Module I

INTRODUCTION TO OPERATING SYSTEM

What is an Operating System?

An operating system is a system software that acts as an intermediary between a user of acomputer and the computer hardware.
It is a software that manages the computer hardware.
Os allows the user to execute programs in a convenient and efficient manner.

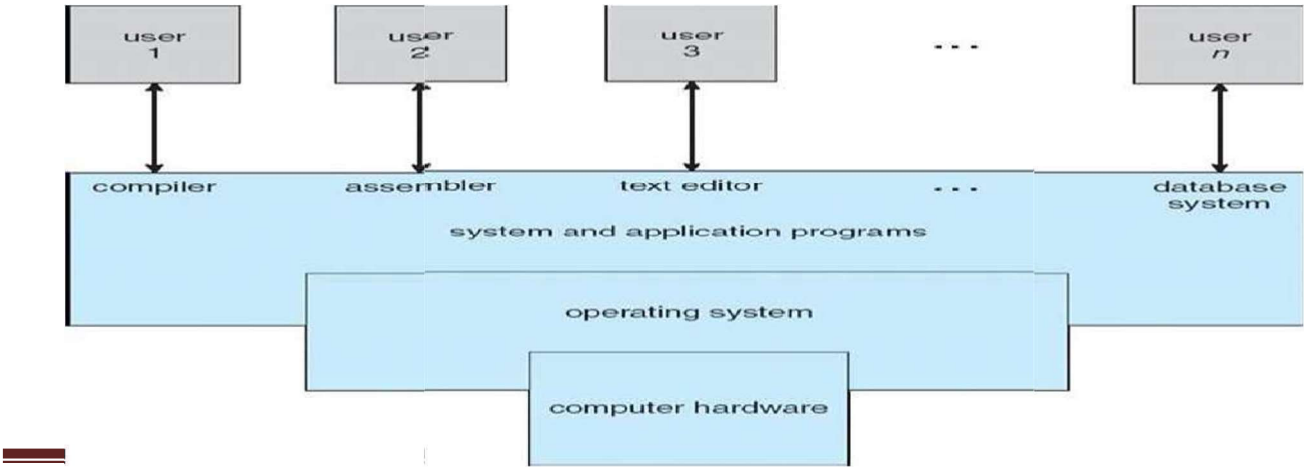
Operating system goals:

- Make the computer system convenient to use. It hides the difficulty in managing the hardware.
- Use the computer hardware in an efficient manner
- Provide and environment in which user can easily interface with computer.
- It is a **resource allocator**

Computer System Structure (Components of Computer System)

Computer system mainly consists of four components-

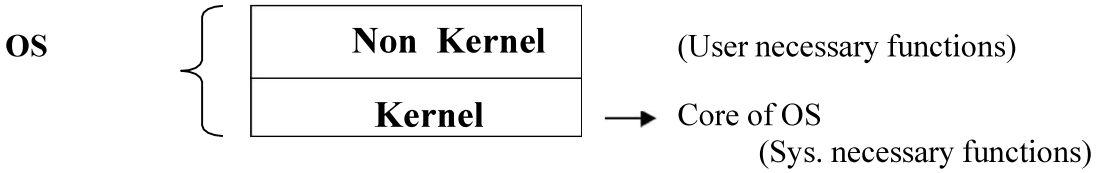
- Hardware – provides basic computing resources
  - ✓ CPU, memory, I/O devices
- Operating system
  - ✓ Controls and coordinates use of hardware among various applications andusers
- Application programs – define the ways in which the system resources are used to solve the computing problems of the users
  - ✓ Word processors, compilers, web browsers, database systems, videogames
- Users
  - ✓ People, machines, other computers



The basic hardware components comprises of CPU, memory, I/O devices. The application program uses these components. The OS controls and co-ordinates the use of hardware, among various application programs (like compiler, word processor etc.) for various users.

The OS allocates the resources among the programs such that the hardware is efficiently used.

The operating system is the program running at all the times on the computer. It is usually called as the kernel.



Kernel functions are used always in system, so always stored in memory. Non kernel functions are stored in hard disk, and it is retrieved whenever required.

Views of OS

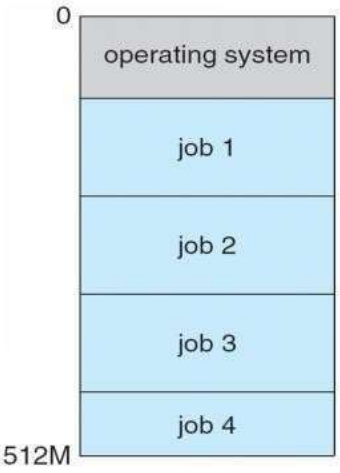
Operating System can be viewed from two viewpoints–
User views & System views

1. User Views:-
   The user's view of the operating system depends on the type of user.
   i. If the user is using **standalone** system, then OS is designed for ease of use and high performances. Here resource utilization is not given importance.
   ii. If the users are at different **terminals** connected to a mainframe or minicomputers, by sharing information and resources, then the OS is designed to maximize resource utilization. OS is designed such that the CPU time, memory and i/o are used efficiently and no single user takes more than the resource allotted to them.
   iii. If the users are in **workstations**, connected to networks and servers, then the user have a system unit of their own and shares resources and files with other systems. Here the OS is designed for both ease of use and resource availability (files).
   iv. Users of **hand held** systems, expects the OS to be designed for ease of use and performance per amount of battery life.
   v. Other systems like embedded systems used in home devies (like washing m/c) & automobiles do not have any user interaction.LEDs to

2. System Views:-
   Operating system can be viewed as a resource allocator and control program.
   a) Resource allocator - The OS acts as a manager of hardware and software resources. CPU time, memory space, file-storage space, I/O devices, shared files etc. are the different resources required during execution of a program. There can be conflicting request for these resources by different programs running in same system. The OS assigns the resources to the requesting program depending on the priority.
   b) Control Program – The OS is a control program and manage the execution of user program to prevent errors and improper use of the computer.

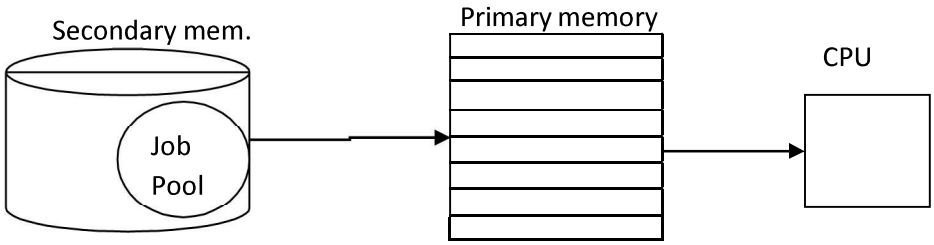
Operating-System Structure

One of the most important aspects of operating systems is the ability to multiprogram. A single user cannot keep either the CPU or the I/O devices busy at all times. **Multiprogramming** increases CPU utilization by organizing jobs, so that the CPU always has one to execute.



The operating system keeps several jobs in memory simultaneously as shown in figure. This set of jobs is a subset of the jobs kept in the job pool. Since the number of jobs that can be kept simultaneously in memory is usually smaller than the number of jobs that can be kept in the job pool(in secondary memory). The operating system picks and begins to execute one of the jobs in memory. Eventually, the job may have to wait for some task, such as an I/O operation, to complete. In a non-multiprogrammed system, the CPU would sit idle. In a multiprogrammed system, the operating system simply switches to, job. When *that* job needs to wait, the CPU is switched to *another* job, andso on.

Eventually, the first job finishes waiting and gets the CPU back. Thus the CPU is never idle.

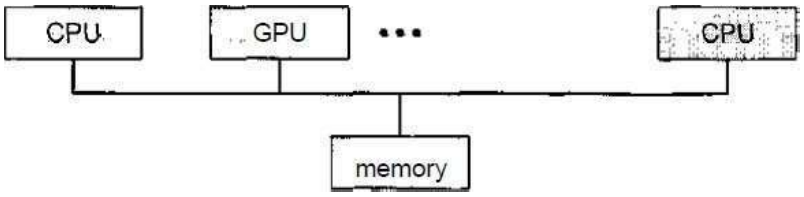


Multiprogrammed systems provide an environment in which the various system resources (for example, CPU, memory, and peripheral devices) are utilized effectively, but they do not provide for user interaction with the computer system.

In **Time sharing** (or **multitasking) systems,** a single 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. The user feels that all the programs are being executed at the same time. Time sharing requires an **interactive** (or **hands-on) computer system,** which provides direct communication between the user and the system. The user gives instructions to theoperating system or to a program directly, using a input device such as a keyboard or a mouse, and waits for immediate results on an output device. Accordingly, the **response time** should be short—typically less than one second.

A time-shared operating system allows many users to share the computer simultaneously. 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 his use only, even though it is being shared among many users.

A **multiprocessor system** is a computer system having two or more CPUs within a single computer system, each sharing main memory and peripherals. Multiple programs are executed by multiple processors parallel.



Operating-System Operations

Modern operating systems are **interrupt driven.** If there are no processes to execute, no I/O devices to service, and no users to whom to respond, an operating system will sit quietly, waiting for something to happen. Events are signaled by the occurrence of an interrupt or a trap. A **trap (or** an **exception)** is a software-generated interrupt. For each type of interrupt, separate segments of code in the operating system determine what action should be taken. An interrupt service routine is provided that is responsible for dealing with the interrupt.

a) Dual-Mode Operation

Since the operating system and the user programs share the hardware and software resources of the computer system, it has to be made sure that an error in a user program cannot cause problems to other programs and the Operating System running in the system.

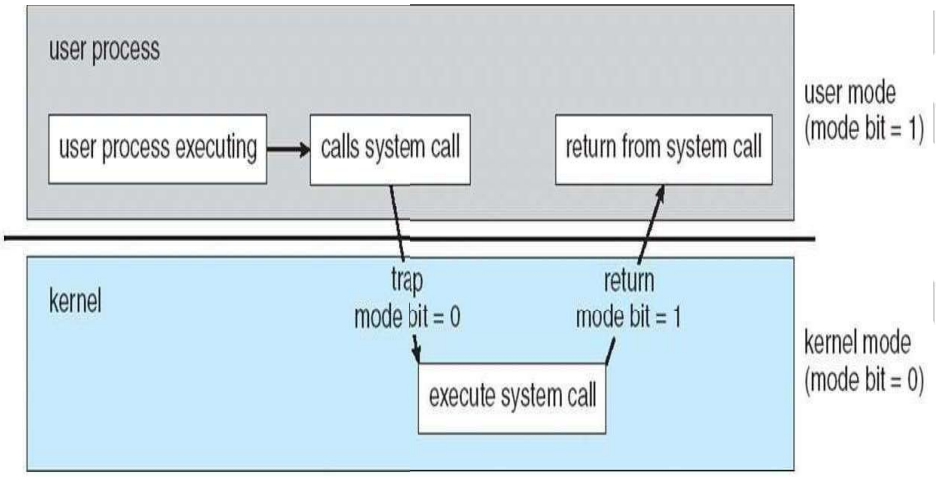
The approach taken is to use a hardware support that allows us to differentiate among various modes of execution.

The system can be assumed to work in two separate **modes** of operation:

- **user mode** and
- **kernel mode** (**supervisor mode, system mode, or privileged mode).**

A hardware bit of the computer, called the **mode bit**, is used to indicate the current mode: kernel (0) or user (1). With the mode bit, we are able to distinguish between a task that is executed by the operating system and one that is executed by the user.

When the computer system is executing a user application, the system is in user mode. When a user application requests a service from the operating system (via a system call), the transition from user to kernel mode takes place.



At system boot time, the hardware starts in kernel mode. The operating system is then loaded and starts user applications in user mode. Whenever a trap or interrupt occurs, the hardware switches from user mode to kernel mode (that is, changes the mode bit from 1 to 0). Thus, whenever the

operating system gains control of the computer, it is in kernel mode.

The dual mode of operation provides us with the means for protecting the operating system from errant users—and errant users from one another.

The hardware allows privileged instructions to be executed only in kernel mode. If an attempt is made to execute a privileged instruction in user mode, the hardware does not execute the instruction but rather treats it as illegal and traps it to the operating system. The instruction to switch to user mode is an example of a privileged instruction.

Initial control is within the operating system, where instructions are e ecuted in kernel mode. When control is given to a user application, the mode is set to user mode. Eventually, control is switched back to the operating system via an interrupt, a trap, or a system call.

b) Timer

Operating system uses timer to control the CPU. A user program cannot hold CPU for a long time, this is prevented with the help of timer.

A timer can be set to interrupt the computer after a specified period. The period may be **fixed** (for example, 1/60 second) or **variable** (for example, from 1 millisecond to 1 second).

Fixed timer – After a fixed time, the process under execution is interrupted.

Variable timer – Interrupt occurs after varying interval. This is implemented using a fixed-rate clock and a counter. The operating system sets the counter. Every time the clock ticks, the counter is decremented. When the counter reaches 0, an interrupt occurs.

Before changing to the user mode, the operating system ensures that the timer is set to interrupt. If the timer interrupts, control transfers automatically to the operating system, which may treat the interrupt as a fatal error or may give the program more time.

2.1 Operating-System Services

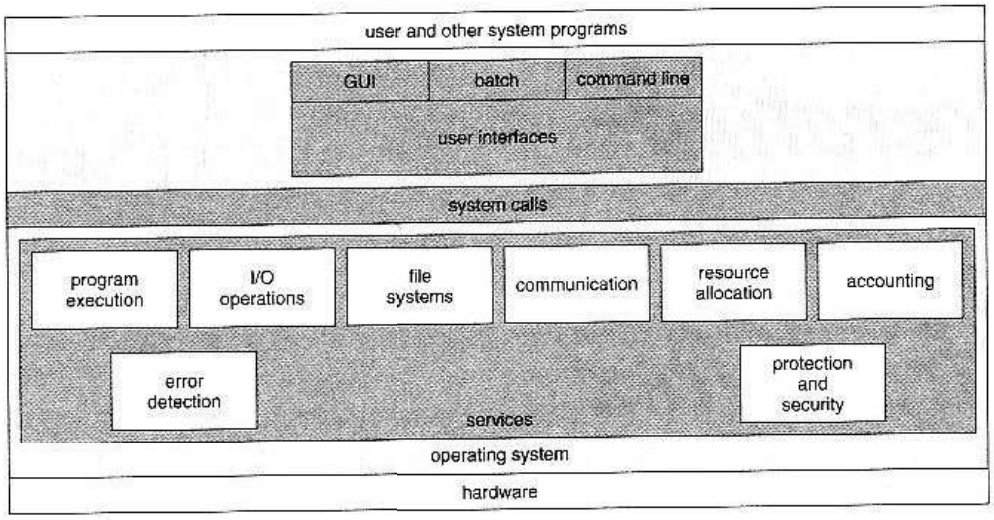


Figure 2.1 A view of operating system services.

An operating system provides an environment for the execution of programs.

It provides certain services to programs and to the users of those programs.

OS provide services for the users of the system, including:

- **User Interfaces -** Means by which users can issue commands to the system. Depending on the operating system these may be a **command-line interface** (e.g. sh, csh, ksh, tcsh, etc.), a **Graphical User Interface** (e.g. Windows, X-Windows, KDE, Gnome, etc.), or a **batch command systems**. In Command Line Interface(CLI)- commands are given to the system. In Batch interface – commands and directives to control these commands are put in a file and then the file is executed. In GUI systems- windows with pointing device to get inputs and keyboard to enter the text.
- **Program Execution -** The OS must be able to load a program into RAM, run the program, and terminate the program, either normally or abnormally.
- **I/O Operations -** The OS is responsible for transferring data to and from I/O devices, including keyboards, terminals, printers, and files. For specific devices, special functions are provided(device drivers) by OS.

- **File-System Manipulation** – Programs need to read and write files or directories. The services required to create or delete files, search for a file, list the contents of a file and change the file permissions are provided by OS.
- **Communications -** Inter-process communications, IPC, either between processes running on the same processor, or between processes running on separate processors or separate machines. May be implemented by using the service of OS- like shared memory or message passing.
- **Error Detection -** Both hardware and software errors must be detected and handled appropriately by the OS. Errors may occur in the CPU and memory hardware (such as power failure and memory error), 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).

OS provide services for the efficient operation of the system, including:

- **Resource Allocation –** Resources like CPU cycles, main memory, storage space, and I/O devices must be allocated to multiple users and multiple jobs at the same time.
- **Accounting –** There are services in OS to keep track of system activity and resource usage, either for billing purposes or for statistical record keeping that can be used to optimize future performance.
- **Protection and Security** – The owners of information(file) in multiuser or networked computer system may want to control the use of that information. When several separate processes execute concurrently, one process should not interfere with other or with OS. Protection involves ensuring that all access to system resources is controlled. Security of the system from outsiders must also be done, by means of a password.

2.2 User Operating-System Interface

There are several ways for users to interface with the operating system.

1) Command-line interface, or command interpreter, allows users to directly enter commands to be performed by the operating system.
2) Graphical user interface(GUI), allows users to interface with the operating system using pointer device and menu system.

Command Interpreter

Command Interpreters are used to give commands to the OS. There are multiple command interpreters known as shells. In UNIX and Linux systems, there are several different shells, like the *Bourne shell, C shell, Bourne-Again shell, Korn shell,* and others.

The main function of the command interpreter is to get and execute the user-specified command. Many of the commands manipulate files: create, delete, list, print, copy, execute, and so on.

The commands can be implemented in two general ways-

1) The command interpreter itself contains the code to execute the command. For example, a command to delete a file may cause the command interpreter to jump to a particular section of its code that sets up the parameters and makes the appropriate system call.

2) The code to implement the command is in a function in a separate file. The interpreter searches for the file and loads it into the memory and executes it by passing the parameter. Thus by adding new functions new commands can be added easily to the interpreter without disturbing it.

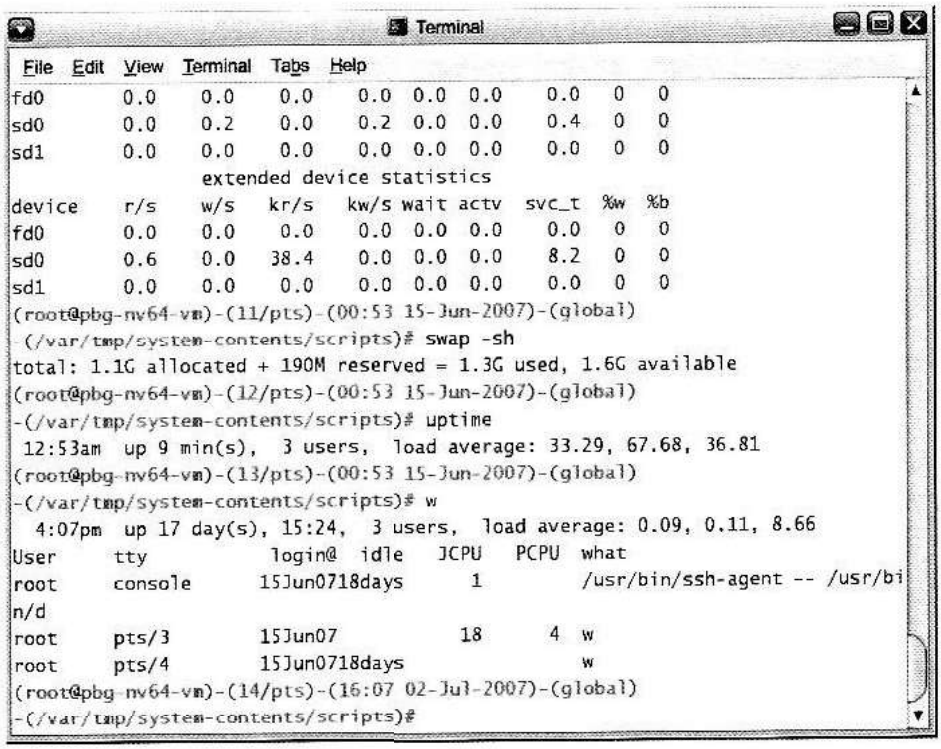


Figure 2.2 The Bourne shell command interpreter in Solaris 10.

Graphical User Interface, GUI

Another way of interfacing with the operating system is through a user friendly graphical user interface, or GUI. Here, rather than entering commands directly via a command-line interface, users employ a mouse-based window and menu system. The user moves the mouse to position its

pointer on images, or icons on the screen (the desktop) that represent programs, files, directories, and system functions. Depending on the mouse pointer's location, clicking a button on the mouse can invoke a program, select a file or directory-known as a folder-or pull down a menu that contains commands.

Graphical user interfaces first appeared on the Xerox Alto computer in 1973.

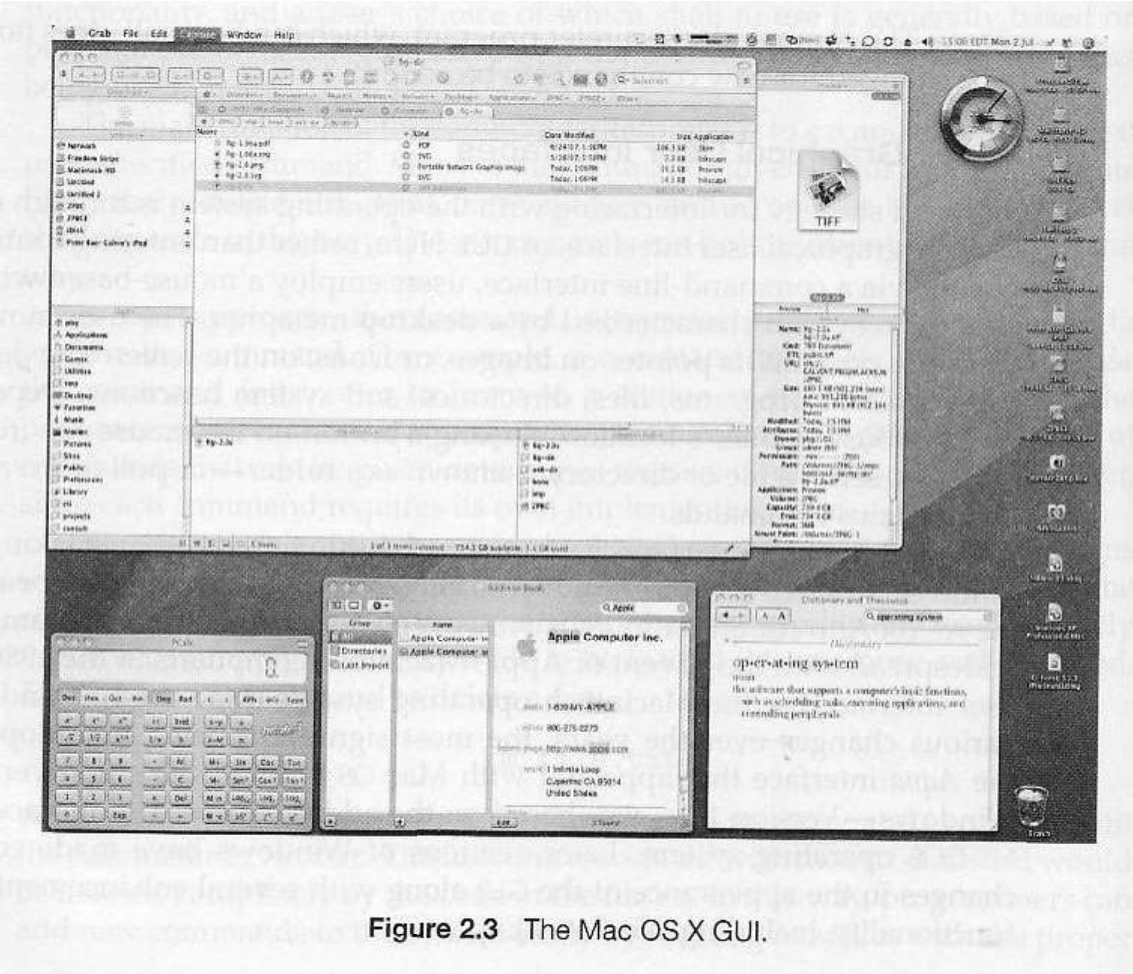


Figure 2.3 The Mac OS X GUI.

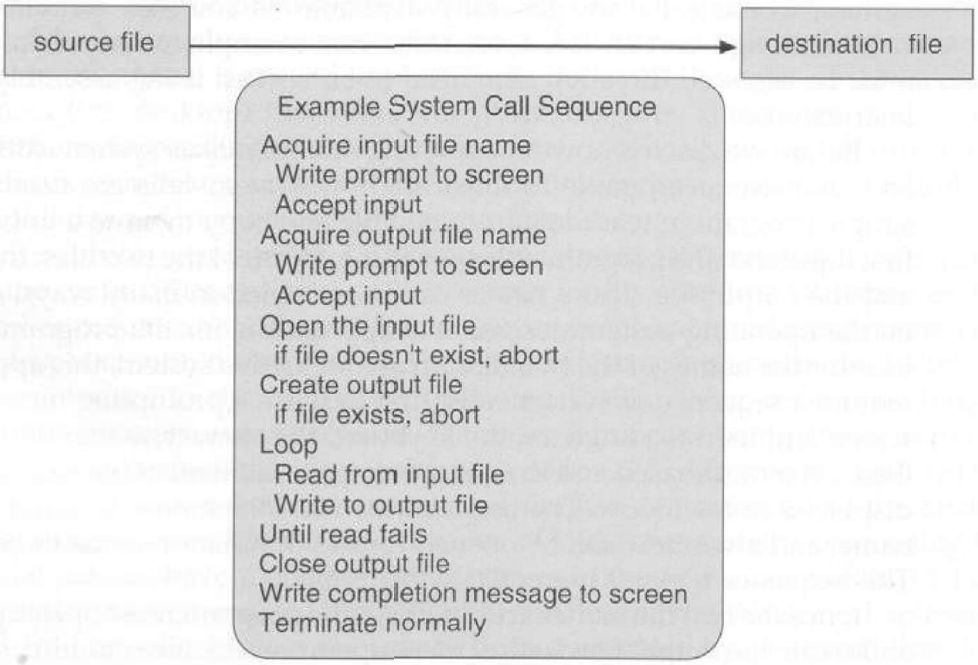
Most modern systems allow individual users to select their desired interface, and to customize its operation, as well as the ability to switch between different interfaces as needed.

2.3 System Calls

- System calls is a means to access the services of the operating system.

- Generally written in C or C++, although some are written in assembly for optimal performance.

- The below figure illustrates the sequence of system calls required to cop a file content from one file(input file) to another file (output file).



There are number of system calls used to finish this task. The first system call is to write a message on the screen (monitor). Then to accept the input filename. Then another system call to write message on the screen, then to accept the output filename. When the program tries to open the input file, it may find that there is no file of that name or that the file is protected against access. In these cases, the program should print a message on the console(another system call) and then terminate abnormally (another system call) and create a new one (another system call).

Now that both the fileas are opened, we enter a loop that reads from the input file(another system call) and writes to output file (another system call).

Finally, after the entire file is copied, the program may close both files (another system call), write a message to the console or window(system call), and finally terminate normally (final system call).

- Most programmers do not use the low-level system calls directly, but instead use an "Application Programming Interface", API.

- The APIs instead of direct system calls provides for greater program portability between different systems. The API then makes the appropriate system calls through the system call interface, using a system call table to access specific numbered system calls, as shown in Figure 2.6.

- Each system call has a specific numbered system call. The system call table (consisting of system call number and address of the particular service) invokes a particular service routine for a specific system call.
- The caller need know nothing about how the system call is implemented or what it does during execution.

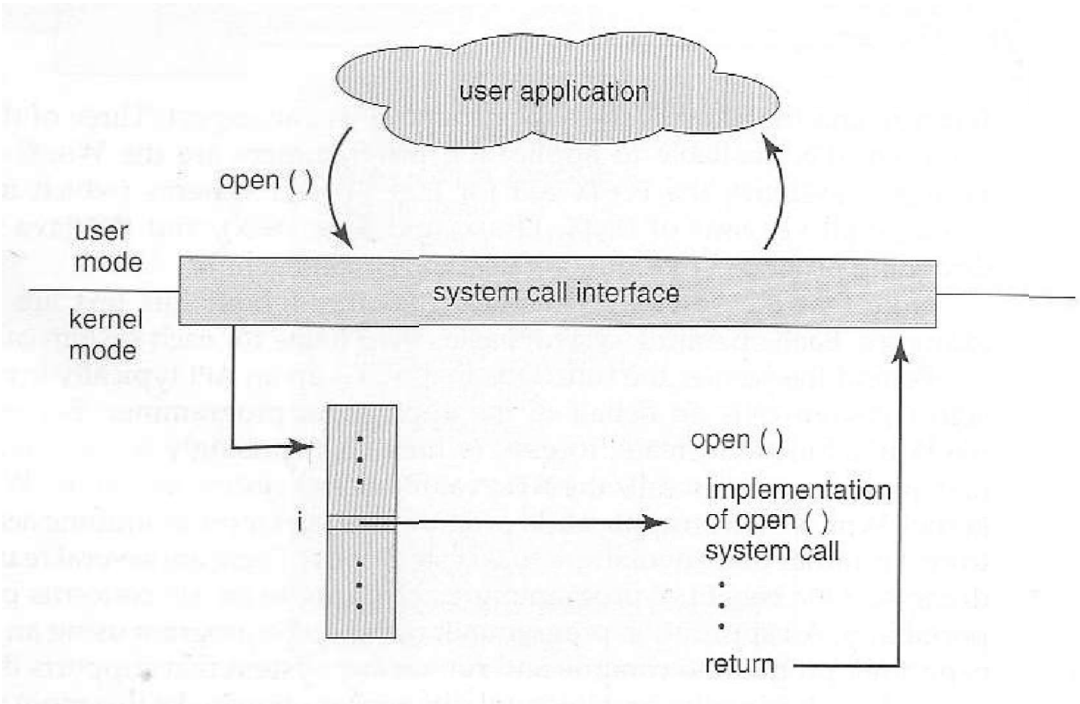


Figure 2.6 The handling of a user application invoking the open() system call.

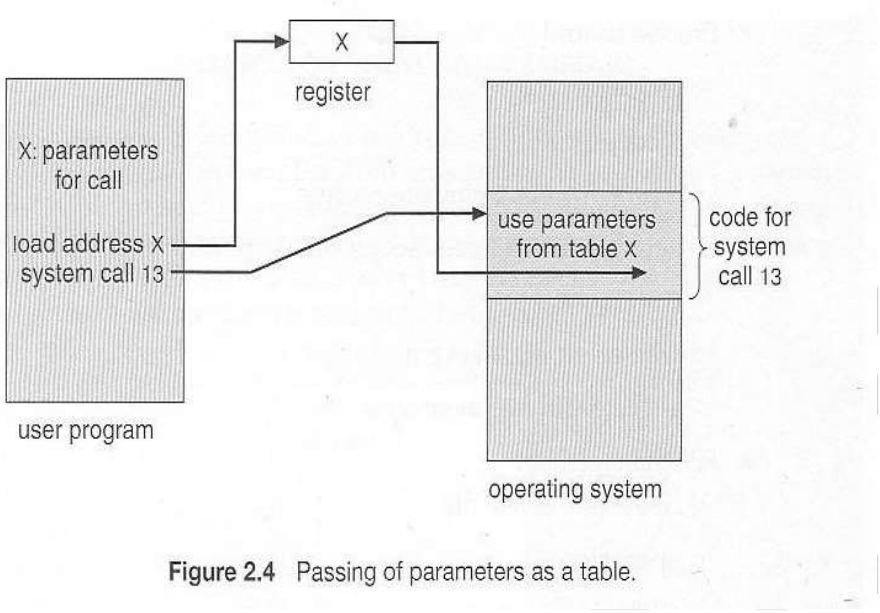


Figure 2.4 Passing of parameters as a table.

Three general methods used to pass parameters to OS are –

- To pass parameters in registers
- If parameters are large blocks, address of block (where parameters are stored in memory) is sent to OS in the register. (Linux & Solaris).
- Parameters can be pushed onto the stack by program and popped off the stack by OS.

2.3.1 Types of System Calls

The system calls can be categorized into six major categories:

- Process Control
- File management
- Device management
- Information management
- Communications
- Protection

- Process control
  - end, abort
  - load, execute
  - create process, terminate process
  - get process attributes, 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

EXAMPLES OF WINDOWS AND UNIX SYSTEM CALLS

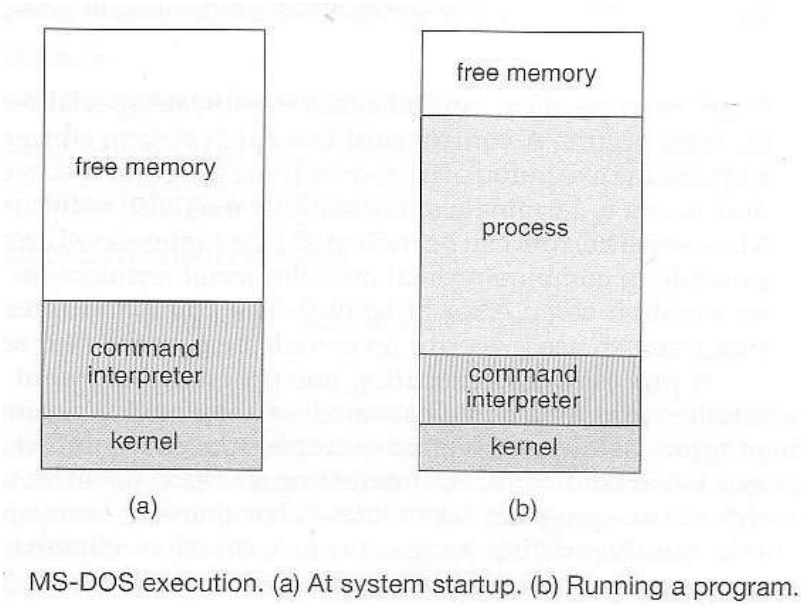
	Windows	Unix
Process Control	CreateProcess()	fork()
	ExitProcess()	exit()
	WaitForSingleObject()	wait()
File Manipulation	CreateFile()	open()
	ReadFile()	read()
	WriteFile()	write()
	CloseHandle()	close()
Device Manipulation	SetConsoleMode()	ioctl()
	ReadConsole()	read()
	WriteConsole()	write()
Information Maintenance	GetCurrentProcessID()	getpid()
	SetTimer()	alarm()
	Sleep()	sleep()
Communication	CreatePipe()	pipe()
	CreateFileMapping()	shmget()
	MapViewOfFile()	mmap()
Protection	SetFileSecurity()	chmod()
	InitlializeSecurityDescriptor()	umask()
	SetSecurityDescriptorGroup()	chown()

a) Process Control

- Process control system calls include end, abort, load, execute, create process, terminate process, get/set process attributes, wait for time or event, signal event, and allocate and free memory.
- Processes must be created, launched, monitored, paused, resumed, and eventually stopped.

- When one process pauses or stops, then another must be launched or resumed
- Process attributes like process priority, max. allowable execution time etc. are set and retrieved by OS.
- After creating the new process, the parent process may have to wait (wait time), or wait for an event to occur(wait event). The process sends back a signal when the event has occurred (signal event).

- In DOS, the command interpreter loaded first.Then loads the process and transfers control to it. The interpreter does not resume until the process has completed, as shown in Figure 2.10:



MS-DOS execution. (a) At system startup. (b) Running a program.

Figure 2.10

- Because UNIX is a multi-tasking system, the command interpreter remains completely resident when executing a process, as shown in Figure 2.11 below.
  - The user can switch back to the command interpreter at any time, and can place the running process in the background even if it was not originally launched as a background process.
  - In order to do this, the command interpreter first executes a "fork" system call, which creates a second process which is an exact duplicate (clone) of the original command interpreter. The original process is known as the parent, and the cloned process is known as the child, with its own unique process ID and parent ID.
  - The child process then executes an "exec" system call, which replaces its code with that of the desired process.
  - The parent (command interpreter) normally waits for the child to complete before issuing a new command prompt, but in some cases it can also issue a new prompt right away, without waiting for the child process to complete. (The child is then said to be running "in the background", or "as a background process".)

b) File Management

The file management functions of OS are –

- File management system calls include create file, delete file, open, close, read, write, reposition, get file attributes, and set file attributes.
- After **creating** a file, the file is **opened**. Data is **read** or **written** to a file.
- The file pointer may need to be **repositioned** to a point.
- The file **attributes** like filename, file type, permissions, etc. are set and retrieved using system calls.
- These operations may also be supported for directories as well as ordinary files.

c) Device Management

- Device management system calls include **request device**, **release** device, **read**, **write**, **reposition**, **get/set** device attributes, and logically **attach** or **detach** devices.
- When a process needs a resource, a request for resource is done. Then the control is granted to the process. If requested resource is already attached to some other process, the requesting process has to wait.
- In multiprogramming systems, after a process uses the device, it has to be returned to OS, so that another process can use the device.
- Devices may be physical (e.g. disk drives), or virtual / abstract (e.g. files, partitions, and RAM disks).

d) Information Maintenance

- Information maintenance system calls include calls to get/set the time, date, system data, and process, file, or device attributes.
- These system calls care used to transfer the information between user and the OS. Information like current time & date, no. of current users, version no. of OS, amount of free memory, disk space etc. are passed from OS to the user.

e) Communication

- Communication system calls create/delete communication connection, send/receive messages, transfer status information, and attach/detach remote devices.
- The **message passing** model must support calls to:
    - Identify a remote process and/or host with which to communicate.
    - Establish a connection between the two processes.
    - Open and close the connection as needed.

  - Transmit messages along the connection.
  - Wait for incoming messages, in either a blocking or non-blocking state.
  - Delete the connection when no longer needed.
- The **shared memory** model must support calls to:
  - Create and access memory that is shared amongst processes (and threads.)
  - Free up shared memory and/or dynamically allocate it as needed.
- Message passing is simpler and easier, (particularly for inter-computer communications), and is generally appropriate for small amounts of data. It is easy to implement, but there are system calls for each read and write process.
- Shared memory is faster, and is generally the better approach where large amounts of data are to be shared. This model is difficult to implement, and it consists of only few system calls.

f) Protection

- Protection provides mechanisms for controlling which users / processes have access to which system resources.
- System calls allow the access mechanisms to be adjusted as needed, and for non-priveleged users to be granted elevated access permissions under carefully controlled temporary circumstances.

2.4 System Programs

A collection os programs that provide a convenient environment for program development and execution (other than OS) are called system programs or system utilities.

- It is not a part of the kernel or command interpreters.
- System programs may be divided into five categories:
  - **File management -** programs to create, delete, copy, rename, print, list, and generally manipulate files and directories.
  - **Status information -** Utilities to check on the date, time, number of users,processes running, data logging, etc. System **registries** are used to store and recall configuration information for particular applications.
  - **File modification -** e.g. text editors and other tools which can change file contents.
  - **Programming-language support -** E.g. Compilers, linkers, debuggers, profilers, assemblers, library archive management, interpreters for common languages, and support for make.
  - **Program loading and execution -** loaders, dynamic loaders, overlay loaders, etc., as well as interactive debuggers.
  - **Communications -** Programs for providing connectivity between processes and users, including mail, web browsers, remote logins, file transfers, and remote command execution.

2.5 Operating-System Design and Implementation

2.5.1 Design Goals

Any system to be designed must have its own goals and specifications. Similarly the OS to be built will have its own goals depending on the type of system in which it will be used, the type of hardware used in the system etc.

- **Requirements** define properties which the finished system must have, and are a necessary steps in designing any large complex system. The requirements may be of two basic groups:

    1. User goals (User requirements)
    2. System goals (system requirements)

    - **User requirements** are features that users care about and understand like system should be convenient to use, easy to learn,reliable, safe and fast.
    - **System requirements** are written for the developers, ie. People who design the OS. Their requirements are like easy to design, implement and maintain, flexible, reliable, error free and efficient.

2.5.2 Mechanisms and Policies

- Policies determine *what* is to be done. Mechanisms determine *how* it is to be implemented.
- Example: in timer, counter and decrementing counter is the mechanism and deciding how long the time has to be set is the policies.
- Policies change overtime. In the worst case, each change in policy would require a change in the underlying mechanism.
- If properly separated and implemented, policy changes can be easily adjusted without re-writing the code, just by adjusting parameters or possibly loading new data / configuration files.

2.5.3 Implementation

- Traditionally OS were written in assembly language.
- In recent years, Os are written in C, or C++. Critical sections of code are still written in assembly language.
- The first OS that was not written in assembly language was the Master Control Program (MCP).
- The advantages of using a higher-level language for implementing operating systems are: The code can be written faster, more compact, easy to port to other systems and is easier to understand and debug.
- The only disadvantages of implementing an operating system in a higher-level language are reduced speed and increased storage requirements.

2.7 Operating-System Structure

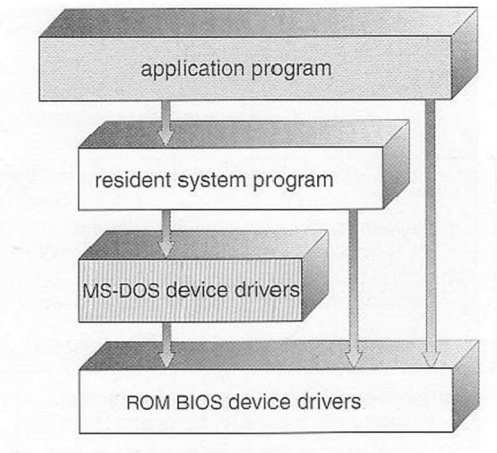
OS structure must be carefully designed. The task of OS is divided into small components and then interfaced to work together.

2.7.1 Simple Structure

Many operating systems do not have well-defined structures. They started as small, simple, and limited systems and then grew beyond their original scope. Eg: MS-DOS. In MS-DOS, the interfaces and levels of functionality are not well separated. Application programs can access basic I/O routines to write directly to the display and disk drives. Such freedom leaves MS-DOS in bad state and the entire system can crash down when user programs fail.

UNIX OS consists of two separable parts: the kernel and the system programs. The kernel is further separated into a series of interfaces and device drivers. The kernel provides the file system, CPU scheduling, memory management, and other operating-system functions through system calls.

MS-DOS Layer Structure



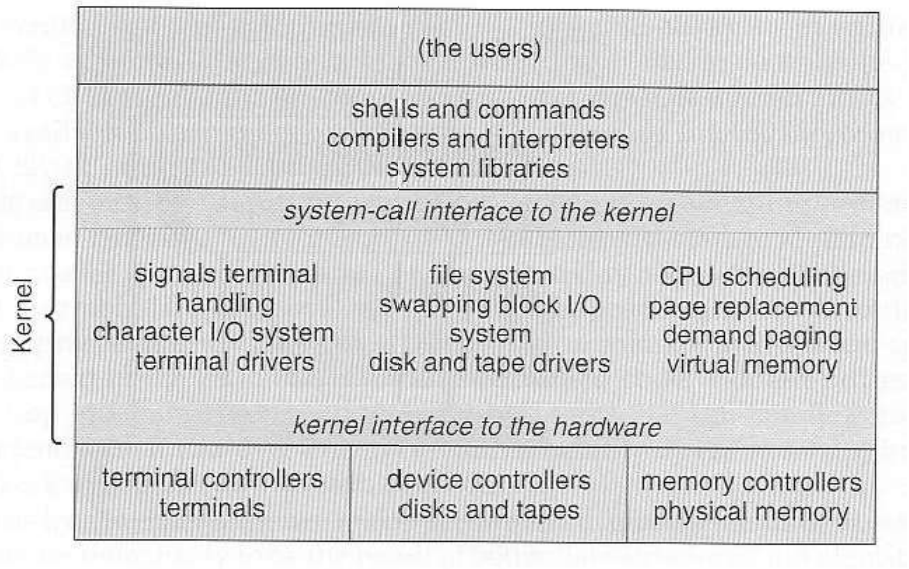


Figure 2.13 UNIX System Structure

2.7.2 Layered Approach

- The OS is broken into number of layers (levels). Each layer rests on the layer below it, and relies on the services provided by the next lower layer.

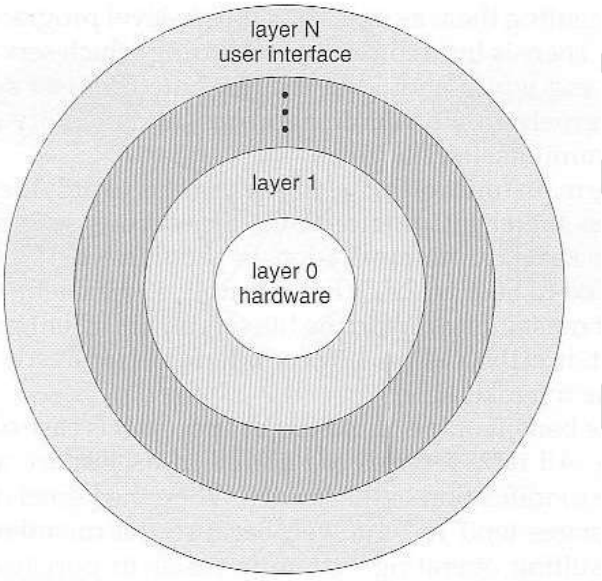
- Bottom layer(layer 0) is the hardware and the topmost layer is the user interface.
- A typical layer, consists of data structure and routines that can be invoked by higher-level layer.

Advantage of layered approach is simplicity of construction and debugging.

The layers are selected so that each uses functions and services of only lower-level layers. So simplifies debugging and system verification. The layers are debugged one by one from the lowest and if any layer doesn't work, then error is due to that layer only, as the lower layers are already debugged. Thus the design and implementation is simplified.

A layer need not know how its lower level layers are implemented. Thus hides the operations from higher layers.

Figure 2.14 A layered Operating System



Disadvantages of layered approach:

- The various layers must be appropriately defined, as a layer can use only lower level layers.
- Less efficient than other types, because any interaction with layer 0 required from top layer. The system call should pass through all the layers and finally to layer 0. This is an overhead.

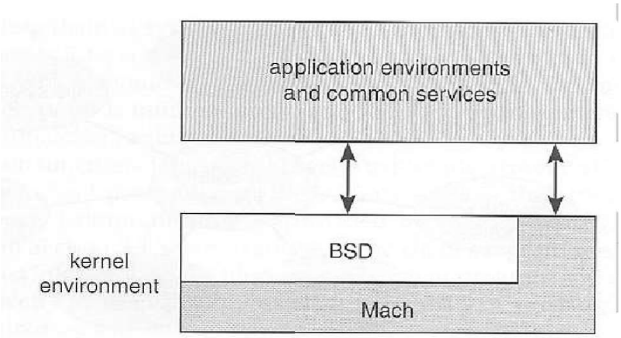


Figure 2.14 The Mac OS X structure.

2.7.3 Modules

- Modern OS development is object-oriented, with a relatively small core kernel and a set of ***modules*** which can be linked in dynamically.
- Modules are similar to layers in that each subsystem has clearly defined tasks and interfaces, but any module is free to contact any other module, eliminating the problems of going through multiple intermediary layers.
- The kernel is relatively small in this architecture, similar to microkernels, but the kernel does not have to implement message passing since modules are free to contact each other directly. Eg: Solaris, Linux and MacOSX.

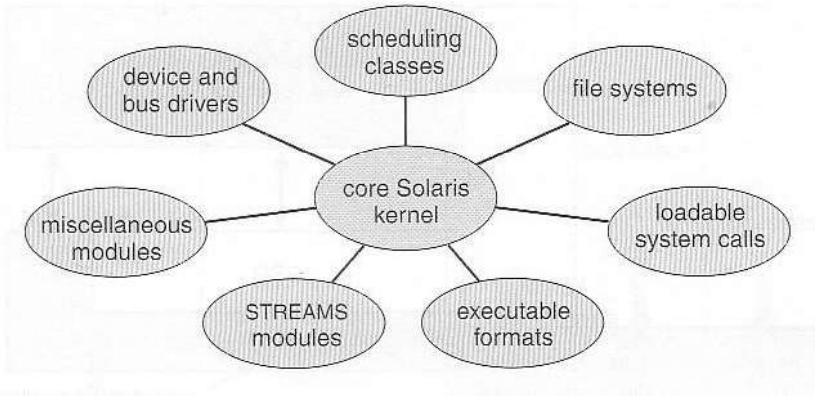


Figure 2.15 Solaris loadable modules

- The Max OSX architecture relies on the Mach microkernel for basic system management services, and the BSD kernel for additional services. Application services and dynamically loadable modules (kernel extensions) provide the rest of the OS functionality.
- Resembles layered system, but a module can call any other module.
- Resembles microkernel, the primary module has only core functions and the knowledge of how to load and communicate with other modules.