OPERATING SYSTEMS

Module-1

Chapter-1

Introduction to Operating Systems

An operating system (OS) is a large & complex system software designed & created piece by piece. Each of these pieces are designed & created with carefully defined inputs, outputs & functions.

Generally, now-a-days, all electronic computing systems are having OS in one (or) another format.

Personal computers, mobiles, laptops, fully automated washing machines, ovens, aircraft management systems, etc doesn't work without OS.

Operating system is an essential part of any computer system. Similarly, a course on OS is an essential part of any CS education.

1.1 What operating systems do?

An operating system is a collection of programs that manages the computer hardware.

An operating system acts as an intermediary between user of a computer & the computer hardware.

The purpose of an OS is to provide an environment in which the user can execute the programs in a convenient & efficient manner.

So, any OS has 2 goals

*) efficient use of computer system

*) user convenience.

But, no OS provides both. Efficient use is important, when a computer system is shared by several users while user convenience is

important in personal computers.

A computer system can be divided roughly into 4 components.

*) Hardware
*) OS
*) Application program
*) User

Hardware consists of memory, CPU, ALU, I/O devices, peripheral devices, etc which provides basic computing resources for the system.

Application programs such as word processors, spreadsheets, compilers & web browsers define the ways in which these resources are used to solve users computing problems.

OS controls the hardware & co-ordinates its use among the various application programs for the various users.

An OS is similar to government. Like a government, it performs no useful function by itself. It simply provides an environment within which other programs can do useful work.

An OS is responsible for co-ordinating all of the computer's individual components so that they work together according to a single plan.

Summary :- Major goals of any OS are

*) execute user programs & make solving user problems easier
*) make the computer system convenient to use
*) use the computer hardware in an efficient manner.

Thus, an OS can be viewed in 2 points of view,

1) User view

2) System view

User view

Always, user wants their OS in a convenient manner i.e., ease of use. So, user view of the computer depends on the interface used i.e., either GUI (or) command line.

*) Some users may use pc's. In this type, systems are designed so that only one user can utilize the resources & mostly for ease of use where the attention is mainly on performances & not on the resource utilization.

*) Some users may use a terminals connected to mainframe (or) minicomputers. Some other users are accessing the same computer through other terminals.

Here, users at different terminals may exchange information & may share the resources.

In this case, OS is designed to maximize resource utilization.

*) In other case, users may sit at workstations connected to networks of other workstations & servers.

There users have dedicated resources at their end but they also share resources such as networking & servers. Therefore, their OS is designed to compromise between individual usability & as well as resource utilization.

*) But some computers have little user view (or) no user view at all.

For ex, embedded computers in home devices & automobiles may have numeric keypads & may turn indicator lights ON (or) OFF to show some status, but they & their OS are designed primarily to run without user intervention.

System view

From the system view (Computer's point of view), OS is the program mainly involved with the hardware. In this context, we can view an OS as a resource allocator.

Computer resources may be cpu time, memory space, file-storage space, I/O devices & so on.

In order to solve any problem, a computer system has to use one or more computer resources.

OS acts as the manager of these resources. OS must decide how to allocate these resources to programs & to the users so that it can operate the computer system efficiently & fairly.

An another different view of OS is that, it need to control various I/O devices & user programs i.e., OS is a control program used to manage the execution of user program to prevent errors & improper use of the computer.

Every OS must support the following tasks

*) should provide facility to create, modification of programs & data files using editors.

*) access to compilers for translating the user program from high level language to machine language.

*) should provide a loader program to move the compiled program code to computer's memory for execution.

*) should provide routines that handle the details of I/O programming.

Why we are using computer?

Fundamental goal of computer system is to execute user programs & to solve user problem easier. In order to achieve this, hardware's are used.

But, hardware's alone can't do all. So, application programs are written to manage & control these hardwares.

Set of programs for controlling & allocating resources are then brought together into one piece of software called operating system.

When we install any OS in our computer system, complete OS won't run, instead an essential part of the OS that will run all the time on the computer called Kernel.

Along with the kernel, there are 2 other types of programs.

1) System programs
2) Application programs

System programs are associated with the OS but are not part of the Kernel.

Application programs includes all programs but not associated with the operation of the system.

1.2 Computer-System Organization

We know, how the individual units of an computer are connected together to form an complete computer system.

1.2.1) Computer-system operation

When we power-ON the computer, one program will run initially. This initial program is called as bootstrap program, which is stored in ROM (or) EEPROM.

This bootstrap program must know how to load OS & how to start executing that system. To do this, bootstrap code should locate the OS kernel & then it should load that OS kernel from secondary memory to the main memory.

Then OS starts executing the first process called "init" & waits for some event to occur.

EEPROM → Electrically Erasable Programable ROM] These ROM (or) EEPROM are present within the computer hardware.

Usually, events occurs in the form of interrupts/exceptions.

1.5 Operating System Operations

Modern OS's are interrupt driven. If there are no processes to execute, no I/O devices to service & no users to whom to respond, an OS will sit quietly, waiting for something to happen.

Always, events are happened/occured in the form of interrupt & trap.

A trap (also called as exception) is a software-generated interrupt which occurs either due to

*) error (for ex, division by zero, invalid memory access)

*) specific request from a user program

When interrupt is raised, then OS executes the corresponding Interrupt Service Routine (ISR) & completes the request (action).

We know that, both hardware & software resources of the computer system are shared by OS & as well as user programs. So, care should be taken when any error occurs due to user program.

With sharing, many processes could be adversely affected by a bug in one program. For example, if a process get stuck in a infinite loop, this loop could prevent the correct operation of many other processes.

More subtle errors can occur in a multiprogramming system, where one erroneous program might modify another program, data of another program (or) even OS itself.

If there is no protection against these type of errors, then it is better to execute only one process at a time otherwise, we need to suspect outputs of all the programs in the multiprogrammed systems.

So care should be taken while designing the proper OS such that any incorrect program should not affect/harm other programs.

1.5.1 Dual-mode operation

In order to ensure the proper execution of OS, we must be able to distinguish between the execution of OS code & user-defined code.

There are 2 modes of operation

1) user mode

2) Kernel mode (also called as supervisor mode, system mode (or) privileged mode)

Single bit called mode bit is used to identify the mode in which OS is running.

If Mode bit = 0 → OS is running in kernel mode

Mode bit = 1 → OS is running in user mode

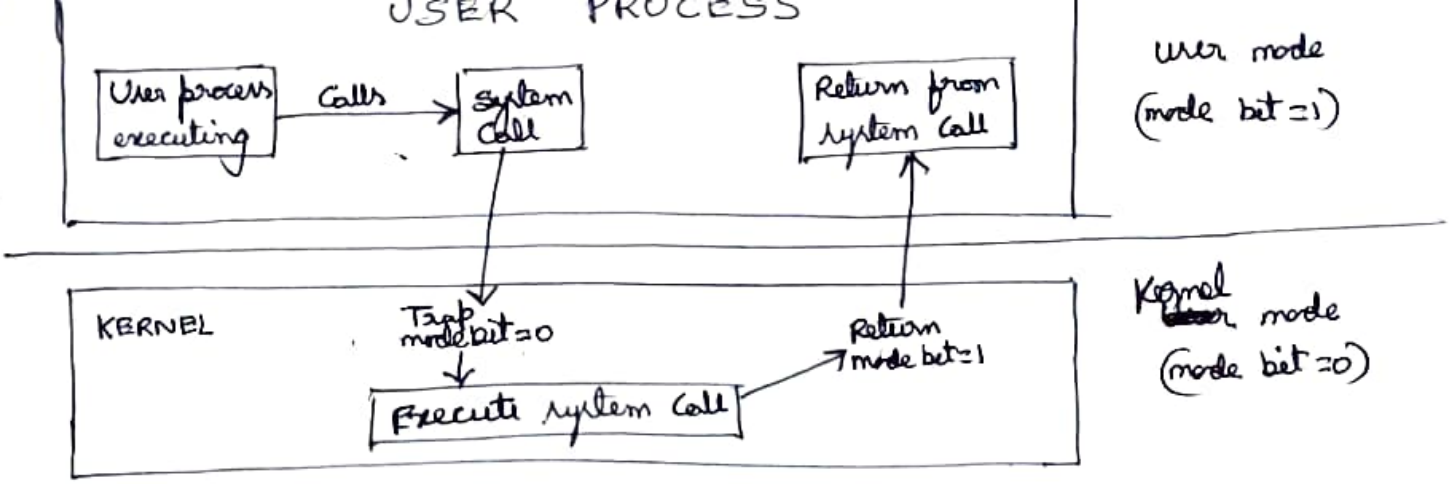

fig 1.5 Transition from user to Kernel mode

After booting OS, normally user performs some action i.e., some user process is executing. Whenever, user process needs some service it requests the OS through some system call. Then, OS switches from user mode to Kernel mode by making mode bit = 0. Then it executes corresponding system call code. This is as shown in fig 1.5.

When system starts booting, hardware starts in Kernel mode. Then, OS is loaded & starts user applications in user mode.

Whenever a trap or interrupt occurs, hardware switches from user mode to Kernel mode by changing the status of mode bit as 0.

Thus, whenever OS gains control of the computer, it is in Kernel mode.

System always switches to user mode before passing control to user program.

Out of all instructions, in the instruction set, some of the instructions are designated as privileged instructions. These privileged instructions are executed only in the Kernel mode.

When any user program tries to execute these privileged instructions in user mode, then it is treated as illegal operation.

1.5.2 Timer

When a user executes some program in user mode, it should not stuck in an infinite loop. Similarly, process should not stuck in Kernel mode for long time. At regular intervals, it should switch the mode. For this purpose, timer is used.

A timer can be set to interrupt the computer after a specified period. This period may be fixed (1/60 second) or variable (from 1m sec to 1 sec).

The OS sets the counter. Every time, clock ticks, counter is decremented. When the counter reaches 0, interrupt occurs. Thus, we can use timer to prevent user program from running too long.

1.6 Process management

14

1X3

chp 2 :- SYSTEM STRUCTURES

We know that, an OS is a software that manages the Computer hardware & also OS provides an environment for application programs to run.

Moreover, OS provides interfacing facility to the user in order to use the computer system.

There are so many types of OS which are different in their structure & functionalities.

The design of new OS is a major task. We can view an OS from 3 points

1) Services that OS provides
2) interface that it makes available to users & programmers
3) OS components & their interconnections.

In this chapter, we will discuss about following

→ what services an OS provides
→ How the services are provided to the user
→ if there are any error, bug in the services, how to debug it
→ various methodologies for designing an OS.
→ how OS's are created & how computer starts its OS.

2.1 OS services

OS provides some services to programs & as well as to the users of those programs.

Services provided by an OS differ from one OS to another but there is some common classes.

OS provides some services for the convenience of programmer to make programming easy. A view of OS services is as shown in the fig 2.1

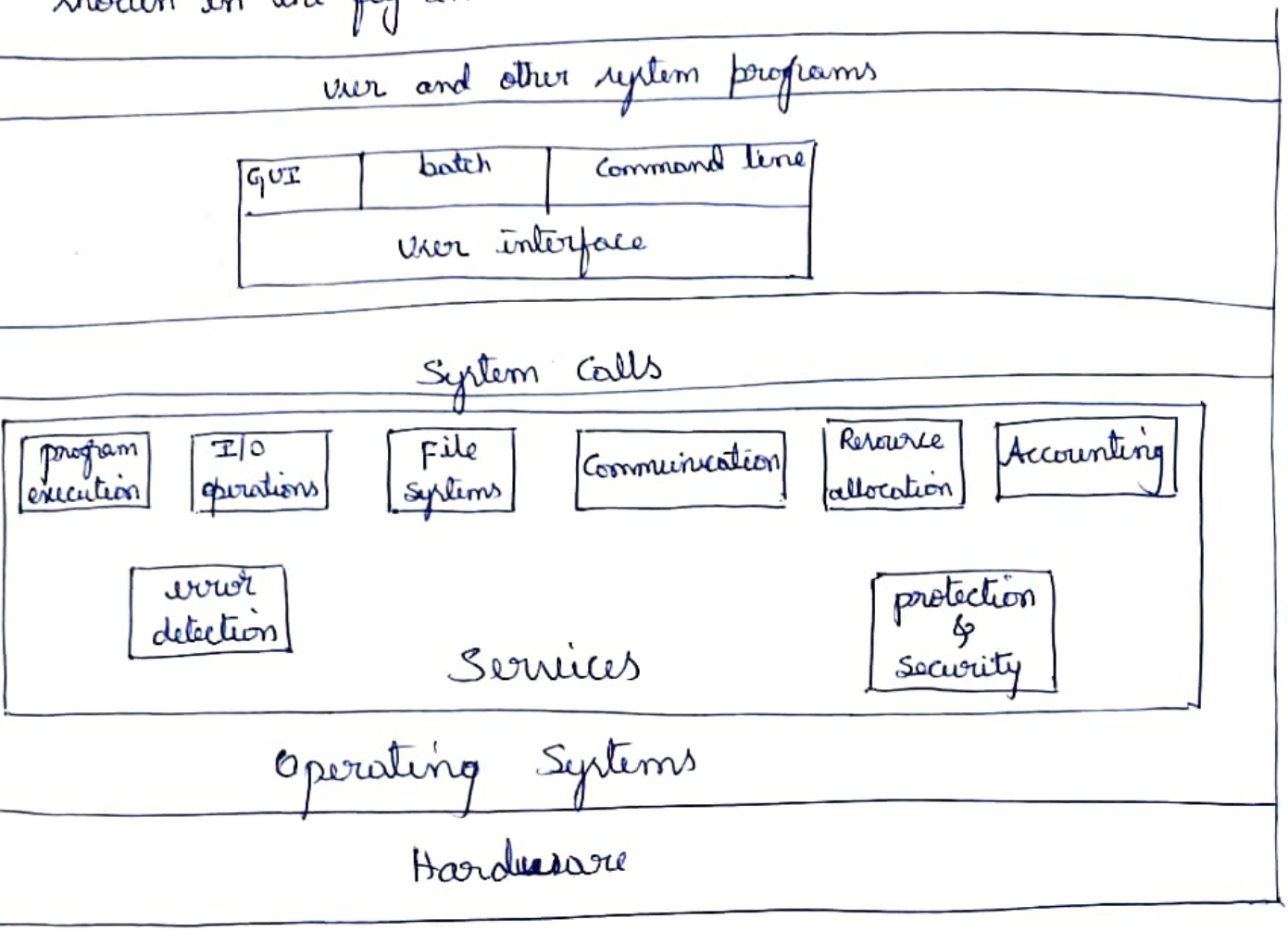

Fig 2.1 A view of OS services

Common services provided by any OS are

1) User interface (UI)

Almost all OS have UI. Whenever user wants to interact with the computer, then UI has to be used. Any OS provides UI in 3 forms

a) Command line interface which uses text commands at the command prompt
b) batch interface in which commands, methods & directives are entered into files & those files are executed.
c) Graphical UI (GUI) which is most common in almost

all the OS.

2) Program execution

OS must be able to load the program from secondary memory to main memory & then OS should run that program. After execution, OS should end that program either normally (Succesful case) or abnormally (indicating error).

3) I/O operations

A running program (process) may require I/O like file or an I/o device. For effeciency & protection, users usually cannot control I/O devices directly. Therefore, OS must provide a means for controlling I/o devices

4) File system manipulation

Always, user performs actions on files only like reading, writing, searching, creating, deleting the files by their names. Sometimes, programs includes permission management to allow or deny access to files or directories based on file ownership.

For all there, OS should provide some effecient features & facilety to perform file system manipulation.

5) Communications

In some situations, one process needs to exchange information with another process. This type of communication may occur between 2 processes that are executing on the same computer or between the processes that are executing on different computer system in the network.

So, OS should implement/provide there type of communication. This is done by OS via shared memory or message passing.

6) error detection

Errors may occur in

→ CPU

→ memory hardware (such as memory full, power failure)

→ I/O devices (parity error on tape, connection failure on network or lack of paper in printer)

→ user program (arithmetic overflow, divide by zero, accessing illegal memory location)

So, OS should aware of possible errors. For each type of error, OS should take appropriate action to ensure Correct & Consistent Computing.

So far we discussed 6 services provided by the OS in order to help user. Now, we will see some other services which provides efficient operation of the system itself.

7) Resource allocation

When multiple users logs onto the system or when multiple jobs are running, resources must be allocated to each of them. So, it is responsibility of OS to manage the available resources.

(Some resources may need some special allocation codes & others may have some general request & release code)

8) Accounting

MADHUSUDHAN M.V.
Dept. of CSE

OS should keep track of which users use how many & what kind of resources. This record keeping may be used for accounting. This accounting data may be used for statistics or billing. It can also be used to improve the system efficiency.

9) protection & security

In multiprocess environment, it is possible that one process may interfere with the other or with OS itself. Some users, stores their information in multiuser computer system or networked computer system, such data should be protected.

Security starts with each user having authenticated to the system, usually by means of password.

External I/O devices must also be protected from invalid access.

2.2 User Operating-System interface

We know that, an OS is a collection of system programs that controls & co-ordinates the overall operations of a computer systems.

Main functionality of an OS is to provide an interface for the user to interact with the computer.

There are several ways for user to interact with the computer, among them most commonly & widely used 2 approaches are 1) Command line interface (Command interpreter
2) Graphical user interface (GUI)

Command line interface (or Command interpreter)

Some OS include this Command interpreter in the Kernel. Others such as Windows XP & Unix, treat this Command interpreter as a special program running when some job is initiated or when user logs in to the system first time.

On some systems, there are multiple command interpreters, user has to choose one among them. In such cases, those interpreters are called as shells.

For ex, on Unix & Linux systems, user may choose among several different shells like Bourne-shell, C-shell, Bourne-Again shell, Korn shell & etc. (Freendly Interaka Shell, Fish) etc

In command line interface (CLI), user has to enter the commands at the command prompt. For every task there will be some commands & user has to remember all those commands & their formats. So, it is not so user friendly.

Some of the basic commands used in Linux are ls, cd, cp, mv, rm, mkdir, rmdir, etc.

Graphical user interface (GUI)

GUI is more user friendly than CLI, because user can simply perform any action without remembering any command.

Here, just user moves the mouse to position its pointer on images or icons on the screen that represents programs, files, directories & system functions.

Here, everything is represented in graphical form. For example, application programs, commands, disk drives, files, etc are presented in the form of icons.

Usually, command is given to the computer by clicking with mouse on the icon. GUI also provides menus, buttons & other graphical objects to the user to perform different tasks. GUI is very easy to interact with the computer.

2.3 System Calls

When the computer is turned ON, the program that gets executed first is called the OS. It controls pretty much all activity in the Computer. This includes who logs in, how disks are used, how memory is used; how cpu is used & how it communicates with the other computers.

We know that, OS provides interface for user to use the Computer & its resources.

User communicate with the Computer through OS. Then, OS has to Communicate with the Computer resources, this is done through system calls.

System calls are built on top of the OS. System calls interact with Kernel area of OS. System calls can also interact with hardware part of a system like Keyboard, mouse, printer, cpu, etc.

System calls are generally available as routines written in C, C++ & most of the times in assembly language instructions.

We can define, "system call as an interface between the OS & a process that allows a process to invoke OS functions".

System calls transfers the control from user level to Kernel level. When a user wants to access a resource (memory, cpu, time, etc), first user requests through a system call to OS, then OS will decide what to do next.

Common ex^s of system calls are read(), write(), open(), etc

Handling user application invoking the open() system call is as shown in the fig 2.2.

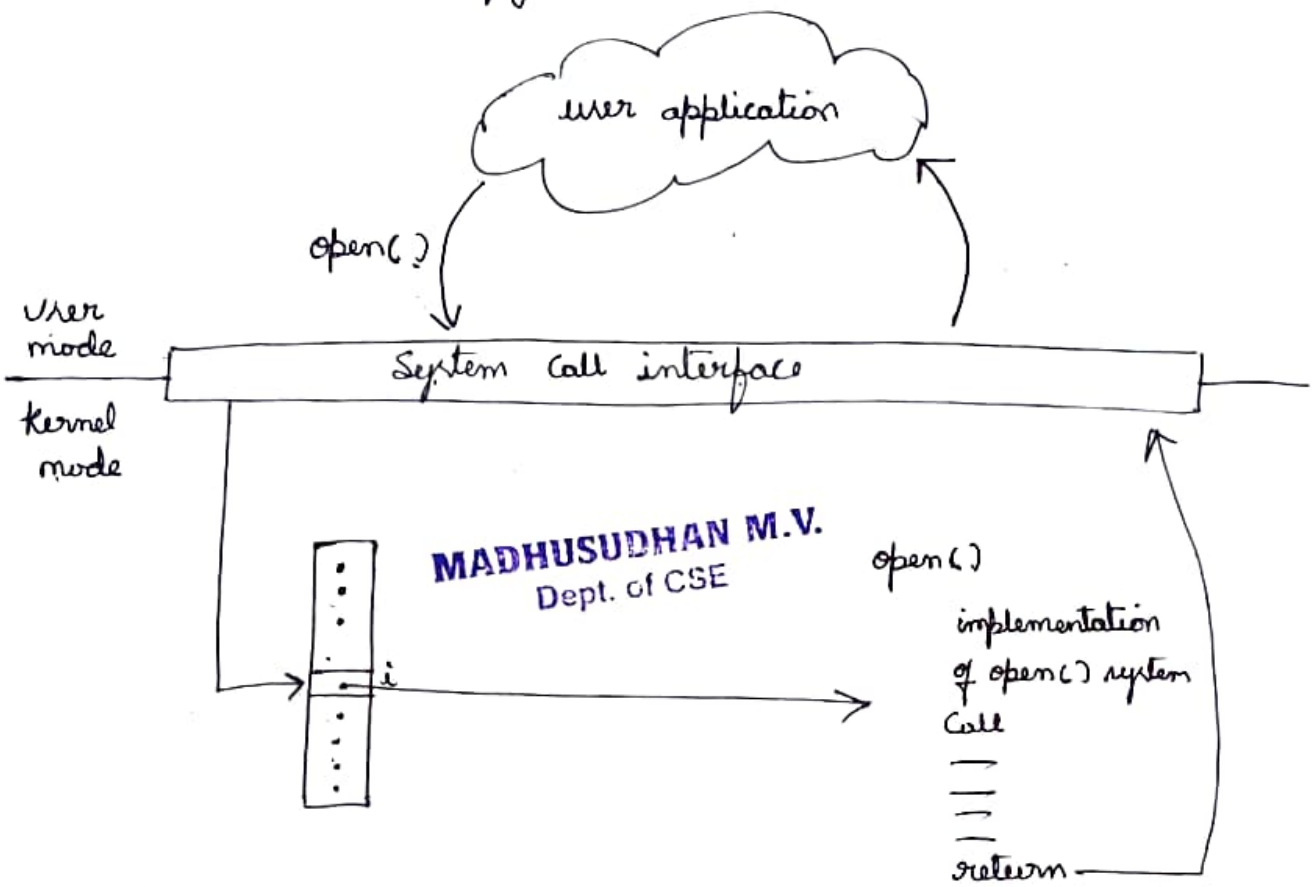

Fig 2.2 Handling user application invoking open() system call

When a user invokes the system call, control transfers from user mode to Kernel mode through system call interface.

Every system call has its own unique number & these numbers are maintained in an table index.

When system call is (open()) invoked, system call interface checks that corresponding number in the table index & runs the appropriate code. After, it returns the control back to user.

When a system call is invoked, it may pass some of the parameters or no parameters.

Ans

There are 3 generals ways/methods to pass parameters to the OS.

1) Through registers.

This is effecient when we have very less number of parameters to pass.

2) Through block or table

When number of parameters are more then, parameters are stored in a block or table in memory & address of the block is passed as a parameter in register. It is as shown in the fig 2.3. Generally used in Linux & Solaris

3) Through stack

parameters can also be placed or pushed on to the stack (at the sending side) by the calling program & popped off the stack by the OS (at the receiving side).

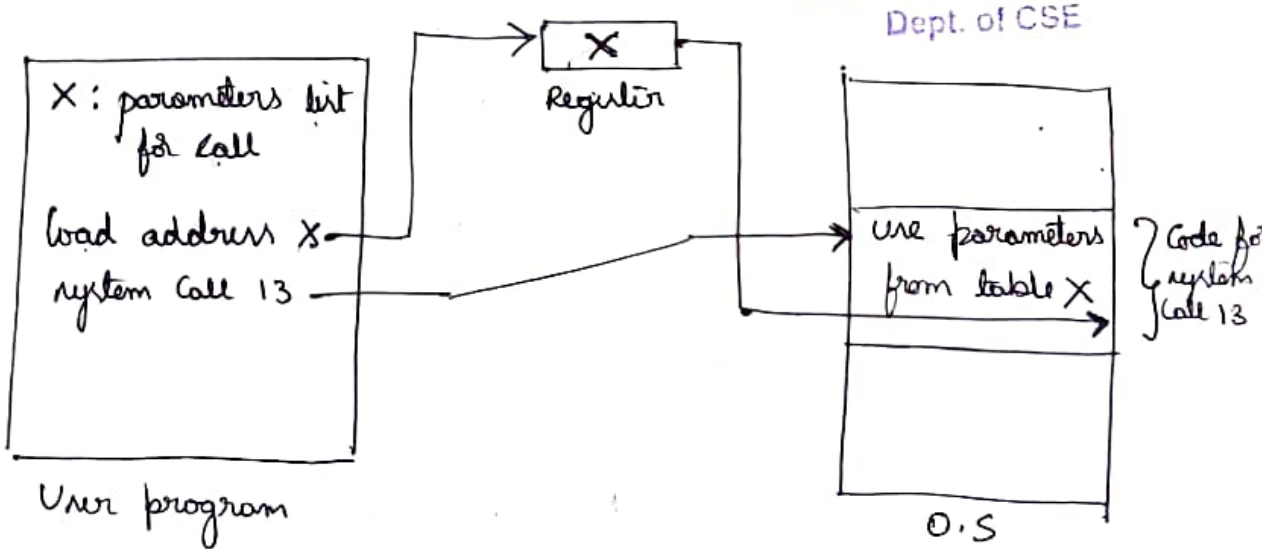

fig 2.3 passing the parameters as through table

Some OS's prefer the block or stack method because those approaches do not limit the number or length of parameters being passed.

1) process control → end, abort, load, execute, create process, terminate process allocate, free memory, etc

2) File management → create file, delete file, open, close, write, read, etc

3) Device management → Request device, release device, read, write, attach, & detach devices

4) Information maintenance → get time or date, set time or date, get system data, set system data, etc,

5) Communication → Create, delete Communication Connection, Send, recieve messages, attach or detach remote devices.

6) protection → user accounts creation & deletion, Creating & deleting groups, authentication, permissions, etc

Fig 2.5

1) process control

A running program is called as process. In order to control these processes some of the system calls are used.

CreateProcess() in windows & fork() in Unix are used to create the process.

ExitProcess() in windows & exit() in Unix are used to terminate the process.

Sometimes, process is to be halted normally using (end) & abnormally using (abort).

A process or job executing one program may want to load & execute another program. Selected program is loaded into memory via exec() system call & then that program is get executed.

2) File Management

There are some system calls which are used to create, manage & control the files & directories/folders.

CreateFile() in Windows & Open() in Unix are used to create the file if it is not present otherwise it opens the existing file.

ReadFile() in Windows & read() in Unix are used to read the content of existing file similarly to write into the file WriteFile() & write() is used in windows & Unix respectively.

After job is over, we need to close that file, for that closeHandle() in windows & close() in Unix are used.

File attributes include the file name, file type, protection codes, accounting information & so on. Atleast 2 system calls, get file attribute & set file attribute are required for this function.

3) Device management

A process may need several resources to complete its job. Resources may be physical resources (main memory, disk drives, I/O devices, etc) & virtual resources (access to files & folders).

Every resources (virtual & physical devices) which are controlled by the OS are considered as devices.

If the resources are available, then they are easily granted to the processes otherwise the process has to wait until sufficient resources are available.

In order to use any resource, process has to request for that device first & after its job over, process has to release it. These functions are similar to open() & close() system calls for files.

Once the device is allocated, then it can be read & write as we can do with files.

4) Information maintenance

Most of the times, we use system calls to exchange the information between the user program & the OS.

But some times, we need some information about the system such as number of current users, OS version, amount of free memory, time, date, etc.

In addition, OS keeps information about all its processes & system calls are used to access this information. Generally, calls are also used to reset the process information like get process attributes & set process attributes.

5) Communication

Sometimes, 2 processes within the same system has to communicate each other & sometimes between 2 processes between 2 computers which are networked. In both the cases, communication channel has to be created in order to carry communication.

This type of communication is called as Interprocess Communication (IPC).

which are the processes used this info, which are the processes users which are the system calls, etc.

Ipc is done through either message-passing or shared memory model.

Each system connected to a network is having an host name. In order to communicate, both the systems should know the host name of each other.

Similarly each process has a process name. System calls, gethostid() & get processid() are used to know the host name & process name.

6) Protection

Protection provides a mechanism for controlling access to the resources provided by a computer system. Since, we are using multiprogramming & multiuser system, protection is must.

System calls that are used for protection purpose are set permission & get permission, which manipulates the permission settings of resources such as files & disks. The allow user & deny user system calls are used to specify whether particular user can be allowed or not to access certain resources.

2.5 System programs

Generally, there are 2 types of software; Application software & System Software.

Application software is computer software designed to help the user to perform specific tasks.

Application software runs on the ~~system~~ top of system software. It interacts with system software which in turn

2.6 Operating-System Design & Implementation

Let us discuss some of the problems/issues/challenges faced by the designers of OS.

- setting the goals
- generality
- portable
- maintained
- compatibility

2.6.1 Design goals

Very first problem in designing an OS is to define goals & its specifications.

Due to the modern technology, there is a rapid growth & revolutionary in both Computer hardware & also software. So, what we are using today that will be outdated in the next decade, so our OS also should change/update.

First of all, design of OS will be affected by the choice of hardware & the type of system: batch, time-shared, single user, multiuser, distributed, real time or general purpose.

Requirements of an OS can be divided into 2 basic groups 1) user goals &
2) System goals

Always user expects, Convenient use of the system,
easy to learn
reliability
Safety &
fastness.

In the same way, people who are designing, creating, maintaining & operating the system also expects some of the things from the OS such as easy to design,
easy to implement,
easy to maintain
flexible
relocable
error free & effeciency.

Another goal is, OS designers really do not have a good idea of how their systems will be used, so designers should consider generality.

Modern OS's are generally designed to be portable, i.e., they have to run on multiple hardware platforms. This is another major goals while designing an OS.

Final one is the frequent need to be backward compatability with some previous OS.

2.6.2 Mechanisms & Policies

Mechanisms determines how to do something whereas policies determines what will be done.

While designing & implementing an OS, one important principle is the separation of policy from mechanism.

Separation of policy & mechanism is important for flexibility. policies are likely to change across place or over time.

Let us take an real-time example. Consider a restaurant. It has the mechanism for serving diners, including tables, plates, waiters, a kitchen full of equipment, agreements with credit card companies & so on. The policy is set by chefs, namely, what is on the menu. If the chef decides that an any item is out & another one is in, this new policy can be handled by the existing mechanism.

Let us consider an OS example, allowing programs to be loaded into the Kernel. The mechanism concerns how they are inserted, how they are linked, what system calls they can make, what system calls can be made on them.

The policy is determining who is allowed to load a programs into the Kernel & which programs.

As mechanisms are changing policies can also change. But, when policies are changing no need to change mechanism. But change in the policies should not affect the mechanism.

2.6.3 Implementation

Once an OS is degined, it must be implemented. Traditionally, OS's have been written in assembly language. Now, they are most commonly written in C & C++.

Advantages of using higher-level language for implementing OS's are

*) Application programs are also written using high-level languages so OS can be written in more compact,

*) easy to debug

*) easy to understand

*) portability – to move to some other hardware.

*) Improvements in Compiler technology will improve the generated code for the entire OS by simple recompilation.

The only possible disadvantage of implementing an OS in a high-level language are reduced speed & increased storage requirements. But this is not a major issue in todays technology.

2.7 Operating-System Structure

We know that, OS is an huge collection of programs. No one person can sit down at a PC & dash-off a serious OS in a few months. All current versions of UNIX exceeds 3 millions lines of code (3×10^6); Windows Vista has over 5 millions lines of Kernel code (& over 50 million lines of total code) No one person can understand 3-5 millions lines of code.

So, it is very complex to design, & to make functioning, to organize, to manage such a ~~big~~ large set of programs.

So, it is better to partition the task into small components rather than having one monolithic system.

Each of these modules/components should be well-defined portion of the system, with carefully defined inputs, outputs & functions.

In this section, we discuss how these components are interconnected & melded into a Kernel.

2.7.1 Simple Structure

Many commercial OS's do not have well-defined structures. MS-DOS is an example of such a system. It was originally designed & implemented by a few people who had no idea that it would become so popular.

MS-DOS was written to provide the most functionality in the least space, so it was not divided into modules carefully. Fig 2.6 shows its structure.

In MS-DOS, interfaces & levels of functionality are not well separated. It does not break the system into

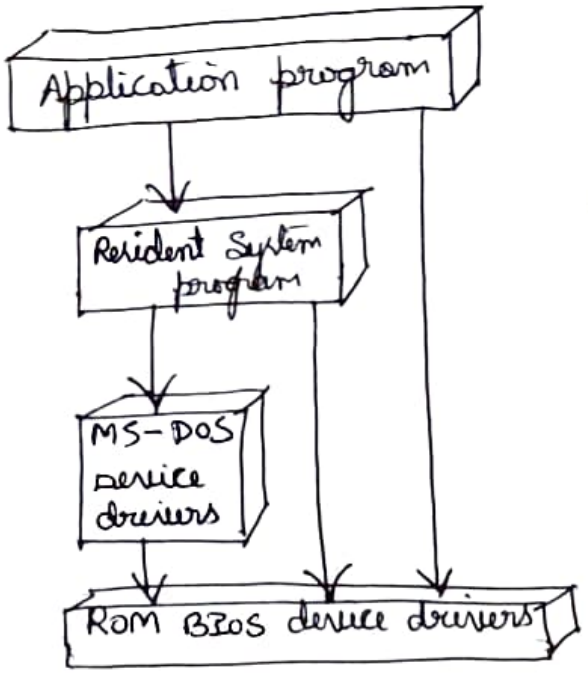

fig 2.6 MS-DOS layer structure

sub-systems.

MS-DOS has no distinction between user mode & Kernel mode. So, it allows all programs directly to access the underlying hardware. Such freedom leaves MS-DOS vulnerable to errant (malicious) programs, causing entire system to crash.

2.7.2 Layered approach

Designers have much control over the computer & the applications that make use of that computer if OS is break down in ~~many~~ many levels/layers/subsystems.

One such method is layered approach in which OS is broken into number of smaller layers, each of which rests on the layer below it & relies solely on the services provided by the next lower layer.

This layered approach allows each layer to be developed & debugged independently, with the assumption that all lower layers have already been debugged & are trusted to deliver proper services.

But the problem is, deciding in what order to place the layer as no layer can call upon the services of any higher layer (just like chicken-&-egg situation).

This layered approach is as shown in fig 2.7.

If OS is divided into modules/layers/subsystems carefully, then designer has more choice

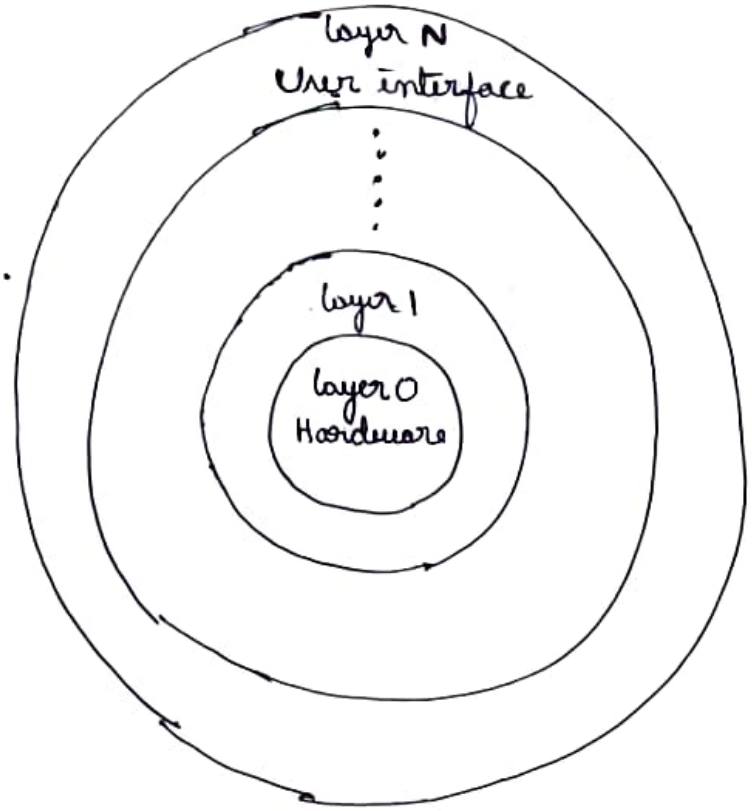

Fig 2.7. A layered OS.

The bottom layer is the hardware & top layer N is the user interface.

A typical OS layer, say layer M consists of data structures & a set of routines that can be invoked by higher level layers. Layer-M in-turn can call operations on lower-level-layers.

Another disadvantage of layered approach is that, as a request for service from a higher layer has to filter through all lower layers before it reaches the hardware, possibly with significant processing at each step.

2.7.3 Microkernels

As the services provided by the OS increases, kernel size also becomes large & difficult to manage.

So, the basic idea behind microkernels is to remove all non-essential services from the kernel & implement them as system applications, thereby making the kernel as small & efficient as possible.

This microkernels provides minimum (very basic) process management & memory management in addition to communication facility.

The main function of microkernel is to provide Communication facility between the client program & the various services that are running in user mode.

Generally, Communication is provided by message passing. Any new services can be added easily in the user space, without affecting the kernel. Thus, we can avoid rebuilding the kernel.

This microkernel also provides more security & reliability, since most services are running in user mode rather than kernel mode.

If any particular service fails then that can be corrected separately without touching rest of the OS.

System expansion can also be easier, because it only involves adding more system applications, not rebuilding a new kernel.

Mach was the first & most widely known microkernel & now forms a major component of Mach OSX.

Windows NT was originally microkernel but suffered from performance problems relative to windows 95.

Another microkernel example is QNX which is Real Time OS (RTOS) for embedded systems.

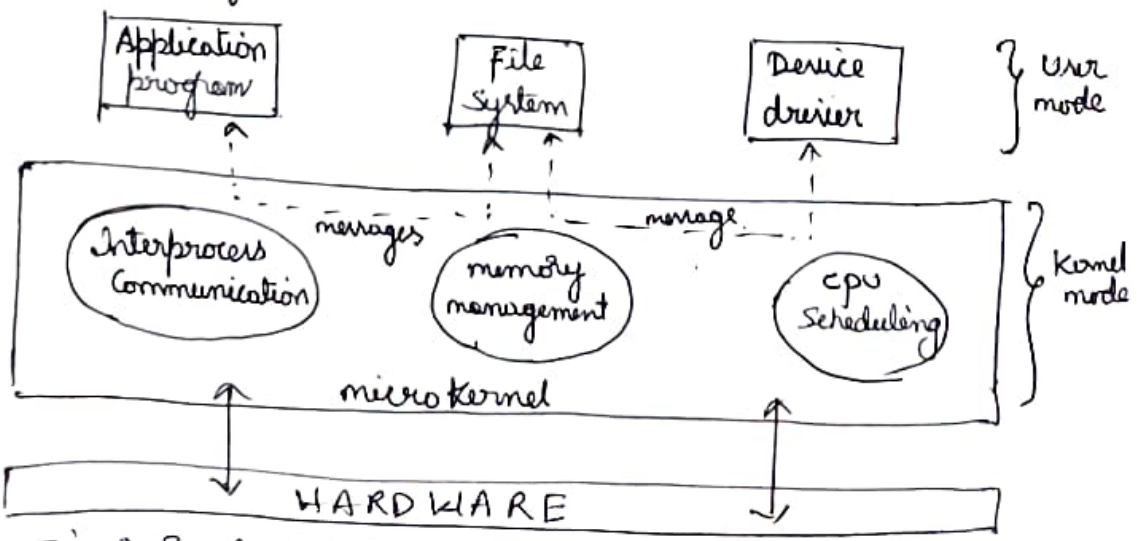

Fig 2.8 Architecture of typical microkernel.

2.7.4 Modules

perhaps, the best current methodology for OS design is using object-oriented programming techniques to create modular kernel.

Here, Kernel has set of core components & it has some links to other services which are created during boot time or run time.

Modules are similar to layers in that each sub-system has clearly defined tasks & interfaces, but any module is free to contact any other module, eliminating the problems of going through multiple intermediary layers.

Here, Kernel is too small similar to microkernels but the Kernel does not have to implement message passing since modules are free to contact directly with each other.

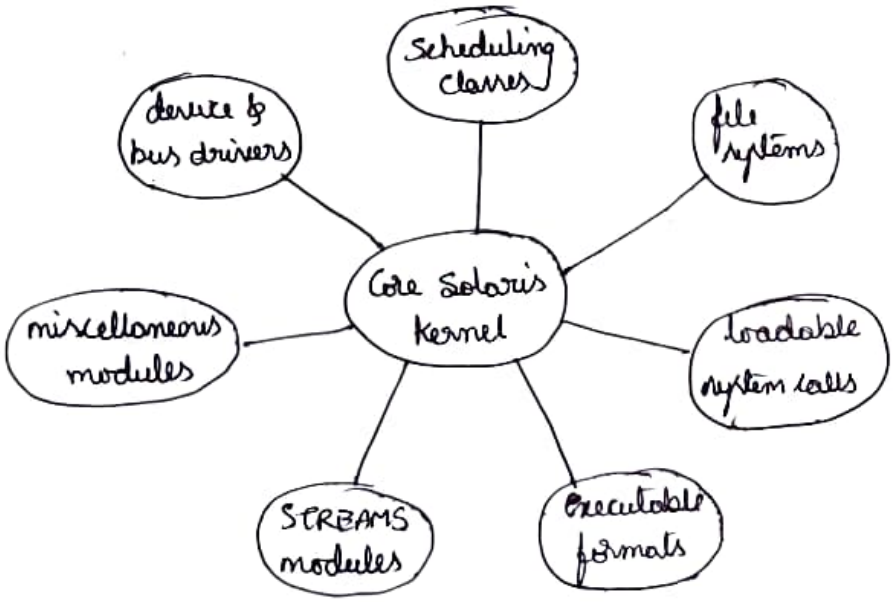

Fig 2.9 Solaris loadable modules.