UNIT – I
OPERATING SYSTEMS OVERVIEW

Objective of the course
1. To provide a general explanation of the component of operating systems
2. To provide the general organization of the computer systems and the relation between the computer structure and operating systems.

What is an Operating System?
A program that acts as an intermediary between a user of a computer and the computer hardware.

Why we need an Operating system?
Generally an operating system is needed for the following reasons:
1. Execute user programs and make solving user problems easier.
2. Make the computer system convenient to use.
3. Use the computer hardware in an efficient manner.
4. OS is a resource allocator
5. Manages all resources
6. Decides between conflicting requests for efficient and fair resource use
7. OS is a control program
8. Controls execution of programs to prevent errors and improper use of the computer
9. No universally accepted definition
10. “The one program running at all times on the computer” is the kernel. Everything else is either a system program (ships with the operating system) or an application program.

1. What Operating Systems Do

Computer systems
Computer systems can be divided into four components
1. Hardware –provides basic computing resources CPU, memory, I/O devices
2. Operating system-Controls and coordinates use of hardware among various applications and users
3. 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, video games
4. Users
5. People, machines, other computers

Four Components of a Computer System

**User View**
The user view of computer varies by the interface being used. The operating systems are designed mostly for ease of use. Others are designed to maximize resource utilization. Other operating systems are designed to compromise between individual usability and resource utilization.

**System view**
From the computer’s point of view, the OS is a

1. **Resource allocator**: Manages all resources and decides between conflicting requests for efficient and fair resource use.
2. **Control program**: Controls execution of programs to prevent errors and improper use of the computer

**Computer Startup**
1. Bootstrap program is loaded at power-up or reboot
2. Typically stored in ROM or EPROM, generally known as firmware
3. Initializes all aspects of system
4. Loads operating system kernel and starts execution
2. **Computer System Organization**

1. Computer-system operation
2. One or more CPUs, device controllers connect through common bus providing access to shared memory
3. Concurrent execution of CPUs and devices competing for memory cycles.

![Fig: A Modern Computer System](image)

**Computer-System Operation**

A modern, general-purpose computer system consists of CPU and a number of device controllers that connected through a common bus that provides access to shared memory system, CPU other devices can execute concurrently competing for memory cycles.

1. I/O devices and the CPU can execute concurrently
2. Each device controller is in charge of a particular device type
3. Each device controller has a local buffer
4. CPU moves data from/to main memory to/from local buffers
5. I/O is from the device to local buffer of controller
6. Device controller informs CPU that it has finished its operation by causing An interrupt

**Booting:** It is the operation of bringing operating system kernel from the secondary storage and put it in main storage to execute it in CPU. There is a program bootstrap which is performing this operation when computer is powered up or rebooted.

**Bootstrap software:** It is an initial program and simple it is stored in read-only memory (ROM) such as firmware or EEPROM within the computer hardware.

**Jobs of Bootstrap program:**

1. Initialize all the aspect of the system, from CPU registers to device controllers to memory contents.
2. Locate and load the operating system kernel into memory then the operating system starts executing the first process, such as “init” and waits for some event to occur.

3. The operating system then waits for some event to occur. Types of events are either software events (system call) or hardware events (signals from the hardware devices to the CPU through the system bus and known as an interrupt).

**Note:** All modern operating system are “interrupt driven”.

**Trap (exception):** it is a software-generated interrupt caused either by an error (ex: division by zero or invalid memory access) or by a specific request from a user program that an operating system service be performed.

**Interrupt vector (IV):** it is a fixed locations (an array) in the low memory area (first 100 locations of RAM) of operating system when the interrupt occur the CPU stops what it’s doing and transfer execution to a fixed location (IV) contain starting address of the interrupt service routine(ISR), on completion the CPU resumes the interrupted computation.

**Interrupt Service Routine:** is it a routine provided to be responsible for dealing with the interrupt.

**Common Functions of Interrupts**

1. Interrupt transfers control to the interrupt service routine generally, through the interrupt vector, which contains the addresses of all the service routines
2. Interrupt architecture must save the address of the interrupted instruction
3. Incoming interrupts are disabled while another interrupt is being processed to prevent a lost interrupt.
4. A trap is a software-generated interrupt caused either by an error or a user request
5. An operating system is interrupt driven.

**Interrupt Handling**

1. The operating system preserves the state of the CPU by storing registers and the program counter
2. Determines which type of interrupt has occurred:
   a. polling
   b. vectored interrupt system
3. Separate segments of code determine what action should be taken for each type of interrupt.

**Interrupt Timeline**

![Interrupt Timeline](image)

**Fig: Interrupt timeline for a single process doing output**

**Storage Structure**

1. Main memory – only large storage media that the CPU can access directly
2. Secondary storage – extension of main memory that provides large nonvolatile storage capacity
3. Magnetic disks – rigid metal or glass platters covered with magnetic recording material
4. Disk surface is logically divided into tracks, which are subdivided into sectors
5. The disk controller determines the logical interaction between the device and the computer.

**Storage Hierarchy**

1. Storage systems organized in hierarchy
2. Speed
3. Cost
4. Volatility

**Caching** – copying information into faster storage system; main memory can be viewed as a last cache for secondary storage.
Caching

1. Important principle, performed at many levels in a computer (in hardware, operating system, software)
2. Information in use copied from slower to faster storage temporarily
3. Faster storage (cache) checked first to determine if information is there
4. If it is, information used directly from the cache (fast)
5. If not, data copied to cache and used there
6. Cache smaller than storage being cached
7. Cache management important design problem
8. Cache size and replacement policy.

Direct Memory Access Structure

- Used for high-speed I/O devices able to transmit information at close to memory speeds
- Device controller transfers blocks of data from buffer storage directly to main memory without CPU intervention
- Only one interrupt is generated per block, rather than the one interrupt per byte
**Fig: How a modern computer system works.**

**I/O Structure**

1. After I/O starts, control returns to user program only upon I/O completion
2. Wait instruction idles the CPU until the next interrupt
3. Wait loop (contention for memory access)
4. At most one I/O request is outstanding at a time, no simultaneous I/O processing
5. After I/O starts, control returns to user program without waiting for I/O completion
6. System call – request to the operating system to allow user to wait for I/O completion
7. Device-status table contains entry for each I/O device indicating its type, address, and state
8. Operating system indexes into I/O device table to determine device status and to modify table entry to include interrupt

**3. Computer System Architecture**

1. Most systems use a single general-purpose processor (PDAs through mainframes)
2. Most systems have special-purpose processors as well
3. Multiprocessors systems growing in use and importance
4. Also known as parallel systems, tightly-coupled systems

**Advantages**

1. Increased throughput
2. Economy of scale
3. Increased reliability – graceful degradation or fault tolerance

Two types

1. Asymmetric Multiprocessing
2. Symmetric Multiprocessing

**Symmetric Multiprocessing Architecture**

![Symmetric Multiprocessing Architecture Diagram]

**A Dual-Core Design**

![A Dual-Core Design Diagram]
Mainframe System:
1. Mainframe systems are mainly used for scientific & commercial applications.
2. An OS may process its workload serially where the computer runs only one application or concurrently where computer runs many applications.

Batch Systems:
1. Early computers where physically large machines.
2. The common I/P devices are card readers & tape drives.
3. The common O/P devices are line printers, tape drives & card punches.
4. The user does not interact directly with computers but we use to prepare a job with the program, data & some control information & submit it to the computer operator.
5. The job was mainly in the form punched cards.
6. At later time the O/P appeared and it consisted of result along with dump of memory and register content for debugging.

The OS of these computers was very simple. Its major task was to transfer control from one job to the next. The OS was always resident in the memory. The processing of job was very slow. To improve the processing speed operators batched together the jobs with similar needs and processed it through the computers. This is called Batch Systems.

1. In batch systems the CPU may be idle for some time because the speed of the mechanical devices slower compared to the electronic devices.
2. Later improvement in technology and introduction of disks resulted in faster I/O devices.
3. The introduction of disks allowed the OS to store all the jobs on the disk. The OS could perform the scheduling to use the resources and perform the task efficiently.

Disadvantages of Batch Systems:
1. Turn around time can be large from user.
2. Difficult to debug the program.
3. A job can enter into infinite loop.
4. A job could corrupt the monitor.
5. Due to lack of protection scheme, one job may affect the pending jobs.
Multiprogrammed System:

1. If there are two or more programs in the memory at the same time sharing the processor, this is referred as multiprogrammed OS.
2. It increases the CPU utilization by organizing the jobs so that the CPU will always have one job to execute.
3. Jobs entering the systems are kept in memory.
4. OS picks the job from memory & it executes it.
5. Having several jobs in the memory at the same time requires some form of memory management.
6. Multiprogrammed systems monitors the state of all active program and system resources and ensures that CPU is never idle until there are no jobs.
7. While executing a particular job, if the job has to wait for any task like I/O operation to be complete then the CPU will switch to some other jobs and starts executing it and when the first job finishes waiting the CPU will switch back to that.
8. This will keep the CPU & I/O utilization busy. The following figure shows the memory layout of multiprogrammed OS.
Time sharing Systems:

1. Time sharing system or multi tasking is logical extension of multi programming systems. The CPU executes multiple jobs by switching between them but the switching occurs so frequently that user can interact with each program while it is running.
2. An interactive & hands on system provides direct communication between the user and the system. The user can give the instruction to the OS or program directly through keyboard or mouse and waits for immediate results.
3. A time shared system allows multiple users to use the computer simultaneously. Since each action or commands are short in time shared systems only a small CPU time will be available for each of the user.
4. A time shared systems uses CPU scheduling and multi programming to provide each user a small portion of time shared computers. When a process executes it will be executing for a short time before it finishes or need to perform I/O. I/O is interactive i.e. O/P is to a display for the user and the I/O is from a keyboard, mouse etc.
5. Since it has to maintain several jobs at a time, system should have memory management & protection.
6. Time sharing systems are complex than the multi programmed systems. Since several jobs are kept in memory they need memory management and protection. To obtain less response time jobs are swapped in and out of main memory to disk. So disk will serve as backing store for main memory. This can be achieved by using a technique called virtual memory that allows for the execution of job i.e. not completes in memory.
7. Time sharing system should also provide a file system & file system resides on collection of disks so this need disk management. It supports concurrent execution, job synchronization & communication.

Desktop systems:

1. Pc’s appeared in 1970’s and during this they lacked the feature needed to protect an OS from user program& they even lack neither multi user nor multi tasking.
2. The goals of those OS changed later with the time and new systems include Microsoft Windows & Apple Macintosh.
3. The Apple Macintosh OS ported to more advanced hardware & includes new features like virtual memory & multi tasking.
4. Micro computers are developed for single user in 1970’s & they can accommodate software with large capacity & greater speeds. MS-DOS is an example for micro computer OS & are used by commercial, educational, government enterprises.

**Single-Processor Systems**

1. One main CPU which manages the computer and runs user apps.
2. Other specialized processors (disk controllers, GPUs, etc.) do not run user apps.

**Multi Processor Systems:** (Parallel systems or tightly coupled systems)

1. Multi processor systems include more than one processor in close communication.
2. They share computer bus, the clock, m/y & peripheral devices.
3. Two processes can run in parallel.

Multi processor systems are of two types

1. Symmetric Multi processors (SMP)
2. Asymmetric Multi processors.

i. In symmetric multi processing, each processors runs an identical copy of OS and they communicate with one another as needed. All the CPU shares the common memory.

ii. In asymmetric multi processing, each processors is assigned a specific task. It uses a master slave relationship. A master processor controls the system. The master processors schedules and allocates work to slave processors. The following figure shows asymmetric multi processors.

iii. SMP means all processors are peers i.e. no master slave relationship exists between processors. Each processor concurrently runs a copy of OS.

iv. The differences between symmetric & asymmetric multi processing may be result of either H/w or S/w.

   Special H/w can differentiate the multiple processors or the S/w can be written to allow only master & multiple slaves.

**Advantages of Multi Processor Systems:**

1. **Increased Throughput:**- By increasing the Number of processors we can get more work done in less time. When multiple processes cooperate on task, a certain amount of overhead is incurred in keeping all parts working correctly.
2. **Economy of Scale:**- Multi processor system can save more money than multiple single processors, since they share peripherals, mass storage & power supplies. If many programs
operate on same data, they will be stored on one disk & all processors can share them instead of maintaining data on several systems.

3. Increased Reliability: If a program is distributed properly on several processors, than the failure of one processor will not halt the system but it only slows down.

This ability to continue providing service proportional to the level of surviving hardware is called “graceful degradation” is also called “fault tolerant”.

There are different architectures for multiprocessor systems.

**Symmetric Multiprocessing Architecture**

![Symmetric Multiprocessing Architecture Diagram]

**A Dual-Core Design**

![A Dual-Core Design Diagram]

**Distributed Systems:** A network is a communication path between two or more systems. Distributed systems depend on networking for their functionality. Using communicates, distributed systems are able to share computational tasks, and provide a rich set of set of feature to users.

1. client-server systems
2. peer-to-peer systems

Some operating system benefits from ideas of networking and distributed systems in build network operating system.
Clustered Systems

1. Like multiprocessor systems, but multiple systems working together
2. Usually sharing storage via a storage-area network (SAN)
3. Provides a high-availability service which survives failures
   a. Asymmetric clustering has one machine in hot-standby mode
   b. Symmetric clustering has multiple nodes running applications, monitoring each other.
4. Some clusters are for high-performance computing (HPC): Applications must be written to use parallelization

Fig: General structure of a clustered system.

Real-Time Systems: Special purpose operating system, it is used when there are rigid time requirements on the operation of a processor or the flow of data, thus it is often used as a control device in dedicated application.

Real time system need that the processing must be done within the defined time constraints or the system will fail.

There are two flavors of real time system:
1. Hard real-time system
2. Soft real time system
**Handheld Systems:** Handheld systems include personal digital assistants (PDAs). Developers of handheld systems and applications face many challenges (due to the limited size of such devices) such as speed of processor, limited size of memory, and small display screen.

4. **Operating-System Structure**
   1. Multiprogramming needed for efficiency
   2. Single user cannot keep CPU and I/O devices busy at all times
   3. Multiprogramming organizes jobs (code and data) so CPU always has one to Execute
   4. A subset of total jobs in system is kept in memory
   5. One job selected and run via job scheduling
   6. When it has to wait (for I/O for example), OS switches to another job
   7. Timesharing (multitasking) is logical extension in which CPU switches jobs so frequently that users can interact with each job while it is running, creating interactive computing
   8. Response time should be < 1 second
   9. Each user has at least one program executing in memory [process
   10. If several jobs ready to run at the same time CPU scheduling
   11. If processes don’t fit in memory, swapping moves them in and out to run
   12. Virtual memory allows execution of processes not completely in memory

**Memory Layout for Multiprogrammed System**

![Memory layout for a multiprogramming system.](attachment:image.png)

5. **Operating-System Operations**
   1. Interrupt driven by hardware
   2. Software error or request creates exception or trap
   3. Division by zero, request for operating system service
4. Other process problems include infinite loop, processes modifying each other or the operating system.
5. Dual-mode operation allows OS to protect itself and other system components.
6. User mode and kernel mode.
7. Mode bit provided by hardware.

**Dual-Mode Operation:**
1. To ensure proper operation, we must protect the operating system and all other programs and their data from any malfunctioning program.
2. The approach taken by many operating systems provides hardware support that allows us to differentiate among various modes of execution.
3. A bit, called the mode bit is added to the hardware of the computer to indicates the current mode: monitor (0) or user (1) with mode bit we could distinguish between a task that is executed on behalf of the operating system, and one that is executed on behalf of the user.
4. Provides ability to distinguish when system is running user code or kernel code.
5. Some instructions designated as **privileged**, only executable in kernel mode.
6. System call changes mode to kernel, return from call resets it to user.

**Transition from User to Kernel Mode**

1. Timer to prevent infinite loop / process hogging resources.
2. Set interrupt after specific period.
3. Operating system decrements counter.
4. When counter zero generate an interrupt.
5. Set up before scheduling process to regain control or terminate program that exceeds allotted time.

**Fig:** Transition from user to kernel mode.
Timer

1. Before the kernel begins executing user code, a timer is set to generate an interrupt.
2. The timer interrupt handler reverts control back to the kernel.
3. This assures that no user process can take over the system.
4. Timer control is a privileged instruction (requiring kernel mode).

OPERATING SYSTEM FUNCTIONS

6. Process Management

1. A process is a program in execution. It is a unit of work within the system. Program is a passive entity, process is an active entity.
2. Process needs resources to accomplish its task
3. CPU, memory, I/O, files
4. Initialization data
5. Process termination requires reclaim of any reusable resources
6. Single-threaded process has one program counter specifying location of next instruction to execute
7. Process executes instructions sequentially, one at a time, until completion
8. Multi-threaded process has one program counter per thread
9. Typically system has many processes, some user, some operating system running concurrently on one or more CPUs
10. Concurrency by multiplexing the CPUs among the processes / threads

Process Management Activities

1. The operating system is responsible for the following activities in connection with process management:
2. Creating and deleting both user and system processes
3. Suspending and resuming processes
4. Providing mechanisms for process synchronization
5. Providing mechanisms for process communication
6. Providing mechanisms for deadlock handling
7. Memory Management

1. All data in memory before and after processing
2. All instructions in memory in order to execute
3. Memory management determines what is in memory when
4. Optimizing CPU utilization and computer response to users

Memory management activities

1. Keeping track of which parts of memory are currently being used and by whom
2. Deciding which processes (or parts thereof) and data to move into and out of memory
3. Allocating and deallocating memory space as needed.

8. Storage Management: OS provides uniform, logical view of information storage

1. File system management
   1. Abstracts physical properties to logical storage unit - **file**
   2. Each medium is controlled by device (i.e., disk drive, tape drive)
   3. Varying properties include access speed, capacity, data-transfer rate, access method (sequential or random)
   4. File-System management
   5. Files usually organized into directories
   6. Access control on most systems to determine who can access what

   **OS activities include**
   1. Creating and deleting files and directories
   2. Primitives to manipulate files and directories
   3. Mapping files onto secondary storage
   4. Backup files onto stable (non-volatile) storage media

2. Mass-Storage Management

1. Usually disks used to store data that does not fit in main memory or data that must be kept for a “long” period of time
2. Proper management is of central importance
3. Entire speed of computer operation hinges on disk subsystem and its algorithms.

**Mass-Storage Management activities**

i. Free-space management
Storage allocation

Disk scheduling

Some storage need not be fast

Tertiary storage includes optical storage, magnetic tape. Still must be managed

Varies between WORM (write-once, read-many-times) and RW (read-write).

3. Caching

1. There are many cases in which a smaller higher-speed storage space serves as a cache, or temporary storage, for some of the most frequently needed portions of larger slower storage areas.

2. The hierarchy of memory storage ranges from CPU registers to hard drives and external storage.

3. The OS is responsible for determining what information to store in what level of cache, and when to transfer data from one level to another.

4. The proper choice of cache management can have a profound impact on system performance.

5. Data read in from disk follows a migration path from the hard drive to main memory, then to the CPU cache, and finally to the registers before it can be used, while data being written follows the reverse path. Each step (other than the registers) will typically fetch more data than is immediately needed, and cache the excess in order to satisfy future requests faster. For writing, small amounts of data are frequently buffered until there is enough to fill an entire "block" on the next output device in the chain.

6. The issues get more complicated when multiple processes (or worse multiple computers) access common data, as it is important to ensure that every access reaches the most up-to-date copy of the cached data (amongst several copies in different cache levels).

Data read in from disk follows a migration path from the hard drive to main memory, then to the CPU cache, and finally to the registers before it can be used, while data being written follows the reverse path. Each step (other than the registers) will typically fetch more data than is immediately needed, and cache the excess in order to satisfy future requests faster. For writing, small amounts of data are frequently buffered until there is enough to fill an entire "block" on the next output device in the chain.

The issues get more complicated when multiple processes (or worse multiple computers) access common data, as it is important to ensure that every access reaches the most up-to-date copy of the cached data (amongst several copies in different cache levels).
Performance of Various Levels of Storage

<table>
<thead>
<tr>
<th>Level</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>registers</td>
<td>cache</td>
<td>main memory</td>
<td>disk storage</td>
</tr>
<tr>
<td>Typical size</td>
<td>&lt; 1 KB</td>
<td>&gt; 16 MB</td>
<td>&gt; 16 GB</td>
<td>&gt; 100 GB</td>
</tr>
<tr>
<td>Implementation technology</td>
<td>custom memory with multiple ports, CMOS</td>
<td>on-chip or off-chip CMOS SRAM</td>
<td>CMOS DRAM</td>
<td>magnetic disk</td>
</tr>
<tr>
<td>Access time (ns)</td>
<td>0.25 – 0.5</td>
<td>0.5 – 25</td>
<td>80 – 250</td>
<td>5,000.000</td>
</tr>
<tr>
<td>Bandwidth (MB/sec)</td>
<td>20,000 – 100,000</td>
<td>5000 – 10,000</td>
<td>1000 – 5000</td>
<td>20 – 150</td>
</tr>
<tr>
<td>Managed by</td>
<td>compiler</td>
<td>hardware</td>
<td>operating system</td>
<td>operating system</td>
</tr>
<tr>
<td>Backed by</td>
<td>cache</td>
<td>main memory</td>
<td>disk</td>
<td>CD or tape</td>
</tr>
</tbody>
</table>

Migration of Integer A from Disk to Register

1. Multitasking environments must be careful to use most recent value, no matter where it is stored in the storage hierarchy

2. Multiprocessor environment must provide cache coherency in hardware such that all CPUs have the most recent value in their cache

3. Distributed environment situation even more complex

4. Several copies of a datum can exist.

4. I/O Subsystem

1. One purpose of OS is to hide peculiarities of hardware devices from the user

2. I/O subsystem responsible for

3. Memory management of I/O including buffering (storing data temporarily while it is being transferred), caching (storing parts of data in faster storage for performance), spooling (the overlapping of output of one job with input of other jobs)

4. General device-driver interface
5. Drivers for specific hardware devices.

9. Protection and Security

Protection – any mechanism for controlling access of processes or users to resources defined by the OS.

Security – defense of the system against internal and external attacks

1. Huge range, including denial-of-service, worms, viruses, identity theft, theft of service
2. Systems generally first distinguish among users, to determine who can do what
3. User identities (user IDs, security IDs) include name and associated number, one per user
4. User ID then associated with all files, processes of that user to determine access control
5. Group identifier (group ID) allows set of users to be defined and controls managed, then also associated with each process, file
6. Privilege escalation allows user to change to effective ID with more rights

10. Distributed Systems

1. A distributed system is one in which H/w or S/w components located at the networked computers communicate & co ordinate their actions only by passing messages.
2. A distributed systems looks to its user like an ordinary OS but runs on multiple, Independent CPU’s.
3. Distributed systems depends on networking for their functionality which allows for communication so that distributed systems are able to share computational tasks and provides rich set of features to users.
4. N/w may vary by the protocols used, distance between nodes & transport media. Protocols->TCP/IP, ATM etc. Network-> LAN, MAN, WAN etc. Transport Media-> copper wires, optical fibers & wireless transmissions.

Advantages of Distributed Systems:

1. Resource sharing.
2. Higher reliability.
4. Shorter response time
5. Higher throughput.
11. SPECIAL-PURPOSE SYSTEMS

Clustered Systems

1. Like parallel systems the clustered systems will have multiple CPU but they are composed of two or more individual system coupled together.
2. Clustered systems share storage & closely linked via LAN N/w.
3. Clustering is usually done to provide high availability.
4. Clustered systems are integrated with H/w & S/w. H/w clusters means sharing of high performance disk.

S/w clusters are in the form of unified control of a computer system in a cluster.

1. A layer of S/w cluster runs on the cluster nodes. Each node can monitor one or more of the others. If the monitored M/c fails the monitoring M/c take ownership of its storage and restart the application that were running on failed M/c.
2. Clustered systems can be categorized into two groups
   i. Asymmetric Clustering &
   ii. Symmetric clustering
   i. In asymmetric clustering one M/c is in hot standby mode while others are running the application. The hot standby M/c does nothing but it monitors the active server. If the server fails the hot standby M/c becomes the active server.
   ii. In symmetric mode two or more hosts are running the Application & they monitor each other. This mode is more efficient since it uses all the available H/w.
   iii. Parallel clustering and clustering over a LAN is also available in clustering.
   Parallel clustering allows multiple hosts to access the same data on shared storage.
   iv. Clustering provides better reliability than the multi processor systems.
   v. It provides all the key advantages of a distributed system.
   vi. Clustering technology is changing & include global clusters in which M/c could be anywhere in the world.

Real-Time Embedded Systems

1. Real time system is one which was originally used to control autonomous systems like satellites, robots, hydroelectric dams etc.
2. Real time system is one that must react to I/p & responds to them quickly.
3. A real time system should not be late in response to one event.
4. A real time should have well defined time constraints.

Real time systems are of two types
   i. Hard Real Time Systems
   ii. Soft Real Time Systems

1. A hard real time system guarantees that the critical tasks to be completed on time. This goal requires that all delays in the system be bounded from the retrieval of stored data to time that it takes the OS to finish the request.
2. In soft real time system is a less restrictive one where a critical real time task gets priority over other tasks & retains the property until it completes. Soft real time system is achievable goal that can be mixed with other type of systems. They have limited utility than hard real time systems.
3. Soft real time systems are used in area of multimedia, virtual reality & advanced scientific projects. It cannot be used in robotics or industrial controls due to lack of deadline support.
4. Real time OS uses priority scheduling algorithm to meet the response requirement of a real time application.
5. Soft real time requires two conditions to implement, CPU scheduling must be priority based & dispatch latency should be small.
6. The primary objective of file management in real time systems is usually speed of access, rather than efficient utilization of secondary storage.

**Multimedia systems:** Streams of data must be delivered according to time restrictions

**Handheld systems:**

1. PDAs, smart phones, limited CPU, memory, power
2. Reduced feature set OS, limited I/O.
12. Computing Environments

All above systems are used in variety of computing environments settings.

1. Traditional Computing.
2. Web-Based computing.

**Traditional computer**

1. Blurring over time
2. Office environment
   a. PCs connected to a network, terminals attached to mainframe or minicomputers providing batch and timesharing
   b. Now portals allowing networked and remote systems access to same resources
3. Home networks
   i. Used to be single system, then modems
   ii. Now firewalled, networked

**Client-Server Computing**

1. Dumb terminals supplanted by smart PCs
2. Many systems now servers, responding to requests generated by clients
3. Compute-server provides an interface to client to request services (i.e. database)
4. File-server provides interface for clients to store and retrieve files.
Peer-to-Peer Computing

1. Another model of distributed system
2. P2P does not distinguish clients and servers
3. Instead all nodes are considered peers
4. May each act as client, server or both
5. Node must join P2P network
   i. Registers its service with central lookup service on network, or
   ii. Broadcast request for service and respond to requests for service via discovery protocol.

6. Examples include Napster and Gnutella

Web-Based Computing

1. Web has become ubiquitous
2. PCs most prevalent devices
3. More devices becoming networked to allow web access
4. New category of devices to manage web traffic among similar servers: load balancers
5. Use of operating systems like Windows 95, client-side, have evolved into Linux and Windows XP, which can be clients and servers.
OPERATING SYSTEMS STRUCTURES

1. OPERATING SYSTEM SERVICES:
An OS provides services for the execution of the programs and the users of such programs. The services provided by one OS may be different from other OS. OS makes the programming task easier. The common services provided by the OS are

1. User interface - Almost all operating systems have a user interface (UI). Varies between Command-Line (CLI), Graphics User Interface (GUI), and Batch.
2. Program Execution:- The OS must able to load the program into memory & run that program. The program must end its execution either normally or abnormally.
3. I/O Operation:- A program running may require any I/O. This I/O may be a file or a specific device user can’t control the I/O device directly so the OS must provide a means for controlling I/O devices.
4. File System Interface:- Program need to read or write a file. The OS should provide permission for the creation or deletion of files by names.
5. Communication:- In certain situation one process may need to exchange information with another process. This communication May takes place in two ways.
   a. Between the processes executing on the same computer.
   b. Between the processes executing on different computer that are connected by a network. This Communication can be implemented via shared memory or by OS.
6. Error Detection:- Errors may occur in CPU, I/O devices or in M/y H/w. The OS constantly needs to be aware of possible errors. For each type of errors the OS should take appropriate actions to ensure correct & consistent computing.

OS with multiple users provides the following services,

a. Resource Allocation:- When multiple users logs onto the system or when multiple jobs are running, resources must be allocated to each of them. The OS manages different types of OS resources. Some resources may need some special allocation codes & others may have some general request & release code.

b. Accounting:- We need to 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 system efficiency.
c. Protection: Protection ensures that all the access to the system are controlled. Security starts with each user having authenticated to the system, usually by means of a password. External I/O devices must also be protected from invalid access. In multi process environment it is possible that one process may interface with the other or with the OS, so protection is required.

**A View of Operating System Services**

<table>
<thead>
<tr>
<th>user and other system programs</th>
</tr>
</thead>
<tbody>
<tr>
<td>GUI</td>
</tr>
<tr>
<td>user interfaces</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>system calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>program execution</td>
</tr>
<tr>
<td>error detection</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>services</th>
</tr>
</thead>
<tbody>
<tr>
<td>operating system</td>
</tr>
<tr>
<td>hardware</td>
</tr>
</tbody>
</table>

2. **User Operating System Interface - CLI**

1. Command Line Interface (CLI) or command interpreter allows direct command entry
   i. Sometimes implemented in kernel, sometimes by systems program
   ii. Sometimes multiple flavors implemented – shells
   iii. Primarily fetches a command from user and executes it

2. Sometimes commands built-in, sometimes just names of programs
3. If the latter, adding new features doesn’t require shell modification

**Bourne Shell Command Interpreter**
User Operating System Interface - GUI

1. User-friendly desktop metaphor interface
2. Usually mouse, keyboard, and monitor
3. Icons represent files, programs, actions, etc
4. Various mouse buttons over objects in the interface cause various actions (provide information, options, execute function, open directory (known as a folder))
5. Invented at Xerox PARC
6. Many systems now include both CLI and GUI interfaces
7. Microsoft Windows is GUI with CLI “command” shell
8. Apple Mac OS X as “Aqua” GUI interface with UNIX kernel underneath and shells available
9. Solaris is CLI with optional GUI interfaces (Java Desktop, KDE)

The Mac OS X GUI

3. System Calls

1. Programming interface to the services provided by the OS
2. Typically written in a high-level language (C or C++)
3. Mostly accessed by programs via a high-level Application Program Interface (API) rather than direct system call use. Three most common APIs are Win32 API for Windows, POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X), and Java API for the Java virtual machine (JVM).

**Example of System Calls:** Illustrates the sequence of system calls required to copy a file:

![System Call Diagram]

**Example of Standard API**

1. Consider the `ReadFile()` function in the
2. Win32 API—a function for reading from a file

```c
BOOL ReadFile (HANDLE file, LPVOID buffer, DWORD bytesToRead, LPOVERLAPPED ovl);
```

A description of the parameters passed to `ReadFile()`

1. `HANDLE file`—the file to be read
2. `LPVOID buffer`—a buffer where the data will be read into and written from
3. `DWORD bytesToRead`—the number of bytes to be read into the buffer
4. LPDWORD bytesRead—the number of bytes read during the last read
5. LPOVERLAPPED ovl—indicates if overlapped I/O is being used

API – System Call – OS Relationship

![Diagram showing the handling of a user application invoking the open() system call]

Fig: Handling of user application invoking the open() system call

```c
#include <stdio.h>
int main ()
{
    ...
    printf ("Greetings");
    ...
    return 0;
}
```

![Diagram showing the standard C library and system calls]

user node
standard C library
write ()
write () system call
System Call Implementation

1. Typically, a number associated with each system call
2. System-call interface maintains a table indexed according to these
3. Numbers
4. The system call interface invokes intended system call in OS kernel and returns status of the system call and any return values
5. The caller need know nothing about how the system call is implemented
6. Just needs to obey API and understand what OS will do as a result call
7. Most details of OS interface hidden from programmer by API
8. Managed by run-time support library (set of functions built into libraries included with compiler)

System Call Parameter Passing

1. Often, more information is required than simply identity of desired system call
2. Exact type and amount of information vary according to OS and call
3. Three general methods used to pass parameters to the OS
4. Simplest: pass the parameters in registers
   a. In some cases, may be more parameters than registers
5. Parameters stored in a block, or table, in memory, and address of block passed as a parameter in a register
   i. This approach taken by Linux and Solaris
6. Parameters placed, or pushed, onto the stack by the program and popped off the stack by the operating system
7. Block and stack methods do not limit the number or length of parameters being passed.
Parameter Passing via Table

Fig: Passing of Parameters as a table

4. Types of System Calls

1. Process control
   i. End, abort
   ii. Load, execute
   iii. Create process, terminate process
   iv. Get process attributes, set process attributes
   v. Wait for time
   vi. Wait event, signal event
   vii. Allocate and free memory

2. File management
   i. Create file, delete file
   ii. Open, close file
   iii. Read, write, reposition
   iv. Get and set file attributes

3. Device management
   i. Request device, release device
   ii. Read, write, reposition
   iii. Get device attributes, set device attributes
   iv. Logically attach or detach devices
4. **Information maintenance**
   
i. Get time or date, set time or date

ii. Get system data, set system data

iii. Get and set process, file, or device attributes.

5. **Communications**
   
i. Create, delete communication connection

ii. Send, receive messages

iii. Transfer status information

iv. Attach and detach remote devices.

**Examples of Windows and Unix System Calls**
Process Control

1. 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.

2. Processes must be created, launched, monitored, paused, resumed, and eventually stopped.

3. When one process pauses or stops, then another must be launched or resumed.

4. When processes stop abnormally it may be necessary to provide core dumps and/or other diagnostic or recovery tools.

5. Compare DOS (a single-tasking system) with UNIX (a multi-tasking system).

6. When a process is launched in DOS, the command interpreter first unloads as much of itself as it can to free up memory, then loads the process and transfers control to it. The interpreter does not resume until the process has completed.

MS-DOS execution

Because UNIX is a multi-tasking system, the command interpreter remains completely resident when executing a process. 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.

1. 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.

2. The child process then executes an "exec" system call, which replaces its code with that of the desired process.

3. 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").

**FreeBSD Running Multiple Programs**

![Diagram of process hierarchy]

**File Management**

1. File management system calls include create file, delete file, open, close, read, write, reposition, get file attributes, and set file attributes.

2. These operations may also be supported for directories as well as ordinary files.
Device Management

1. Device management system calls include request device, release device, read, write, reposition, get/set device attributes, and logically attach or detach devices.
2. Devices may be physical (e.g. disk drives), or virtual / abstract (e.g. files, partitions, and RAM disks).
3. Some systems represent devices as special files in the file system, so that accessing the "file" calls upon the appropriate device drivers in the OS. See for example the /dev directory on any UNIX system.

Information Maintenance

1. Information maintenance system calls include calls to get/set the time, date, system data, and process, file, or device attributes.
2. Systems may also provide the ability to dump memory at any time, single step programs pausing execution after each instruction, and tracing the operation of programs, all of which can help to debug programs.

Communication

1. Communication system calls create/delete communication connection, send/receive messages, transfer status information, and attach/detach remote devices.
2. The message passing model must support calls to:
3. Identify a remote process and/or host with which to communicate.
4. Establish a connection between the two processes.
5. Open and close the connection as needed.
6. Transmit messages along the connection.
7. Wait for incoming messages, in either a blocking or non-blocking state.
8. Delete the connection when no longer needed.
9. The shared memory model must support calls to:
10. Create and access memory that is shared amongst processes (and threads.)
11. Provide locking mechanisms restricting simultaneous access.
12. Free up shared memory and/or dynamically allocate it as needed.
13. Message passing is simpler and easier, (particularly for inter-computer communications), and is generally appropriate for small amounts of data.

14. Shared memory is faster, and is generally the better approach where large amounts of data are to be shared, (particularly when most processes are reading the data rather than writing it, or at least when only one or a small number of processes need to change any given data item).

**Protection**

1. Protection provides mechanisms for controlling which users / processes have access to which system resources.
2. System calls allow the access mechanisms to be adjusted as needed, and for non-privileged users to be granted elevated access permissions under carefully controlled temporary circumstances.
3. Once only of concern on multi-user systems, protection is now important on all systems, in the age of ubiquitous network connectivity.

5. System Programs

System programs provide a convenient environment for program development and execution. The can be divided into:

1. File manipulation
2. Status information
3. File modification
4. Programming language support
5. Program loading and execution
6. Communications
7. Application programs

Most users’ view of the operation system is defined by system programs, not the actual system calls

1. Provide a convenient environment for program development and execution
2. Some of them are simply user interfaces to system calls; others are considerably more complex
1. **File management** - Create, delete, copy, rename, print, dump, list, and generally manipulate files and directories

2. **Status information**
   1. Some ask the system for info - date, time, amount of available memory, disk space, number of users
   2. Others provide detailed performance, logging, and debugging information
   3. Typically, these programs format and print the output to the terminal or other output devices
   4. Some systems implement a registry - used to store and retrieve configuration information

3. **File modification**
   1. Text editors to create and modify files
   2. Special commands to search contents of files or perform transformations of the text

4. **Programming-language support** - Compilers, assemblers, debuggers and interpreters sometimes provided

5. **Program loading and execution** - Absolute loaders, relocatable loaders, linkage editors, and overlay-loaders, debugging systems for higher-level and machine language

6. **Communications** - Provide the mechanism for creating virtual connections among processes, users, and computer systems.
   i. Allow users to send messages to one another’s screens, browse web pages, send electronic-mail messages, log in remotely, transfer files from one machine to another
   ii. Most operating systems today also come complete with a set of **application programs** to provide additional services, such as copying files or checking the time and date.
   iii. Most users views of the system is determined by their command interpreter and the application programs. Most never make system calls, even through the API, (with the exception of simple (file I/O in user-written programs.)

6. **Operating-System Structure**
   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:
Simple 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.)

MS-DOS Layer Structure

UNIX

i. UNIX – limited by hardware functionality, the original UNIX operating system had limited structuring. The UNIX OS consists of two separable parts

ii. Systems programs

iii. The kernel

   a. Consists of everything below the system-call interface and above the physical hardware

   b. Provides the file system, CPU scheduling, memory management, and other operating-system functions; a large number of functions for one level.
Traditional UNIX System Structure

Layered Approach

1. 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.
2. 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.
3. 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.
4. 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.

Layered Operating System
Microkernels

1. 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.
2. Most microkernels provide basic process and memory management, and message passing between other services, and not much more.
3. Security and protection can be enhanced, as most services are performed in user mode, not kernel mode.
4. System expansion can also be easier, because it only involves adding more system applications, not rebuilding a new kernel.
5. Mach was the first and most widely known microkernel, and now forms a major component of Mac OSX.
6. 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.
7. Another microkernel example is QNX, a real-time OS for embedded systems.

Mac OS X Structure
Fig: Architecture of a typical microkernel

**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. 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 Modular Approach**
7. Operating-System Generation

1. OS may be designed and built for a specific HW configuration at a specific site, but more commonly they are designed with a number of variable parameters and components, which are then configured for a particular operating environment.

2. Systems sometimes need to be re-configured after the initial installation, to add additional resources, capabilities, or to tune performance, logging, or security.

3. Information that is needed to configure an OS include:
   i. What CPU(s) are installed on the system, and what optional characteristics does each have?
   ii. How much RAM is installed? (This may be determined automatically, either at install or boot time.)
   iii. What devices are present? The OS needs to determine which device drivers to include, as well as some device-specific characteristics and parameters.
   iv. What OS options are desired, and what values to set for particular OS parameters. The latter may include the size of the open file table, the number of buffers to use, process scheduling (priority) parameters, disk scheduling algorithms, number of slots in the process table, etc.

4. At one extreme the OS source code can be edited, re-compiled, and linked into a new kernel.

5. More commonly configuration tables determine which modules to link into the new kernel, and what values to set for some key important parameters. This approach may require the configuration of complicated make files, which can be done either automatically or through interactive configuration programs. Then make is used to actually generate the new kernel specified by the new parameters.

6. At the other extreme a system configuration may be entirely defined by table data, in which case the "rebuilding" of the system merely requires editing data tables.

7. Once a system has been regenerated, it is usually required to reboot the system to activate the new kernel. Because there are possibilities for errors, most systems provide some mechanism for booting to older or alternate kernels.
8. System Boot

1. The general approach when most computers boot up goes something like this:

2. When the system powers up, an interrupt is generated which loads a memory address into the program counter, and the system begins executing instructions found at that address. This address points to the "bootstrap" program located in ROM chips (or EPROM chips) on the motherboard.

3. The ROM bootstrap program first runs hardware checks, determining what physical resources are present and doing power-on self tests (POST) of all HW for which this is applicable. Some devices, such as controller cards may have their own on-board diagnostics, which are called by the ROM bootstrap program.

4. The user generally has the option of pressing a special key during the POST process, which will launch the ROM BIOS configuration utility if pressed. This utility allows the user to specify and configure certain hardware parameters as where to look for an OS and whether or not to restrict access to the utility with a password.

Some hardware may also provide access to additional configuration setup programs, such as for a RAID disk controller or some special graphics or networking cards.

5. Assuming the utility has not been invoked, the bootstrap program then looks for a non-volatile storage device containing an OS. Depending on configuration, it may look for a floppy drive, CD ROM drive, or primary or secondary hard drives, in the order specified by the HW configuration utility.

6. Assuming it goes to a hard drive, it will find the first sector on the hard drive and load up the fdisk table, which contains information about how the physical hard drive is divided up into logical partitions, where each partition starts and ends, and which partition is the "active" partition used for booting the system.

7. There is also a very small amount of system code in the portion of the first disk block not occupied by the fdisk table. This bootstrap code is the first step that is not built into the hardware, i.e. the first part which might be in any way OS-specific. Generally this code knows just enough to access the hard drive, and to load and execute a (slightly) larger boot program.
8. For a single-boot system, the boot program loaded off of the hard disk will then proceed to locate the kernel on the hard drive, load the kernel into memory, and then transfer control over to the kernel. There may be some opportunity to specify a particular kernel to be loaded at this stage, which may be useful if a new kernel has just been generated and doesn't work, or if the system has multiple kernels available with different configurations for different purposes. (Some systems may boot different configurations automatically, depending on what hardware has been found in earlier steps.)

9. For dual-boot or multiple-boot systems, the boot program will give the user an opportunity to specify a particular OS to load, with a default choice if the user does not pick a particular OS within a given time frame. The boot program then finds the boot loader for the chosen single-boot OS, and runs that program as described in the previous bullet point.

10. Once the kernel is running, it may give the user the opportunity to enter into single-user mode, also known as maintenance mode. This mode launches very few if any system services, and does not enable any logins other than the primary login on the console. This mode is used primarily for system maintenance and diagnostics.

11. When the system enters full multi-user multi-tasking mode, it examines configuration files to determine which system services are to be started, and launches each of them in turn. It then spawns login programs (gettys) on each of the login devices which have been configured to enable user logins.

   The getty program initializes terminal I/O, issues the login prompt, accepts login names and passwords, and authenticates the user. If the user's password is authenticated, then the getty looks in system files to determine what shell is assigned to the user, and then "execs" (becomes) the user's shell. The shell program will look in system and user configuration files to initialize itself, and then issue prompts for user commands. Whenever the shell dies, either through logout or other means, then the system will issue a new getty for that terminal device.