USN | | | | | | | | | | |

Internal Assessment Test 1 – JAN 2019

| Sub: | OPERATING SYSTEMS | | | | | Sub Code: | 15CS64 | Branch: | CSE | |
|---|---|---|---|---|---|---|---|---|---|---|
| Date: | 06/03/2019 | Duration: | 90 min's | Max Marks: | 50 | Sem / Sec: | VI(A,B,C) | | OBE | |

Answer any FIVE FULL Questions

| | | MARKS | CO | RBT |
|---|---|---|---|---|
| 1 (a) | Define Operating System. With neat daigram explain Dual Mode operation of OS. | [06] | CO1 | L1 |
| (b) | Differentiate Multiprogramming, Multitasking and Multiprocessing. | [04] | CO1 | L2 |
| 2 (a) | Discuss the Services provided by Operating System that are useful to user and system. | [06] | CO1 | L2 |
| (b) | Write a short note on Scheduler and its types. | [04] | CO2 | L1 |
| 3 (a) | Summarize the handling of a user application invoking System call with neat diagram. | [06] | CO1 | L2 |
| (b) | Compare Layered Approach of an Operating System with Modules approach. | [04] | CO1 | L2 |
| 4 (a) | Define Process. Explain the different States of Process with state diagram. Also explain the PCB. | [10] | CO2 | L1 |
| 5 (a) | Describe Inter Process Communication and its models. | [10] | CO2 | L2 |
| 6 (a) | Define Multi-threading. Explain the Multithreading models. | [04] | CO2 | L1 |
| (b) | Explain the Queueing diagram representation of Process Scheduling. | [06] | CO2 | L2 |

USN | | | | | | | | | | |

Internal Assessment Test 1 – JAN 2019

| Sub: | OPERATING SYSTEMS | | | | | Sub Code: | 15CS64 | Branch: | CSE | |
|---|---|---|---|---|---|---|---|---|---|---|
| Date: | 06/03/2019 | Duration: | 90 min's | Max Marks: | 50 | Sem / Sec: | VI(A,B,C) | | OBE | |

Answer any FIVE FULL Questions

| | | MARKS | CO | RBT |
|---|---|---|---|---|
| 1 (a) | Define Operating System. With neat daigram explain Dual Mode operation of OS. | [06] | CO1 | L1 |
| (b) | Differentiate Multiprogramming, Multitasking and Multiprocessing. | [04] | CO1 | L2 |
| 2 (a) | Discuss the Services provided by Operating System that are useful to user and system. | [06] | CO1 | L2 |
| (b) | Write a short note on Scheduler and its types. | [04] | CO2 | L1 |
| 3 (a) | Summarize the handling of a user application invoking System call with neat diagram. | [06] | CO1 | L2 |
| (b) | Compare Layered Approach of an Operating System with Modules approach. | [04] | CO1 | L2 |
| 4 (a) | Define Process. Explain the different States of Process with state diagram. Also explain the PCB. | [10] | CO2 | L1 |
| 5 (a) | Describe Inter Process Communication and its models. | [10] | CO2 | L2 |
| 6 (a) | Define Multi-threading. Explain the Multithreading models. | [04] | CO2 | L1 |
| (b) | Explain the Queueing diagram representation of Process Scheduling. | [06] | CO2 | L2 |

1 (a) Define Operating System. With neat daigram explain Dual Mode operation of OS.

Operating System

- An OS is a program that acts as an intermediary between
  → computer-user and
  → computer-hardware.
- It also provides a basis for application-programs
- Goals of OS:
  - To execute programs.
  - To make solving user-problems easier.
  - To make the computer convenient to use.
- The OS (also called kernel) is the one program running at all times on the computer.

**Dual Mode Operation**

- Problem: We must be able to differentiate between the execution of
  → OS code and
  → user-defined code.

  Solution: Most computers provide hardware-support.
- We have two modes of operation (Figure 1.9):
  1. User mode and
  2. Kernel mode
- A *mode bit* is a bit added to the hardware of the computer to indicate the current mode:
  i.e. kernel (0) or user (1)

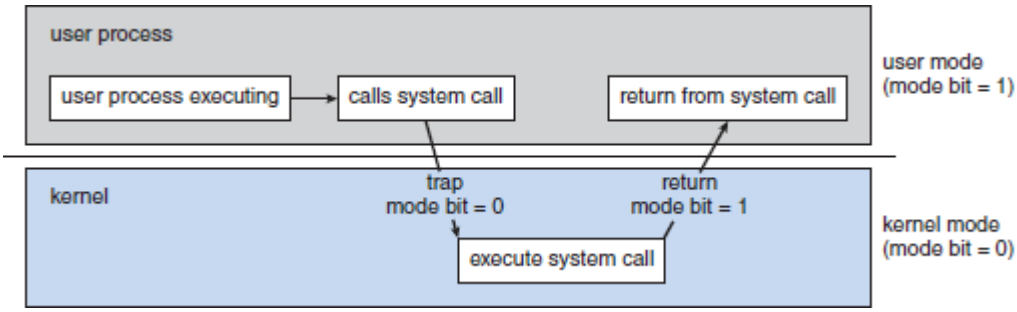


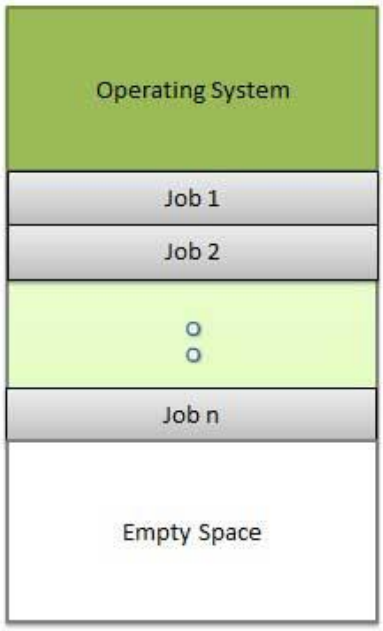

Figure 1.9 Transition from user to kernel mode

- Working principle:
  1. At system boot time, the hardware starts in kernel-mode.
  2. The OS is then loaded and starts user applications in user-mode.
  3. Whenever a trap or interrupt occurs, the hardware switches from user-mode to kernel-mode (that is, changes the state of the mode bit to 0).
  4. The system always switches to user-mode (by setting the mode bit to 1) before passing control to a user-program.
- Dual mode protects
  → OS from errant users and
  → errant users from one another.
- *Privileged instruction* is executed only in kernel-mode.
- If an attempt is made to execute a privileged instruction in user-mode, the hardware treats it as illegal and traps it to the OS.
- A *system calls* are called by user-program to ask the OS to perform the tasks on behalf of the user program.

(b) Differentiate Multiprogramming, Multitasking and Multiprocessing.

**Multiprogramming**

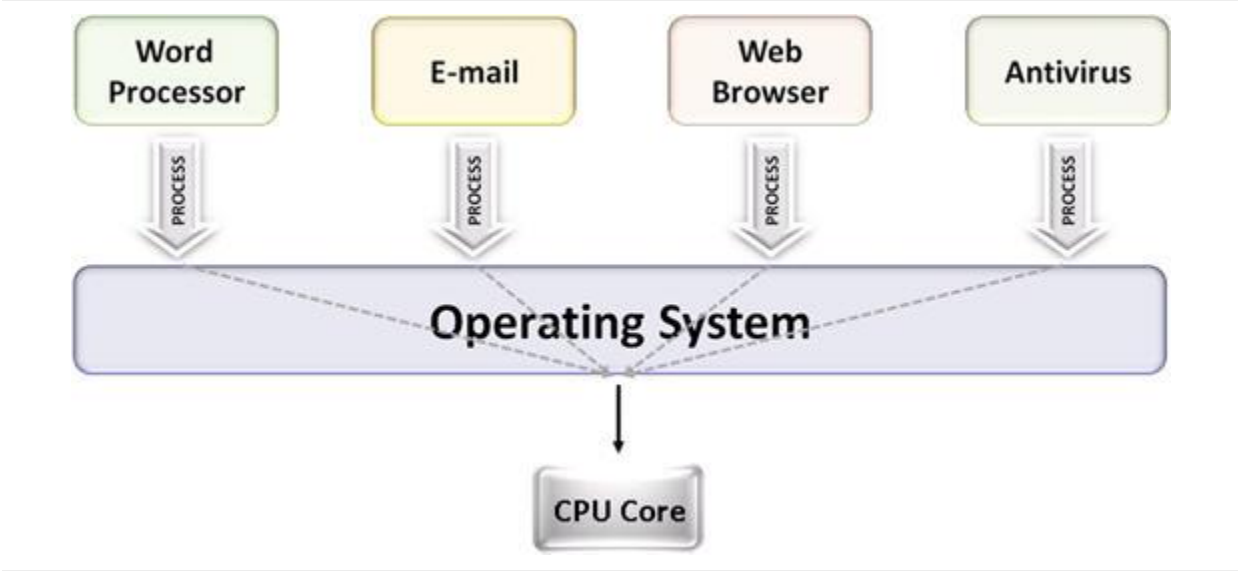
Multiprogramming is also the ability of an operating system to execute more than one program on a single processor machine. More than one task/program/job/process can reside into the main memory at one point of time. A computer running excel and firefox browser simultaneously is an example of multiprogramming.




Memory layout for Multiprogramming System

**Multitasking**

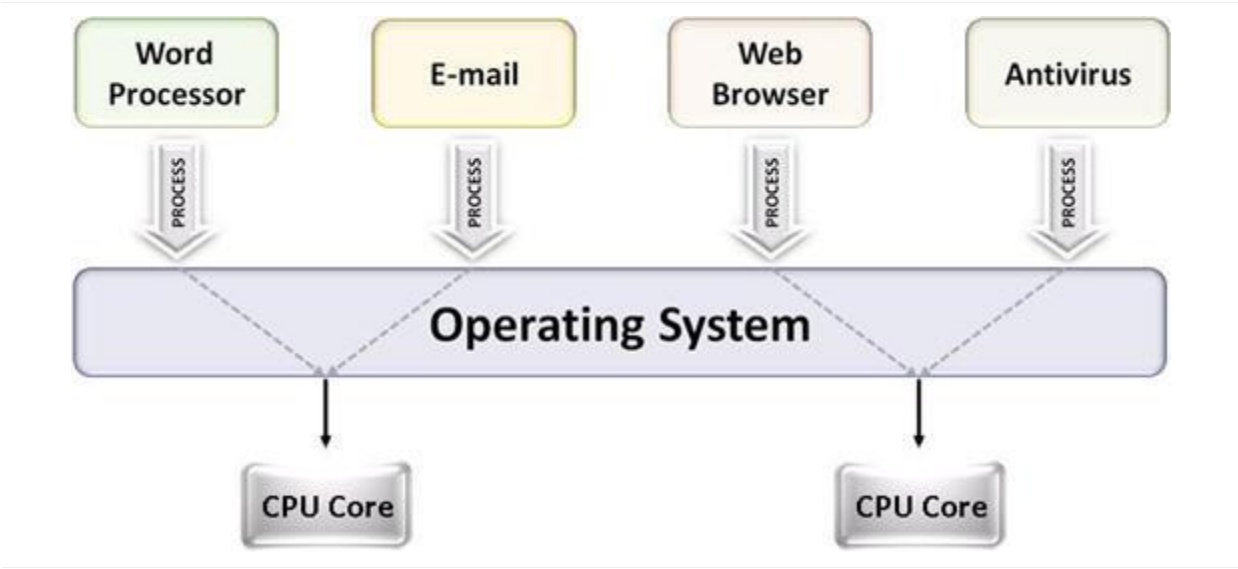
Multitasking is the ability of an operating system to execute more than one task simultaneously on a single processor machine. Though we say so but in reality no two tasks on a single processor machine can be executed at the same time. Actually CPU switches from one task to the next task so quickly that appears as if all the tasks are executing at the same time. More than one task/program/job/process can reside into the same CPU at one point of time.




Multitasking System

**Multiprocessing**

Multiprocessing is the ability of an operating system to execute more than one process simultaneously on a multi processor machine. In this, a computer uses more than one CPU at a time.




Multiprocessing System

| | |
|---|---|
| 2 (a) | Discuss the Services provided by Operating System that are useful to user and system. |
| (b) | Write a short note on Scheduler and its types. |

**Operating System Services**

- An OS provides an environment for the execution of programs.
- It provides services to
  → programs and

→ users.

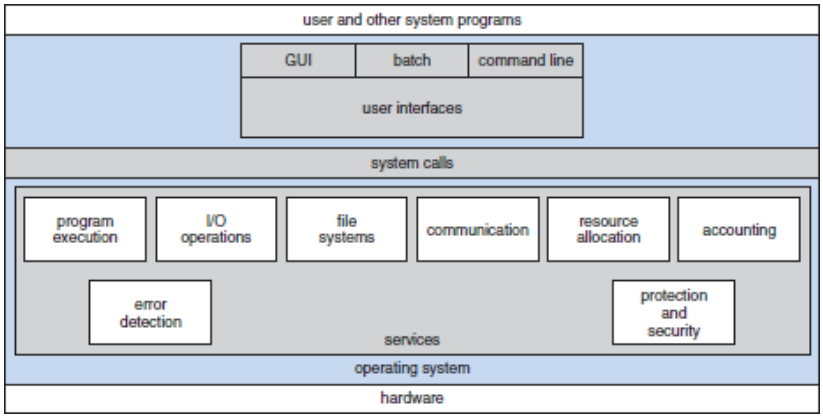



Figure 1.12 A view of OS services

- Common functions helpful to the user are (Figure 1.12):

***1. User Interface***

➢ Almost all OS have a user-interface (UI).

➢ Different interfaces are:

**i) CLI (Command line Interface)**

➢ This uses

→ text commands and

→ method for entering the text commands.

**ii) Batch interface**

➢ Commands & directives to control those commands are entered into files, and those files are executed.

**iii) GUI (Graphical User Interface)**

➢ The interface is a window-system with a pointing-device to

→ direct I/0

→ choose from menus and

→ make selections.

***2. Program Execution***

➢ The system must be able to

→ load a program into memory and

→ run the program.

➢ The program must be able to end its execution, either normally or abnormally.

***3. I/O Operations***

➢ The OS must provide a means to do I/O operations because users cannot control I/O devices directly.

➢ For specific devices, special functions may be desired (ex: to blank CRT screen).

***4. File-system Manipulation***

➢ Programs need to

→ read & write files (or directories)

→ create & delete files

→ search for a given file and

→ allow or deny access to files.

***5. Communications***

➢ In some situations, one process needs to communicate with another process.

➢ Communications may be implemented via

1. Shared memory or
2. Message passing

➢ In *message passing*, packets of information are moved between processes by OS.

***6. Error Detection***

➢ Errors may occur in

→ CPU & memory-hardware (ex: power failure)

→ I/O devices (ex: lack of paper in the printer) and

→ user program (ex: arithmetic overflow)

➢ For each type of error, OS should take appropriate action to ensure correct & consistent computing.

- Common functions for efficient operation of the system are:

***1. Resource Allocation***

➢ When multiple users are logged on the system at the same time, resources must be allocated to each of them.

➢ The OS manages different types of resources.

➢ Some resources (say CPU cycles) may have special allocation code.

Other resources (say I/O devices) may have general request & release code.

***2. Accounting***

➢ We want to keep track of

→ which users use how many resources and

→ which kinds of resources.

➢ This record keeping may be used for

→ accounting (so that users can be billed) or

→ gathering usage-statistics.

***3. Protection***

➢ When several separate processes execute concurrently, it should not be possible for one process to interfere with the others or with the OS itself.

➢ Protection involves ensuring that all access to resources is controlled.

➢ Security starts with each user having authenticated to the system by means of a password.

**Schedulers**

- Three types of schedulers:
  1. Long-term scheduler
  2. Short-term scheduler and
  3. Medium-term schedulers

| ***Long-term Scheduler*** | ***Short-term Scheduler*** |
|---|---|
| Also called job scheduler. | Also called CPU scheduler. |
| Selects which processes should be brought into the ready-queue. | Selects which process should be executed next and allocates CPU. |
| Need to be invoked only when a process leaves the system and therefore executes much less frequently. | Need to be invoked to select a new process for the CPU and therefore executes much more frequently. |
| May be slow ‘,’ minutes may separate the creation of one new process and the next. | Must be fast ‘,’ a process may execute for only a few milliseconds. |
| Controls the degree of multiprogramming. | |

- Processes can be described as either:

***1. I/O-bound Process***

➢ Spends more time doing I/O operation than doing computations.

➢ Many short CPU bursts.

***2. CPU-bound Process***

➢ Spends more time doing computations than doing I/O operation.

➢ Few very long CPU bursts.

- Why long-term scheduler should select a good process mix of I/O-bound and CPU-bound processes?

Ans: 1) If all processes are I/0 bound, then

i) Ready-queue will almost always be empty, and

ii) Short-term scheduler will have little to do.

1) If all processes are CPU bound, then

i) I/0 waiting queue will almost always be empty (devices will go unused) and

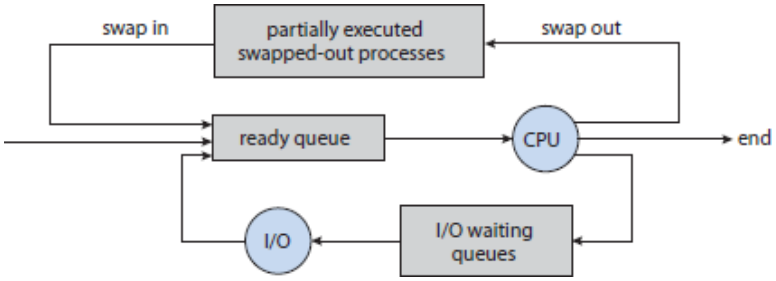
ii) System will be unbalanced.

- Some time-sharing systems have **medium-term scheduler** (Figure 2.6).

➢ The scheduler removes processes from memory and thus reduces the degree of multiprogramming.

➢ Later, the process can be reintroduced into memory, and its execution can be continued where it left off. This scheme is called **swapping**.

➢ The process is swapped out, and is later swapped in, by the scheduler.




| | |
|---|---|
| 3 (a) | Summarize the handling of a user application invoking System call with neat diagram. |
| (b) | Compare Layered Approach of an Operating System with Modules approach. |

**Layered Approach**

- The OS is divided into a number of layers.
- Each layer is built on the top of another layer.
- The bottom layer is the hardware.
  The highest is the user interface (Figure 1.19).
- A layer is an implementation of an abstract-object.
  i.e. The object is made up of
  → data and
  → operations that can manipulate the data.
- The layer consists of a set of routines that can be invoked by higher-layers.
- Higher-layer
  → does not need to know how lower-layer operations are implemented
  → needs to know only what lower-layer operations do.
- Advantage:
  1. Simplicity of construction and debugging.
- Disadvantages:
  1. Less efficient than other types.
  2. Appropriately defining the various layers.('.' a layer can use only lower-layers, careful planning is necessary).

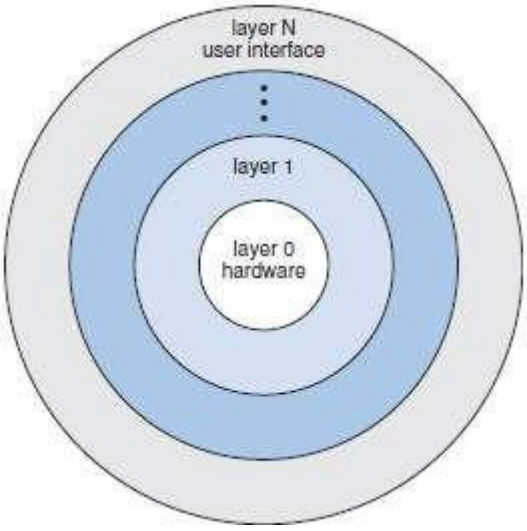



Figure 1.19 A layered OS

**Modules**

- The kernel has
  → set of core components and
  → dynamic links in additional services during boot time( or run time).
- Seven types of modules in the kernel (Figure 1.21):
  1. Scheduling classes
  2. File systems
  3. Loadable system calls
  4. Executable formats
  5. STREAMS modules
  6. Miscellaneous
  7. Device and bus drivers
- The top layers include
  → application environments and
  → set of services providing a graphical interface to applications.
- Kernel environment consists primarily of
  → Mach microkernel and
  → BSD kernel.
- Mach provides
  → memory management;
  → support for RPCs & IPC and
  → thread scheduling.
- BSD component provides
  → BSD command line interface
  → support for networking and file systems and
  → implementation of POSIX APIs
- The kernel environment provides an I/O kit for development of
  → device drivers and
  → dynamic loadable modules (which Mac OS X refers to as *kernel extensions*).

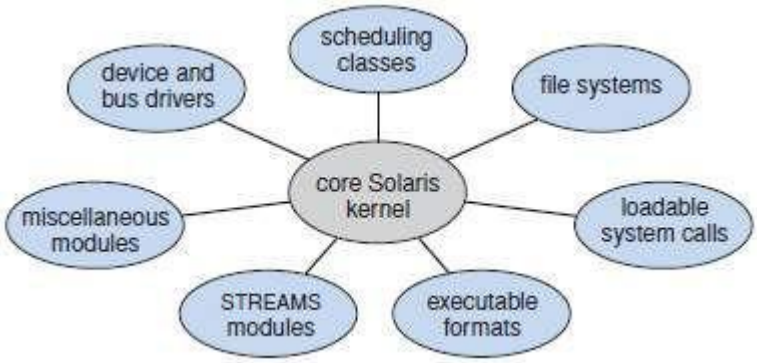



Figure 1.21 Solaris loadable modules

6.a. Define Multi-threading. Explain the Multithreading models.

**Multithreading Models**

- Many-to-One
- One-to-One
- Many-to-Many

**Many-to-One**

Many user-level threads mapped to single kernel thread
->Examples:
->Solaris Green Threads
->GNU Portable Threads

**One-to-One**

1. Each user-level thread maps to kernel thread
2. Examples
- Windows NT/XP/2000
- Linux
- Solaris 9 and later

**Many-to-Many Model**

1. Allows many user level threads to be mapped to many kernel threads.
2. Allows the operating system to create a sufficient number of kernel threads.
3. Solaris prior to version 9.
4. Windows NT/2000 with the *ThreadFiber* package

6.b. Explain the Queueing diagram representation of Process Scheduling.

**Scheduling Queues**

- Three types of scheduling-queues:
  - ***1. Job Queue***
    - This consists of all processes in the system.
    - As processes enter the system, they are put into a job-queue.
  - ***2. Ready Queue***
    - This consists of the processes that are
      → residing in main-memory and
      → ready & waiting to execute (Figure 2.4).
    - This queue is generally stored as a **linked list**.
    - A ready-queue header contains pointers to the first and final PCBs in the list.
    - Each PCB has a pointer to the next PCB in the ready-queue.
  - ***3. Device Queue***
    - This consists of the processes that are waiting for an I/O device.
    - Each device has its own device-queue.
- When the process is executing, one of following events could occur (Figure 2.5):
  1. The process could issue an I/0 request and then be placed in an I/0 queue.
  2. The process could create a new subprocess and wait for the subprocess's termination.
  3. The process could be interrupted and put back in the ready-queue.

Figure 1 The ready-queue and various I/O device-queues

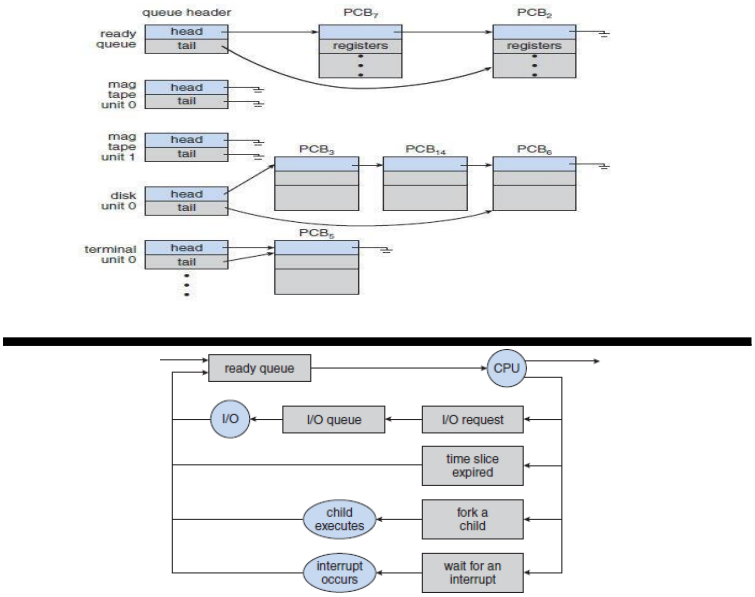



Figure 2 Queueing-diagram representation of process scheduling

.

5 (a) Describe Inter Process Communication and its models.

**Inter Process Communication (IPC)**

- Processes executing concurrently in the OS may be either
  1. Independent processes or 2. Co-operating processes.
  1. A process is ***independent*** if
     i) The process cannot affect or be affected by the other processes.
     ii) The process does not share data with other processes.
  2. A process is ***co-operating*** if

     i) The process can affect or be affected by the other processes.
     ii) The process shares data with other processes.
- Advantages of process co-operation:
  ***1. Information Sharing***
  ➢ Since many users may be interested in same piece of information (ex: shared file).
  ***2. Computation Speedup***
  ➢ We must break the task into subtasks.
  ➢ Each subtask should be executed in parallel with the other subtasks.
  ➢ The speed can be improved only if computer has multiple processing elements such as
  → CPUs or
  → I/O channels.
  ***3. Modularity***
  ➢ Divide the system-functions into separate processes or threads.
  ***4. Convenience***
  ➢ An individual user may work on many tasks at the same time.
  ➢ For ex, a user may be editing, printing, and compiling in parallel.
- Two basic models of IPC (Figure 2.9):
  1) Shared-memory and
  2) Message passing.

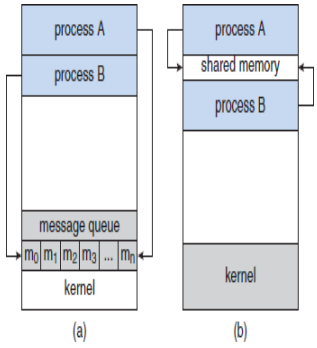


Figure 3 Communications models. (a) Message passing. (b) Shared-memory

**Shared-Memory Systems**

- Communicating-processes must establish a region of shared-memory.
- A shared-memory resides in address-space of the process creating the shared-memory.
  Other processes must attach their address-space to the shared-memory.
- The processes can then exchange information by reading and writing data in the shared-memory.
- The processes are also responsible for ensuring that they are not writing to the same location simultaneously.
- For ex, **producer-consumer problem**:
  Producer-process produces information that is consumed by a consumer-process
- Two types of buffers can be used:
  1. ***Unbounded-buffer*** places no practical limit on the size of the buffer.
  2. ***Bounded-buffer*** assumes that there is a fixed buffer-size.
- Advantages:
  1. Allows maximum speed and convenience of communication.
  2. Faster.

**Message-Passing Systems**

- These allow processes to communicate and to synchronize their actions without sharing the same address-space.
- For example, a chat program used on the WWW.
- Messages can be of either
  1) Fixed size or 2) Variable size.
  1. If ***fixed-sized messages*** are used, the system-level implementation is simple.
     - However, the programming task becomes more difficult.
  2. If ***variable-sized messages*** are used, the system-level implementation is complex.
     - However, the programming task becomes simpler.
- A communication-link must exist between processes to communicate
- Three methods for implementing a link:
  1. Direct or indirect communication.
  2. Symmetric or asymmetric communication.
  3. Automatic or explicit buffering.
- Two operations:
  1. send(P,message): Send a message to process P.
  2. receive(Q,message): Receive a message from process Q.
- Advantages:
  1. Useful for exchanging smaller amounts of data (‘.’ No conflicts need be avoided).
  2. Easier to implement.
  3. Useful in a distributed environment.

**Naming**

- Processes that want to communicate must have a way to refer to each other. They can use either direct or indirect communication.

| ***Direct Communication*** | ***Indirect Communication*** |
|---|---|
| Each process must explicitly name the recipient/sender. | Messages are sent to/received from mailboxes (or ports). |
| <u>Properties of a communication link:</u><br>➢ A link is established automatically between every pair of processes that want to communicate. The processes need to know only each other’s identity to communicate.<br>➢ A link is associated with exactly two processes. | <u>Properties of a communication link:</u><br>➢ A link is established between a pair of processes only if both members have a shared mailbox.<br>➢ A link may be associated with more than two processes.<br>➢ A number of different links may exist |

| | |
|---|---|
| ➢ Exactly one link exists between each pair of processes. | between each pair of communicating processes. |
| Symmetric addressing:<br>➢ Both sender and receiver processes must name the other to communicate. | Mailbox owned by a process:<br>➢ The owner can only receive, and the user can only send.<br>➢ The mailbox disappears when its owner process terminates. |
| Asymmetric addressing:<br>➢ Only the sender names the recipient; the recipient needn't name the sender. | Mailbox owned by the OS:<br>➢ The OS allows a process to:<br>1. Create a new mailbox<br>2. Send & receive messages via it<br>3. Delete a mailbox. |

**Synchronization**

- Message passing may be either blocking or non-blocking (also known as synchronous and asynchronous).

| ***Synchronous Message Passing*** | ***Asynchronous Message Passing*** |
|---|---|
| Blocking send:<br>➢ The sending process is blocked until the message is received by the receiving process or by the mailbox. | Non-blocking send:<br>➢ The sending process sends the message and resumes operation. |
| Blocking receive:<br>➢ The receiver blocks until a message is available. | Non-blocking receive:<br>➢ The receiver retrieves either a valid message or a null. |

**Buffering**

- Messages exchanged by processes reside in a temporary queue.
- Three ways to implement a queue:

***1. Zero Capacity***

➢ The queue-length is zero.
➢ The link can't have any messages waiting in it.
➢ The sender must block until the recipient receives the message.

***2. Bounded Capacity***

➢ The queue-length is finite.
➢ If the queue is not full, the new message is placed in the queue.
➢ The link capacity is finite.
➢ If the link is full, the sender must block until space is available in the queue.

***3. Unbounded Capacity***

➢ The queue-length is potentially infinite.
➢ Any number of messages can wait in the queue.
➢ The sender never blocks.

4 (a) Define Process. Explain the different States of Process with state diagram. Also explain the PCB.

**Process Concept**

- A process is the unit-of-work.
- A system consists of a collection of processes:

1. **OS process** can execute system-code and
2. **User process** can execute user-code.

**The Process**

- A process is a program in execution.
- It also includes (Figure 2.1):
    1. **Program counter** to indicate the current activity.
    2. **Registers content** of the processor.
    3. **Process stack** contains temporary data.
    4. **Data section** contains global variables.
    5. **Heap** is memory that is dynamically allocated during process run time.
- A program by itself is not a process.
    1. A process is an active-entity.
    2. A program is a passive-entity such as an executable-file stored on disk.
- A program becomes a process when an executable-file is loaded into memory.
- If you run many copies of a program, each is a separate process.
    The text-sections are equivalent, but the data-sections vary.

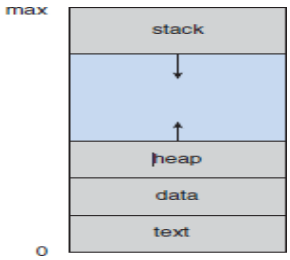



Figure 4 Process in memory

**Process State**

- As a process executes, it changes state.
- Each process may be in one of the following states (Figure 2.2):
    1. **New**: The process is being created.
    2. **Running**: Instructions are being executed.
    3. **Waiting:** The process is waiting for some event to occur (such as I/0 completions).
    4. **Ready**: The process is waiting to be assigned to a processor.
    5. **Terminated**: The process has finished execution.
- Only one process can be *running* on any processor at any instant.

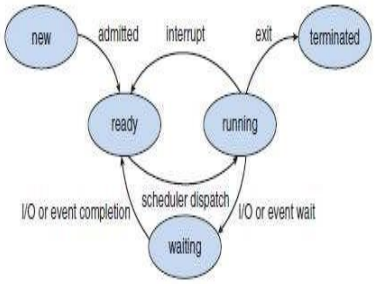
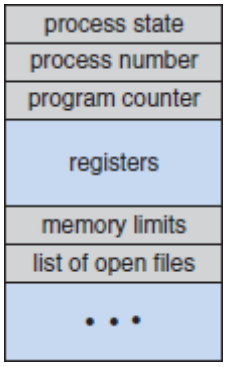



Figure 5 Diagram of process state

**Process Control Block**

- In OS, each process is represented by a PCB (Process Control Block).



- PCB contains following information about the process (Figure 2.3):

***1. Process State***

- The current state of process may be
  - → new
  - → ready
  - → running
  - → waiting or
  - → halted.

***2. Program Counter***

- This indicates the address of the next instruction to be executed for the process.

***3. CPU Registers***

- These include
  - → accumulators (AX)
  - → index registers (SI, DI)
  - → stack pointers (SP) and
  - → general-purpose registers (BX, CX, DX).

***4. CPU Scheduling Information***

- This includes
  - → priority of process
  - → pointers to scheduling-queues and
  - → scheduling-parameters.

***5. Memory Management Information***

- This includes
  - → value of base- & limit-registers and
  - → value of page-tables( or segment-tables).

***6. Accounting Information***

- This includes
  - → amount of CPU time
  - → time-limit and
  - → process-number.

***7. I/O Status Information***

- This includes
  - → list of I/O devices
  - → list of open files.