UNIT I

OPERATING SYSTEM

An operating system is a program that manages the computer hardware. It also provides a basis for application programs and acts as an intermediary between a user of a computer and the computer hardware.

The purpose of an operating system is to provide an environment in which a user can execute programs.

Goals of an Operating System

- The primary goal of an operating system is thus to make the computer system convenient to use.
- The secondary goal is to use the computer hardware in an efficient manner.

BASIC ELEMENTS OF A COMPUTER SYSTEM

An operating system is an important part of almost every computer system. A computer system can be divided roughly into four components.

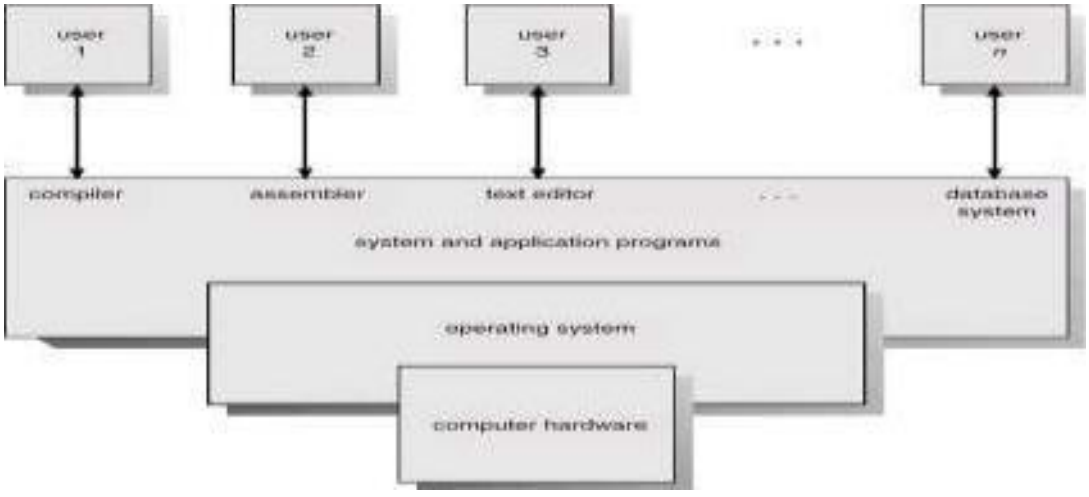
- Hardware
- Operating system
- The application program
- Users

The hardware - the central processing unit (CPU), the memory, and the Input/output (I/O) devices-provides the basic computing resources.

The application programs- such as word processors, spreadsheets, compilers, and web browsers- define the ways in which these resources are used to solve the computing problems of the users.

An operating system is similar to a government. The OS simply provides an environment within which other programs can do useful work.

Abstract view of the components of a computer system.



Operating system can be viewed as a resource allocator. The OS acts as the manager of the resources (such as CPU time, memory space, file storage space, I/O devices) and allocates them to specific programs and users as necessary for tasks.

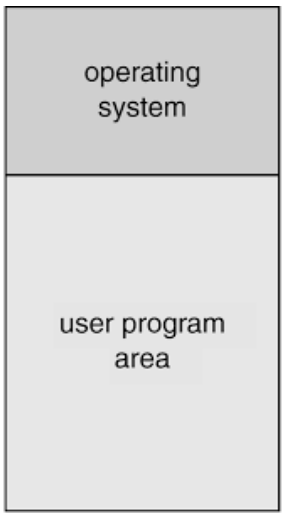
An operating system is a control program. It controls the execution of user programs to prevent errors and improper use of computer.

CLASSIFICATION OF OPERATING SYSTEM

1. Mainframe Systems

Early computers were physically enormous machines run from a console. The common input devices were card readers and tape drives. The common output devices were line printers, tape drives, and card punches. The user did not interact directly with the computer systems. Rather, the user prepared a job - which consisted of the program, the data, and some control information about the nature of the job (control cards)-and submitted it to the computer operator. The job was usually in the form of punch cards.

The operating system in these early computers was fairly simple. Its major task was to transfer control automatically from one job to the next. The operating system was always resident in memory.



Memory layout for a simple batch system

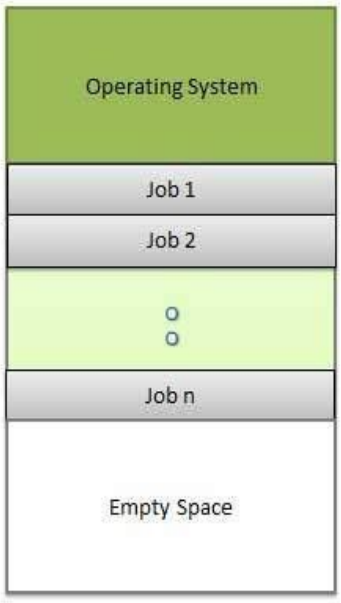
A batch operating system, thus normally reads a stream of separate jobs. When the job is complete its output is usually printed on a line printer. The definitive feature of batch system is the lack of interaction between the user and the job while the job is executing. Spooling is also used for processing data at remote sites.

2. Multi-programmed Systems

A pool of jobs on disk allows the OS to select which job to run next, to increase CPU utilization. Multiprogramming increases CPU utilization by organizing jobs such that the CPU always has one to execute.

The idea is as follows: The operating system keeps several simultaneously jobs in memory. This set of jobs is a subset of the jobs kept in job pool. The operating system picks and begins to execute one of the jobs in the memory.

Memory layout for a multiprogramming system.



3. Time-Sharing Systems

Time sharing (or multitasking) is a logical extension of multiprogramming. The CPU executes multiple jobs by switching among them, but the switches occur so frequently that the users can interact with each program while it is running.

A time-shared operating system allows many users to share the computer simultaneously. Since each action or command in a time-shared system tends to be short, only a little CPU time is needed for each user.

As the system switches rapidly from one user to the next, each user is given the impression that the entire computer system is dedicated to her use, even though it is being shared among many users.

4. Desktop Systems

As hardware costs have decreased, it has once again become feasible to have a computer system dedicated to a single user. These types of computer systems are usually referred to as personal computers (PCS). They are microcomputers that are smaller and less expensive than mainframe computers.

Operating systems for these computers have benefited from the development of operating systems for mainframes in several ways.

5. Multiprocessor Systems

Multiprocessor systems (also known as parallel systems or tightly coupled systems) have more than one processor in close communication, sharing the computer bus, the clock, and sometimes memory and peripheral devices.

Multiprocessor systems have three main advantages.

- Increased throughput
- Economy of scale
- Increased reliability

If functions can be distributed properly among several processors, then the failure of one processor will not halt the system, only slow it down. If we have ten processors and one fails, then each of the remaining nine processors must pick up a share of the work of the failed processor.

Thus, the entire system runs only 10 percent slower, rather than failing altogether. This ability to continue providing service proportional to the level of surviving hardware is called graceful degradation. Systems designed for graceful degradation are also called fault tolerant.

Continued operation in the presence of failures requires a mechanism to allow the failure to be detected, diagnosed, and, if possible, corrected.

The most common multiple-processor systems now use symmetric multiprocessing (SMP), in which each processor runs an identical copy of the operating system, and these copies communicate with one another as needed.

Some systems use asymmetric multiprocessing, in which each processor is assigned a specific task. A master processor controls the system; the other processors either look to the master for instruction or have predefined tasks. This scheme defines a master-slave relationship. The master processor schedules and allocates work to the slave processors.

6. Distributed Systems

In contrast to the tightly coupled systems, the processors do not share memory or a clock. Instead, each processor has its own local memory.

The processors communicate with one another through various communication lines, such as high speed buses or telephone lines. These systems are usually referred to as loosely coupled systems, or distributed systems.

Advantages of distributed systems

- Resource Sharing
- Computation speedup
- Reliability
- Communication

7. Real-Time Systems

Systems that control scientific experiments, medical imaging systems, industrial control systems, and certain display systems are real-time systems.

Some automobile-engine fuel-injection systems, home-appliance controllers, and weapon systems are also real-time systems. A real-time system has well-defined, fixed time constraints.

Real-time systems come in two flavors: hard and soft.

Hard real –time system

A hard real-time system guarantees that critical tasks be completed on time. Missing the deadline may have disastrous consequences. The usefulness of result produced by a hard real time system decreases abruptly and may become negative if tardiness increases. Tardiness means how late a real time system completes its task with respect to its deadline.

Ex: Flight controller system

Soft real time system

This type of system can miss its deadline occasionally with some acceptably low probability. Missing the deadline has no disastrous consequences. The usefulness of result produced by a soft real time system decreases gradually with increase in tardiness.

Ex: Telephone switches

OPERATING SYSTEM COMPONENTS

There are eight major operating system components. They are:

- Process management
- Main-memory management
- File management
- I/O-system management
- Secondary-storage management
- Networking
- Protection system
- Command-interpreter system

Process Management

A process can be thought of as a program in execution. A batch job is a process. A time shared user program is a process. It needs certain resources-including CPU time, memory, files, and I/O devices-to accomplish its task.

A program by itself is not a process; a program is a passive entity, such as the contents of a file stored on disk, whereas a process is an active entity, with a program counter specifying the next instruction to execute. A process is the unit of work in a system.

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

Main – Memory Management

Main memory is a large array of words or bytes, ranging in size from hundreds of thousands to billions. Each word or byte has its own address. It is a repository of quickly accessible data shared by the CPU and I/O devices.

To improve both the utilization of the CPU and the speed of the computer's response to its users, we must keep several programs in memory.

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

- Keeping track of which parts of memory are currently being used and by whom
- Deciding which processes to load when memory space becomes available
- Allocate and de-allocate memory space as needed

File Management

File management is one of the most visible components of an operating system. The operating system is responsible for the following activities in connection with file management:

- 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 (nonvolatile) storage media

I/O System management

One of the purposes of an operating system is to hide the peculiarities of specific hardware devices from the user. This is done using the I/O subsystem.

The I/O subsystem consists of a memory-management component that includes

- Buffering, caching, and spooling.
- A general device-driver interface
- Drivers for specific hardware devices

Secondary storage management

Because main memory is too small to accommodate all data and programs, and because the data that it holds are lost when power is lost, the computer system must provide secondary storage back up main memory.

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

- Free-space management
- Storage allocation
- Disk Scheduling

Networking

A distributed system is a collection of processors that do not share memory, peripheral devices, or a clock.

Instead, each processor has its own local memory and clock, and the processors communicate with one another through various communication lines, such as high-speed buses or networks.

The processors in the system are connected through a communication network, which can be configured in a number of different ways.

Protection System

Various processes must be protected from one another's activities. For that purpose, mechanisms ensure that the files, memory segments, CPU, and other resources can be operated on by only those processes that have gained proper authorization from the operating system.

Protection is any mechanism for controlling the access of programs, processes, or users to the resources defined by a computer system.

Protection can improve reliability by detecting latent errors at the interfaces between component subsystems.

Command-Interpreter System

One of the most important systems programs for an operating system is the command interpreter. It is the interface between the user and the operating system.

Some operating systems include the command interpreter in the kernel. Other operating systems, such as MS-DOS and UNIX, treat the command interpreter as a special program that is running when a job is initiated, or when a user first logs on (on time-sharing systems).

Many commands are given to the operating system by control statements which deal with:

- Process creation and management
- I/O handling
- Secondary –storage management
- Main-memory management
- File system access

- Protection
- Networking

When a new job is started in a batch system, or when a user logs on to a time-shared system, a program that reads and interprets control statements is executed automatically. This program is sometimes called the control-card interpreter or **the command-line interpreter, and is often known as the shell.**

OPERATING-SYSTEM SERVICES

The OS provides certain services to programs and to the users of those programs.

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

2. **I/O operation**
   A running program may require I/O. This I/O may involve a file or an I/O device.

3. **File-system manipulation**
   The program needs to read, write, create and delete files.

4. **Communications**
   In many circumstances, one process needs to exchange information with another process. Such communication can occur in two major ways. The first takes place between processes that are executing on the same computer; the second takes place between processes that are executing on different computer systems that are tied together by a computer network.

5. **Error detection**
   The operating system constantly needs to be aware of possible errors. Errors may occur in the CPU and memory hardware (such as a memory error or a power failure), in I/O devices (such as a parity error on tape, a connection failure on a network, or lack of paper in the printer), and in the user program (such as an arithmetic overflow, an attempt to access an illegal memory location, or a too-great use of CPU time). For each type of error, the operating system should take the appropriate action to ensure correct and consistent computing.

6. **Resource allocation**
   Different types of resources are managed by the Os. When there are multiple users or multiple jobs running at the same time, resources must be allocated to each of them.

7. **Accounting**
   Keeps track which users use how many and which kinds of computer resources. This record keeping may be used for accounting or simply for accumulating usage statistics.

8. Protection

The owners of information stored in a multiuser computer system may want to control use of that information. Security of the system is also important.

SYSTEM CALLS

System calls provide the interface between a process and the operating system. These calls are generally available as assembly-language instructions. System calls can be grouped roughly into five major categories:

- Process control
- File management
- Device management
- Information maintenance
- Communications.

Process Control

- end, abort
- load, execute
- Create process and terminate process
- get process attributes and set process attributes.
- Wait for time, wait event, signal event
- Allocate and free memory.

File Management

- Create file, delete file
- Open , close
- Read, write, reposition
- Get file attributes, set file attributes.

Device Management

- Request device, release device
- Read, write, reposition
- Get device attributes, set device attributes
- Logically attach or detach devices

Information maintenance

- Get time or date, set time or date
- Get system data, set system data
- Get process, file, or device attributes
- Set process, file or device attributes

Communications

- Create, delete communication connection
- Send, receive messages
- Transfer status information
- Attach or detach remote devices

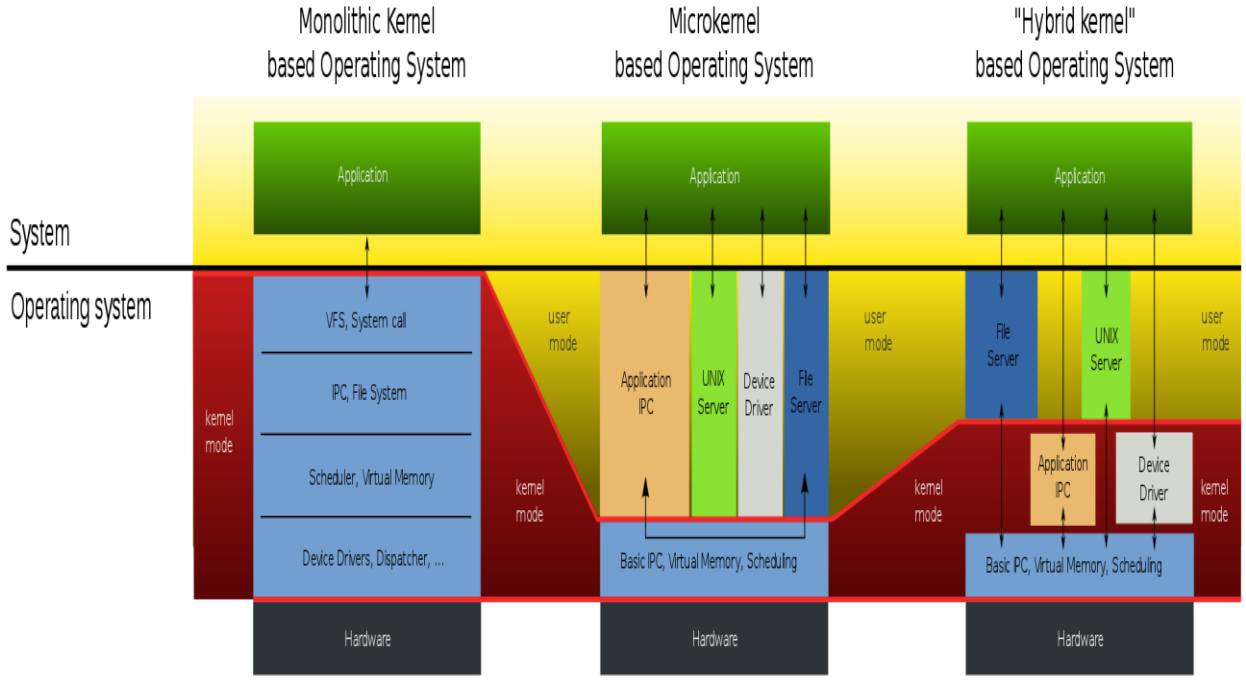
Two types of communication models

- Message passing model
- Shared memory model

ARCHITECTURE AND DESIGN OF OPERATING SYSTEM

The kernel is the core of an operating system. It is the software responsible for running programs and providing secure access to the machine's hardware. Since there are many programs, and resources are limited, the kernel also decides when and how long a program should run. This is called scheduling.

Accessing the hardware directly can be very complex, since there are many different hardware designs for the same type of component. Kernels usually implement some level of hardware abstraction (a set of instructions universal to all devices of a certain type) to hide the underlying complexity from applications and provide a clean and uniform interface. This helps application programmers to develop programs without having to know how to program for specific devices. The kernel relies upon software drivers that translate the generic command into instructions specific to that device.



Monolithic kernel

It manages system resources between application and hardware, but user services and kernel services are implemented under same address space. It increases the size of the kernel, thus increases size of operating system as well.

This kernel provides CPU scheduling, memory management, file management and other operating system functions through system calls. As both services are implemented under same address space, this makes operating system execution faster.

If any service fails the entire system crashes, and it is one of the drawbacks of this kernel. The entire operating system needs modification if user adds a new service.

Advantages

- It provides CPU scheduling, memory management, file management and other operating system functions through system calls.
- A single large process running entirely in a single address space.
- It is a single static binary file. Example of some Monolithic Kernel based OSs are: Unix, Linux, Open VMS, XTS-400, z/TPF.

Disadvantages

- If any service fails it leads to entire system failure.
- If a user has to add any new service, the entire operating system needs to be modified.

Microkernel

In a microkernel, the user services and kernel services are implemented in different address space. The user services are kept in user address space, and kernel services are kept under kernel address space, thus also reduces the size of kernel and size of operating system as well.

It provides minimal services of process and memory management. The communication between client program/application and services running in user address space is established through message passing, reducing the speed of execution microkernel. The Operating System remains unaffected as user services and kernel services are isolated so if any user service fails it does not affect kernel service. Thus it adds to one of the advantages in a microkernel. It is easily extendable i.e. if any new services are to be added they are added to user address space and hence requires no modification in kernel space. It is also portable, secure and reliable.

Advantages

- The architecture of this kernel is small and isolated hence it can function better.
- Expansion of the system is easier; it is simply added in the system application without disturbing the kernel.

Hybrid Kernel

The hybrid kernel attempts to combine the features and aspects of the microkernel and the monolithic kernel. This means that the kernel structure should be similar to a microkernel but the structure should be implemented like a monolithic kernel.

A well-known example of the hybrid kernel is the Microsoft Windows NT kernel

INTRODUCTION TO PROCESSES

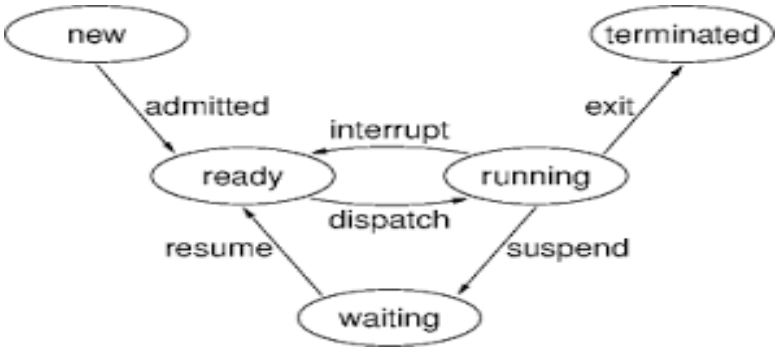
A **process** is a program in execution; process execution must progress in sequential fashion. A process includes:

- Program counter
- Stack
- Data section

Process state

As a process executes, it changes state. They are,

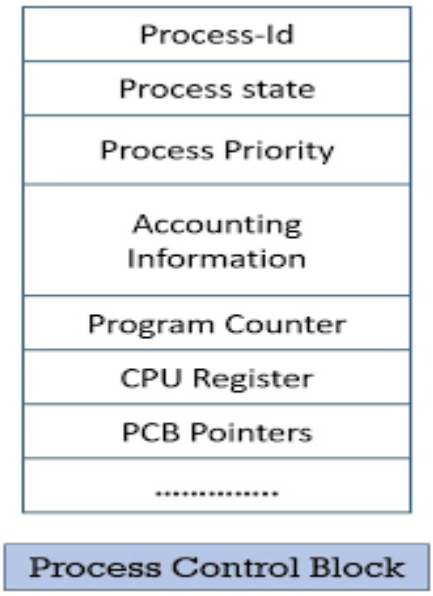
- New- The process is being created
- Running- Instructions are being executed
- Waiting- The process is waiting to be assigned to a process
- Ready- The process is waiting to be assigned to a process
- Terminated - The process has finished execution



PROCESS CONTROL BLOCK (PCB)

It has information associated with process. Every process has its own PCB. It consists of

- Process state
- Program counter
- CPU register
- CPU scheduling information
- Memory-Management information
- Accounting information
- I/O status information



PROCESS SCHEDULING

The objective of multiprogramming is to have some process running at all times, so as to maximize CPU utilization.

Scheduling Queues

There are 3 types of scheduling queues .They are:

- Job Queue
- Ready Queue
- Device Queue

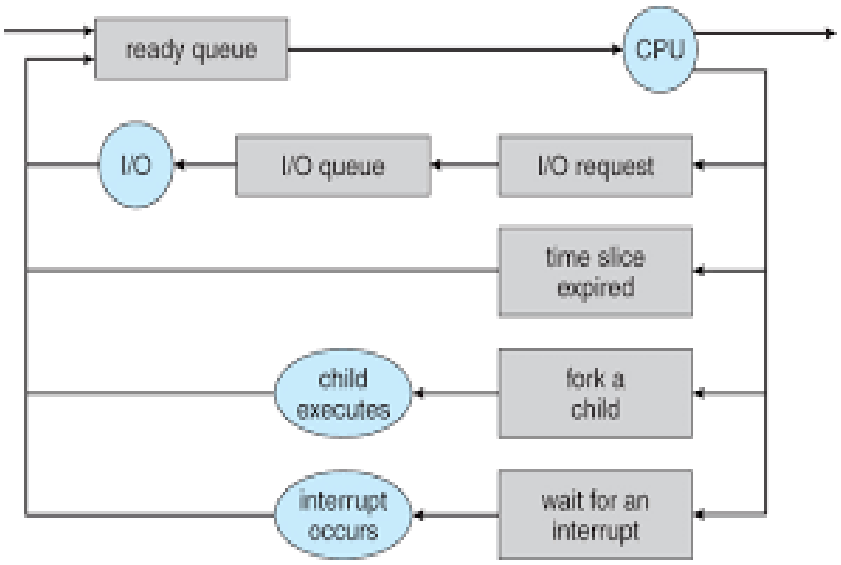
As processes enter the system, they are put into a *job queue*. The processes that are residing in main memory and are ready and waiting to execute are kept on a list called the *ready queue.* The list of processes waiting for an I/O device is kept in a *device queue* for that particular device.

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 several events could occur:

- The process could issue an I/O request, and then be placed in an I/O queue.
- The process could create a new sub-process 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.

A common representation of process scheduling is a queuing diagram.



Schedulers

A process migrates between the various scheduling queues throughout its lifetime. The operating system must select, for scheduling purposes, processes from these queues in some fashion. The selection process is carried out by the appropriate scheduler. There are three different types of schedulers. They are:

- Long-term Scheduler or Job Scheduler
- Short-term Scheduler or CPU Scheduler
- Medium term Scheduler

The long-term scheduler, or job scheduler, selects processes from this pool and loads them into memory for execution. It is invoked very in frequently. It controls the degree of multiprogramming.

The short-term scheduler, or CPU scheduler, selects from among the processes that are ready to execute, and allocates the CPU to one of them. It is invoked very frequently. Processes can be described as either I/O bound or CPU bound.

An I\O-bound process spends more of its time doing I/O than it spends doing computations. A CPU-bound process, on the other hand, generates I/O requests infrequently, using more of its time doing computation than an I/O-bound process uses.

The system with the best performance will have a combination of CPU-bound and I/O-bound processes.

The Medium term Scheduler -some operating systems, such as time-sharing systems, may introduce an additional, intermediate level of scheduling.

The key idea is medium-term scheduler, removes processes from memory and thus reduces the degree of multiprogramming. At some later time, the process can be reintroduced into memory and its execution can be continued where it left off. This scheme is called swapping.

Context switch

When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process.

OPERATIONS ON PROCESSES

Process Creation

A process may create several new processes, during the course of execution. The creating process is called a parent process, whereas the new processes are called the children of that process.

When a process creates a new process, two possibilities exist in terms of execution:

- The parent continues to execute concurrently with its children.
- The parent waits until some or all of its children have terminated.

There are also two possibilities in terms of the address space of the new process:

- The child process is a duplicate of the parent process.
- The child process has a program loaded into it.

In UNIX, each process is identified by its process identifier, which is a unique integer. A new process is created by the fork system call.

Process Termination

A process terminates when it finishes executing its final statement and asks the operating system to delete it by using the exit system call. At that point, the process may return data (output) to its parent process (via the wait system call).

A process can cause the termination of another process via an appropriate system call. A parent may terminate the execution of one of its children for a variety of reasons, such as these:

- The child has exceeded its usage of some of the resources that it has been allocated.
- The task assigned to the child is no longer required.
- The parent is exiting, and the operating system does not allow a child to continue if its parent terminates. On such systems, if a process terminates (either normally or abnormally), then all its children must also be terminated. This phenomenon, referred to as cascading termination, is normally initiated by the operating system.

INTERPROCESS COMMUNICATION

Operating systems provide the means for cooperating processes to communicate with each other via an inter-process communication (PC) facility.

IPC provides a mechanism to allow processes to communicate and to synchronize their actions.IPC is best provided by a message passing system.

Basic Structure:

If processes P and Q want to communicate, they must send messages to and receive messages from each other; a communication link must exist between them.

Physical implementation of the link is done through a hardware bus, network etc. There are several methods for logically implementing a link and the operations:

- Direct or indirect communication
- Symmetric or asymmetric communication
- Automatic or explicit buffering
- Send by copy or send by reference
- Fixed-sized or variable-sized messages

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

Each process that wants to communicate must explicitly name the recipient or sender of the communication.

A communication link in this scheme has the following properties:

- 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.
- A link is associated with exactly two processes.
- Exactly one link exists between each pair of processes.

There are two ways of addressing namely

- Symmetry in addressing
- Asymmetry in addressing

In symmetry in addressing, the send and receive primitives are defined as:

Send (P, message) - Send a message to process P
Receive (Q, message) - Receive a message from Q

In asymmetry in addressing, the send & receive primitives are defined as:

send (p, message) send a message to process p
receive(id, message) receive message from any process,

id is set to the name of the process with which communication has taken place

Indirect Communication

With indirect communication, the messages are sent to and received from mailboxes, or ports.

The send and receive primitives are defined as follows:

send (A, message) Send a message to mailbox A.
receive (A, message) Receive a message from mailbox A.

A communication link has the following properties:

- A link is established between a pair of processes only if both members of the pair have a shared mailbox.
- A link may be associated with more than two processes.
- A number of different links may exist between each pair of communicating processes, with each link corresponding to one mailbox

Buffering

A link has some capacity that determines the number of message that can reside in it temporarily. This property can be viewed as a queue of messages attached to the link.

There are three ways that such a queue can be implemented.

- **Zero capacity:** Queue length of maximum is 0. No message is waiting in a queue. The sender must wait until the recipient receives the message. (Message system with no buffering)
- **Bounded capacity:** The queue has finite length n. Thus at most n messages can reside in it.
- **Unbounded capacity:** The queue has potentially infinite length. Thus any number of messages can wait in it. The sender is never delayed.

Synchronization

Message passing may be either blocking or non-blocking.

- **Blocking Send** - The sender blocks itself till the message sent by it is received by the receiver.
- **Non-blocking Send** - The sender does not block itself after sending the message but continues with its normal operation.
- **Blocking Receive** - The receiver blocks itself until it receives the message.
- **Non-blocking Receive** – The receiver does not block itself.

References:

1. Abraham Silberschatz Peter B. Galvin , G.Gane," Operating system concepts",Sixth Edition, Addison Wesley Publishing Co., 2003
2. Operating Systems, Willam Stalling, Fouth Edition,Pearson Education, 2003