OPERATING SYSTEM

Unit 1

Introduction

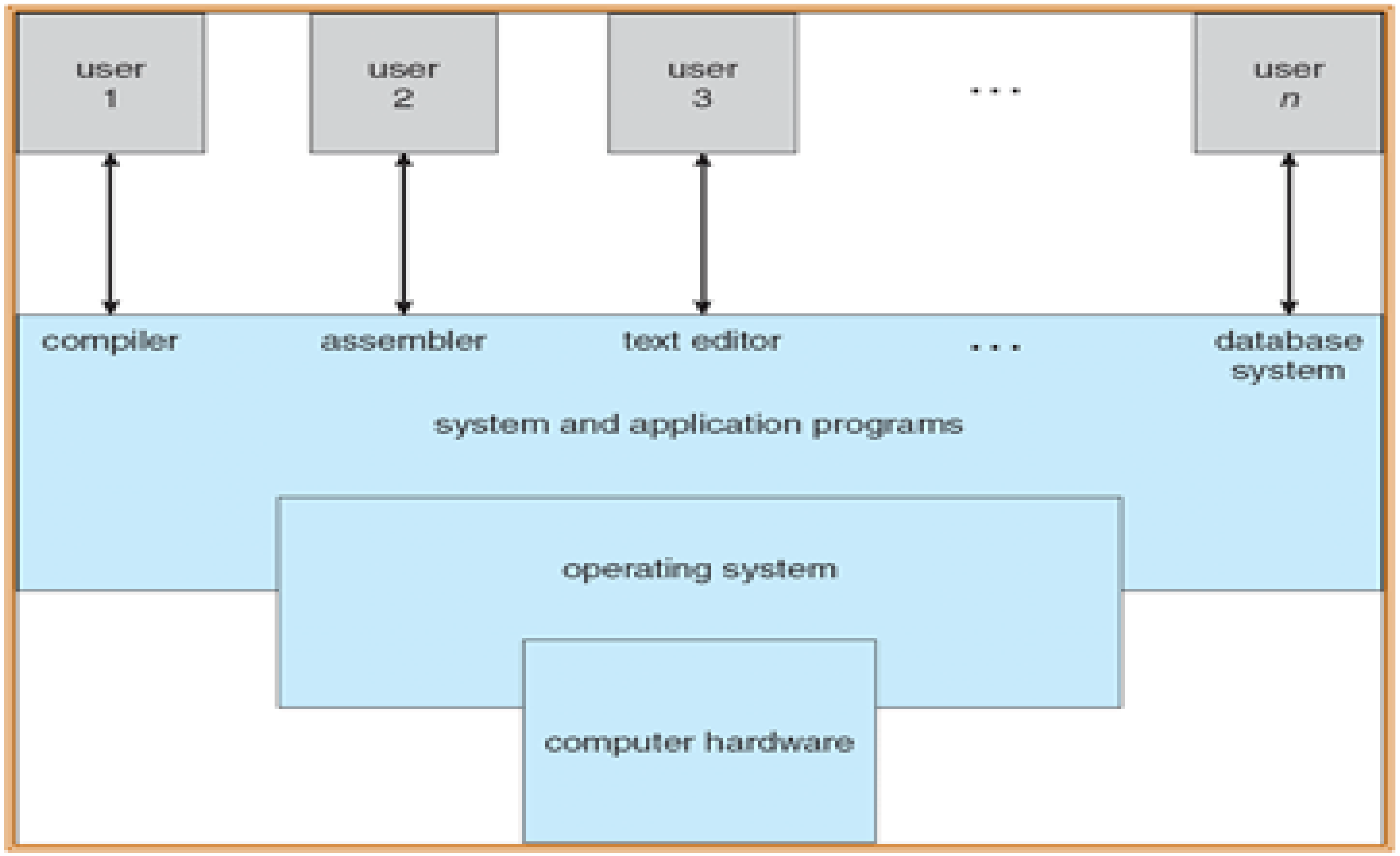
Syllabus

- Introduction to Operating System, definition
- Types of operating system, tasks of operating system
- Concepts of Multiprogramming and Time Sharing,
- Operating System Structures, Components and Services
- System Concept, System programs, Virtual machines
- Process Management: Process concept, Process scheduling, operations on processes
- Co-operating process, Threads, Inter process communication
- CPU scheduling criteria, Scheduling algorithm

Introduction to Operating System, definition

- A program that acts as an intermediary between a user of a computer and the computer hardware
- Hence an operating System can be defined as a system software which provides an interface between an user, application and computer
- 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

Abstract view of the components of a computer system



Abstract view of the components of a computer system

- 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

User view

- Depends on the point of view
- Users want convenience, **ease of use** and **good performance**
  - Don’t care about **resource utilization**
- But shared computer such as **mainframe** or **minicomputer** must keep all users happy
- Users of dedicate systems such as **workstations** have dedicated resources but frequently use shared resources from **servers**
- Handheld computers are resource poor, optimized for usability and battery life
- Some computers have little or no user interface, such as embedded computers in devices and automobiles

System view

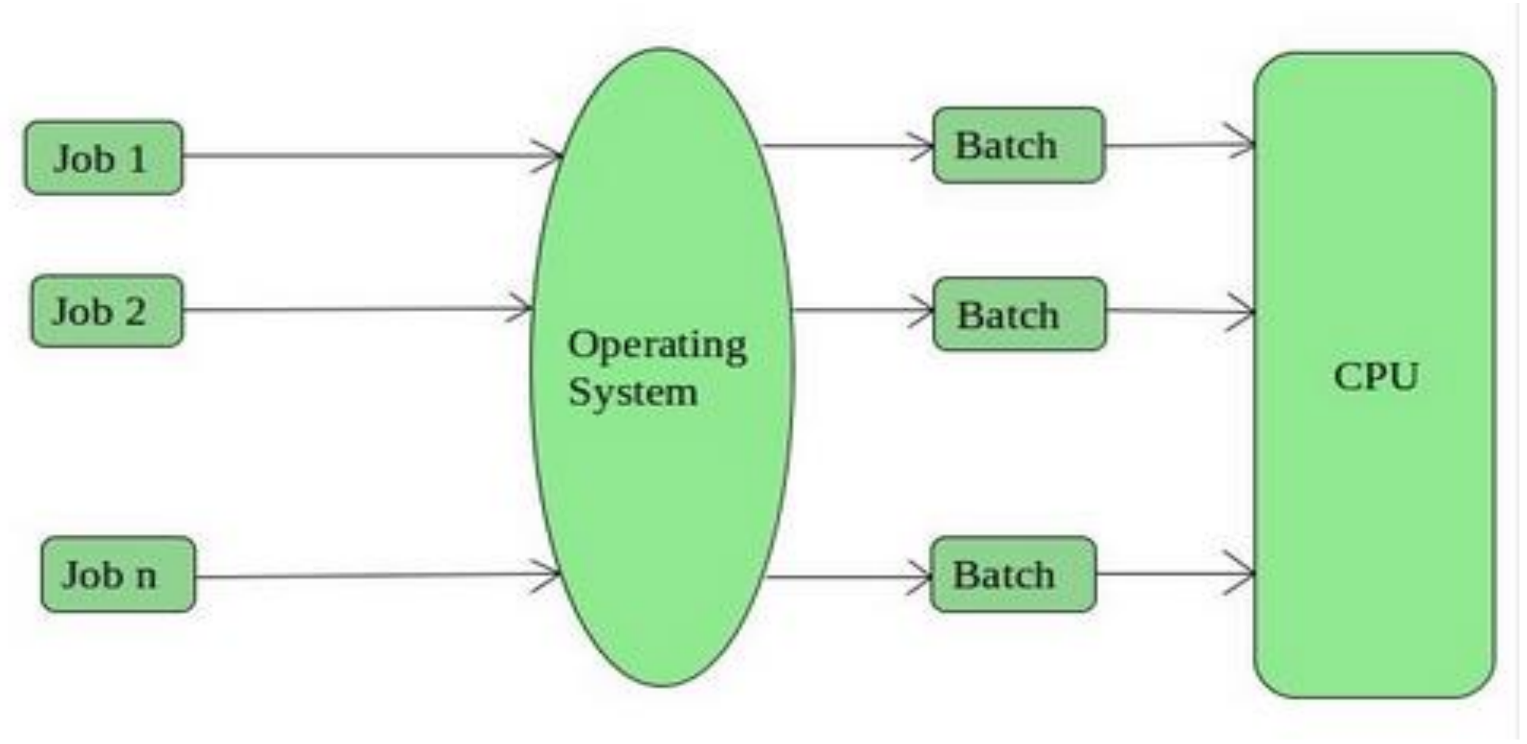
- OS is a **resource allocator**
  - Manages all resources
  - Decides between conflicting requests for efficient and fair resource use
- OS is a **control program**
  - Controls execution of programs to prevent errors and improper use of the computer

Types of Operating Systems

1. Batch Operating System
2. Time-Sharing Operating Systems
3. Distributed Operating System
4. Network Operating System
5. Real-Time Operating System

1. Batch Operating System

- This type of operating system do not interact with the computer directly. There is an operator which takes similar jobs having same requirement and group them into batches. It is the responsibility of operator to sort the jobs with similar needs.



Batch Operating System

Advantages

- Multiple users can share the batch systems
- It is easy to manage large work repeatedly in batch systems

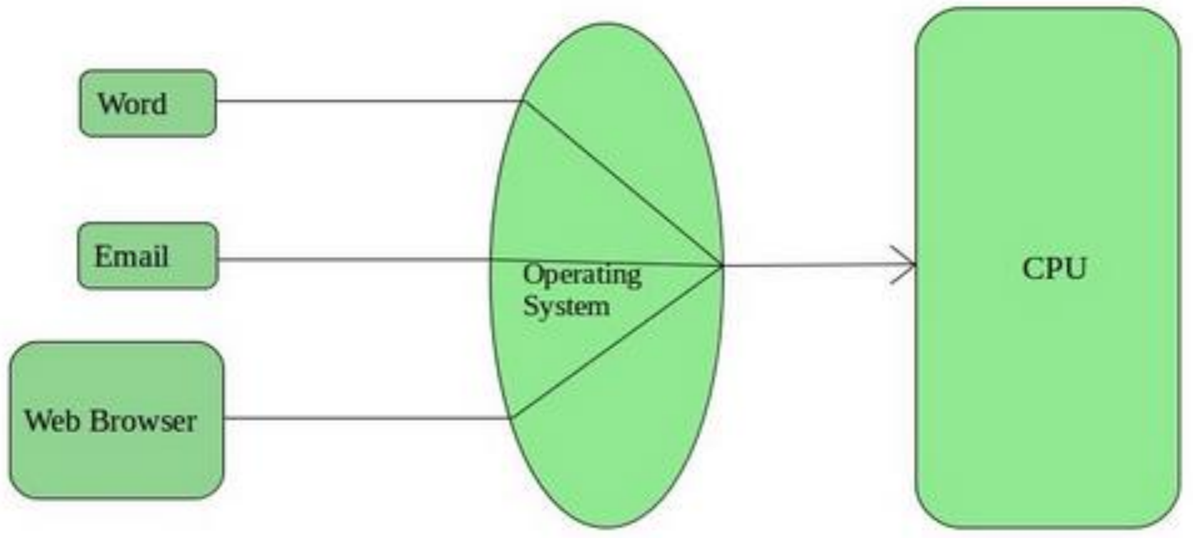
Disadvantages

- Batch systems are hard to debug
- The computer operators should be well known with batch systems
- The other jobs will have to wait for an unknown time if any job fails

Examples IBM’s OS360

2. Time-Sharing Operating Systems

- Each task has given some time to execute, so that all the tasks work smoothly. Each user gets time of CPU as they use single system. These systems are also known as Multitasking Systems.
- The task can be from single user or from different users also. The time that each task gets to execute is called quantum. After this time interval is over OS switches over to next task.



Time-Sharing Operating Systems

Advantages

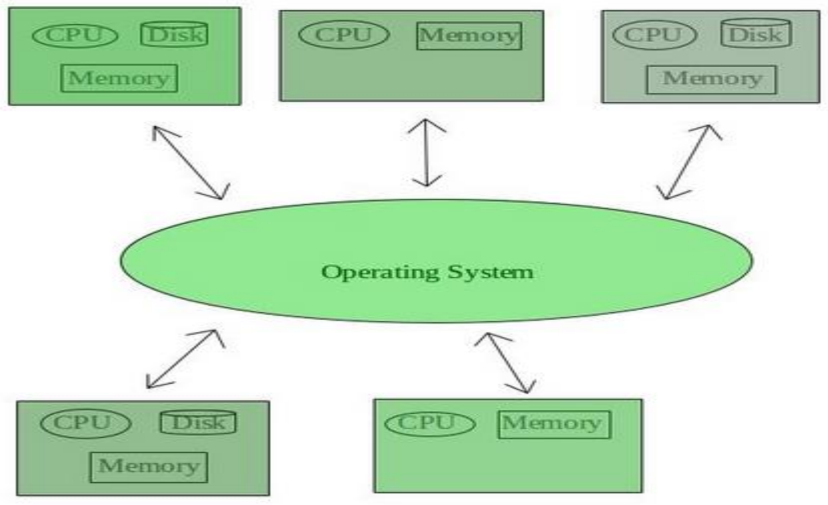
- Each task gets an equal opportunity
- CPU idle time can be reduced

Disadvantages

Examples UNIX, Multics

3. Distributed Operating System

- These are referred as **loosely coupled systems** or distributed systems. These systems processors differ in sizes and functions. The major benefit of working with these types of operating system is that it is always possible that one user can access the files or software which are not actually present on his system but on some other system connected within this network i.e., remote access is enabled within the devices connected in that network.



Distributed Operating System

Advantages

- Failure of one will not affect the other network communication, as all systems are independent from each other
- Since resources are being shared, computation is highly fast and durable

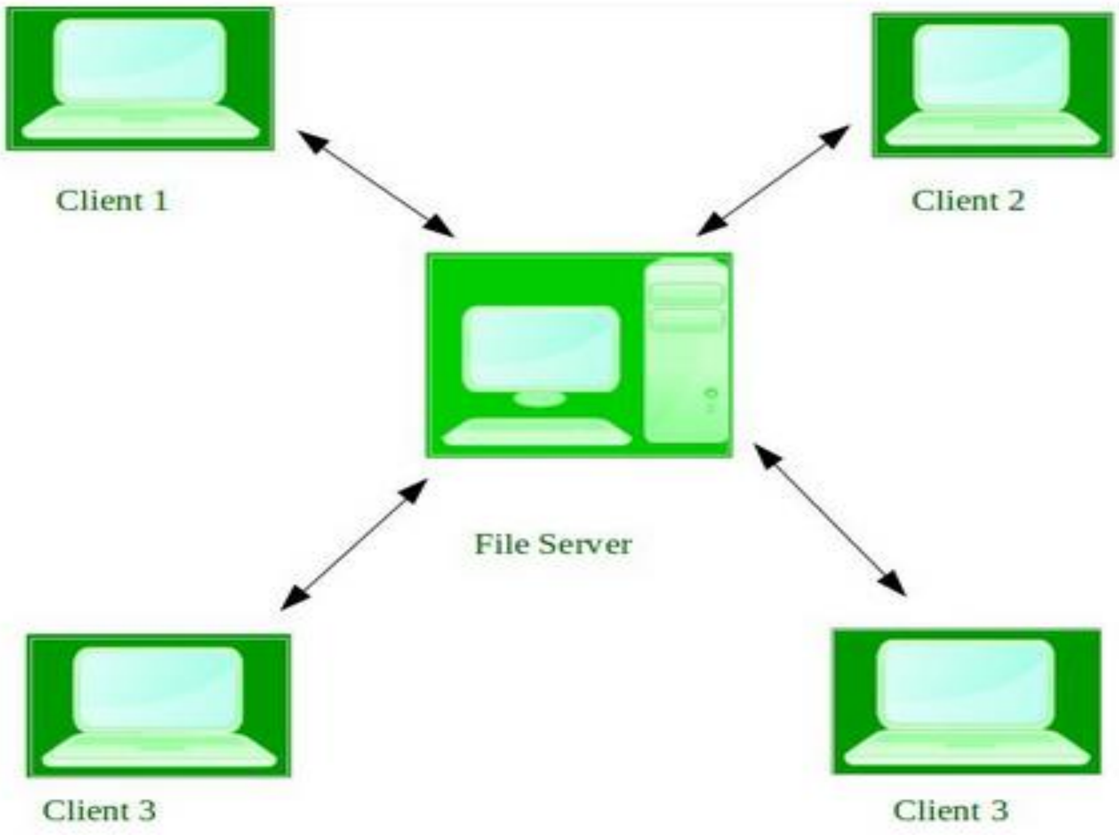
Disadvantages

- Failure of the main network will stop the entire communication

  Examples LOCUS

4.Network Operating System

- These systems runs on a server and provides the capability to manage data, users, groups, security, applications, and other networking functions. These type of operating systems allows shared access of files, printers, security, applications, and other networking functions over a small private network.
- One more important aspect of Network Operating Systems is that all the users are well aware of the underlying configuration, of all other users within the network, their individual connections etc. and that's why these computers are popularly known as **tightly coupled systems**.


Client 1
Client 2
File Server
Client 3
Client 3

Network Operating System

Advantages

- Highly stable centralized servers
- Security concerns are handled through servers

Disadvantages

- Servers are costly
- User has to depend on central location for most operations

Examples Microsoft Windows Server 2003, Mac OS X

5.Real-Time Operating System

- These types of OS serves the real-time systems. The time interval required to process and respond to inputs is very small. This time interval is called **response time**.

➢ **Hard Real-Time Systems:**
These OSs are meant for the applications where time constraints are very strict and even the shortest possible delay is not acceptable. These systems are built for saving life like automatic parachutes or air bags which are required to be readily available in case of any accident. Virtual memory is almost never found in these systems.

➢ **Soft Real-Time Systems:**
These OSs are for applications where for time-constraint is less strict.

Real-Time Operating System

Advantages

- **Real time operating system in embedded system:** Since size of programs are small, RTOS can also be used in embedded systems like in transport and others.

Disadvantages

- **Complex Algorithms:** The algorithms are very complex and difficult for the designer to write on.

Examples : robots, air traffic control systems.

Tasks of operating system

1.**Booting a Computer**

Booting a computer is to load an operating system into the computer's main memory or random access memory (RAM)

2. User-interface

User interfaces. A user interface (UI) refers to the part of an operating system, program, or device that allows a user to enter and receive information

3.Running Programs

To keep the processor busy at all times, if the execution of one program is halted then the operating system switches the processor to run another program

4. Managing Files

The system that an operating system or program uses to organize and keep track of files

5. Memory Management

Memory management is the process of controlling and coordinating computer memory, assigning portions called blocks to various running programs to optimize overall system performance

6. Scheduling Jobs

The system handles prioritized job queues that are awaiting CPU time and it should determine which job to be taken from which queue and the amount of time to be allocated for the job.

7. Managing Devices

Operating system controls all devices attached to computer. All hardware devices are controlled with the help of small software called device drivers

8. Establishing Internet Connection

Operating system provides means to establish internet connection

9. Controlling Network

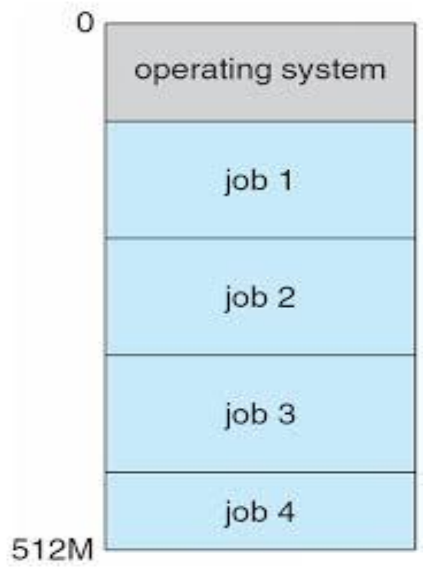
Coordinate the activities of multiple computers across a network. The network operating system acts as a director to keep the network running smoothly.

10. Task Management

The part of the operating system that controls the running of one or more programs (tasks) within the computer at the same time.

Concepts of Multiprogramming and Timesharing

- **Multiprogramming**
- It is 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



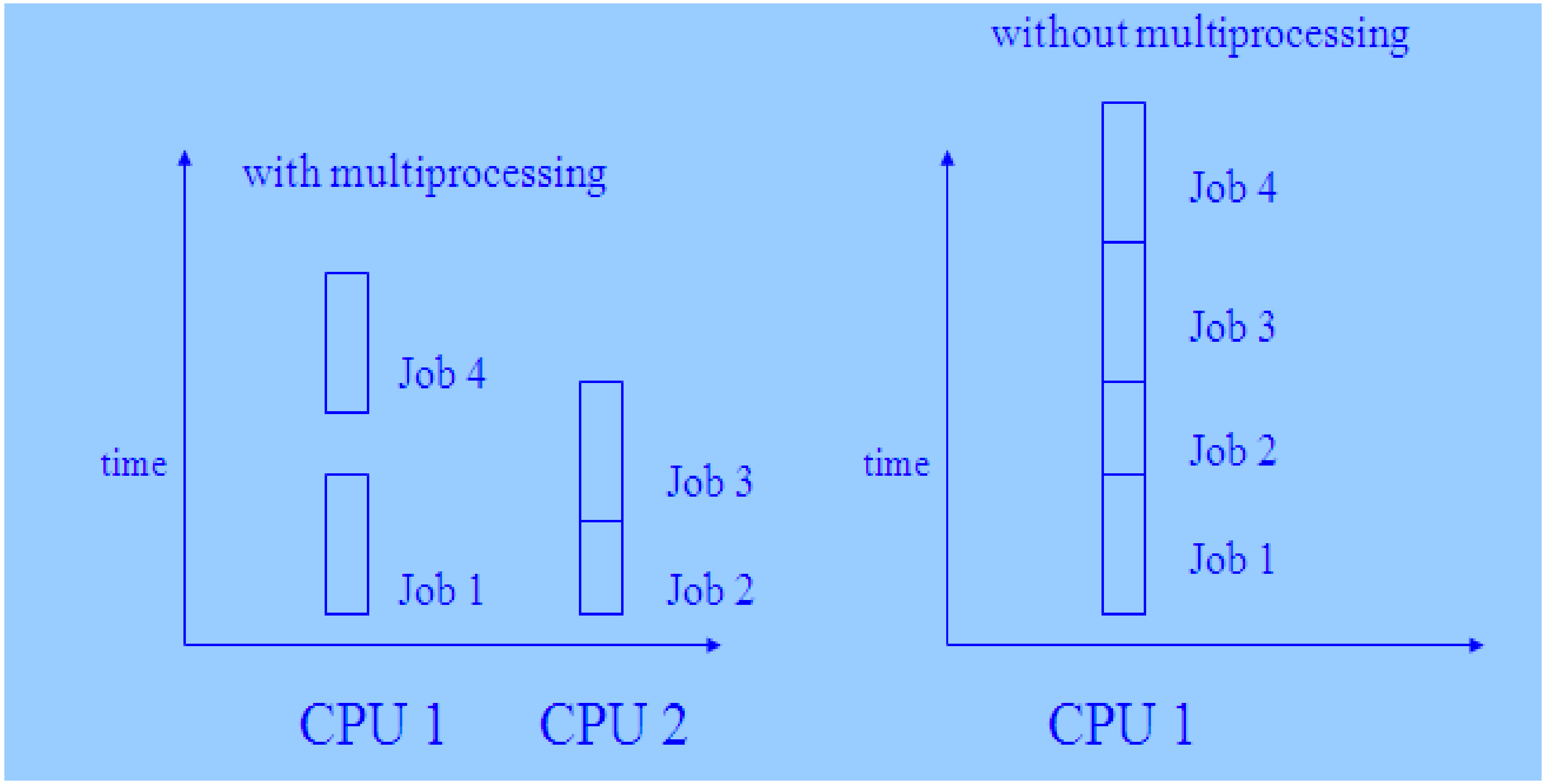
Memory layout for a multiprogramming system

Timesharing

- 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
  - Response time should be less than a second
  - Each user has at least one program executing in memory is called **process**

Parallel processing

- Use 2 or more CPUs to handle jobs

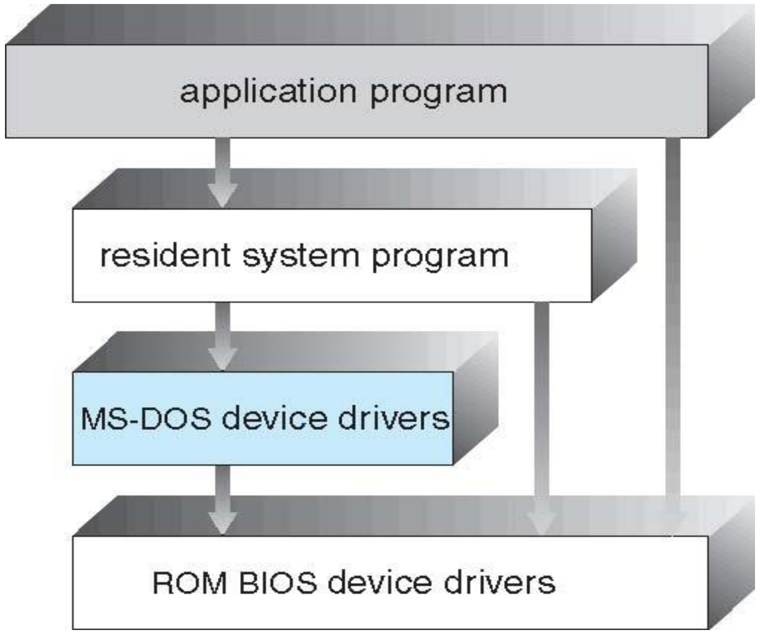


Operating system structure

- General-purpose OS is very large program
- Various ways to structure OS are:-
  - Simple structure – MS-DOS
  - Layered – an abstraction

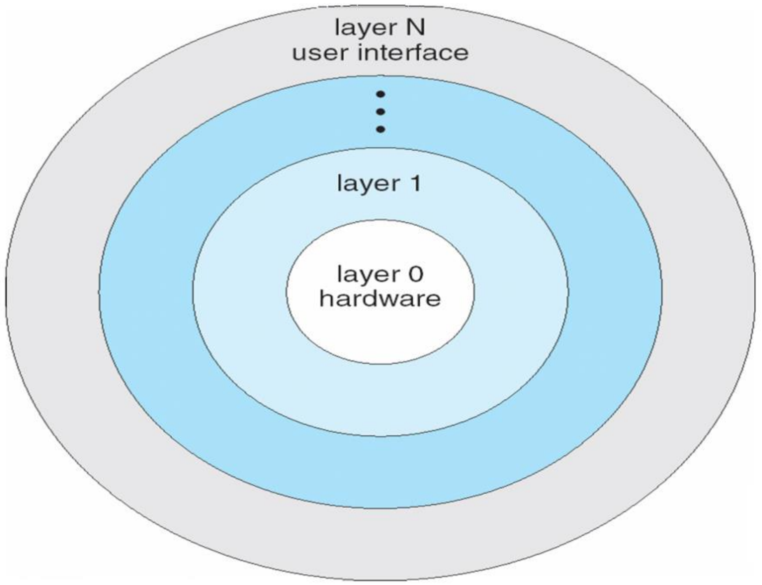
Simple Structure -- MS-DOS

- MS-DOS – written to provide the most functionality in the least space
  - Not divided into modules
  - Although MS-DOS has some structure, its interfaces and levels of functionality are not well separated

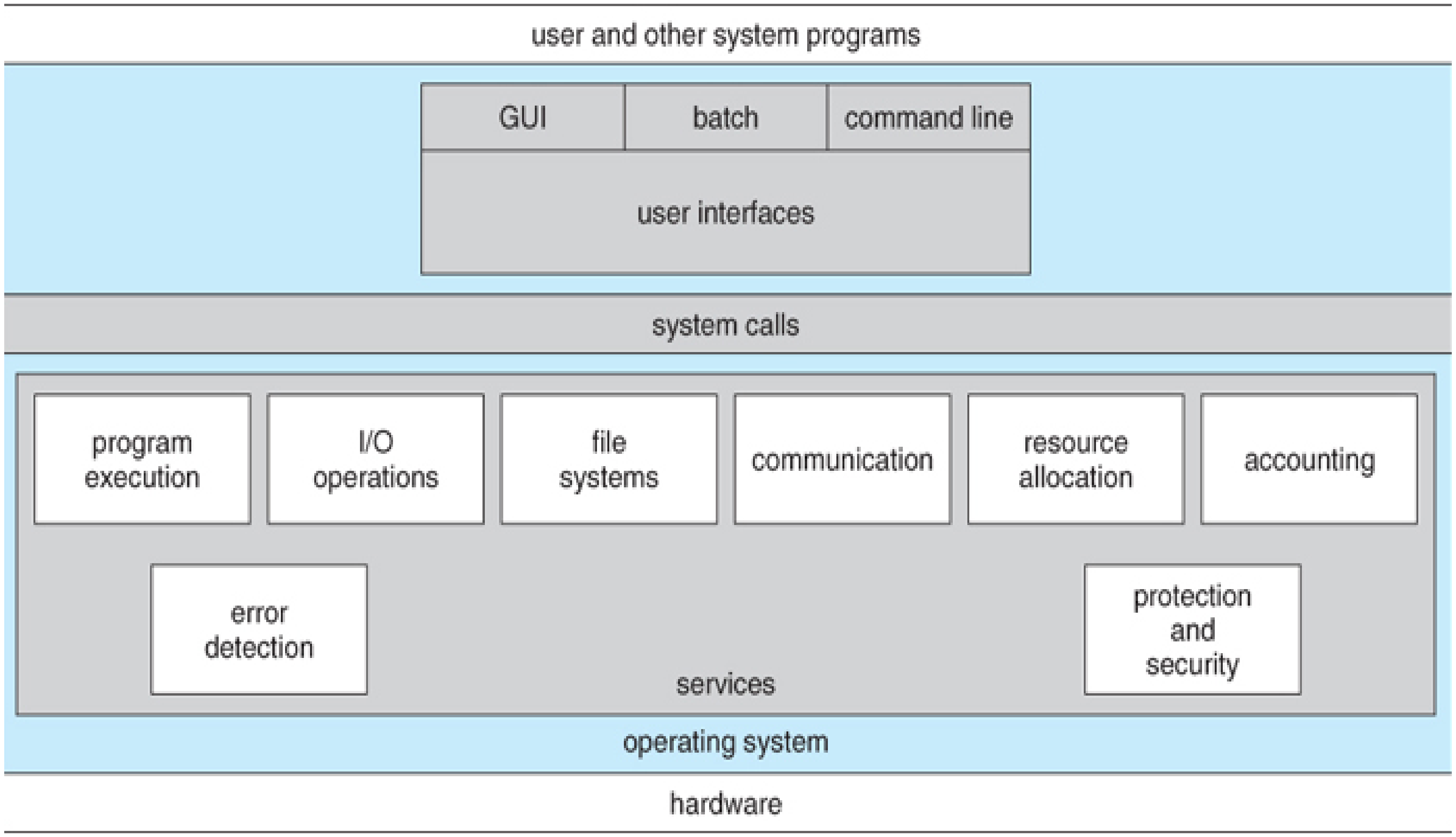


Layered Approach

- The operating system is divided into a number of layers (levels), each 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 such that each uses functions (operations) and services of only lower-level layers



Services of operating system



Services of operating system Conti..

- Operating systems provide an environment for execution of programs and services to programs and users
- One set of 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),   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

Services of operating system Conti..

- **File-system manipulation -** The file system is of particular interest. 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 or through message passing (packets moved by the OS)

Services of operating system Conti..

- **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

System Concept

- User Operating System Interface
- System Calls
- Types of System calls
- System Programs
- Virtual Machines

User Operating System Interface - CLI

CLI or **command interpreter :** Command interpreter is a special program that is running when a job is initiated or when the user first logs on

– The main function of the command interpreter is to get and execute the next user-specified command

– allows direct command entry

– Sometimes implemented in kernel, sometimes by systems program

– Sometimes multiple flavors implemented – **shells**

User Operating System Interface - GUI

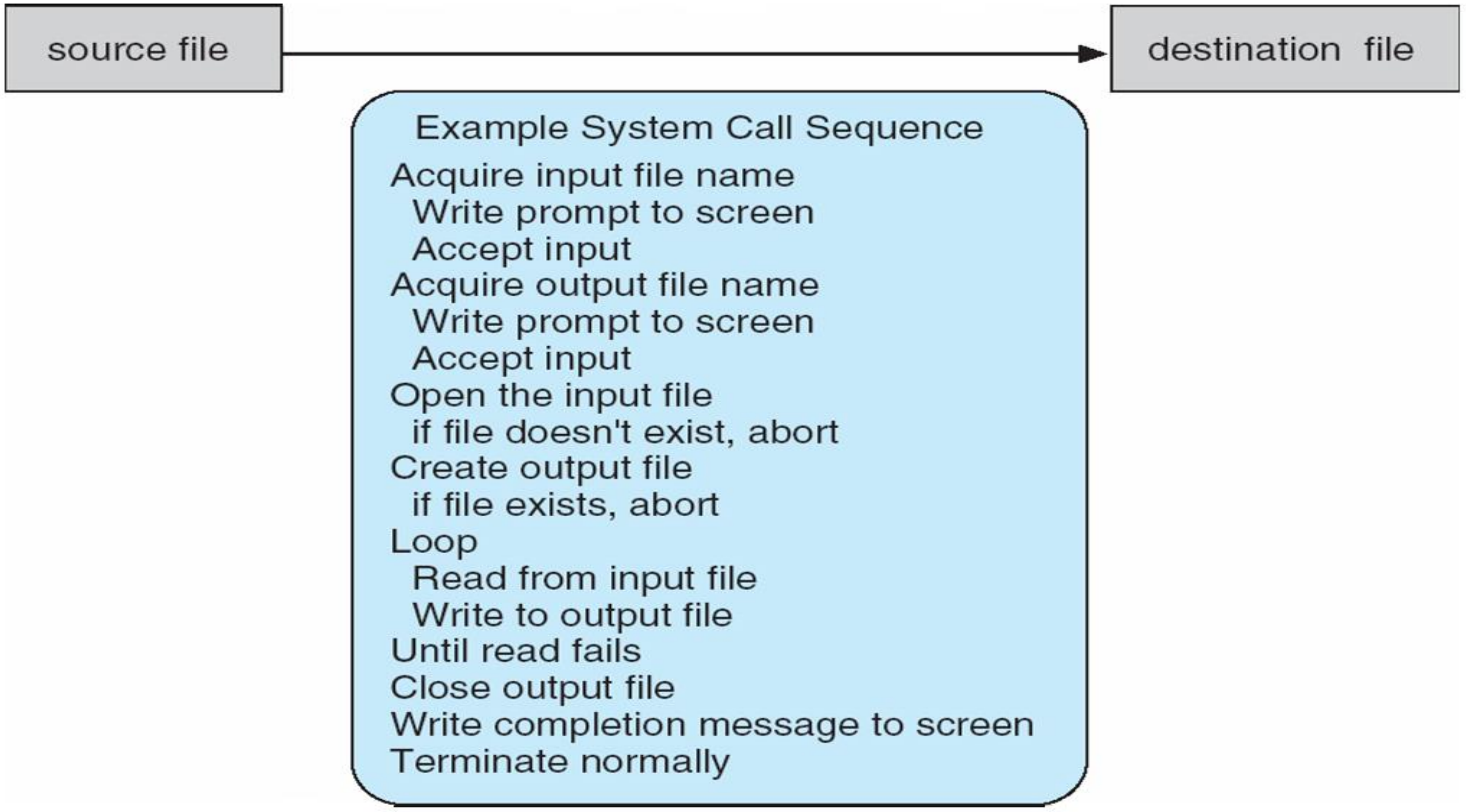
- User-friendly **desktop** interface
  - Usually mouse, keyboard, and monitor
  - Icons represent files, programs, actions, etc
  - GUI provides or allows a mouse based windows and menu system as an interface

System Calls

- System calls provide an interface to the services made available by an operating system
- These calls are generally available as routines written in C and C++.
- Certain low-level tasks may need to be written using assembly-language instructions.
- Users do not see these details.
- Users write functions for APIs which call the system call interface which in turn invokes the System calls

Example of System Calls

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

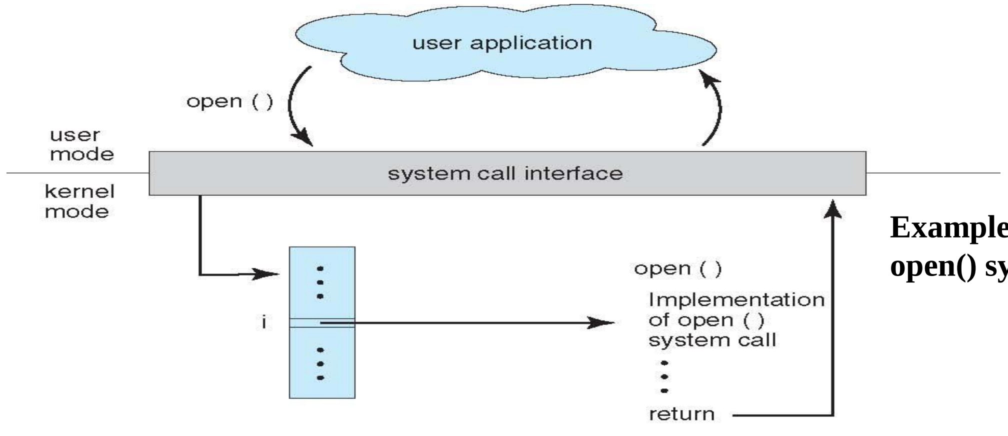


Example System Call Sequence

```
Acquire input file name
  Write prompt to screen
  Accept input
Acquire output file name
  Write prompt to screen
  Accept input
Open the input file
  if file doesn't exist, abort
Create output file
  if file exists, abort
Loop
  Read from input file
  Write to output file
Until read fails
Close output file
Write completion message to screen
Terminate normally
```

System Calls

- Consider a simple program to copy data from one file to another file .
- The first input of the program will be the name of both files.
- These names can be specified depending on the design of Operation System
- Once the file names are obtained, the program opens an input file and create an output file
- Each of these operations requires other System calls and may encounter possible error conditions.



Example: The handling of user application invoking the open() system call

Types of system calls

- **Process control**
  - create process, terminate process
  - end, abort
  - load, execute
  - get process attributes, set process attributes
  - wait for time
  - wait event, signal event
  - allocate and free memory

Types of system calls

- File management
  - create file, delete file
  - open, close file
  - read, write, reposition
  - get and set file attributes
- Device management
  - request device, release device
  - read, write, reposition
  - get device attributes, set device attributes
  - logically attach or detach devices

Types of system calls continued..

- Information maintenance
  - get time or date, set time or date
  - get system data, set system data
  - get and set process, file, or device attributes
- Communications
  - create, delete communication connection
  - send, receive messages if message passing model to host name or process name
    - From client to server
  - Shared-memory model create and gain access to memory regions
  - transfer status information
  - attach and detach remote devices

System Programs(System Utilities)

- System programs provide basic functioning to users so that they do not need to write their own environment for program development (editors, compilers) and program execution (shells).
- In some sense, they are bundles of useful system calls.

  Examples: operating system, interpreter, compiler, editors …..Etc are all system programs.
- They can be divided into these categories:
  - File management
  - Status information
  - File modification
  - Programming language support
  - Program loading and execution
  - Communications

System Programs(System Utilities)

- **File Management:** These programs create, delete, copy, rename, print and generally manipulate files and directories.
- **Status information:** some programs simply ask the system for the date, time, and amount of available memory or disk space, number of users.
- **File Modification:** several text editors may be available to create and modify the content of files stored on disk or other storage devices.
- **Programming language support:** compilers, assemblers debuggers and interpreters
- **Program loading and execution:** once a program is assembled or compiled, it must be loaded into the memory to be executed.
- **Communications:** with help of these programs we can send message to one pc to other pc by remote log in.

Virtual Machines

- In computing, a virtual machine (VM) is an emulation of a computer system.

- The fundamental idea behind a virtual machine is to abstract the hardware of a single computer (the CPU, memory ,disk drives, network interface cards, and so forth) into several different execution environments, thereby creating the illusion that each separate execution environment is running its own private computer.

Benefits:

- the host system is protected from the virtual machines, just as the virtual machines are protected from each other.
- A virtual-machine system is a perfect vehicle for operating-systems research and development

Virtual Machines

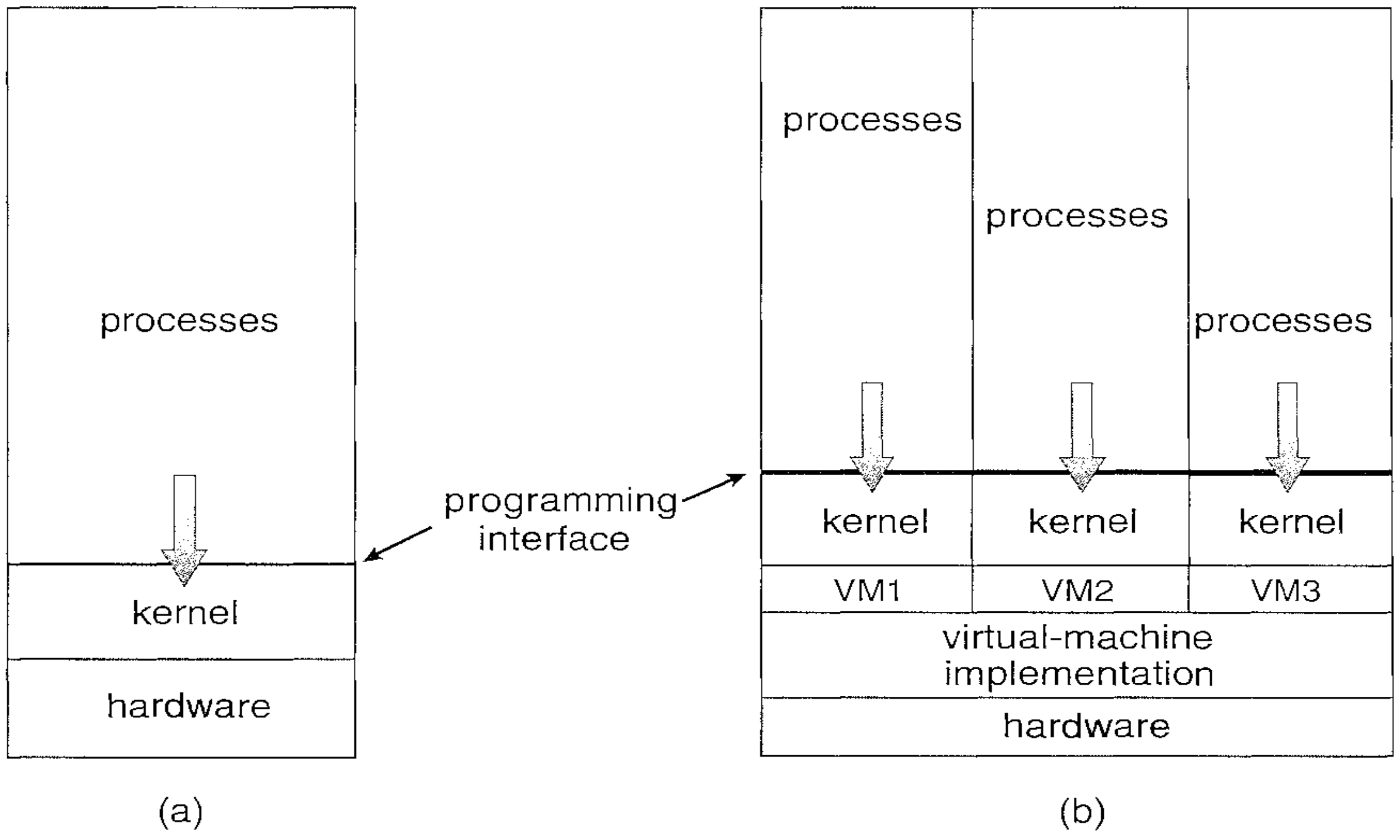


Figure 2.17 System models. (a) Nonvirtual machine. (b) Virtual machine.

Virtual Machines – Examples

Vmware

- Vmware Workstation runs as an application on a host operating system such as Windows or Linux and allows this host system to concurrently run several different guest operating systems as independent virtual machines.

Java Virtual Machine

- The Java Virtual Machine (JVM) is the runtime engine of the Java Platform, which allows any program written in Java or other language compiled into Java bytecode to run on any computer that has a native JVM.

Process Management

Process Concept

- A process can be thought of as a program in execution.
- A process is an active entity.
- A process will need certain resources-such as CPU time, memory, files, and I/O devices to accomplish its task.
- These resources are allocated to the process either when it is created or while it is executing.

Process Concept

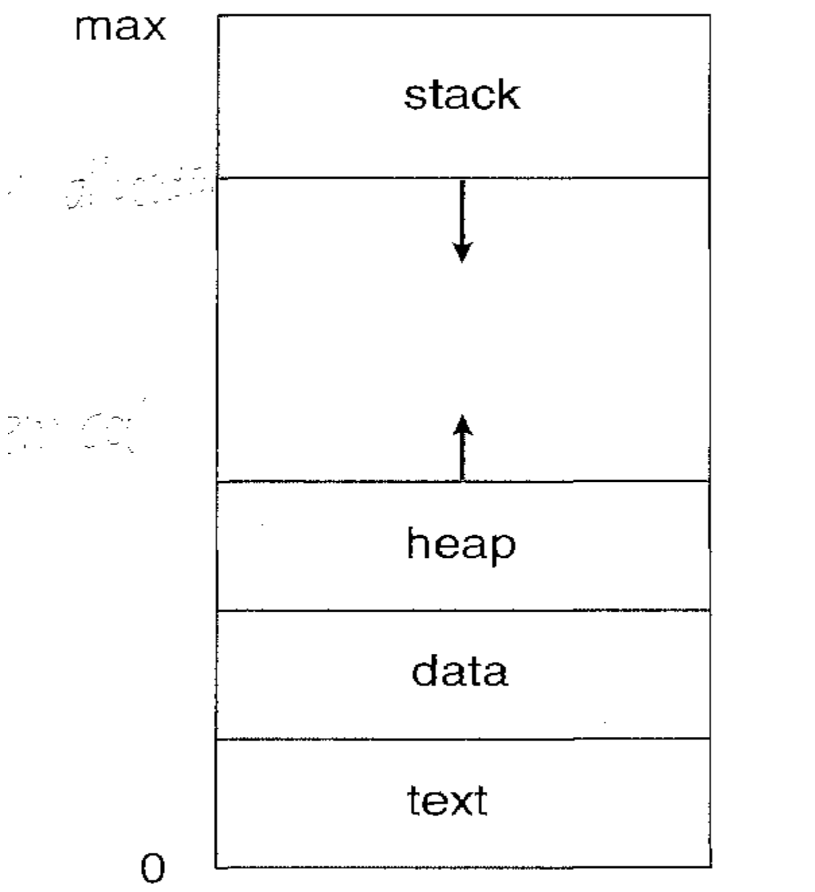


Figure 3.1 Process in memory.

Process in memory

- When a program is loaded into the memory and it becomes a process. Process is represented in memory by using four sections ─ stack, heap, text, and data.

❑ **Stack:** Stack contains the temporary data, such as method/function parameters, return address, and local variables related to a process.

❑ **Heap:** Heap is a dynamically allocated memory to a process when a process is under execution.

❑ **Text:** Text section of the process tells about the current activity which is represented by the value of Program Counter and the contents of the processor's registers.

❑ **Data:** Data section contains the global and static variables.

Process Concept – Process State

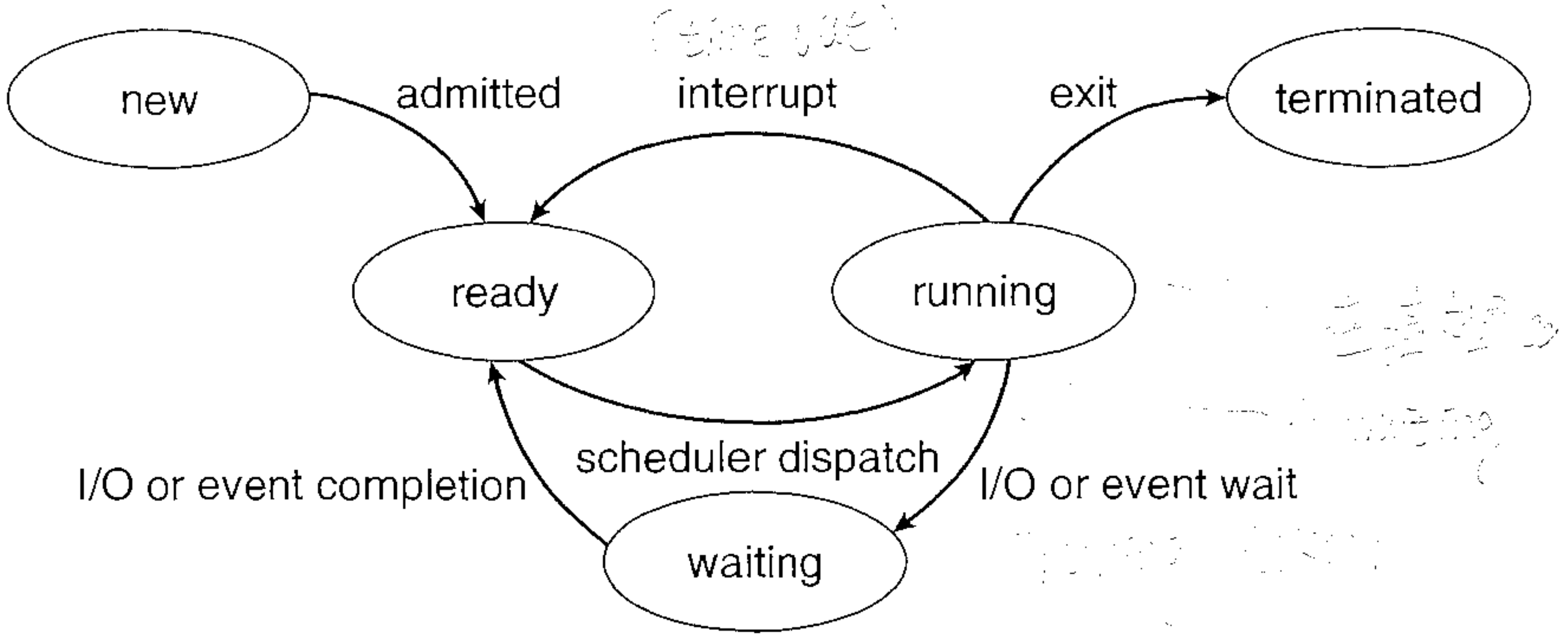


Figure 3.2 Diagram of process state.

Process State

When a process is created and terminated, from beginning to end during its life cycle a process can remain in many states.

- **Start or New:** First state is Start or new state. A process is said to be in new state when it is started or created.
- **Ready:**It is the state next to new state.The process is said to be in ready state when it is waiting to be assigned to a processor. Ready processes are waiting to have the processor allocated to them by the operating system so that they can run. A process may come into ready state in three situation:-

❑ first is just after the Start state,

❑ second situation may be while running and getting interrupted by the scheduler to assign CPU to some other process in this situation it comes to ready state so that CPU can execute the new process.

Process State

- ❑ Third situation from wait state when I/O is completed or wait for an even is completed.
- **Running:** Once the process has been assigned to a processor by the OS scheduler, the process state is set to running and the processor executes its instructions then process is said to be in running state.
- **Waiting:** When a process is in running state then process moves into the waiting state if it needs to wait for a resource, such as waiting for user input, or waiting for a file to become available.
- **Terminated or Exit:** Once the process finishes its execution, or it is terminated by the operating system, it is moved to the terminated state where it waits to be removed from main memory.

Process Concept – PCB/TCB

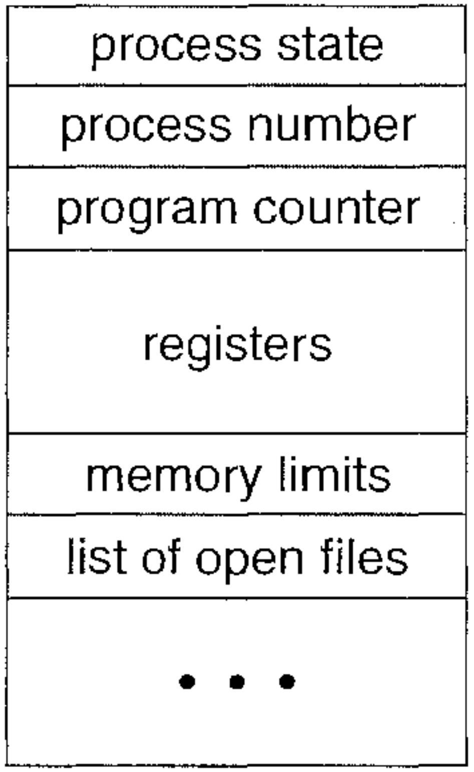


Figure 3.3 Process control block (PCB).

Process Concept – Process Control Block

- As the operating system supports multi-programming, it needs to keep track of all the processes. For this task, the process control block (PCB) is used to track the process’s execution status.
- Each block of memory contains information about the process state, program counter, stack pointer, status of opened files, scheduling algorithms, etc.
- When the process made transitions from one state to another, the operating system must update information in the process’s PCB.
- **Pointer –** It is a stack pointer which is required to be saved when the process is switched from one state to another to retain the current position of the process.
- **Process state –** It stores the respective state of the process.

Process Concept – Process Control Block

- **Process number –** Every process is assigned with a unique id known as process ID or PID which stores the process identifier.
- **Program counter –** It stores the counter which contains the address of the next instruction that is to be executed for the process.
- **Register –** These are the CPU registers which includes: accumulator, base, registers and general purpose registers.
- **Memory limits –** This field contains the information about memory management system used by operating system. This may include the page tables, segment tables etc.
- **Open files list –** This information includes the list of files opened for a process.
- **Miscellaneous accounting and status data –** This field includes information about the amount of CPU used, time constraints, jobs or process number, etc.

Process Scheduling

- The act of determining which process is in the **ready** state, and should be moved to the **running** state is known as **Process Scheduling**.
- Job of the scheduler is to keep the CPU occupied to deliver minimum response time for all programs.

Process Scheduling – Queuing Diagram

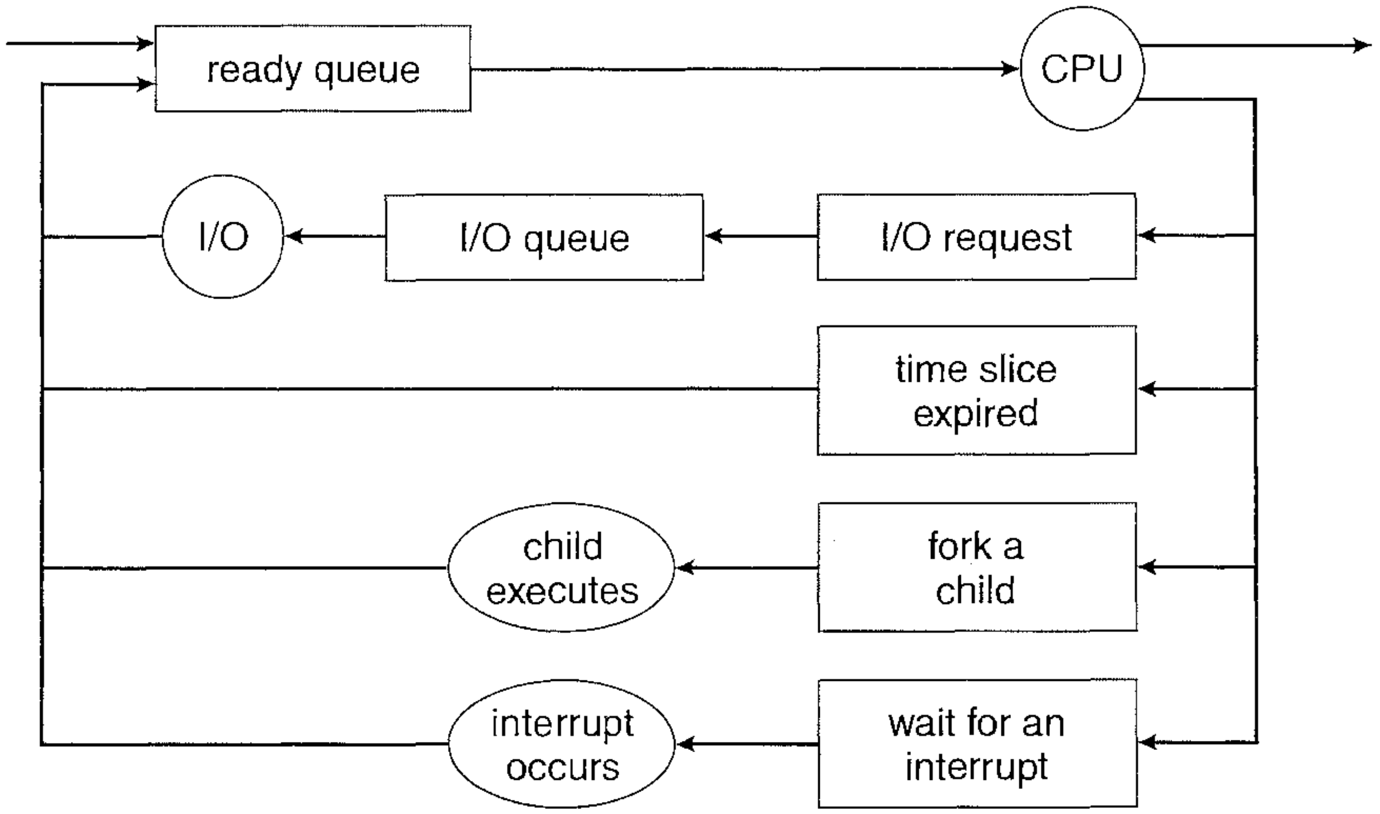


Figure 3.7 Queueing-diagram representation of process scheduling.

What are Scheduling Queues?

- All processes, upon entering into the system, are stored in the **Job Queue**.
- Processes in the Ready state are placed in the **Ready Queue**.
- Processes waiting for a device to become available are placed in **Device Queues**. There are unique device queues available for each I/O device.
- A new process is initially put in the **Ready queue**. It waits in the ready queue until it is selected for execution(or dispatched). Once the process is assigned to the CPU and is executing, one of the following several events can occur:
- The process could issue an I/O request, and then be placed in the **I/O queue**.
- The process could create a new subprocess and wait for its termination.
- The process could be removed forcibly from the CPU, as a result of an interrupt, and be put back in the ready queue.
- In the first two cases, the process eventually switches from the waiting state to the ready state, and is then put back in the ready queue. A process continues this cycle until it terminates

Cooperating 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
- Advantages of process cooperation
  - Information sharing
  - Computation speed-up
  - Modularity
  - Convenience

Schedulers

- **Short-term scheduler** (or **CPU scheduler**) – selects which process should be executed next and allocates CPU
  - Sometimes the only scheduler in a system
  - Short-term scheduler is invoked frequently (milliseconds) $\Rightarrow$ (must be fast)
- **Long-term scheduler** (or **job scheduler**) – selects which processes should be brought into the ready queue
  - Long-term scheduler is invoked infrequently (seconds, minutes) $\Rightarrow$ (may be slow)
  - The long-term scheduler controls the **degree of multiprogramming**

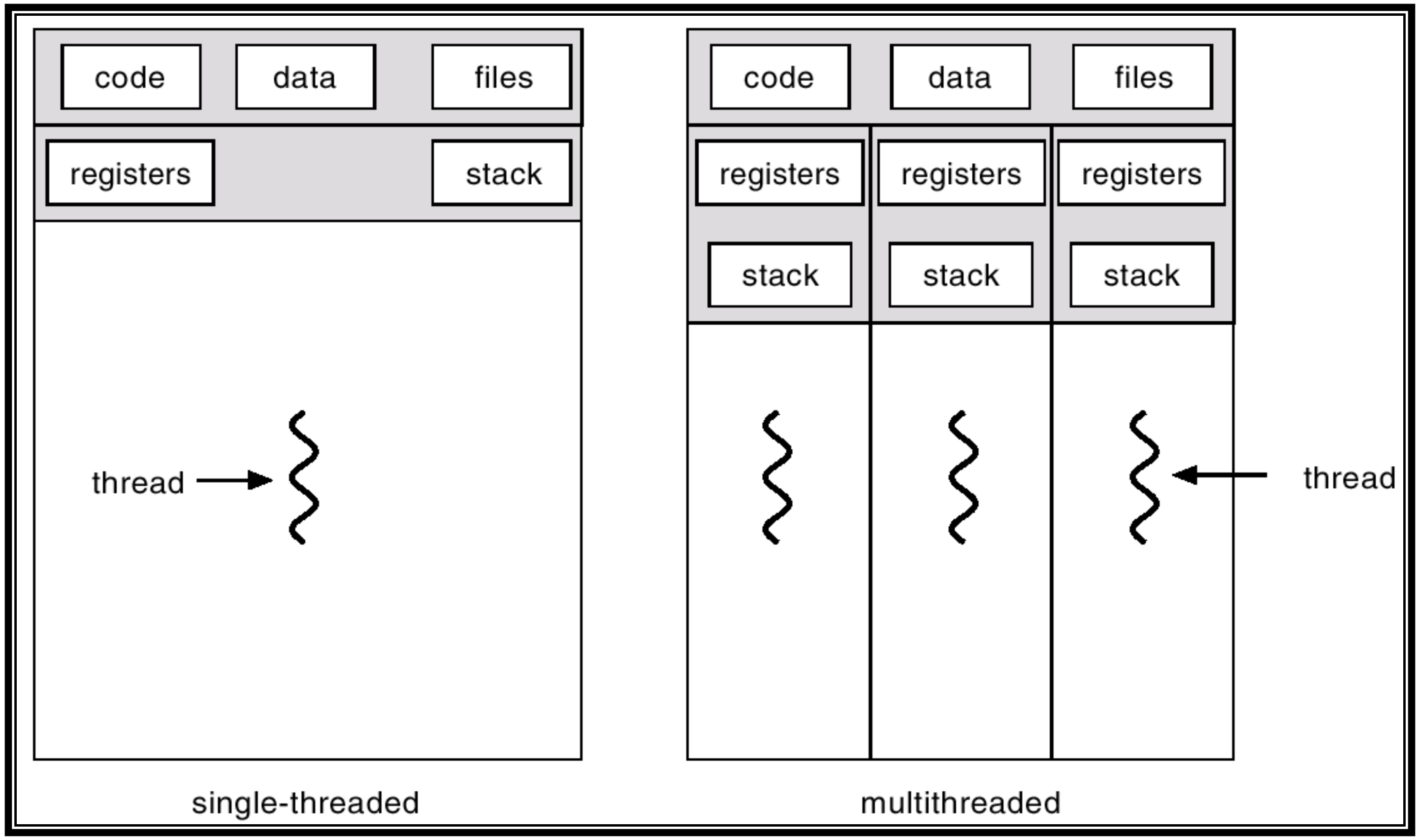
- Processes can be described as either:
  - **I/O-bound process** – spends more time doing I/O than computations, many short CPU bursts
  - **CPU-bound process** – spends more time doing computations; few very long CPU bursts
- Long-term scheduler strives for good ***process mix***

Threads

Overview

- A thread is a basic unit of CPU utilization
- It comprises of a thread ID, a program counter, a register set and a stack
- It shares with other threads belonging to the same process its code section, data section and other OS resources, such as open files and signals
- A traditional(heavyweight) process has a single thread of control
- If a process has multiple threads of control,it can perform more than one task at a time

Single and Multithreaded Processes



Types of threads

There are two types of threads:

- User Threads
- Kernel Threads

User Threads

- User threads are above the kernel and without kernel support
- These are the threads that application programmers use in their programs
- Thread management done by user-level threads library
- Examples

  - POSIX *Pthreads*

  - Mach *C-threads*

  - Solaris *threads*

Kernel Threads

- **Kernel threads** are supported within the kernel of the OS itself
- All modern OSs support kernel level threads, allowing the kernel to perform multiple simultaneous tasks and/or to service multiple kernel system calls simultaneously.
- Examples

  - Windows 95/98/NT/2000

  - Solaris

  - Tru64 UNIX

  - BeOS

  - Linux

Multithreading Models

- How do user and kernel threads map into each other?
- Many-to-One

- One-to-One

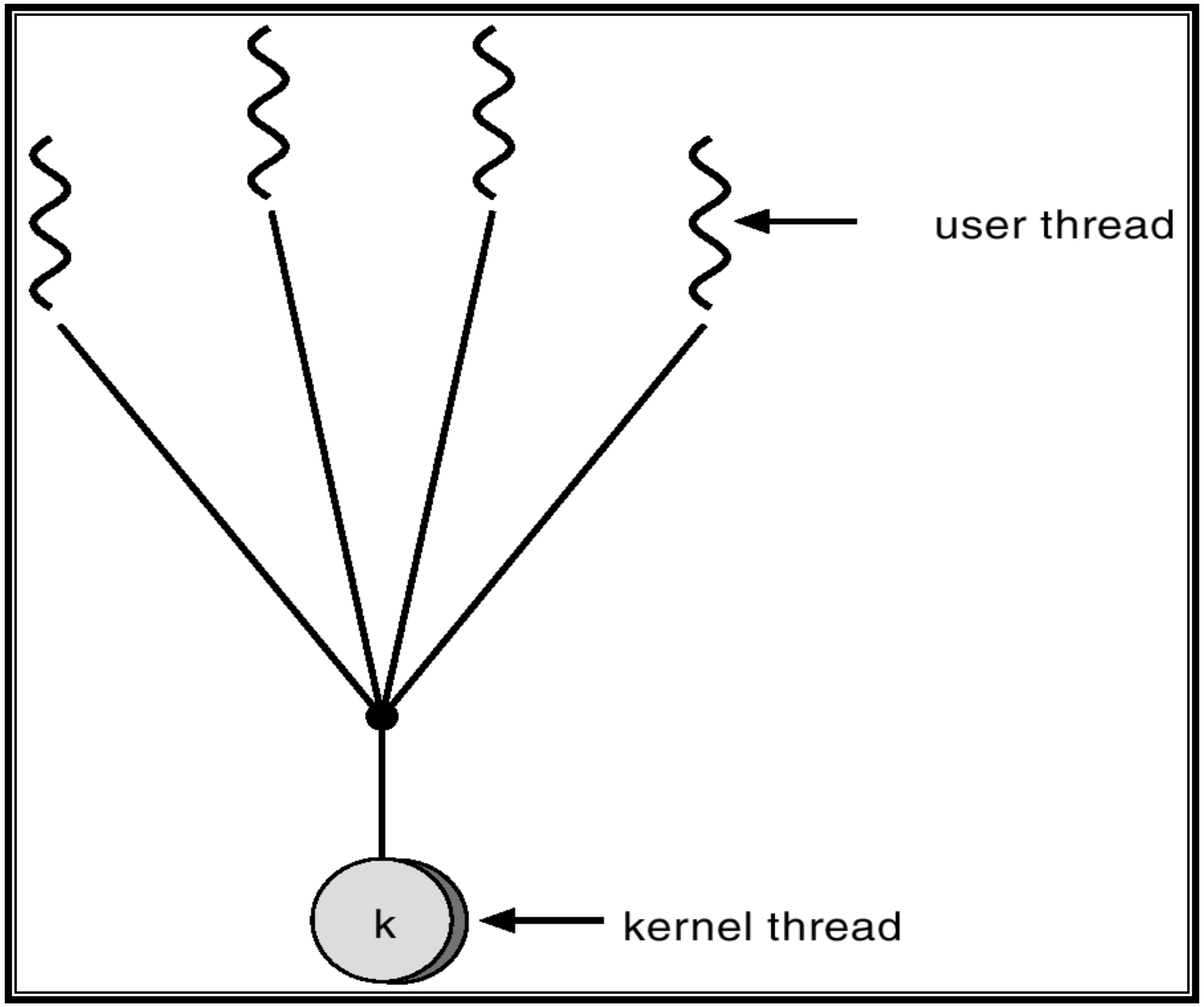
- Many-to-Many

Many-to-One Model

- Many user-level threads mapped to single kernel thread
- Used on systems that do not support kernel threads
- Examples:

  Solaris Green Threads

  GNU Portable Threads


user thread
k
kernel thread

One-to-One Model

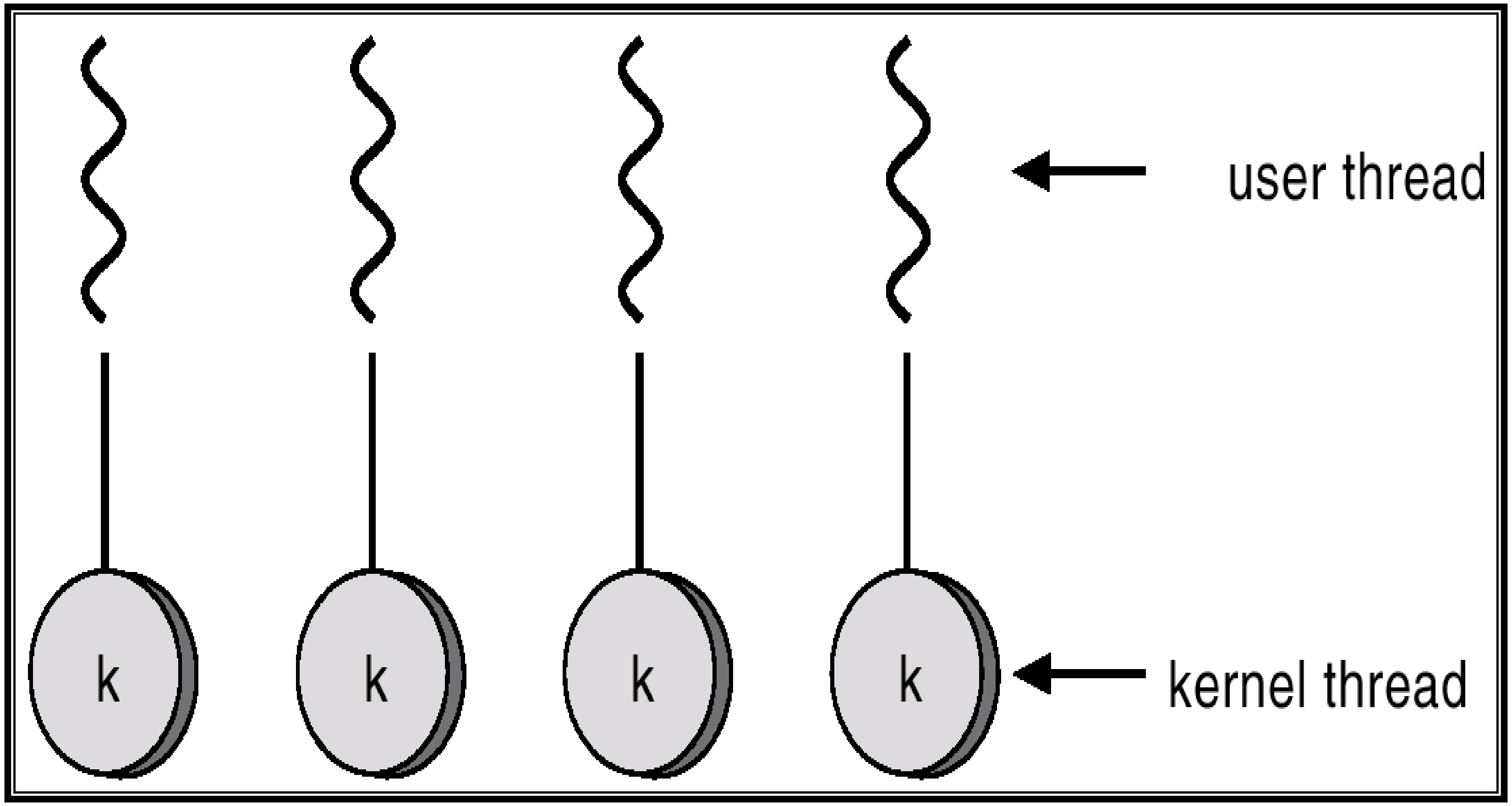
- Each user-level thread maps to kernel thread.

- Examples

  - Windows 95/98/NT/2000

  - Linux

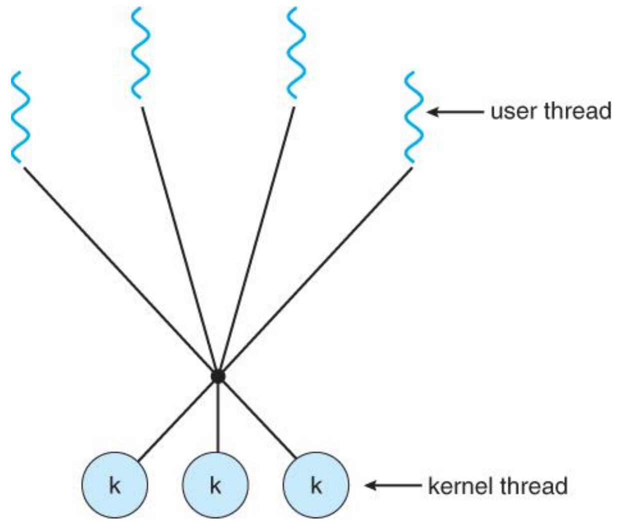
One-to-One Model



Many-to-Many Model

- The **many to many model** multiplexes any number of user **threads** onto an equal or smaller number of kernel **threads**
- It combines the best features of the **one**-to-**one** and **many**-to-**one models**
- and **many**-to-**one models**.
- Users can create any number of the **threads**
- Blocking the kernel system calls does not block the entire process.
- Processes can be split across multiple processors.

Many-to-Many model

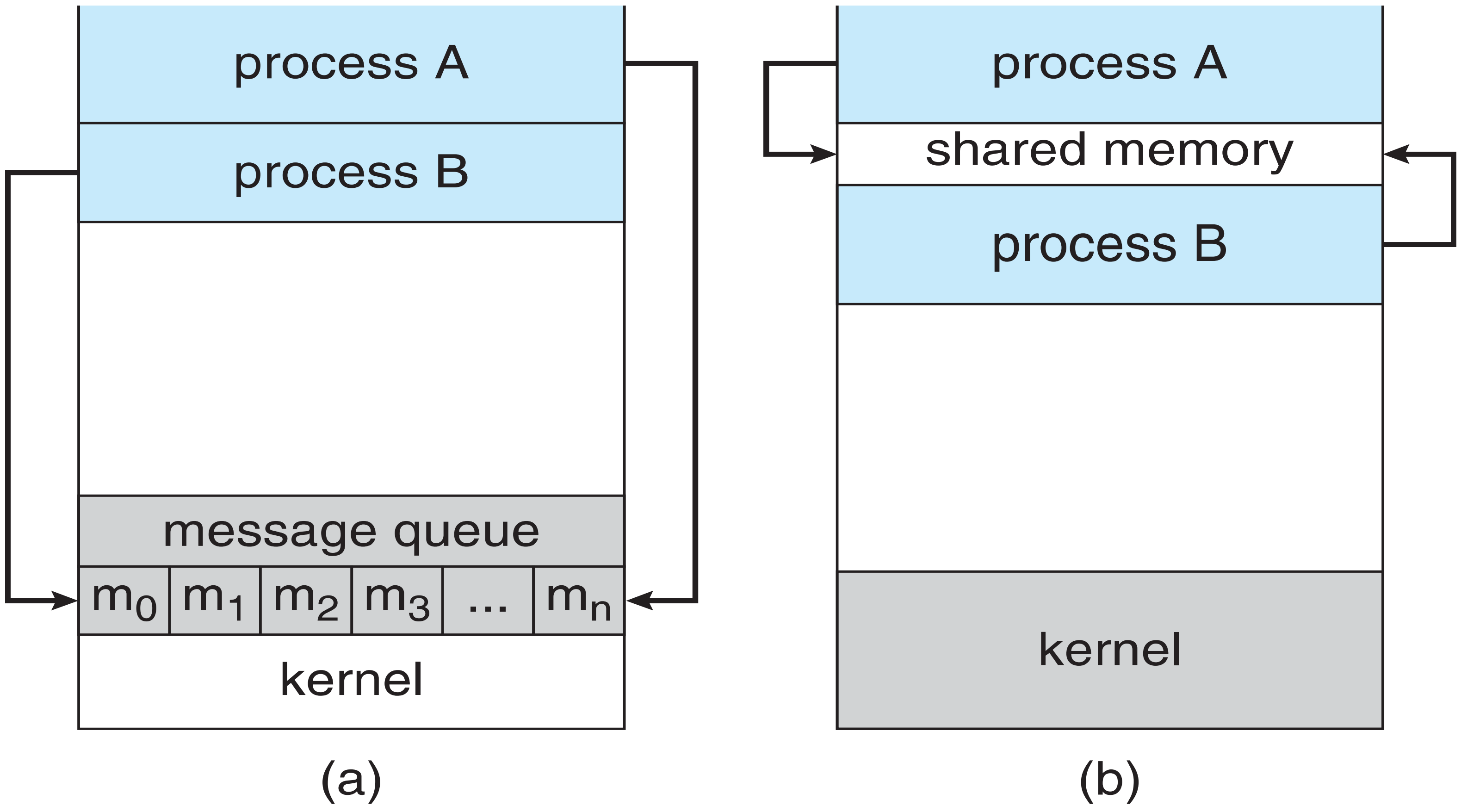


Inter Process Communication

- Processes within a system may be ***independent*** or ***cooperating***
- Cooperating process can affect or be affected by other processes, including sharing data
- Reasons for cooperating processes:
  - Information sharing
  - Computation speedup
  - Modularity
  - Convenience
- Cooperating processes need **interprocess communication** (**IPC**)
- Two models of IPC
  - **Shared memory**
  - **Message passing**

Communications Models

(a) Message passing. (b) shared memory.



Inter process Communication – Shared Memory

- An area of memory shared among the processes that wish to communicate
- The communication is under the control of the users processes not the operating system.
- Major issues is to provide mechanism that will allow the user processes to synchronize their actions when they access shared memory.

Inter process Communication – Message Passing

- Mechanism for processes to communicate and to synchronize their actions

- Message system – processes communicate with each other without resorting to shared variables

- IPC facility provides two operations:
  - **send**(*message*)
  - **receive**(*message*)

- The *message* size is either fixed or variable

CPU Scheduling criteria

- Many criteria have been suggested for comparing CPU scheduling algorithm
- The criteria include the following

1. **CPU utilization-** keep the CPU as busy as possible
2. **Throughput-** # of processes that complete their execution per time unit
3. **Turnaround time-** amount of time to execute a particular process (finishing time – arrival time)
4. **Waiting time-** amount of time a process has been waiting in the ready queue
5. **Response time-**amount of time it takes from when a request was submitted until the first response is produced, **not** output (for time-sharing environment)

Scheduling algorithms

- CPU scheduling deals with the problem of deciding which of the processes in the ready queue is to be allocated the CPU
- Following are various CPU scheduling algorithms

1. First-Come, First-Served (FCFS) Scheduling
2. Shortest-Job-First (SJR) Scheduling
3. Priority Scheduling
4. Round Robin (RR)

FCFS Scheduling (First Come First Serve)

- FCFS Scheduling (First Come First Serve)
- First job that requests the CPU gets the CPU
- Non preemptive – Process continues till the burst cycle
- **Advantages** – Simple – Fair (as long as no process hogs the CPU, every process will eventually run)
- **Disadvantages** – Waiting time depends on arrival order – short processes stuck waiting for long process to complete ends

Shortest Job First (SJF)

- Shortest Job First (SJF) no preemption
- Associate with each process the length of its next CPU burst
  - Use these lengths to schedule the process with the shortest time
- Schedule process with the shortest burst time – FCFS if same
- Advantages – Minimizes average wait time and average response time
- Disadvantages – Not practical : difficult to predict burst time
- Learning to predict future – May starve long jobs

Shortest Remaining Time First -- SRTF (SJF with preemption)

- If a new process arrives with a shorter burst time than remaining of current process then schedule new process
- Further reduces average waiting time and average response time
- Not practical

Round Robin Scheduling

- Each process gets a small unit of CPU time (**time quantum** q), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.
- If there are n processes in the ready queue and the time quantum is q, then each process gets $1/n$ of the CPU time in chunks of at most q time units at once. No process waits more than $(n-1)q$ time units.
- Timer interrupts every quantum to schedule next process
- Performance
  - q large $\Rightarrow$ FIFO
  - q small $\Rightarrow$ q must be large with respect to context switch, otherwise overhead is too high

Advantages

- Fair (Each process gets a fair chance to run on the CPU) – Low average wait time, when burst times vary – Faster response time

Disadvantages –

- Increased context switching
- Context switches are overheads!!! – High average wait time, when burst times have equal lengths

Priority based Scheduling

- A priority number (integer) is associated with each process

- The CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority)
  - Preemptive
  - Nonpreemptive
- SJF is priority scheduling where priority is the inverse of predicted next CPU burst time
- Problem ≡ **Starvation** – low priority processes may never execute
- Solution ≡ **Aging** – as time progresses increase the priority of the process