CSE18R273/ OPERATING SYSTEMS

UNIT I INTRODUCTION TO OPERATING SYSTEMS

Introduction: Concept of Operating Systems, Generations of Operating systems, Types of Operating Systems, OS Services, System Calls, Structure of an OS - Layered, Monolithic, Microkernel Operating Systems, Concept of Virtual Machine. Case study on UNIX and WINDOWS Operating System

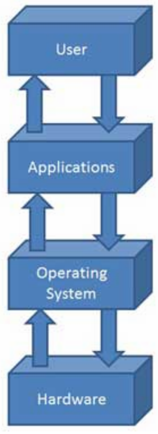
Reference:

1. Abraham Silberschatz, Greg Gagne, and Peter Baer Galvin, "Operating System Concepts, Ninth Edition ", Chapter 1

Operating System

The Operating System is a program with the following features −

- An operating system is a program that acts as an **interface between the software and the computer hardware.**
- It is an integrated set of specialized programs used to **manage overall resources and operations of the computer.**
- It is a specialized software that **controls and monitors the execution of all other programs** that reside in the computer, including **application programs** and other **system software.**



Objectives of Operating System

The objectives of the operating system are −

- To make the computer system **convenient to use** in an efficient manner.
- To **hide the details of the hardware resources** from the users.
- To provide users a convenient **interface to use** the computer system.
- To act as an **intermediary between the hardware and its users**, making it easier for the users to access and use other resources.
- To **manage the resources** of a computer system.
- To keep **track** of who is using which resource, granting resource requests, and mediating conflicting requests from different programs and users.
- To provide efficient and fair sharing of resources among users and programs.

Characteristics of Operating System

Here is a list of some of the most prominent characteristic features of Operating Systems −

- **Memory Management** − Keeps track of the primary memory, i.e. what part of it is in use by whom, what part is not in use, etc. and allocates the memory when a process or program requests it.
- **Processor Management** − Allocates the processor (CPU) to a process and deallocates the processor when it is no longer required.
- **Device Management** − Keeps track of all the devices. This is also called I/O controller that decides which process gets the device, when, and for how much time.
- **File Management** − Allocates and de-allocates the resources and decides who gets the resources.
- **Security** − Prevents unauthorized access to programs and data by means of passwords and other similar techniques.
- **Job Accounting** − Keeps track of time and resources used by various jobs and/or users.

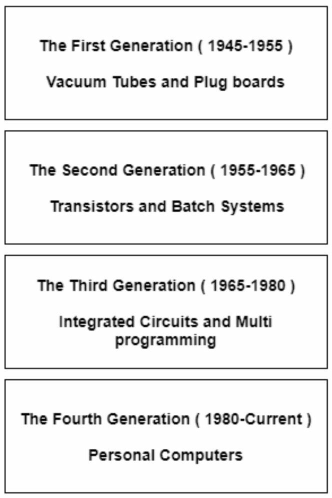
System Boot

- The process of starting a computer by loading the **kernel** is known as **booting** the system
- Small piece of code – **bootstrap loader**, locates the kernel, loads it into memory, and starts it

---

Generations of Operating Systems

Operating Systems have evolved over the years. So, their evolution through the years can be mapped using generations of operating systems. There are four generations of operating systems. These can be described as follows:


The First Generation (1945-1955)
Vacuum Tubes and Plug boards
The Second Generation (1955-1965)
Transistors and Batch Systems
The Third Generation (1965-1980)
Integrated Circuits and Multi programming
The Fourth Generation (1980-Current)
Personal Computers

OPERATING SYSTEM GENERATIONS

The First Generation (1945 - 1955): Vacuum Tubes and Plug boards

Digital computers were not constructed until the Second World War. **Calculating engines with mechanical relays were built at that time**. However, the **mechanical relays** were **very slow** and were later **replaced with vacuum tubes**. These machines were enormous but were **still very slow**.

These early computers were designed, built and maintained by a single group of people. Programming languages were unknown and there were **no operating systems** so **all the programming was done in machine language. All the problems were simple numerical calculations.**

By the 1950's punch cards were introduced and this improved the computer system. Instead of using plug boards, **programs were written on cards and read into the system**.

The Second Generation (1955 - 1965): Transistors and Batch Systems

Transistors led to the development of the computer systems that could be manufactured and sold to paying customers. These machines were **known as mainframes** and were locked in air-conditioned computer rooms with **staff to operate them**.

The **Batch System** was introduced **to reduce the wasted time in the computer**. A tray full of **jobs was collected in the input room and read into the magnetic tape**. After that, the tape was rewound and mounted on a tape drive. Then the batch operating system was loaded in which **read the first job from the tape and ran it**. The **output was written on the second tape**. After the **whole batch was done, the input and output tapes were removed and the output tape was printed.**

The Third Generation (1965 - 1980): Integrated Circuits and Multiprogramming

Until the 1960's, there were two types of computer systems i.e the scientific and the commercial computers. These were combined by IBM in the System/360. This used **integrated circuits** and provided a **major price and performance advantage over the second generation systems**.

The third generation operating systems also introduced **multiprogramming**. This meant that the **processor was not idle while a job was completing its I/O operation**. **Another job was scheduled on the processor so that its time would not be wasted.**

The Fourth Generation (1980 - Present): Personal Computers

Personal Computers were easy to create with the development of **large-scale integrated circuits**. These were chips containing thousands of transistors on a square centimeter of silicon. Because of these, **microcomputers were much cheaper than minicomputers** and that made it possible for a single individual to own one of them.

The advent of personal computers also led to the growth of networks. This **created network operating systems and distributed operating systems.** The users were aware of a network while **using a network operating system and could log in to remote machines and copy files from one machine to another.**

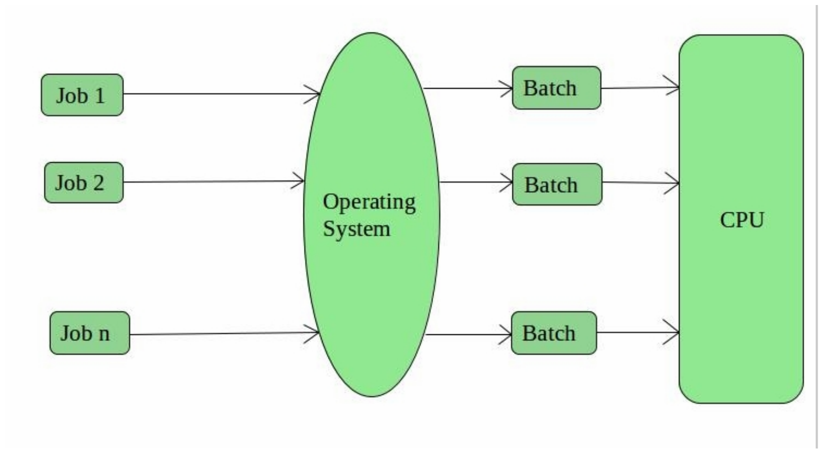
Types of Operating Systems

An Operating System performs all the basic tasks like managing file, process, and memory. Thus operating system acts as manager of all the resources, i.e. resource manager. Thus operating system becomes an interface between user and machine.

Some of the widely used operating systems are as follows-

1. Batch Operating System

This type of operating system does not interact with the computer directly. There is an operator who takes similar jobs having same requirement and group them into batches. It is the responsibility of operator to sort the jobs with similar needs.



Advantages of Batch Operating System:

- It is very difficult to guess or know the time required by any job to complete. Processor of the batch systems knows how long the job would be when it is in queue
- **Multiple users can share** the batch systems
- The idle time batch system is very less
- It is **easy to manage large work** repeatedly in batch systems

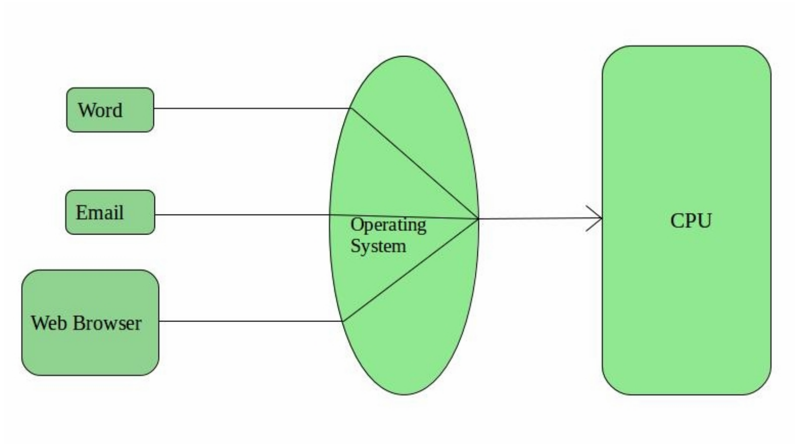
Disadvantages of Batch Operating System:

- The computer operators should be well known with batch systems

- Batch systems are **hard to debug**
- It is sometime **costly**
- The other jobs will have to **wait** for an unknown time if any job fails
- **Examples of Batch based Operating System:** Payroll System, Bank Statements etc.

2. Time-Sharing Operating Systems

Each task has given some time to execute, so that all the tasks work smoothly. Each user gets time of CPU as they use single system. These systems are also known as Multitasking Systems. The task can be from single user or from different users also. The time that each task gets to execute is called quantum. After this time interval is over OS switches over to next task.



Advantages of Time-Sharing OS:

- Each task gets an equal opportunity
- Less chances of duplication of software
- CPU idle time can be reduced

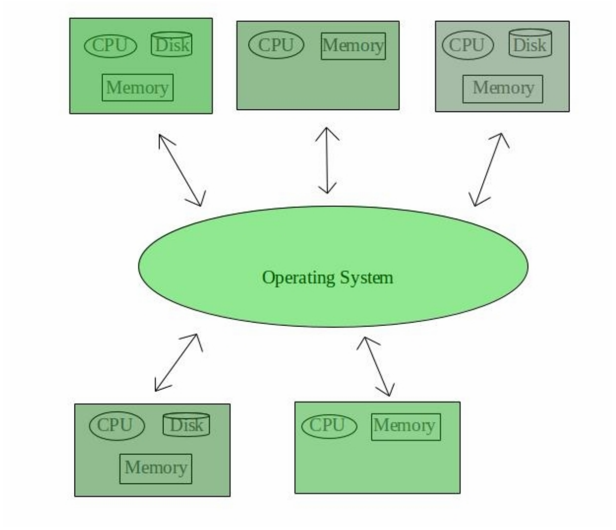
Disadvantages of Time-Sharing OS:

- Reliability problem
- One must have to take care of security and integrity of user programs and data
- Data communication problem

Examples of Time-Sharing OSs are: Multics, Unix etc.

3. Distributed Operating System

These types of operating system is a recent advancement in the world of computer technology and are being widely accepted all-over the world and, that too, with a great pace. Various autonomous interconnected computers communicate each other using a shared communication network. Independent systems possess their own memory unit and CPU. These are referred as loosely coupled systems or distributed systems. These systems processors differ in sizes and functions. The major benefit of working with these types of operating system is that it is always possible that one user can access the files or software which are not actually present on his system but on some other system connected within this network i.e., remote access is enabled within the devices connected in that network.



Advantages of Distributed Operating System:

- Failure of one will not affect the other network communication, as all systems are independent from each other
- Electronic mail increases the data exchange speed
- Since resources are being shared, computation is highly fast and durable
- Load on host computer reduces
- These systems are easily scalable as many systems can be easily added to the network
- Delay in data processing reduces

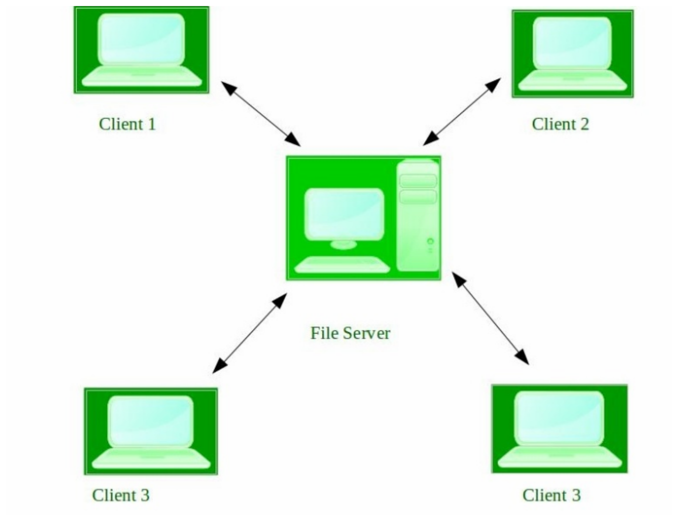
Disadvantages of Distributed Operating System:

- Failure of the main network will stop the entire communication
- To establish distributed systems the language which are used are not well defined yet
- These types of systems are not readily available as they are very expensive. Not only that the underlying software is highly complex and not understood well yet

Examples of Distributed Operating System are- LOCUS etc.

4. Network Operating System

These systems runs on a server and provides the capability to manage data, users, groups, security, applications, and other networking functions. These type of operating systems allows shared access of files, printers, security, applications, and other networking functions over a small private network. One more important aspect of Network Operating Systems is that all the users are well aware of the underlying configuration, of all other users within the network, their individual connections etc. and that's why these computers are popularly known as tightly coupled systems.



Advantages of Network Operating System:

- Highly stable centralized servers
- Security concerns are handled through servers
- New technologies and hardware up-gradation are easily integrated to the system
- Server access are possible remotely from different locations and types of systems

Disadvantages of Network Operating System:

- Servers are costly
- User has to depend on central location for most operations
- Maintenance and updates are required regularly

Examples of Network Operating System are: Microsoft Windows Server 2003, Microsoft Windows Server 2008, UNIX, Linux, Mac OS X, Novell NetWare, and BSD etc.

5. Real-Time Operating System

These types of OSs serves the real-time systems. The time interval required to process and respond to inputs is very small. This time interval is called response time.

Real-time systems are used when there are time requirements are very strict like missile systems, air traffic control systems, robots etc.

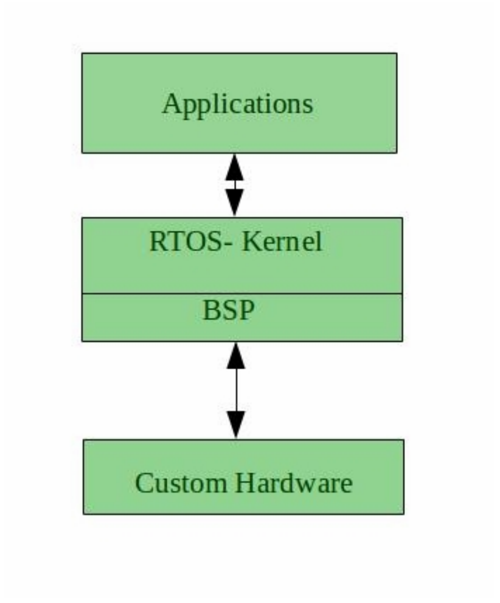
Two types of Real-Time Operating System which are as follows:

Hard Real-Time Systems:

These OSs are meant for the applications where time constraints are very strict and even the shortest possible delay is not acceptable. These systems are built for saving life like automatic parachutes or air bags which are required to be readily available in case of any accident. Virtual memory is almost never found in these systems.

Soft Real-Time Systems:

These OSs are for applications where for time-constraint is less strict.



Advantages of RTOS:

- Maximum Consumption: Maximum utilization of devices and system, thus more output from all the resources
- Task Shifting: Time assigned for shifting tasks in these systems are very less. For example in older systems it takes about 10 micro seconds in shifting one task to another and in latest systems it takes 3 micro seconds.
- Focus on Application: Focus on running applications and less importance to applications which are in queue.
- Real time operating system in embedded system: Since size of programs are small, RTOS can also be used in embedded systems like in transport and others.
- Error Free: These types of systems are error free.
- Memory Allocation: Memory allocation is best managed in these type of systems.

Disadvantages of RTOS:

- Limited Tasks: Very few task run at the same time and their concentration is very less on few applications to avoid errors.
- Use heavy system resources: Sometimes the system resources are not so good and they are expensive as well.
- Complex Algorithms: The algorithms are very complex and difficult for the designer to write on.
- Device driver and interrupt signals: It needs specific device drivers and interrupt signals to response earliest to interrupts.

- Thread Priority: It is not good to set thread priority as these systems are very less prone to switching tasks.

Examples of Real-Time Operating Systems are: Scientific experiments, medical imaging systems, industrial control systems, weapon systems, robots, air traffic control systems, etc.

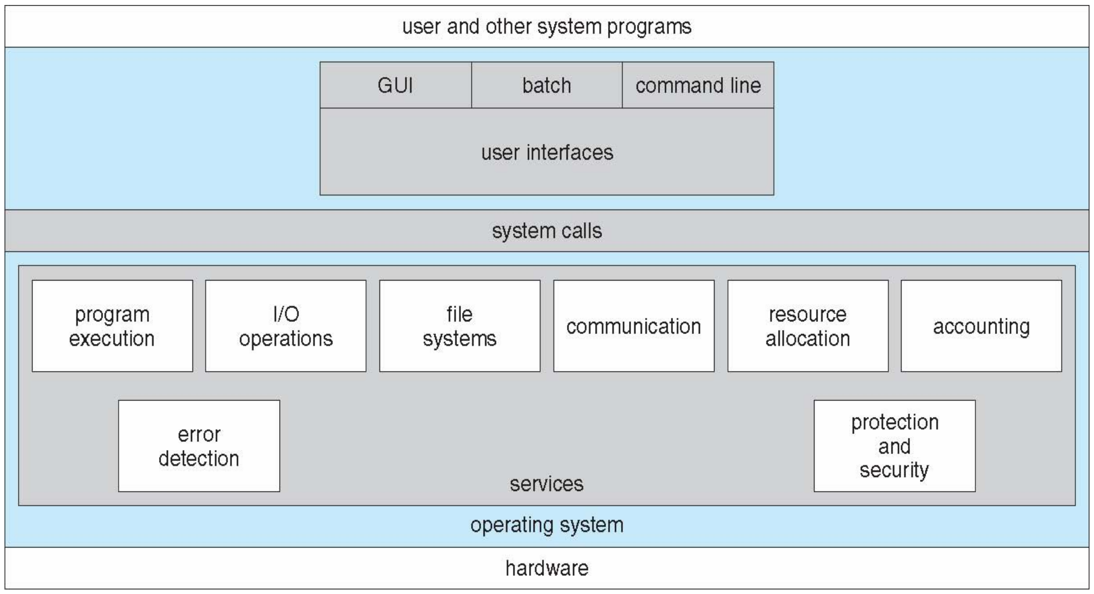
OS Services

An Operating System supplies different kinds of services to both the users and to the programs as well. It also provides application programs an environment to execute it freely. It provides users the services which run various programs in a convenient manner.

Here is a **list of common services** offered by almost all operating systems:

1. User Interface
2. Program Execution
3. File system manipulation
4. Input / Output Operations
5. Communication
6. Resource Allocation
7. Error Detection
8. Accounting
9. Security and protection

A View of Operating System Services



1. **User Interface**

Usually Operating system comes in three forms or types. Depending on the interface, their types have been further subdivided. These are:

- Command line interface
- Batch based interface
- Graphical User Interface

The Command Line Interface (CLI): uses text commands and a method for entering those commands.

The Batch Interface (BI): commands and directives used to manage those commands are entered into files and those files are executed.

Graphical User Interface (GUI): which is a window system with a pointing device (like mouse) to direct I/O, choose from the menus and make selections, and a keyboard to enter text.

2. **Program Execution**

The operating system must have the capability to load a program into memory and execute that program. Furthermore, the program must be able to end its execution, either normally or abnormally / forcefully.

3. **File system manipulation**

Programs need to read and write into files and directories. File handling portion of operating system also allows users to create and delete files by specific name along with extension, search for a given file and/or list file information. Some programs comprise of permissions management for allowing or denying access to files or directories based on file ownership.

4. Input/Output Operations

A program which is currently executing in memory, may require I/O, which may involve file or other I/O device. For efficiency and protection, users cannot directly govern the I/O devices. So, the OS provide a means to do Input/Output operation which means read or write operation with any file.

5. Communication

Process needs to exchange information with other process. Processes executing on same computer system or on different computer systems can communicate using operating system support. Communication between two processes can be done using **shared memory** or via **message passing**.

6. Resource Allocation

When multiple jobs are running concurrently, resources must be allocated to each of them. Resources can be CPU cycles, main memory storage, file storage and I/O devices. CPU scheduling routines are used here to establish how best the CPU can be used.

7. Error Detection

Errors may occur within CPU, memory hardware, I/O devices and in the user program. For each type of error, the OS takes adequate action for ensuring correct and consistent computing.

8. Accounting

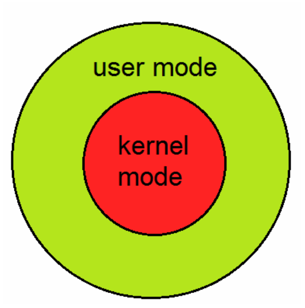
This service of the operating system keeps track of which users are using how much and what kinds of computer resources. It is used for accounting or simply to accumulate usage statistics.

9. Security and protection

Protection involves ensuring that all access to system resources is in a controlled manner. For making a system secure, the user needs to authenticate him or her to the system before using (usually via login ID and password).

System call

To understand system calls, first one needs to understand the difference between **kernel mode** and **user mode** of a CPU. Every modern operating system supports these two modes.



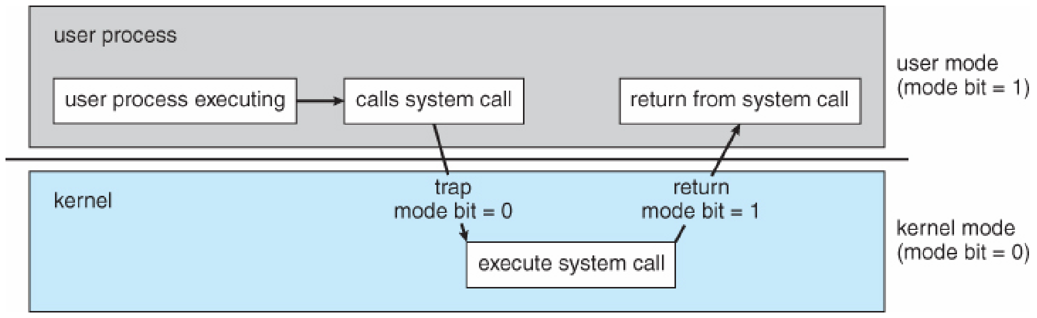
Modes supported by the operating system

1.Kernel Mode

- When CPU is in **kernel mode**, the code being executed can access any memory address and any hardware resource.
- Hence kernel mode is a very privileged and powerful mode.
- If a program crashes in kernel mode, the entire system will be halted.

2. User Mode

- When CPU is in **user mode**, the programs don't have direct access to memory and hardware resources.
- In user mode, if any program crashes, only that particular program is halted.
- That means the system will be in a safe state even if a program in user mode crashes.
- Hence, most programs in an OS run in user mode.



System Call

When a program in user mode requires access to RAM (memory) or a hardware resource, it must ask the kernel to provide access to that resource. This is done via something called a **system call**.

When a program makes a system call, the mode is switched from user mode to kernel mode. This is called a **context switch**.

Then the kernel provides the resource which the program requested. After that, another context switch happens which results in change of mode from kernel mode back to user mode.

Generally, system calls are made by the user level programs in the following situations:

- Creating, opening, closing and deleting files in the file system.
- Creating and managing new processes.
- Creating a connection in the network, sending and receiving packets.
- Requesting access to a hardware device, like a mouse or a printer.

Definition

A system call is a way for programs to interact with the operating system. A computer program makes a system call when it makes a request to the operating system's kernel. System call provides the services of the operating system to the user programs via Application Program Interface(API). System calls are the only entry points into the kernel system. All programs needing resources must use system calls.

Types of System Calls : There are 5 different categories of system calls –

1. Process control
2. File management
3. Device management
4. Information maintenance
5. Communication

1. 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

2. File management

- Create file, delete file
- Open , close
- Read , write , reposition
- Get file attributes, set file attributes
  - File attributes: file name, file type, protection codes, accounting information

3. Device management

- Request device, release device
- Read , write , reposition
- Get device attributes, set device attributes

4. 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

5. Communication

- 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()
	ExitProcess()
	WaitForSingleObject()
		fork()
		exit()
		wait()

File Manipulation	CreateFile()
	ReadFile()
	WriteFile()
	CloseHandle()
		open()
		read()
		write()
		close()

Device	SetConsoleMode()	ioctl()

Manipulation	ReadConsole()
	WriteConsole()
		read()
		write()

Information Maintenance	GetCurrentProcessID()
	SetTimer()
	Sleep()
		getpid()
		alarm()
		sleep()

Communication	CreatePipe()
	CreateFileMapping()
	MapViewOfFile()
		pipe()
		shmget()
		mmap()

Protection	SetFileSecurity()
	InitlializeSecurityDescriptor()
	SetSecurityDescriptorGroup()
		chmod()
		umask()
		chown()

Structure of an OS -

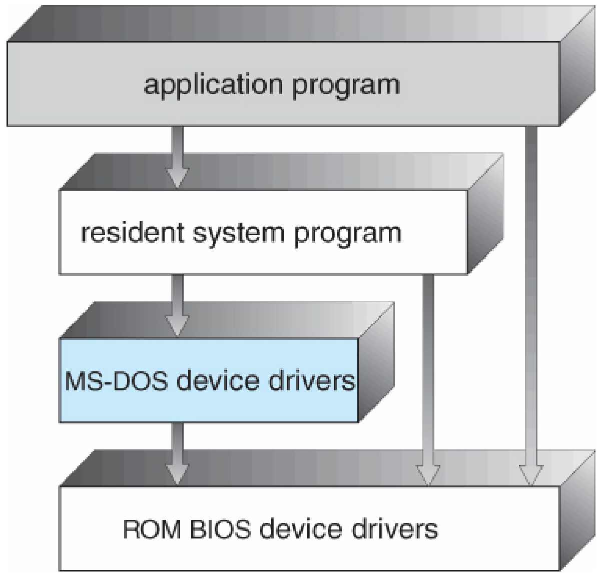
Simple (Monolithic), Layered, Microkernel Operating Systems

For efficient performance and implementation, an OS should be partitioned into separate subsystems, each with carefully defined tasks, inputs, outputs, and performance characteristics. These subsystems can then be arranged in various architectural configurations:

1. Simple/ Monolithic Structure

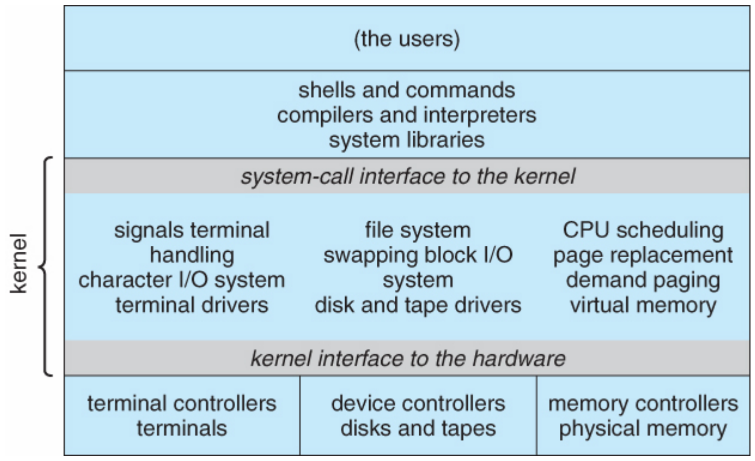
When DOS was originally written its developers had no idea how big and important it would eventually become. It was written by a few programmers in a relatively short amount of time, without the benefit of modern software engineering techniques, and then gradually grew over time to exceed its original expectations. It does not break the system into subsystems, and has no distinction between user and kernel modes, allowing all programs direct access to the underlying hardware. (Note that user versus kernel mode was not supported by the 8088 chip set anyway, so that really wasn't an option back then.)

The original approach, big layer, not layered



MS-DOS layer structure

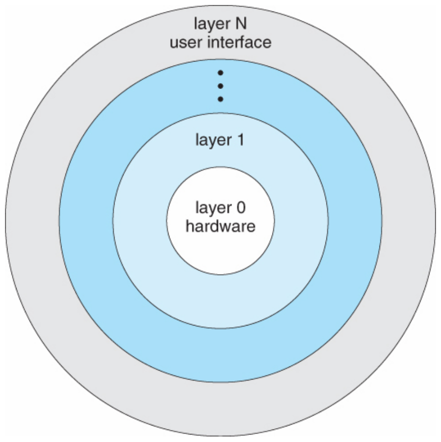
UNIX OS used a simple layered but almost all the OS was in one really breaking the OS down into subsystems:



Traditional UNIX system structure

2. Layered Approach

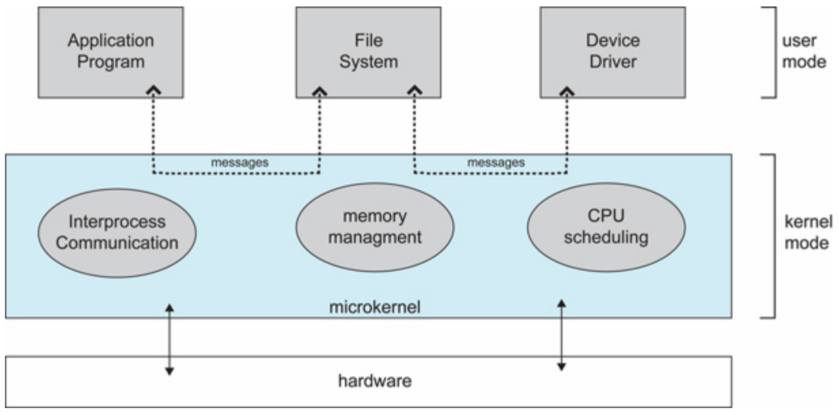
- Another approach is to break the OS into a number of smaller layers, each of which rests on the layer below it, and relies solely on the services provided by the next lower layer.
- This approach allows each layer to be developed and debugged independently, with the assumption that all lower layers have already been debugged and are trusted to deliver proper services.
- The problem is deciding what order in which to place the layers, as no layer can call upon the services of any higher layer, and so many chicken-and-egg situations may arise.
- Layered approaches can also be less efficient, as a request for service from a higher layer has to filter through all lower layers before it reaches the HW, possibly with significant processing at each step.



A layered operating system

3. Micro kernels

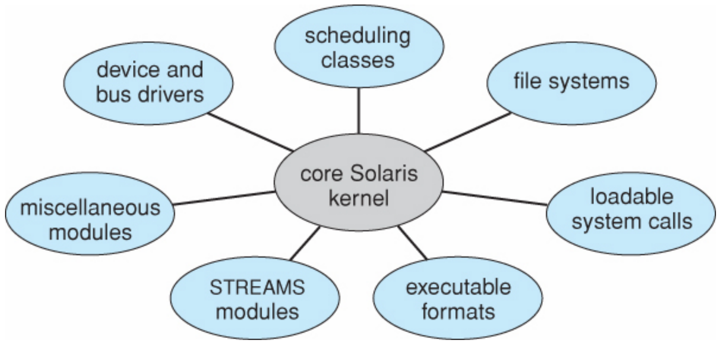
- The basic idea behind micro kernels is to remove all non-essential services from the kernel, and implement them as system applications instead, thereby making the kernel as small and efficient as possible.
- Most microkernels provide basic process and memory management, and message passing between other services, and not much more.
- Security and protection can be enhanced, as most services are performed in user mode, not kernel mode.
- System expansion can also be easier, because it only involves adding more system applications, not rebuilding a new kernel.
- Mach was the first and most widely known microkernel, and now forms a major component of Mac OSX.
- Windows NT was originally microkernel, but suffered from performance problems relative to Windows 95. NT 4.0 improved performance by moving more services into the kernel, and now XP is back to being more monolithic.
- Another microkernel example is QNX, a real-time OS for embedded systems.



Architecture of a typical microkernel

4. Modules

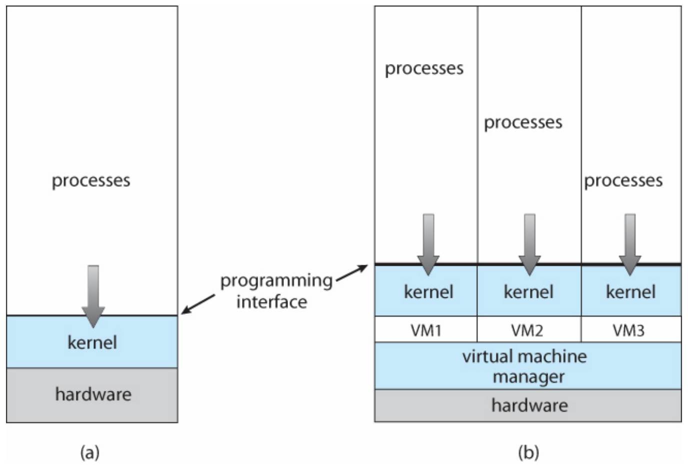
- Modern OS development is object-oriented, with a relatively small core kernel and a set of modules which can be linked in dynamically. See for example the Solaris structure, as shown in Figure 2.13 below.
- Modules are similar to layers in that each subsystem has clearly defined tasks and interfaces, but any module is free to contact any other module, eliminating the problems of going through multiple intermediary layers, as well as the chicken-and-egg problems.
- 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.



Solaris loadable modules

<u>Concept of Virtual Machine</u>

- The concept of a virtual machine is to provide an interface that looks like independent hardware, to multiple different OSes running simultaneously on the same physical hardware. Each OS believes that it has access to and control over its own CPU, RAM, I/O devices, hard drives, etc.
- One obvious use for this system is for the development and testing of software that must run on multiple platforms and/or OSes.
- One obvious difficulty involves the sharing of hard drives, which are generally partitioned into separate smaller virtual disks for each operating OS.



System models. (a) Non virtual machine. (b)Virtual machine.

1. History

- Virtual machines first appeared as the VM Operating System for IBM mainframes in 1972.

2. Benefits

- Each OS runs independently of all the others, offering protection and security benefits.
- (Sharing of physical resources is not commonly implemented, but may be done as if the virtual machines were networked together.)
- Virtual machines are a very useful tool for OS development, as they allow a user full access to and control over a virtual machine, without affecting other users operating the real machine.
- As mentioned before, this approach can also be useful for product development and testing of SW that must run on multiple OSes / HW platforms.

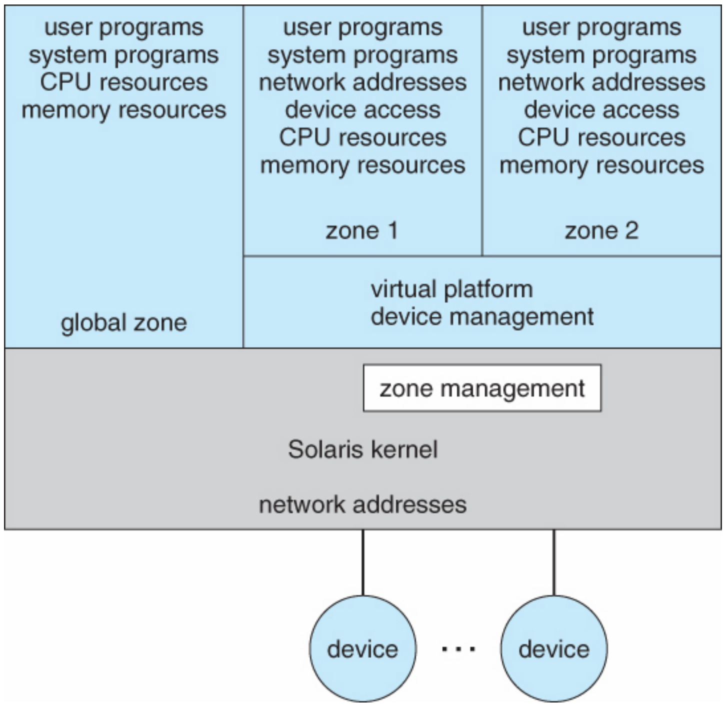
3. Simulation

- An alternative to creating an entire virtual machine is to simply run an ***emulator***, which allows a program written for one OS to run on a different OS.
- For example, a UNIX machine may run a DOS emulator in order to run DOS programs, or vice-versa.
- Emulators tend to run considerably slower than the native OS, and are also generally less than perfect.

4. Para-virtualization

- Para-virtualization is another variation on the theme, in which an environment is provided for the guest program that is ***similar to*** its native OS, without trying to completely mimic it.
- Guest programs must also be modified to run on the para-virtual OS.
- Solaris 10 uses a ***zone*** system, in which the low-level hardware is not virtualized, but the OS and its devices (device drivers) are.
    - Within a zone, processes have the view of an isolated system, in which only the processes and resources within that zone are seen to exist.

- Figure shows a Solaris system with the normal "global" operating space as well as two additional zones running on a small virtualization layer.



Solaris 10 with two zones.

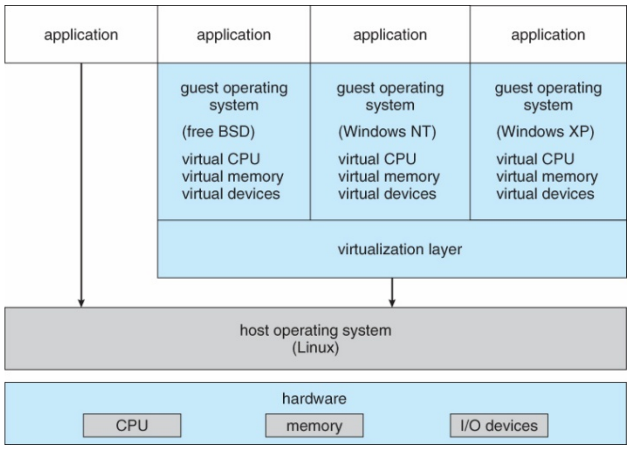
5 Implementation

- Implementation may be challenging, partially due to the consequences of user versus kernel mode.
  - Each of the simultaneously running kernels needs to operate in kernel mode at some point, but the virtual machine actually runs in user mode.
  - So the kernel mode has to be simulated for each of the loaded OSes, and kernel system calls passed through the virtual machine into a true kernel mode for eventual HW access.
- The virtual machines may run slower, due to the increased levels of code between applications and the HW, or they may run faster, due to the benefits of caching. (And virtual devices may also be faster than real devices, such as RAM disks which are faster than physical disks.)

Examples

1. VMware

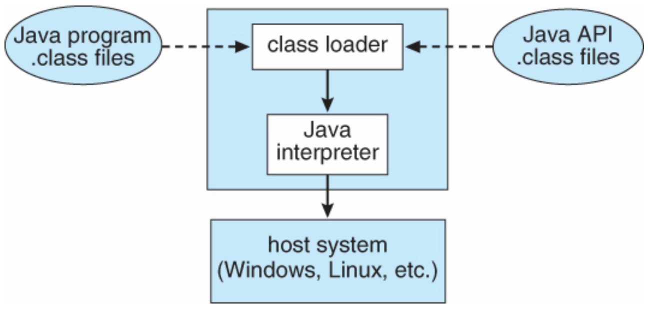
- Abstracts the 80x86 hardware platform, allowing simultaneous operation of multiple Windows and Linux OSes, as shown by example in Figure 2.19:



VMWare Workstation architecture

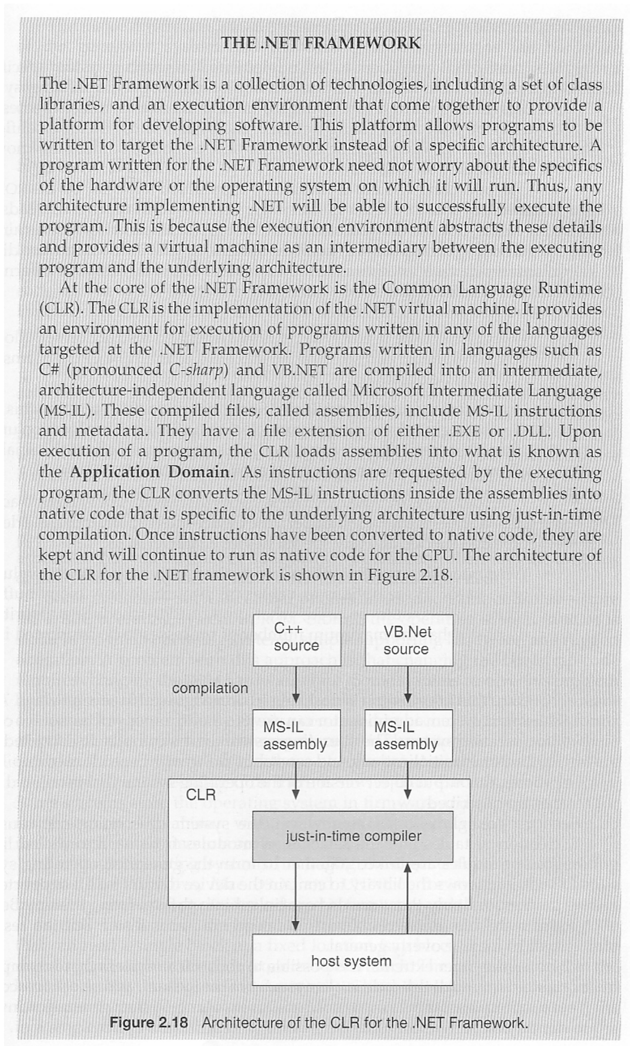
2 The Java Virtual Machine

- Java was designed from the beginning to be platform independent, by running Java only on a Java Virtual Machine, JVM, of which different implementations have been developed for numerous different underlying HW platforms.
- Java source code is compiled into Java byte code in .class files. Java byte code is binary instructions that will run on the JVM.
- The JVM implements memory management and garbage collection.
- Java byte code may be interpreted as it runs, or compiled to native system binary code using just-in-time (JIT) compilation. Under this scheme, the first time that a piece of Java byte code is encountered, it is compiled to the appropriate native machine binary code by the Java interpreter. This native binary code is then cached, so that the next time that piece of code is encountered it can be used directly.
- Some hardware chips have been developed to run Java byte code directly, which is an interesting application of a real machine being developed to emulate the services of a virtual one!



The Java virtual machine

- The .NET framework also relies on the concept of compiling code for an intermediary virtual machine, (Common Language Runtime, CLR), and then using JIT compilation and caching to run the programs on specific hardware, as shown in Figure below:



THE .NET FRAMEWORK

The .NET Framework is a collection of technologies, including a set of class libraries, and an execution environment that come together to provide a platform for developing software. This platform allows programs to be written to target the .NET Framework instead of a specific architecture. A program written for the .NET Framework need not worry about the specifics of the hardware or the operating system on which it will run. Thus, any architecture implementing .NET will be able to successfully execute the program. This is because the execution environment abstracts these details and provides a virtual machine as an intermediary between the executing program and the underlying architecture.

At the core of the .NET Framework is the Common Language Runtime (CLR). The CLR is the implementation of the .NET virtual machine. It provides an environment for execution of programs written in any of the languages targeted at the .NET Framework. Programs written in languages such as C# (pronounced *C-sharp*) and VB.NET are compiled into an intermediate, architecture-independent language called Microsoft Intermediate Language (MS-IL). These compiled files, called assemblies, include MS-IL instructions and metadata. They have a file extension of either .EXE or .DLL. Upon execution of a program, the CLR loads assemblies into what is known as the **Application Domain**. As instructions are requested by the executing program, the CLR converts the MS-IL instructions inside the assemblies into native code that is specific to the underlying architecture using just-in-time compilation. Once instructions have been converted to native code, they are kept and will continue to run as native code for the CPU. The architecture of the CLR for the .NET framework is shown in Figure 2.18.

Figure 2.18 Architecture of the CLR for the .NET Framework.

Case study on UNIX and WINDOWS Operating System

Unix:

Self Study (Refer text book)

Windows:

Self Study (Refer text book)