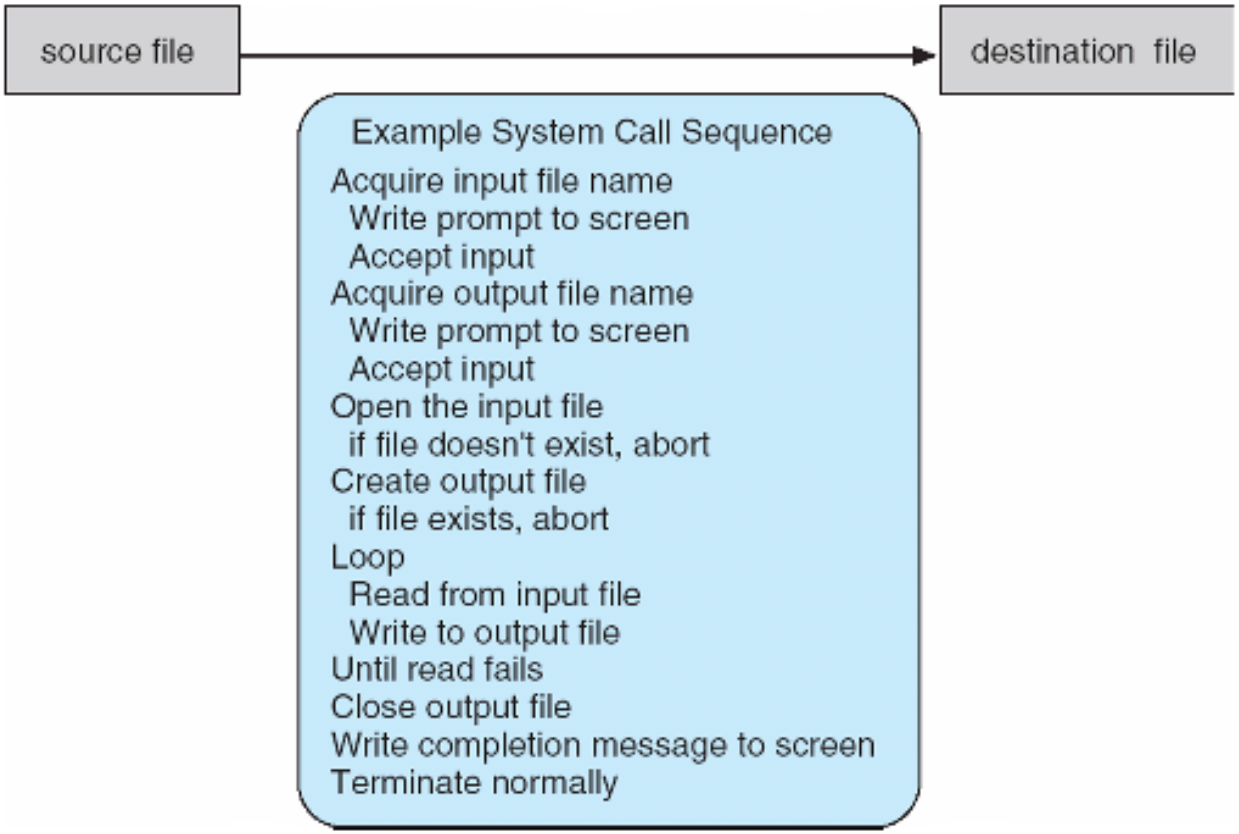
Kernel mode is also called Supervisor mode

System Calls

- System calls provide the interface between a running program and the operating system.
- Generally available as assembly-language instructions.
- Languages defined to replace assembly language for systems programming allow system calls to be made directly (e.g., C, C++)
- Mostly accessed by programs via a high-level Application Program Interface (API) rather than direct system call use.
- Three most common APIs are Win32 API for Windows, POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X), and Java API for the Java virtual machine (JVM)
- Why use APIs rather than system calls?

Example of System Calls

- System call sequence to copy the contents of one file to another file



How system calls are used

API

- An API is a set of functions provided by an operating system or other system software.
- An application program calls the functions to request the services.
- An API clearly defines how to call functions and what the results are. (API is specification, not implementation)
- Examples: APIs for file system, graphics user interface, networking, etc

Advantages of using APIs

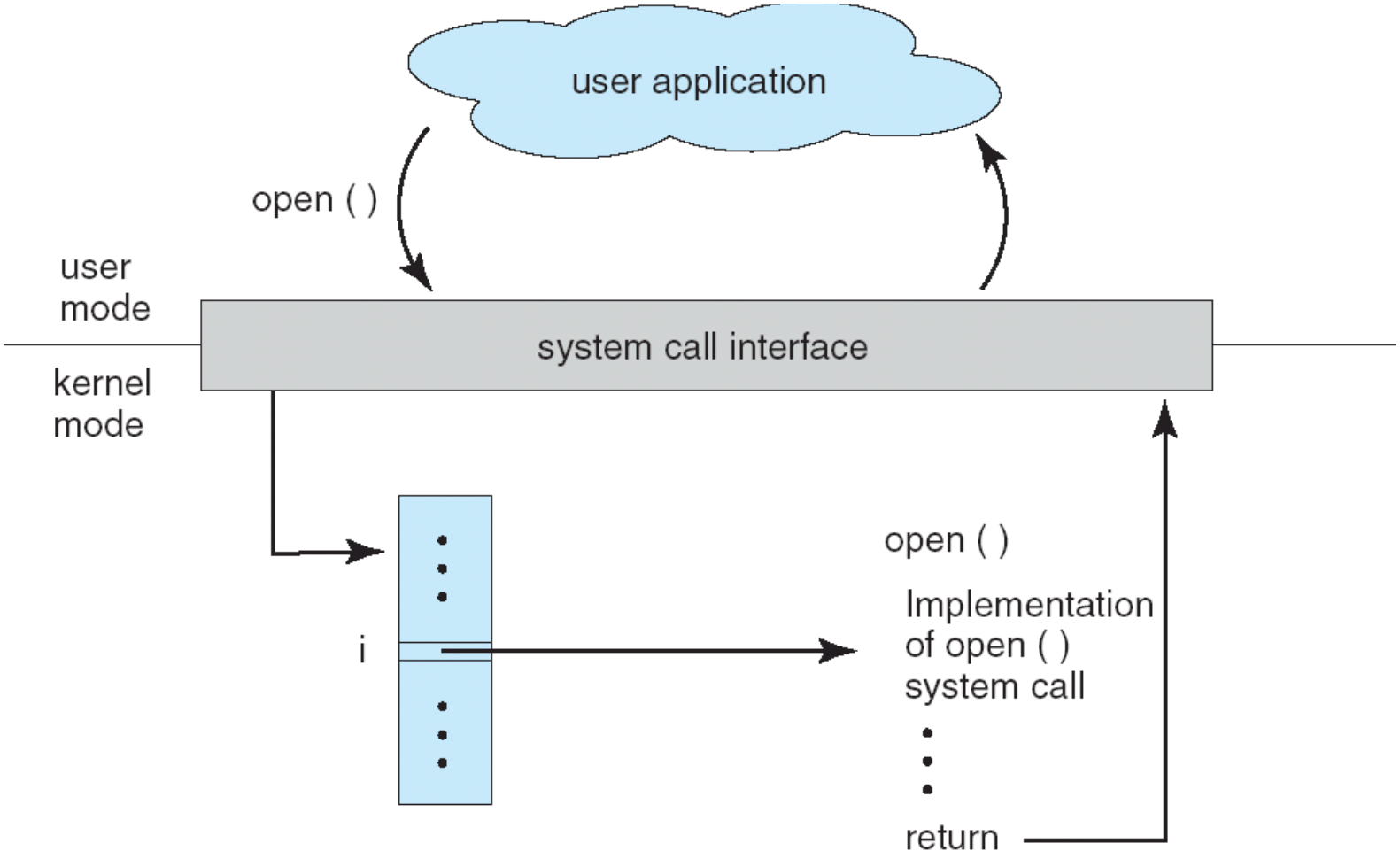
1. Portability: User programs that follow the API's definition are portable.
2. An API can provide a common interface for different implementations of a service.

   For example, the UNIX file system API is the same for all kinds of devices.
3. Using an API allows upgrading system software without changing user programs
4. APIs are faster than system call as there is no need of mode switch.

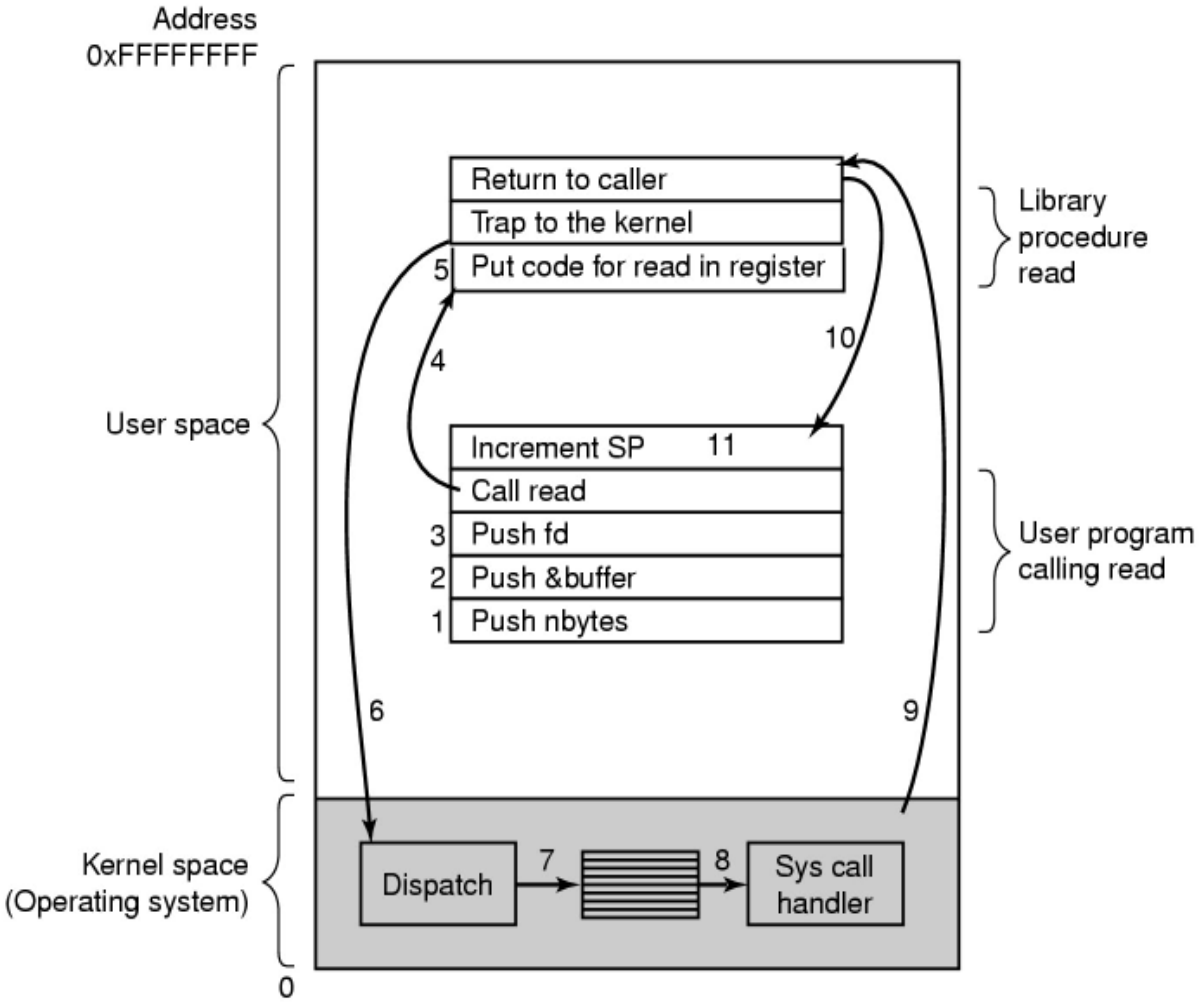
System Call Implementation

- Typically, a number associated with each system call
  - System-call interface maintains a table indexed according to these numbers
- The system call interface invokes intended system call in OS kernel and returns status of the system call and any return values
- The caller need know nothing about how the system call is implemented
  - Just needs to obey API and understand what OS will do as a result call
  - Most details of OS interface hidden from programmer by API
    - Managed by run-time support library (set of functions built into libraries included with compiler)

API – System Call – OS Relationship



Steps in Making a System Call

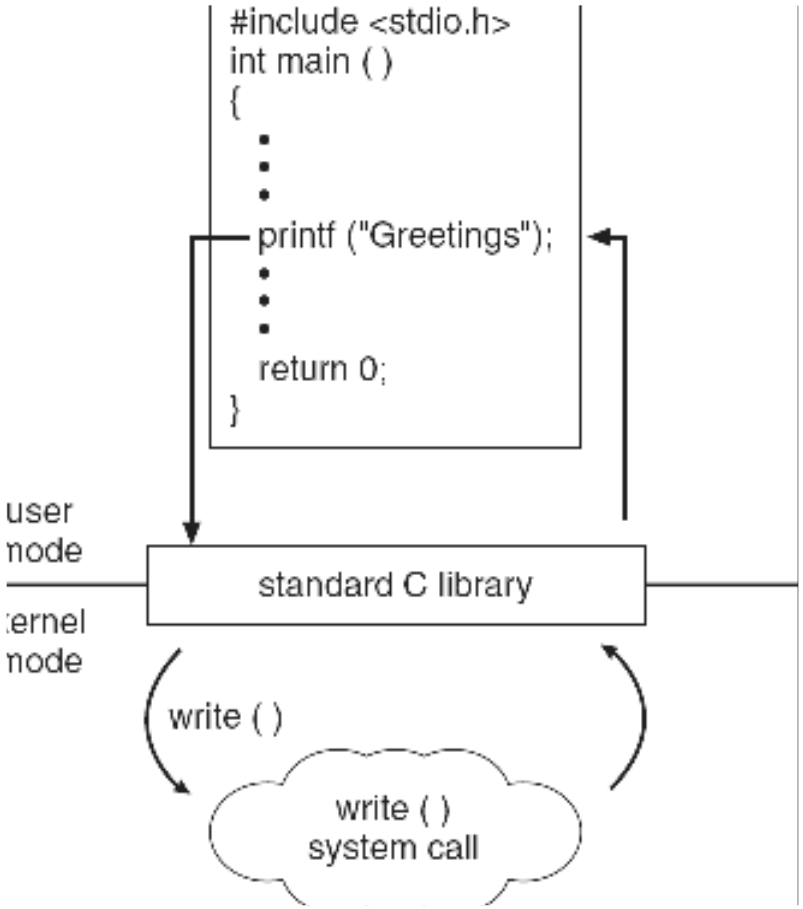


There are 11 steps in making the system call

count=read (fd, buffer, nbytes)

Standard C Library Example

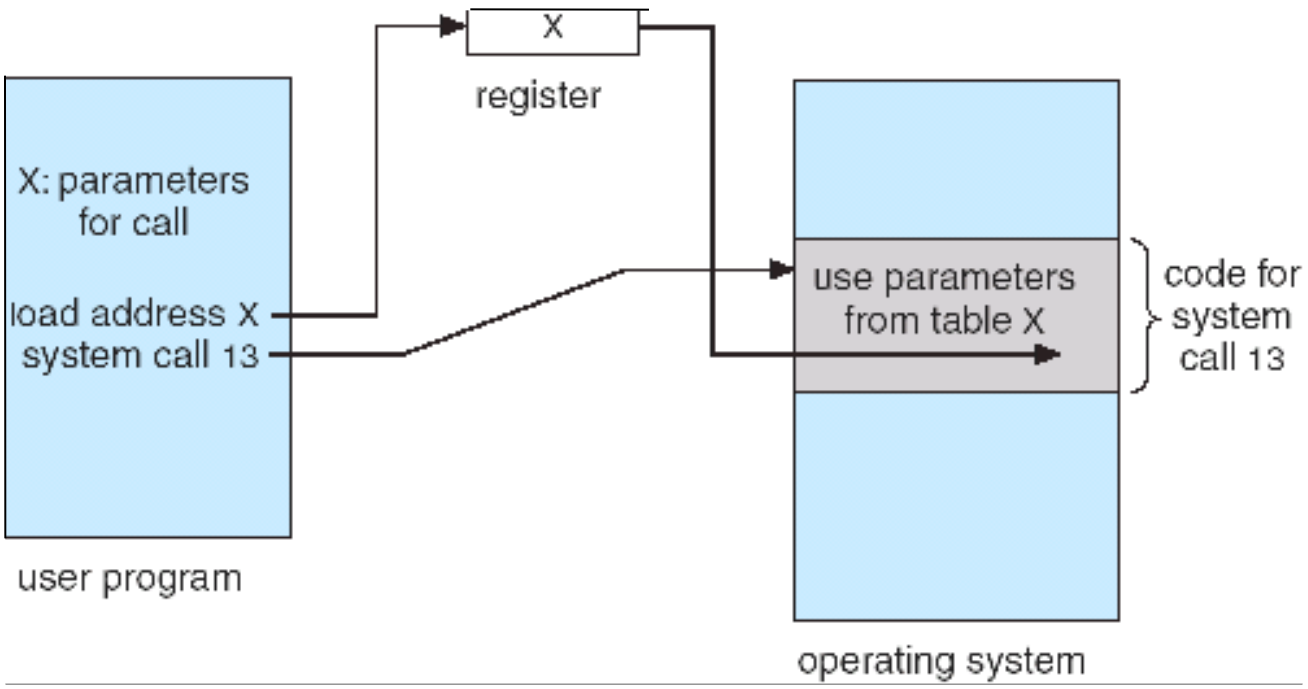
- C program invoking printf() library call, which calls write() system call



System Call Parameter Passing

- Often, more information is required than simply identity of desired system call
  - Exact type and amount of information vary according to OS and call
- Three general methods used to pass parameters to the OS
  - Simplest: pass the parameters in *registers*
    - In some cases, may be more parameters than registers
  - Parameters stored in a *block,* or table, in memory, and address of block passed as a parameter in a register
    - This approach taken by Linux and Solaris
  - Parameters placed, or *pushed,* onto the *stack* by the program and *popped* off the stack by the operating system
  - Block and stack methods do not limit the number or length of parameters being passed

Parameter Passing via Table



Types of System Calls

Process control

File management

Device management

Information maintenance

Communications

Protection

Examples of Windows and Unix System Calls

	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()
Protection	SetFileSecurity() InitlializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

UNIX	Win32	Description
fork	CreateProcess	Create a new process
waitpid	WaitForSingleObject	Can wait for a process to exit
execve	(none)	CreateProcess = fork + execve
exit	ExitProcess	Terminate execution
open	CreateFile	Create a file or open an existing file
close	CloseHandle	Close a file
read	ReadFile	Read data from a file
write	WriteFile	Write data to a file
lseek	SetFilePointer	Move the file pointer
stat	GetFileAttributesEx	Get various file attributes
mkdir	CreateDirectory	Create a new directory
rmdir	RemoveDirectory	Remove an empty directory
link	(none)	Win32 does not support links
unlink	DeleteFile	Destroy an existing file
mount	(none)	Win32 does not support mount
umount	(none)	Win32 does not support mount
chdir	SetCurrentDirectory	Change the current working directory
chmod	(none)	Win32 does not support security (although NT does)
kill	(none)	Win32 does not support signals
time	GetLocalTime	Get the current time

Differences between library functions and system calls

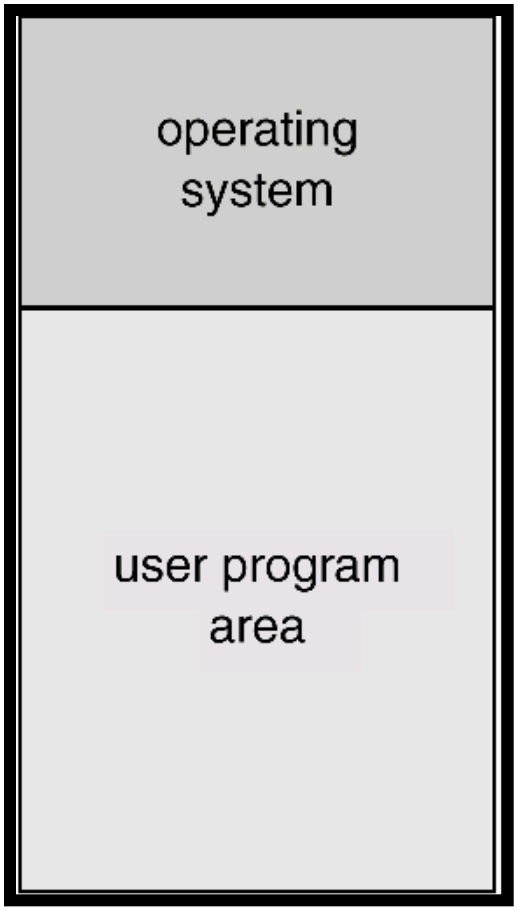
- System calls run in kernel-mode but library functions run in user-mode and may call system calls.
- System calls are not linked to user programs.
- There are only a small number of system calls.

Types of Operating System:

1.Batch OS

- First OS
- Developed around mid 1950s by GM
- IBSYS had become quite famous by early 1960s
- Special piece of software called monitor
- User submits the job on cards or tape to a computer operator who batches the job together sequentially and places the entire batch on an input device for use by the monitor
- Monitor loads one job, executes it, then loads the next and so on

Batch Systems:

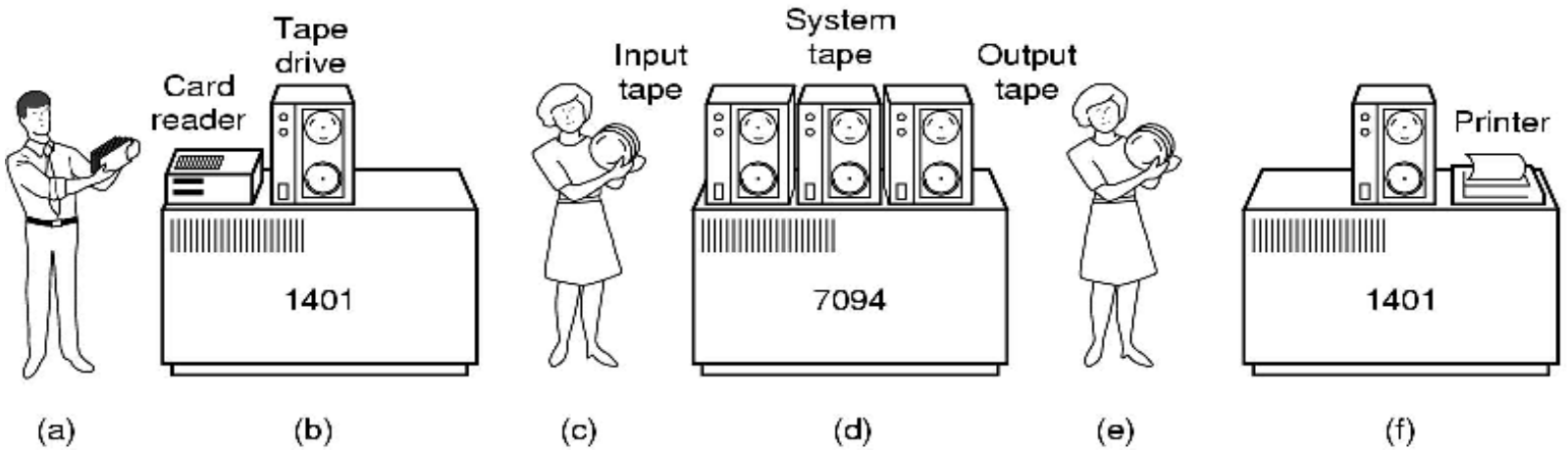


Memory Layout for a Simple Batch System

Batch OS

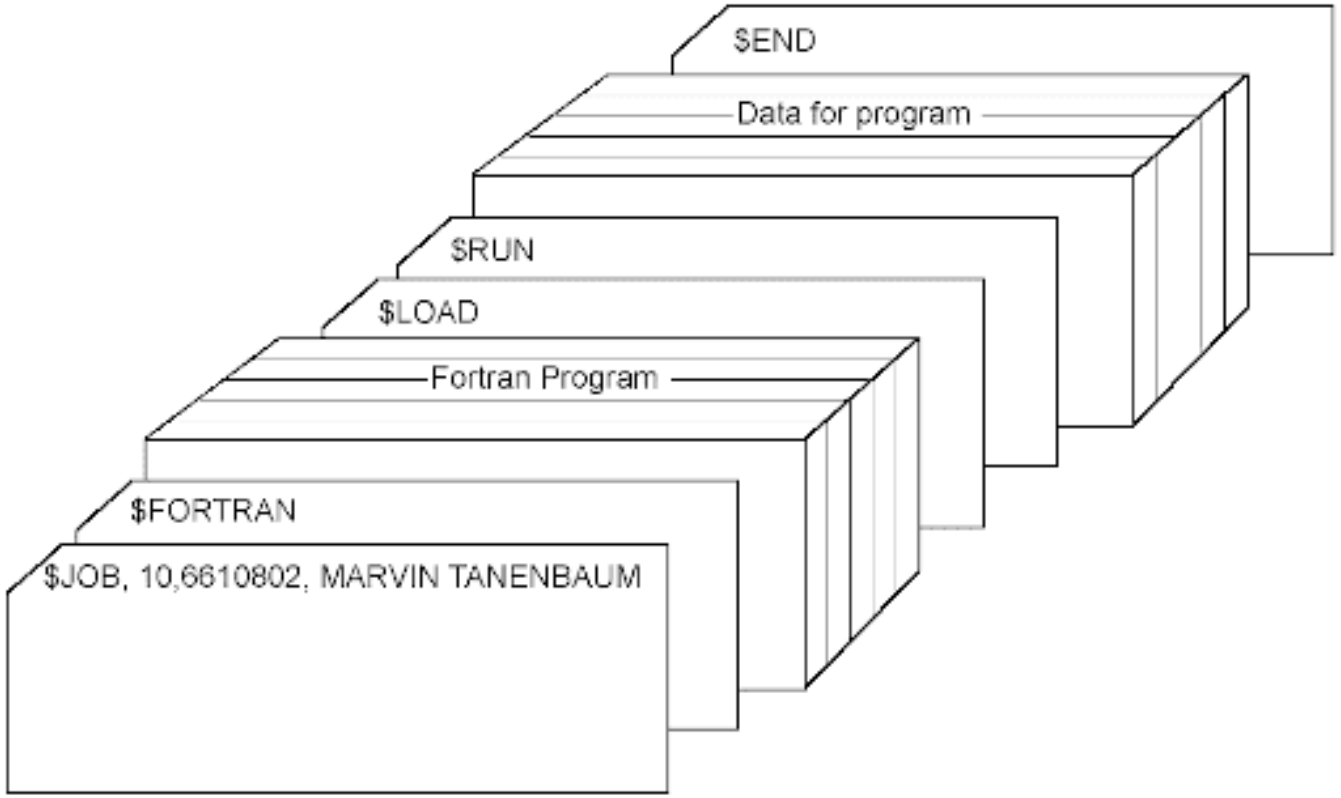
- Monitor controls sequence of events
  - Must always be in memory and available for execution
  - Called **resident monitor**
  - Rest of the monitor is utilities loaded as subroutines when required
  - Executes in **kernel mode** (contrast with **user mode)**
- Job is placed in user program area and control is passed to it
- Upon completion, the job returns the control to the monitor which proceeds with reading the next job
- Results of each job are sent to an output device
- JCL: Job Control Language – meta instructions

Batch Systems:



- Bring cards to 1401
- Read cards onto input tape
- Put input tape on 7094
- Perform the computation, writing results to output tape
- Put output tape on 1401, which prints output

Batch Systems:

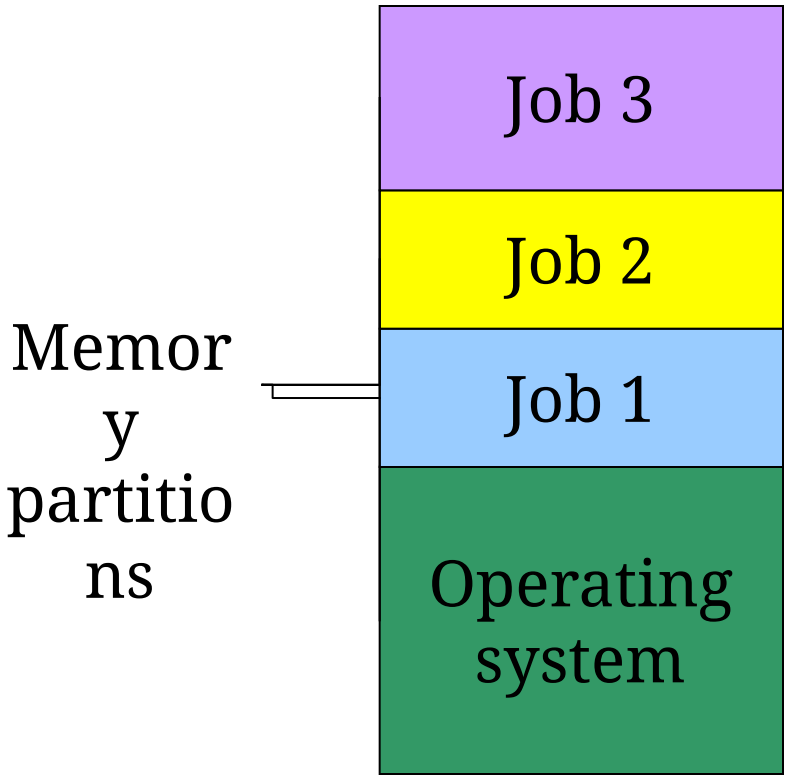


Structure of a typical FMS job

Issues

- CPU idle during I/O operations
- CPU time is wasted; *CPU utilization* is low

2. Multiprogramming Systems:

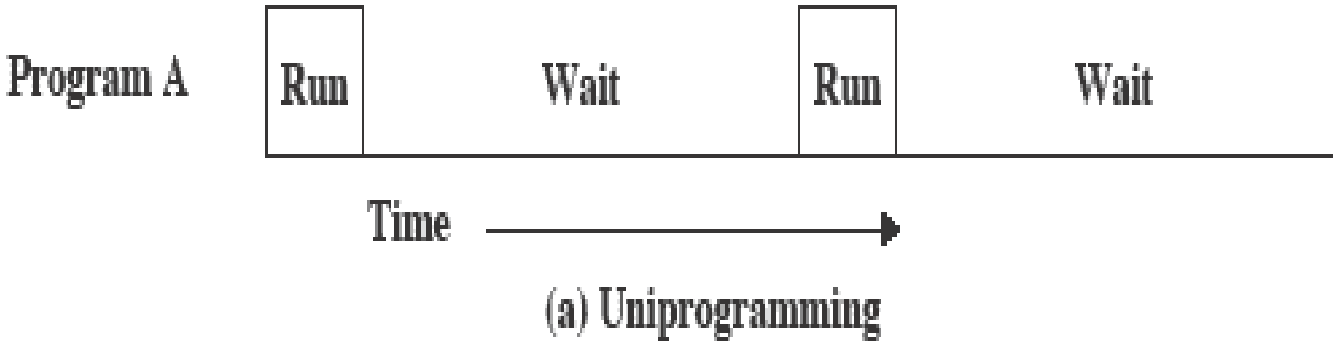


- Multiple jobs in memory
  - Protected from one another
- Operating system protected from each job as well
- Resources (time, hardware) split between jobs
- Still not interactive
  - User submits job
  - Computer runs it
  - User gets results minutes (hours, days) later

Increases CPU utilization
Job pool-jobs waiting on disk for allocation into memory
Job scheduling-selection of job from pool
CPU scheduling

Uniprogramming

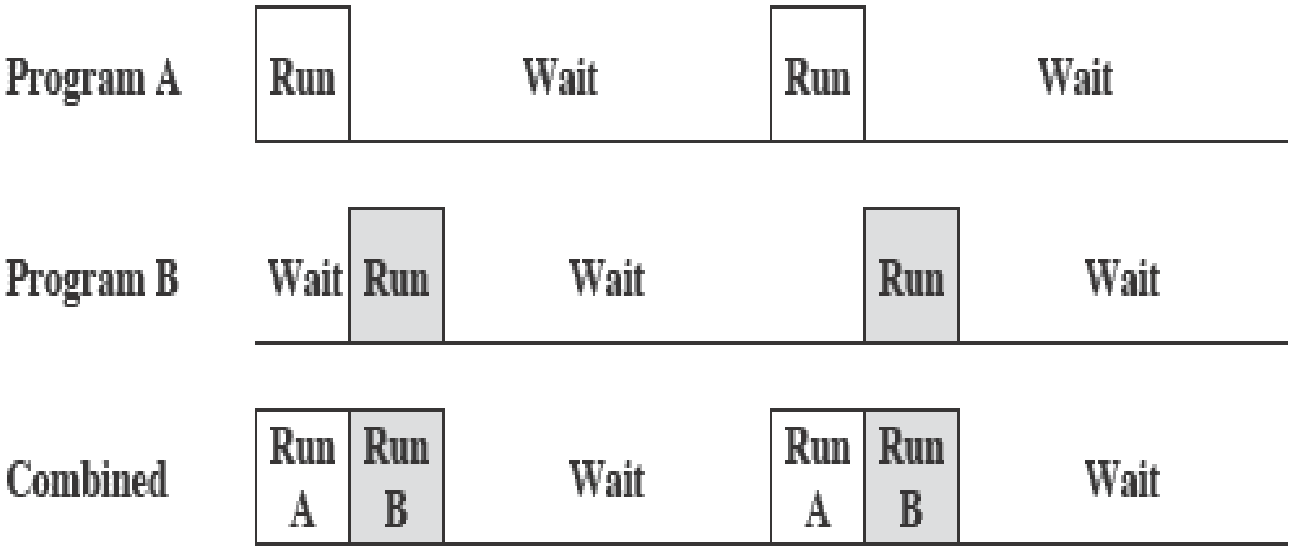
Processor must wait for I/O instruction to complete before preceding



(a) Uniprogramming

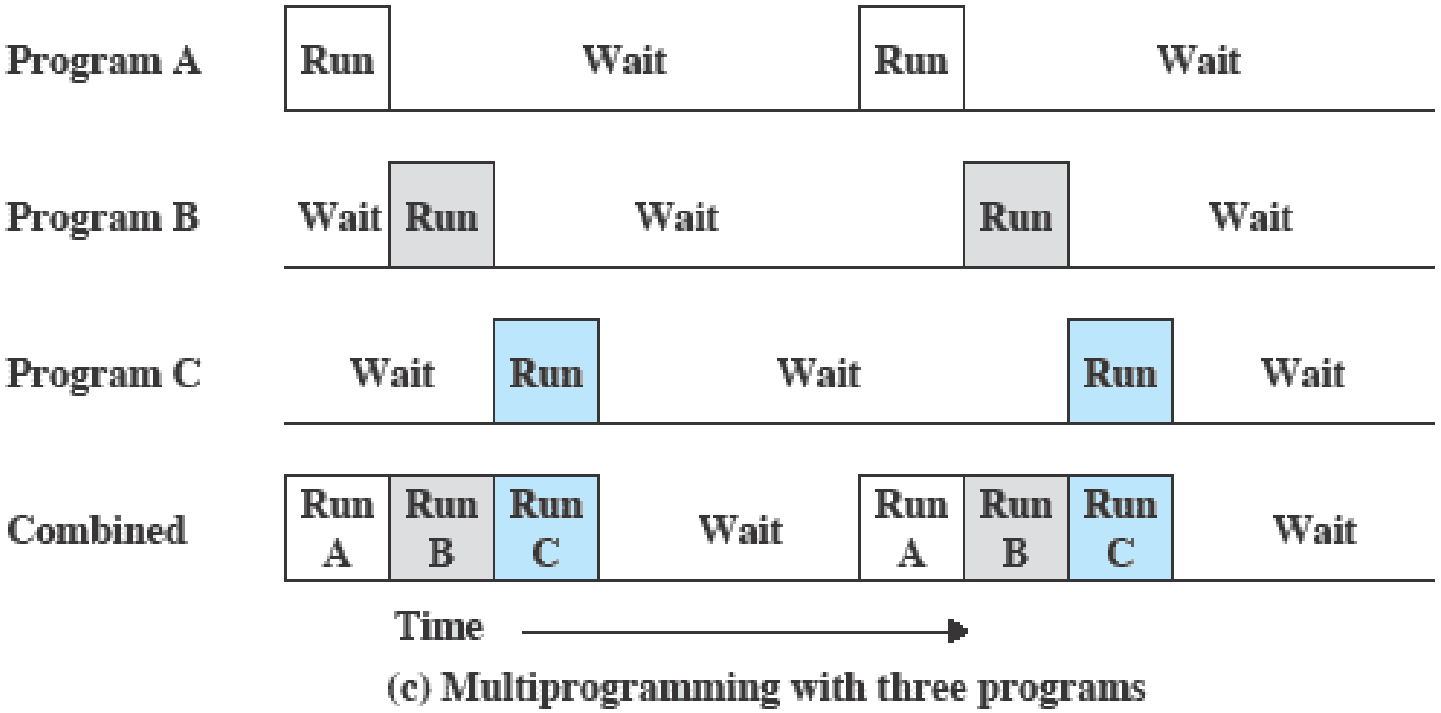
Multiprogramming

- When one job needs to wait for I/O, the processor can switch to the other job



(b) Multiprogramming with two programs

Multiprogramming



(c) Multiprogramming with three programs

It is the central theme of modern operating systems

3. Time-sharing or Multitasking:

- Fast CPU switching between programs : Idea is to give a feeling of concurrency
- CPU switches to another program
  - ✔ When current program blocks
  - ✔ When its currently allocated CPU time-interval ends
- Also **called** **multitasking**
- logical extension in which CPU switches jobs so frequently that users can interact with each job while it is running, creating **interactive** computing

- **Response time** should be < 1 second
- 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

Batch Multiprogramming versus Time Sharing

Table 2.3 Batch Multiprogramming versus Time Sharing

	Batch Multiprogramming	Time Sharing
Principal objective	Maximize processor use	Minimize response time
Source of directives to operating system	Job control language commands provided with the job	Commands entered at the terminal

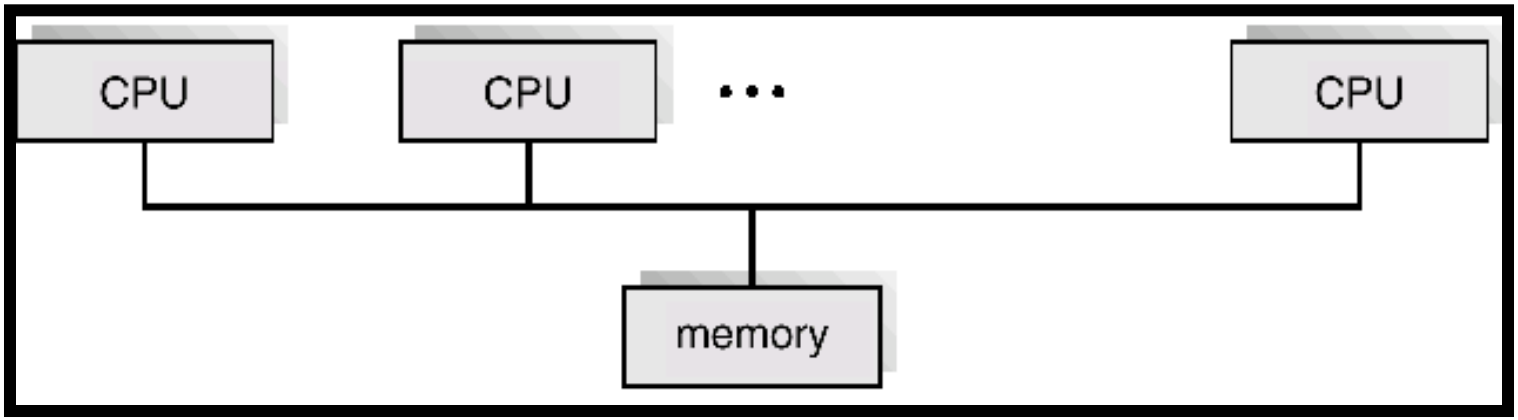
4. Parallel System/Multiprocessor/Tightly coupled:

- As like multiprocessor system have more than one processors in close communication ,sharing the computer, bus clock & sometimes memory & peripheral devices.
- Referred to as tightly coupled system.
- **Advantage**

1. Throughput: get more work done in less time, Increases throughput
2. Economical :It saves money because process can share peripherals, mass storage and power supply.
3. Increases the reliability: Ability to continue providing service proportional to the level of surviving hardware **-graceful degradation -> fault tolerant**

4. Parallel System:

- *Symmetric multiprocessing (SMP)* ← Peer processors
  - Each processor runs an identical copy of the operating system.
  - Many processes can run at once without performance drop.
  - Most modern operating systems support SMP.
  - Ex: Solaries 5,Win2000

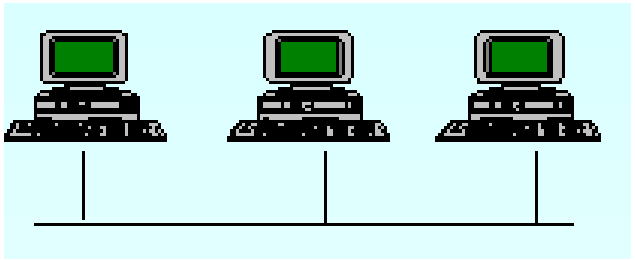


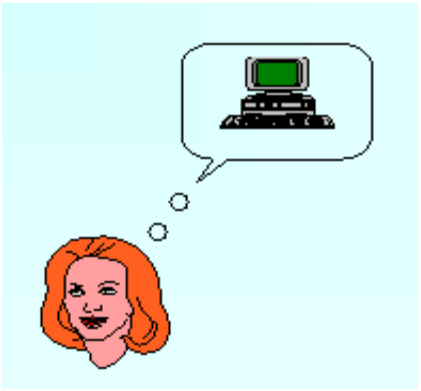
- *Asymmetric multiprocessing*
  - Each processor is assigned a specific task; master processor schedules and allocated work to slave processors.
  - More common in extremely large systems
  - Ex: Solaries 4,

5. Distributed System

A distributed system is:

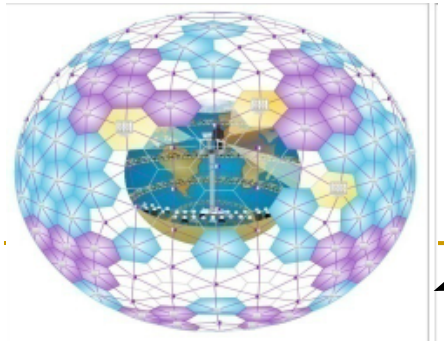
A collection of independent computers that appears to its users as a single coherent system.





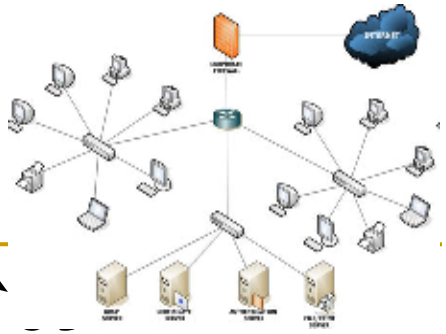
5. Distributed System

- A collection of heterogeneous nodes connected by one or more interconnection networks which provides access to system-wide shared resources and services.
- More than one physical computer, each consisting of CPUs, local memory, and possibly stable storage, and I/O paths to connect it with the environment.
- Interconnections: The processors communicate with one another through various communication lines such as high speed buses or telephone lines.
- Also known as loosely coupled system.

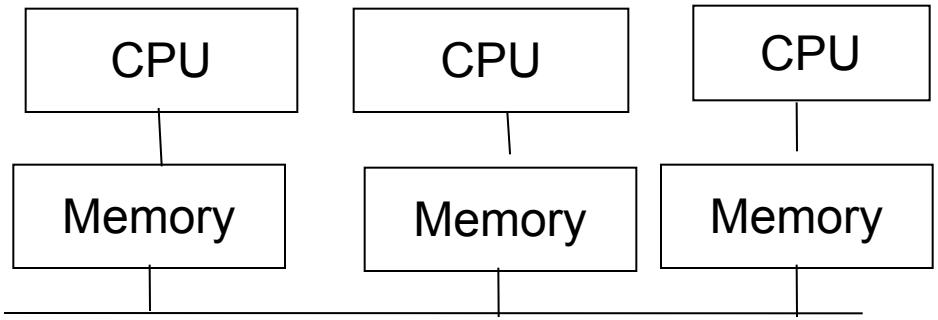


EX: Internet

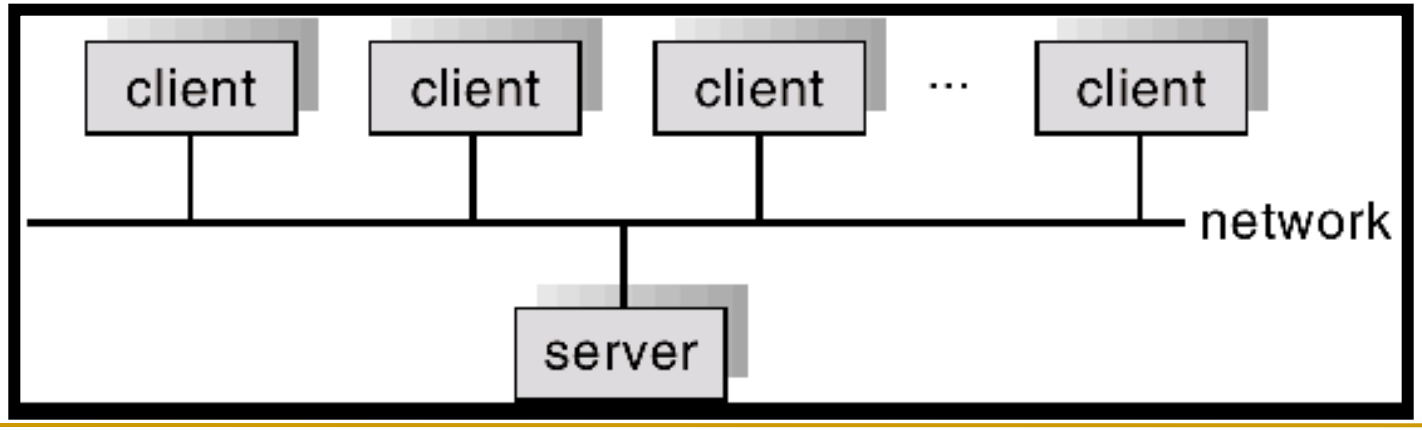
LAN



5. Distributed System



- Advantages of distributed systems.
  - Resources Sharing
  - Computation speed up – load sharing
  - Reliability
  - Communications
- Requires networking infrastructure.
- Ex: Local area networks (LAN) or Wide area networks (WAN)
- May be either client-server or peer-to-peer systems



General Structure of Client-Server

6. Real-time OS.:

- Often used as a control device in a dedicated application such as controlling scientific experiments, medical imaging systems, industrial control systems, and some display systems.
- Well-defined fixed-time constraints.

Ex: Scientific experiments, home appliances

- Real-Time systems may be either hard or soft real-time

6. Real-time OS.:

- Real-Time systems may be either hard or soft real-time
- Hard real-time:
  - Secondary storage limited or absent, data stored in short term memory, or read-only memory (ROM)
  - Conflicts with time-sharing systems, not supported by general-purpose operating systems.
  - Time constraints can be not be relaxed.
- Soft real-time
  - Limited utility in industrial control of robotics
  - Useful in applications (multimedia, virtual reality) requiring advanced operating-system features.
  - Time constraints can be relaxed.

Classes of OS - Summary

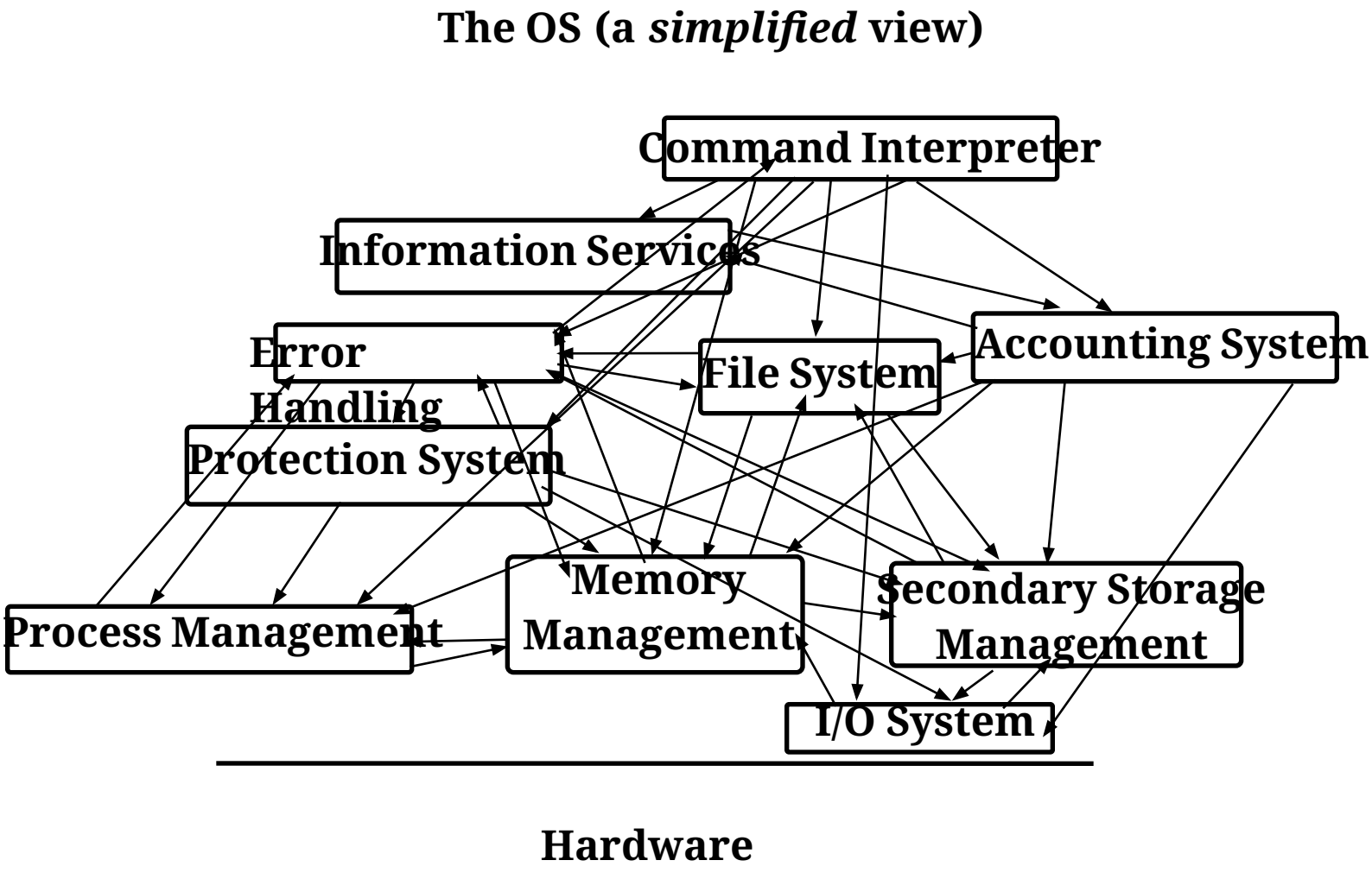
Table 3.2 Key Features of Classes of Operating Systems

OS class	Period	Prime concern	Key concepts
Batch processing	1960s	CPU idle time	Automate transition between jobs
Multiprogramming	1960s	Resource utilization	Program priorities, preemption
Time-sharing	1970s	Good response time	Time slice, round-robin scheduling
Real time	1980s	Meeting time constraints	Real-time scheduling
Distributed	1990s	Resource sharing	Distributed control, transparency

OS Structures

- Monolithic
- Layered
- Ringed
- Virtual Machines ,Hypervisor
- Exo kernels
- Client Server Systems and Microkernels

OS Structure

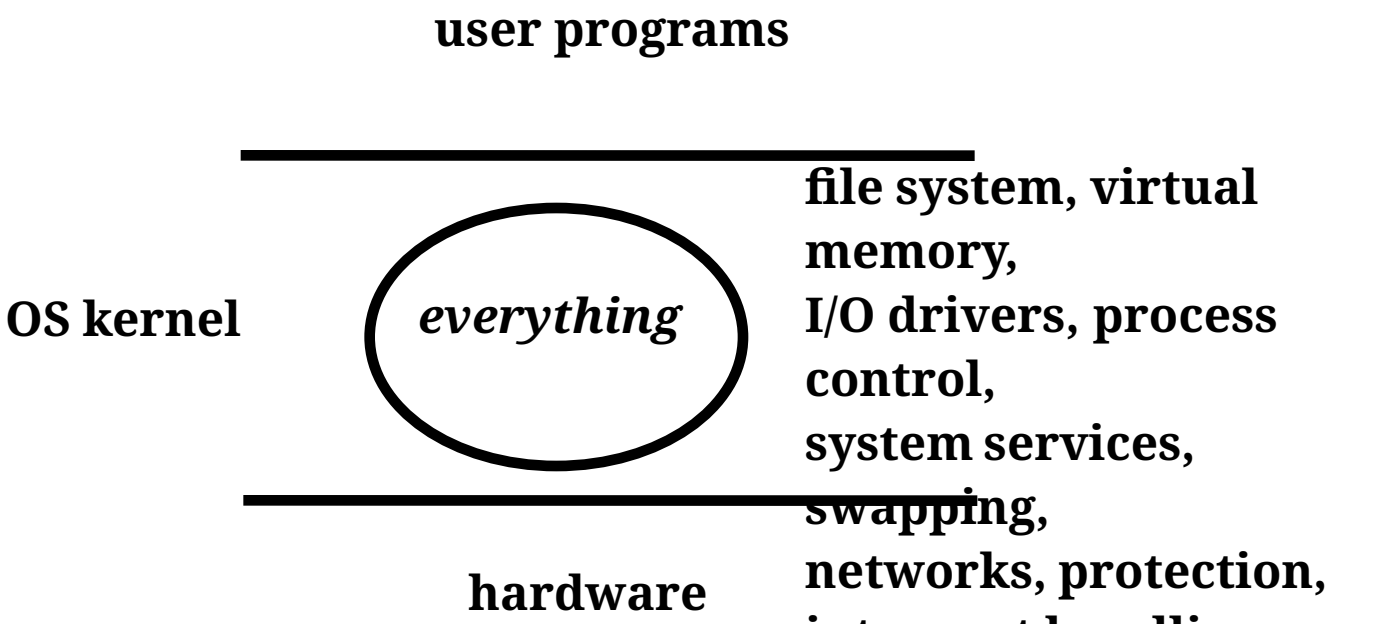


OS Structure

- **An OS consists of all of these components, plus lots of others, plus system service routines, plus system programs (privileged and non-privileged), plus ....**
- **The big issue:**
  - how do we organize all of this?
  - what are the entities and where do they exist?
  - how does these entities cooperate?
- **Basically, how do we build a complex system that's:**
  - performance
  - reliable
  - extensible

Monolithic system

- "The Big mess"
  - Every component contained in kernel
    - direct communication among all elements
    - highly efficient
- Traditionally, systems such as Unix were built as a *monolithic* kernel:



Monolithic system

- Monolithic Systems
  - Entire OS runs as a single program in kernel mode
  - Examples: UNIX, LINUX, DOS, Windows 9x

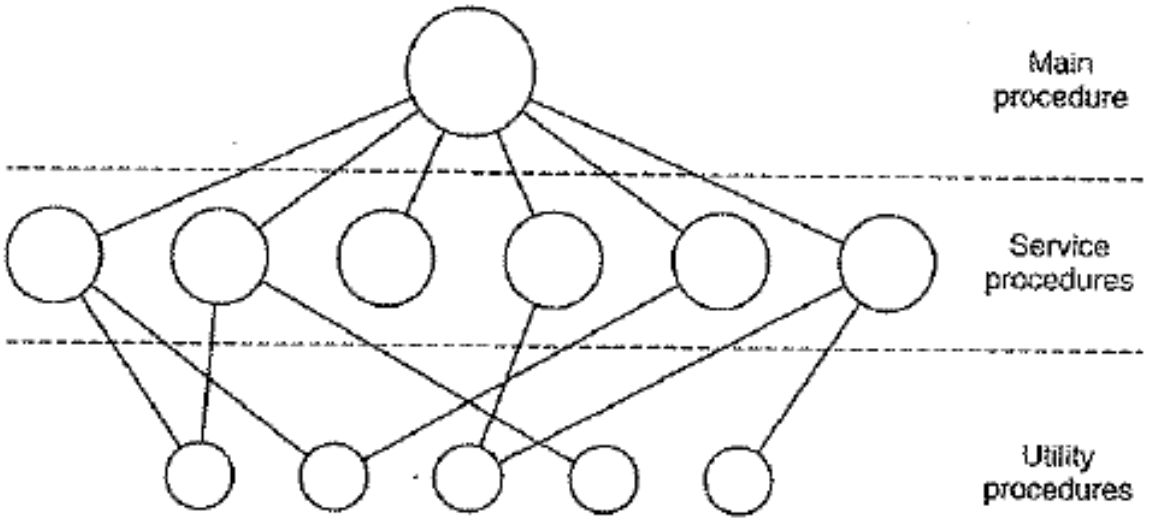


Figure 1-24. A simple structuring model for a monolithic system.

System call/ procedure

Fetch data from user prog.

Monolithic system

- **Problems with monolithic kernels:**
  - hard to understand
  - hard to modify
  - unreliable: a bug *anywhere* causes a system crash
  - hard to maintain
- **Since the beginnings of OS design, people have wanted ways to organize the OS to simplify its design and construction.**

Layered system

- Layered Systems
  - Generalization of previous approach
  - OS constructed as hierarchy of layers
  - The first description of this approach was Dijkstra's THE system.
  - Layers in THE system

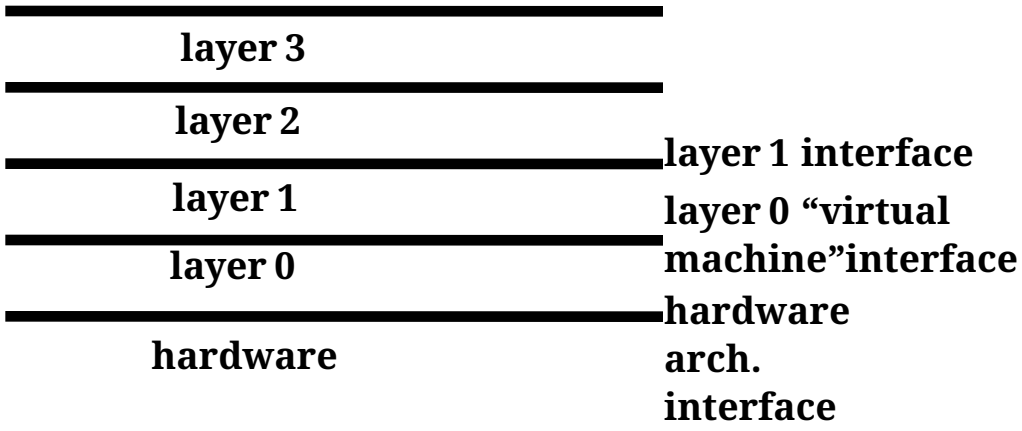
Layer	Function
5	The operator
4	User programs
3	Input/output management
2	Operator-process communication
1	Memory and drum management
0	Processor allocation and multiprogramming

Figure 1-25. Structure of the THE operating system.

Layered system

Traditional approach is layering

- Groups components that perform similar functions into layers
- Each layer communicates only with adjacent layer
- System calls might pass through many layers before completion



Layered system

- **Problems with Layering**
- **Systems must be hierarchical, but real systems are more complex than that, e.g.,**
  - file system would like to be a process layered on VM
  - VM would like to use files for its backing store I/O
- **Approach is not flexible.**
- **Often has poor performance due to layer crossings.**
- **Systems are often modelled as layered structures but not built that way (for better or worse).**

Virtual Machines

OS/360 -> batch system->time sharing

First example: VM/370

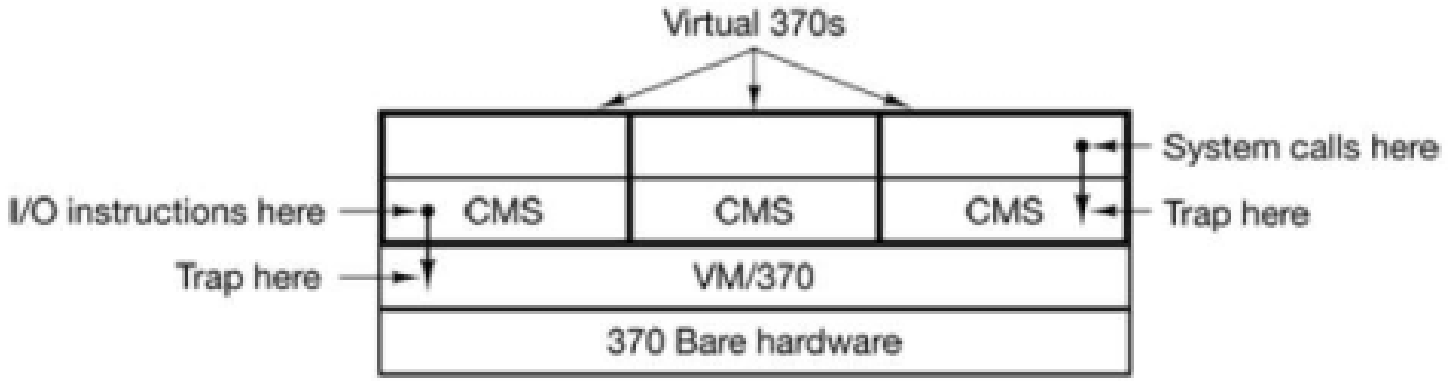
- Virtual machine monitor runs on bare hardware and does multi programming providing several virtual machines to next layer up
- These virtual machines are exact copies of bare hardware
- Can run any OS .

Virtual Machines

CMS: Conversational Monitor System

Uses

- Different Oss can now serve different specific purposes
- Running old MS-DOS programs (16 bit) on 32 bit or 64 bit Windows
- Running Java Programs in JVM

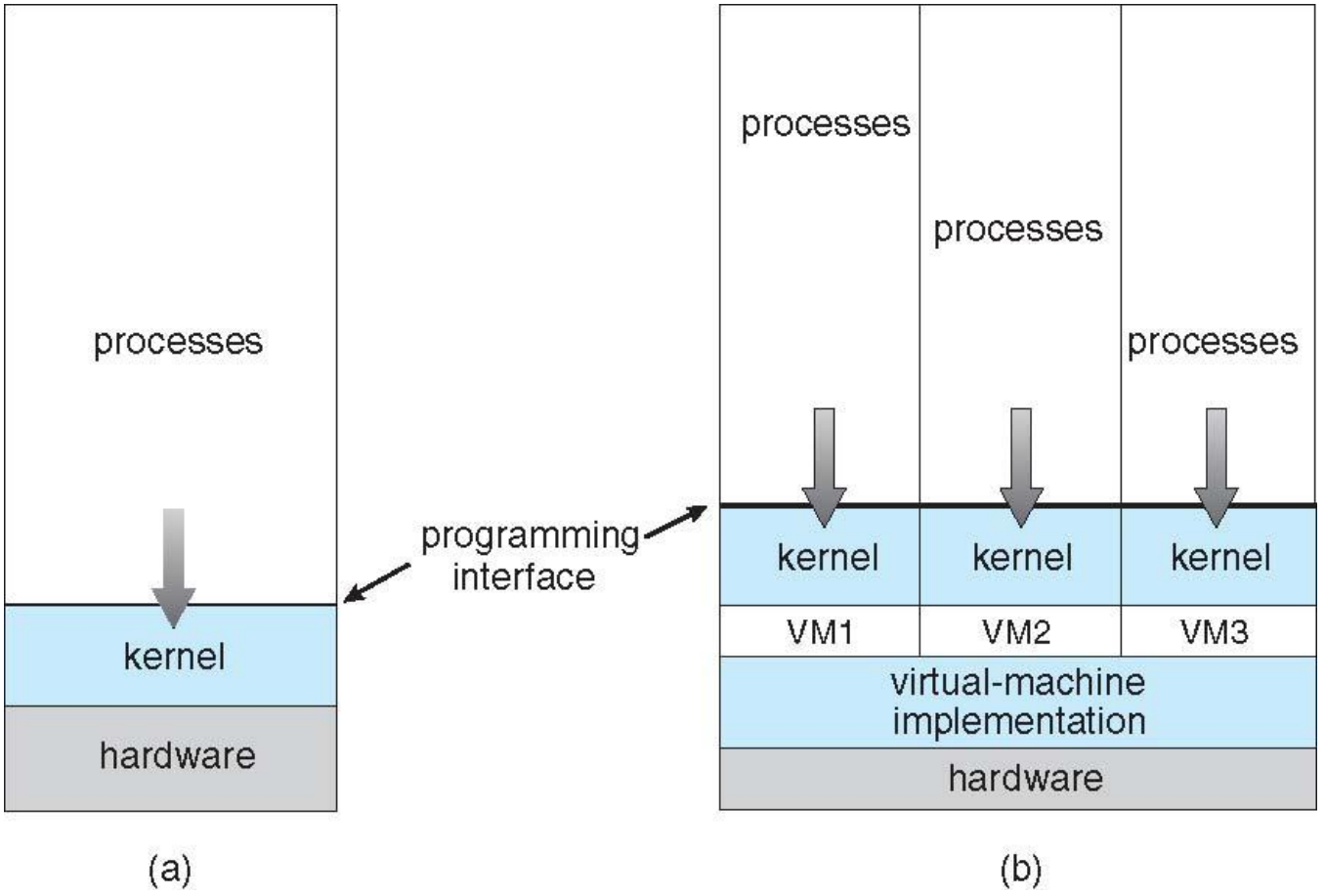


Structure of VM/370 with CMS

Virtual Machines

- A **virtual machine** takes the layered approach to its logical conclusion.
- It treats hardware and the operating system kernel as though they were all hardware.
- The operating system **host** creates the illusion that a process has its own processor and (virtual memory).
- A virtual machine provides an interface *identical* to the underlying bare hardware.
- Each **guest** provided with a (virtual) copy of underlying computer.

Virtual Machines (Cont.)

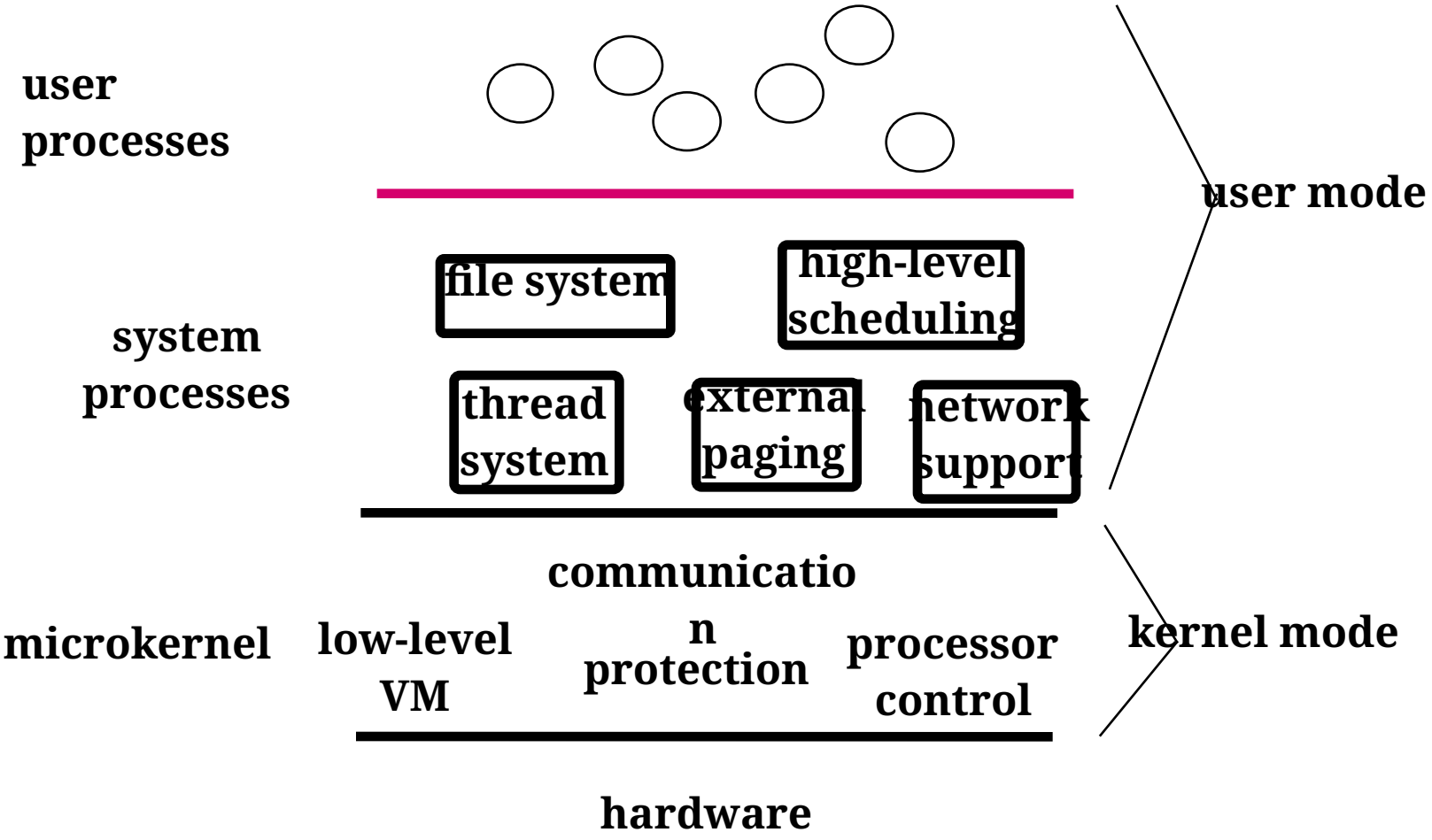


(a) Nonvirtual machine (b) virtual machine

Microkernel

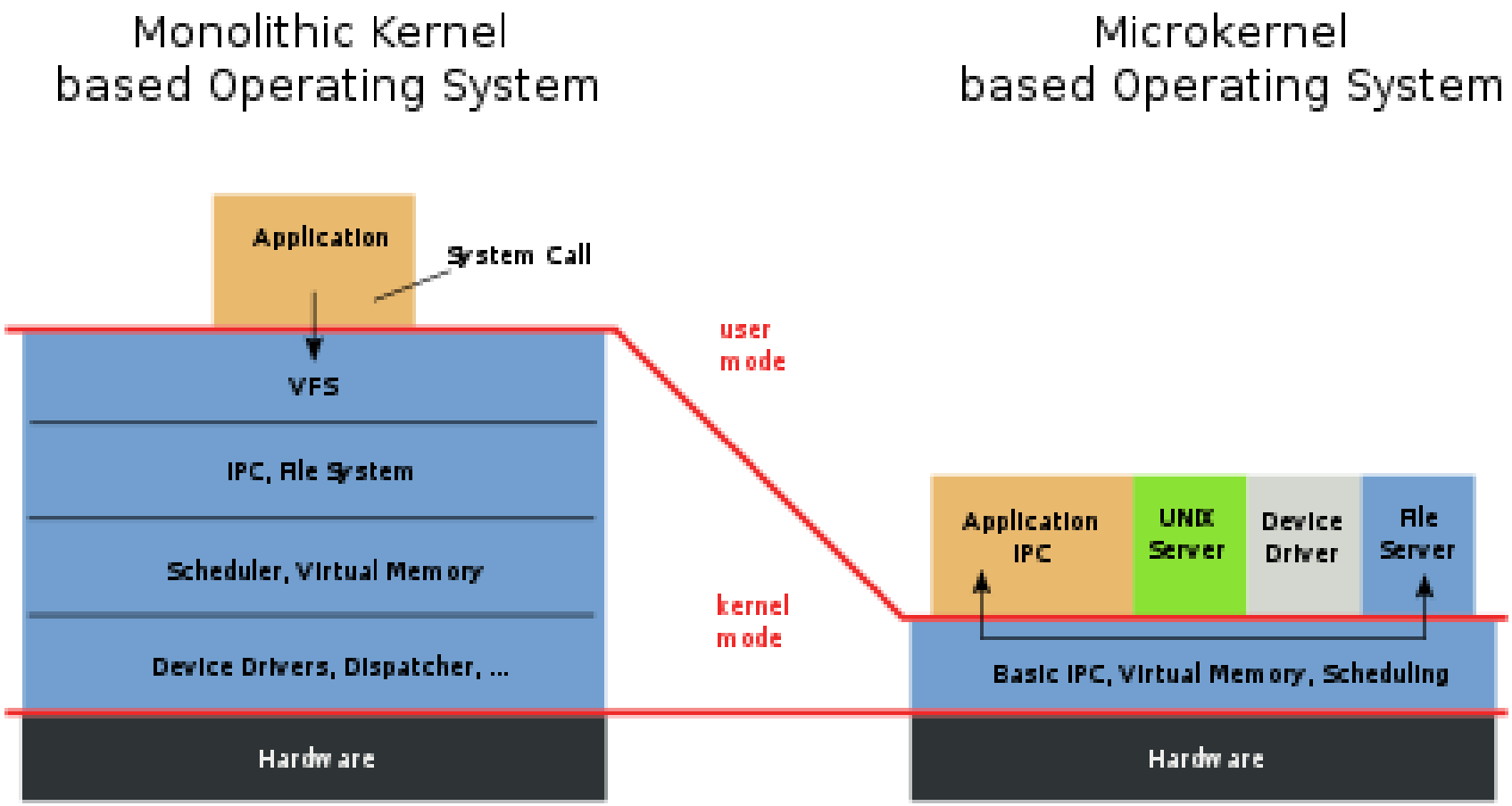
- Moves as much from the kernel into "*user*" space.
- Communication takes place between user modules using message passing
- First microkernel system was Hydra (CMU, 1970)
- Examples of microkernel systems are the CMU Mach system, Chorus (French Unix-like system), and in some ways Microsoft NT/ Windows.
- 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
- Limitation:
  - Performance overhead of user space to kernel space communication

Microkernel



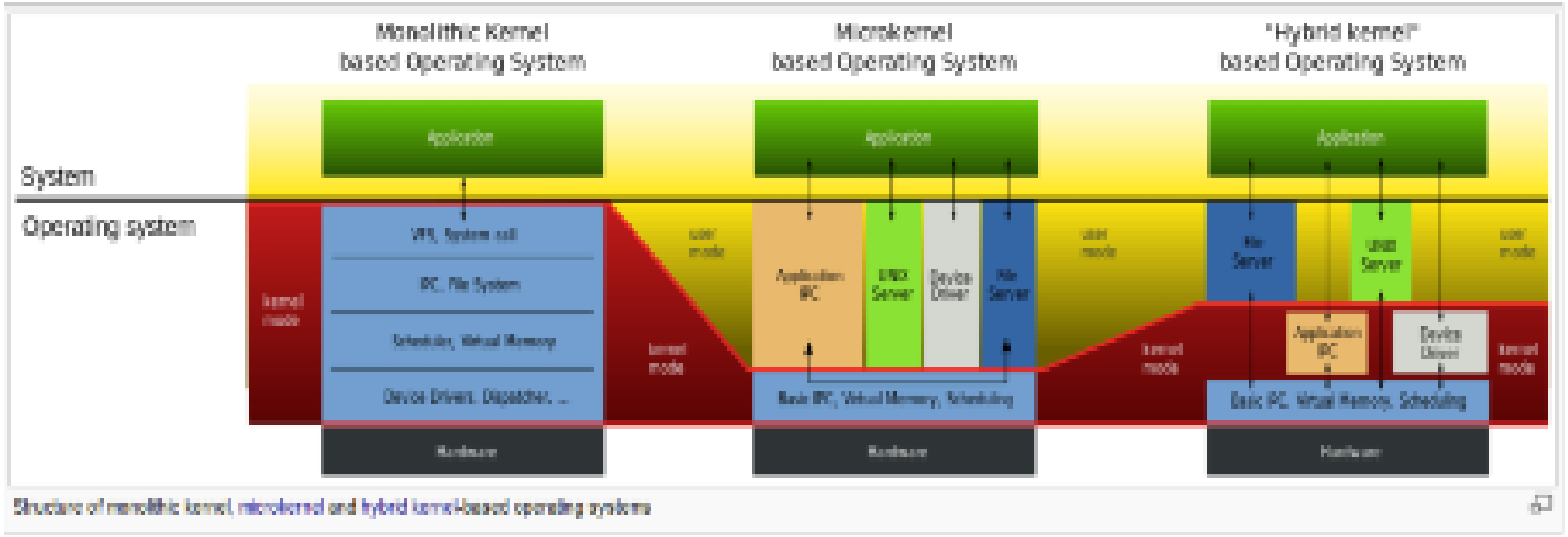
Microkernel System Structure

Monolithic vs Microkernel



Structure of monolithic and microkernel-based operating systems, respectively

Microkernel



Structure of monolithic kernel, microkernel and hybrid kernel-based operating systems