OS

- It is an interface between the user and hardware of the computer
- Controls the execution of application programs
- Also known as resource manager

OS structures
- Monolithic System
- Layered system
- Virtual machines
- Client server model

Objectives of OS

① Convenience - Computer system can be conveniently used due to OS

② Efficiency - Computer system comprises of many resources
- All these resources are utilized by users application in efficient manner due to OS

③ Ability to Evolve - Design of os permits the efficient development, testing. Flexible to work on

Functions of OS

① Process Management
- Control access to shared resources like file, memory, i/o, cpu.
- Control execution of user application
- creation, execution, deletion of user and system process
- Resume a process execution or cancel it
- Scheduling of a process
- Synchronization, communication, deadlock handling

② Memory Management
- Allocates primary and secondary memory to user and system processes
- Reclaim the allocated memory from the completed processes
- Once used block became free, OS allocates it again
- Monitoring and Keeping track of memory usage.

③ File Management
- Creation and deletion of files and directories
- OS offers the service to access the files.
- Keeps backup of files
- Offers the security for files.

④ Device Management
- Device drivers are opened, closed, written by OS.
- Communicate, control and monitor to the device drivers

⑤ Protection and Security
- Resources are protected by OS
- Os makes use of user authentication
- Read, write, encryption, back of data

⑥ User Interface
- offers set of commands
- Software and hardware interaction

⑦ Booting the computer
- Process of starting or restarting the computer
- switched off → Turned on → Cold booting
- Using OS → Restart → Warm booting

⑧ Performs basic computer tasks
- By i/p & o/p devices
- Plug & play → Automatic recognition

Monolithic Kernel

1. It is a single large process running entirely in single address space
2. All kernel services exists and executes in kernel address space
3. Difficult to add new functionality
4. Fast
5. Kernel contains the OS services
6. Larger in size
7. Failure in one component will affect the entire system
8. Advantage
   - Addition / removal is not possible
   - less / zero flexible
9. Disadvantage
   - Intercomponent communication is better
10. The kernel can invoke function directly

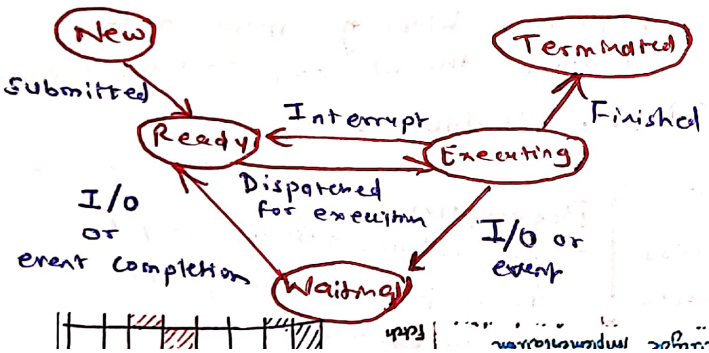
Micro kernel

1. The kernel is broken down into separate processes, known as servers
2. All servers are separate and runs in different address space
3. Easier to add new functionality
4. Slow
5. OS services and kernel are separated
6. Smaller in size
7. Failure in one component will not affect the other components
8. Advantage
   - Flexible for changes
9. Disadvantage
   - Communication Overhead
10. Communication is done via message passing

Process

- It is a program under execution
- It is not as same as program code but a lot more than it.

Process state Transition diagram



1. New state - The new process being created
2. Ready state - A process is ready to run but it is waiting for CPU instruction
3. Executing state - A process is in executing state if currently CPU is allocated to it.
4. Waiting state - A process can't continue the execution because it is waiting for event to happen as I/O completion
5. Terminated state - The process has completed execution

Process

1. Program in execution called as process
2. Processes are heavy weight operation
3. Every process has its own memory space
4. Context switch takes more time
5. New process creation takes more time
6. New process termination takes more time
7. Processes don't share memory with other processes
8. Each process executes the same code but has its own memory

Threads

1. Thread is a part of process
2. Threads are light weight operation
3. Threads use the memory of processes they belong to
4. Context switch takes less time
5. New Thread creation takes less time
6. New Thread termination takes less time
7. Threads share the memory with other threads of the same process
8. All threads can share same set of open files, child processes

User level Thread

1. Implemented by user
2. OS doesn't recognize user level thread
3. Implementation of user level thread is easy
4. Context switch time is less
5. Context switch requires no hardware support
6. If one user level thread performs blocking operation then entire process will be blocked
7. Example - Java Thread, Posix Thread

Kernel Level Thread

1. Implemented by OS
2. Kernel threads are recognized by OS
3. Implementation of Kernel level thread is complicated
4. Context switch time is more
5. Hardware support is needed
6. If one kernel level thread performs blocking operation then another thread can continue execution
7. Ex - Window Solaris

Thread

- A thread is a single stream within a process
- Threads are also called as lightweight processes
- Each thread belongs to exactly one process
- Types of thread
  1. User level thread
  2. Kernel level thread

Multithreading

- It is a type of execution model that allows multiple threads to run independently but share their process resources
- Types of multithreading
  1. Preemptive multithreading
  2. Co-operative multithreading

Preemptive

- It allows taking away CPU from process during execution

Non preemptive

- Once process is allocated to CPU, It does not free CPU until it complete its execution

CPU Utilization

- The amount of time CPU remains busy

Throughput

- Number of jobs processed per unit time

Turnaround Time

- The time difference between completion and arrival time

Response Time

- Time from submission till the first response is produced

Fairness

- Every process should get fair share of CPU time

Burst Time

- Time required by the process to execute

Waiting Time

- Processes waits in ready queue to get CPU.
- Amount of time spent in ready queue is waiting time

Completion time

- The time when process completes its execution

Arrival Time

- The time when the process is arriving into ready state

System calls

① system calls provide an interface to the services made available by an OS.

② 2 modes
- → User mode - All user processes are executed
- → System mode - All privileged operations are executed

③ Advantage
- → security
- → Due to system calls a user program is unable to enter in OS

Concurrency / process synchronization

- Concurrency is the execution of several instruction sequences at the same time
- In an OS, this happens when there are several process threads running in parallel
- These threads communicates with each other through → ① shared memory ② message passing

---

① Open() - A program initializes access to a file in a file system using the open system call.

② Read() - A program that needs to access data from a file stored in a file system uses the read system call.

③ Write() - It writes data from a buffer declared by the user to a given file.
- Primary way to output data from a program by directly using system call

④ Exec() - Exec is a functionality of an OS that runs an executable files in the context of an already existing files, replacing the previous executable

⑤ Fork() - Fork is an operation whereby a process creates a copy of itself
- Primary way of process creation on Unix-like OS.

Application software

① It is used by user to perform specific task

② Appl^n Software installed according to users requirement

③ User interacts with appl^n software

④ Appl^n software can't run independently. System software is needed to run Appl^n softw

⑤ Runs when the user requires

⑥ Ex - Web browsers, spreadsheets etc

System software

① It is used for operating computer hardware

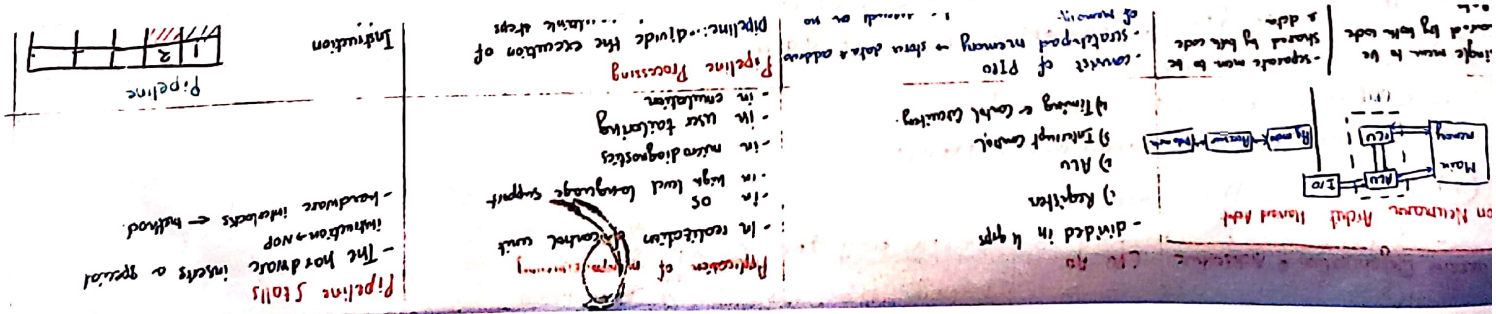
② System softwares are installed when OS is installed

③ User does not interact with system software

④ System software can run independently

⑤ Runs when the system starts and runs till end

⑥ Ex - OS, device drivers



Kernel
- Central part of an OS
- manages operations of the computer and hardware, memory, CPU Time
- When a process makes requests to the kernel, → called as system calls

Shell
- Unix term for Interactive user interface
- known as command interpreter

Inter process Communication

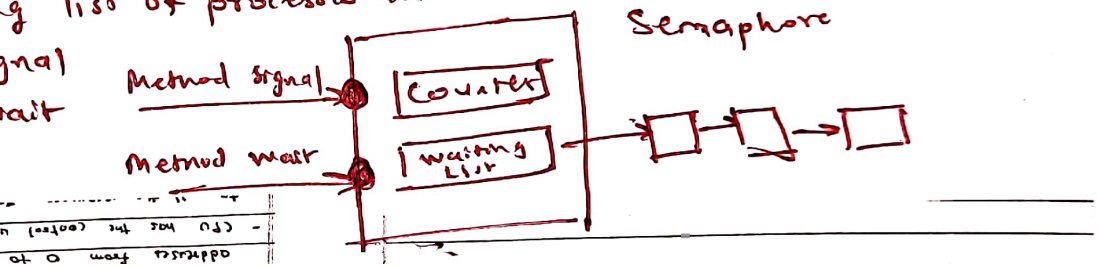
- Synchronization and Communication are two basic requirements which should satisfy when process communicates with each other
- Synchronization of processes is required to achieve the mutual exclusion
- Independent processes do not communicate with each other.
- Cooperative processes may need to exchange info.
- Cooperative process communicates through
  1. shared memory
  2. message passing

Mutual Exclusion

- When a process is accessing a shared variable, the process is said to be in a critical section
- No two process can be in the same critical section at the same time
  This is called mutual exclusion

Semaphores

- A semaphore is an object that consist of a counter, a waiting list of processes and two methods. 1. Signal 2. wait



Shared Memory

- A region of memory that is shared by cooperative processes is established
- Processes can exchange info by reading and writing data to the shared region

Message Passing

- In message passing, communication takes place by means of messages exchanged between the cooperative processes
- Primitives used in message passing
  1. send (destination, message)
  2. receive (source, message)
- A process sends data in the form of a message to another process indicated by a destination
- A process receives data by executing the receive primitive, indicating the source, the message