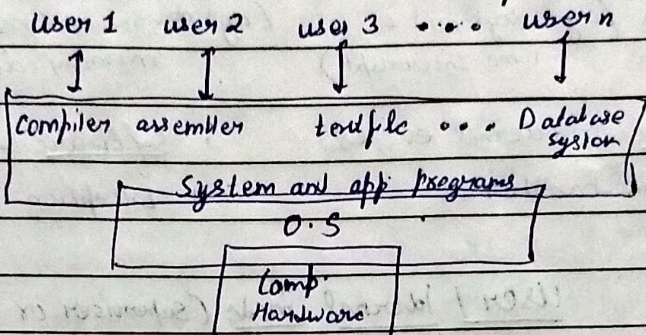
Unit-1

Date ____________

Introduction

O.S → It's a program that manages computer's Hardware, provides a basis for application programs and act as an intermediary b/w end user and comp' hardware



{eg- govt' one. it perform no useful funᶜⁿ but provide env' for every othe program}

. User's p.o.v : • It provide ease of use and stability • Maximum resource utilization

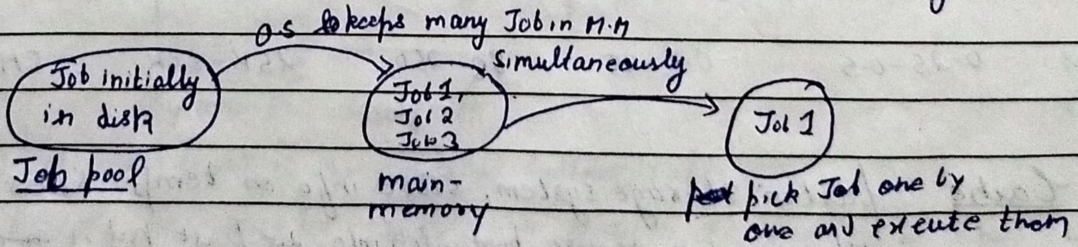
. System's P.O.V : • Resource allocator (C.P.U time, I/O, memory and file storage space) • Control progam {manages user program and prevent error}

↳ O.S is The program running all times on computer usually called Kernel.

Multiprogramming V's Multi-tasking (Time-sharing o.s)

Multiprogramming	Multi-tasking (Time-sharing o.s)
→ Multiple prog' are loaded into memory to share C.P.U {max' C.P.U utilisation}	• Multiple processes execute simultaneously by rapidly switching b/w them {It's so freqvent that gives illusion that all programs are running at same time}
→ Non-premptive {C.P.U switches only when current process completes or blocks}	• premptive scheduling type {Switching based on regular time quantum and priority}
→ Less freqvent context switching	• More freqvent content switching

Main Idea:


O.S keeps many Job in M.M simultaneously
Job initially in disk
Job pool
Job 1, Job 2 Job 3
main-memory
Job 1
pick Job one by one and execute them

· preem' Kernel > Non-preem one b/c →

(i) Improved responsiveness

(ii) Better utilization of resources

→ and points of Multi-tasking.

- Multimode → Multiple mode operation
  - ~~Support~~ allows Virtual machine manager
    - ↱ more privilesged than user mode
    - ↳ less than Kernel

Date

Interrupt | Trap or Execution

Interrupt	Trap or Execution
→ a signal generated by external hardware or internal devices to notify CPU about an event that needs immediate attention (eg- Keyboard or Time Interrupt)	• a signal generated by C.P.U due to an error or a specific conditions in program (eg: divisision by zero, invalid memory access)
→ Triggered by: Hardware or external events	• Software
→ Handling: Interrupt handler	• Exception Handler

User | Kernel mode (Superviser or system mode)

User	Kernel
→ • restricted mode where app run with limited access to system resources	• privilesged mode where O.S has full access to system and resources
• mode bit: '1'	• mode bit: '0'

↓ when user app request service from O.S (via system call), System transistion b/w User ⟺ Kernel mode

• During reboot first Kernel mode starts

- Timer: It's a mechanism that ensures the O.S maintain control over C.P.U, by interrupting a process at regular intervals

It may be fixed or variable, {When counter reaches '0', an interrupt occurs} ~~and OS switches~~ to ~~Kernel mode~~ control transfers to O.S

Level:	1	2	3	4	5
	registers	cache	main-memory	S.S.D	Magnetic disk
(Size) typical	<1 KB	<16 MB	<64 GB	<1 TB	<10 TB
Access time (ns)	0.25-0.5	0.5-25	80-250	25K-50K	5M

- Cache: faster storage system, store info on temp basis
  If we need or to use particular info, we first put it ~~a~~ copy in cache

- Cache Coherency: It ensures that when data is updated in one processors cache, the change is reflected in all other caches that store same data preventing inconsistencies and asynchronization

Unit-2

sh→shell
Scripts

Date

- **O.S Services:**

1) U.I → Cmd line
→ batch interface {.bat, .cmd, .sh)
→ G.U.I

2) Program execution : system load program in memory run it, and program must end either normally or abnormally

3) I/O oper : O.S provides a means to do I/O like access file, etc, ∵ users can't control I/O devices directly

4) file-system manipulation :: program can read, write, dlt, search in file or dir
. also give permission management based on ownership

5) Communication : · Process exchange info in same system (Shared memory or msg passing)
in across system (Network comm)
enable collab b/w Process ↔ System

6) Error detection : Constant monitoring for Interrupt or trap and do action like Halt System (if require), terminate problem prog

More (Not for user but for System efficiency)

1) Resource allocation :: Manage resources like C.P.U cycle, memory, storage etc
· use algo like C.P.U, disk scheduling for this

2) ~~Accounting: Tracks resource usage by~~ users or processes

3) Protection and security : · ensure controlled access to resources
· User authentication for access
· Logs connection attempt
· emphasize : end to end precaution

- System Calls and A.P.I's

System Calls

- Provides an interface to access O.S services ~~services~~. {Written in C, C++ or assembly}

A.P.I

- Provides higher level interface for developers, hiding complex details of System calls
- e.g: Windows API, POSIX API
- Makes portable prog to access across diff O.S ~~supporting~~ same API.

System-Call Interface

: Intercepts A.P.I calls and translate into actual System calls

- System calls typically are Indexed by numbers $\xrightarrow[\text{maps those}]{\text{Interface}}$ Number to kernel's internal calls

- Parameters are passed through → Registers, memory block and Stack

Registers: {fastest methods as they're directly access by C.P.U}

memory block: Para stored in M.B (Table) and its address is passed in registers to O.S

Stack: · Pari are pushed into stack and popped off by O.S during execution
· support variable Spiral no of parameters

Types of System Calls :

(i) Protection management : • System calls manages life cycle of processes — creating, terminating {Normal - end(), abnormal - abort()), and Handling errors {error details examined by debugger to fix it. errors have level (0 to severe), error level can guide recovery actions}.

• Locks (acquire, release lock(), ensure data integrity in concurrent system. , load, exec, wait

(ii) file management : create(), delete(), move(), copy(), get-file-attribute()
open(), read() set-file- " ()

(iii) Device manag : • request device(), release device()
• get - device - att()
set - " - att()

(iv) Info ~~msg~~ mang : get or set time or date, System data, process file

(v) Protection :

(v) Communications : → ~~needs~~ message passing : for small data exchange, and inter comp connections
→ Shared memory : for fast comm but requires synchronization

→ create, delete comm → send, receive msg
→ attach or detach remote device

• Daemons process : Special system processes run in b/g without interaction from user {provide service like httpd (HTTP), apache2 (web server), syslog, }

• System program : • Designed to manage system resources, and help users interact with system
• Higher level utility e.g: {Shell, Diagnostic tools, etc}

Q How O.S handled Interrupt Signal ?
2
(i) CPU stops execution of current process, and saves its state
(ii) It then Jumps to interrupt handler, to address Interrupt
(iii) Once Interrupt handled, CPU restores process and resume normally

Q List any two privileged Instruction?
(a) I/O Inst (b) Memory mana Inst (c) System control Inst
d) Interrupt Control

• O.S Structure :

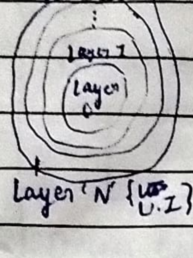
Date ________

(a) Simple Structure : • Straightforward design in which all system components are bundled together in 1 block of code { Monolithic Kernel }

• No division of modules

• e.g- M.S DOS and UNIX Kernel { in early stage }

(b) Layered Structure : • Divides O.S into multiple layers, each of which has specific functionality, and interacts only with the layer below it (organized in Hierarchy)



• Layer 0 : Hardware layer, Layer N : Highest Layer (user interface)

adv' → • Layer can developed and tested Ind'

• easier debugging { once lower layer debugged, it assumed funcn correctly, while debugging above layer }

disadv → • proper planning req' to define layers and its interaction

• less efficient, as each system call must pass through multiple layers, adding overhead

{ M.M, device driver } { below)
(above) { C.P.U Schedul },

(c) Microkernels : • This approach involved designing an O.S by removing all non-essentials component from kernel and implementing system and user level programs, making it more smaller, more modular. them as

• It facilitates comm' b/w client program and services running in user space through message passing.

eg- Mach Kernel (Darwin), QNX

adv → • New services added without modifying Kernel

• Increased Security, smaller and portable O.S

disadv → It suffers from performance overhead due to need for msg passing and communication { problem in Windows NT }

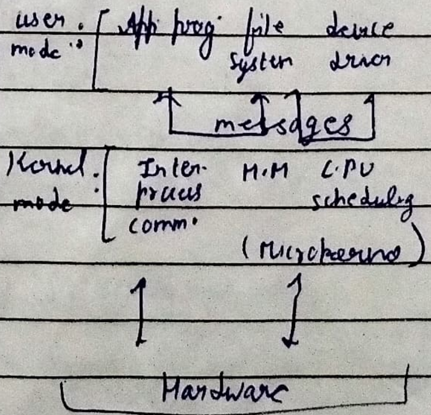
(d) Modules : • This design uses loadable Kernel modules to add flexibility and scalability., instead of building features directly into Kernel, services are implemented as separate modules

• modules can be added ~~to~~ at boot time without need to recompile Kernel

• Comparison

↳ • Similar to Layered, but modules can call any module

• " " Microkernel, but without performance overhead



Spergl (Micro-kernel structure)

Date____________

- In msg. passing - processes don't ~~share~~ have memory, and don't use shared memory, instead they comm' and shared data through comm' system, such as kernel or a msg queue.