The concept of operating system

- A program that acts as an intermediary between a user of a computer and the computer hardware.
- Operating system goals:
  - Execute user programs and make solving user problems easier
  - Make the computer system convenient to use
  - Use the computer hardware in an efficient manner

Structure of OS

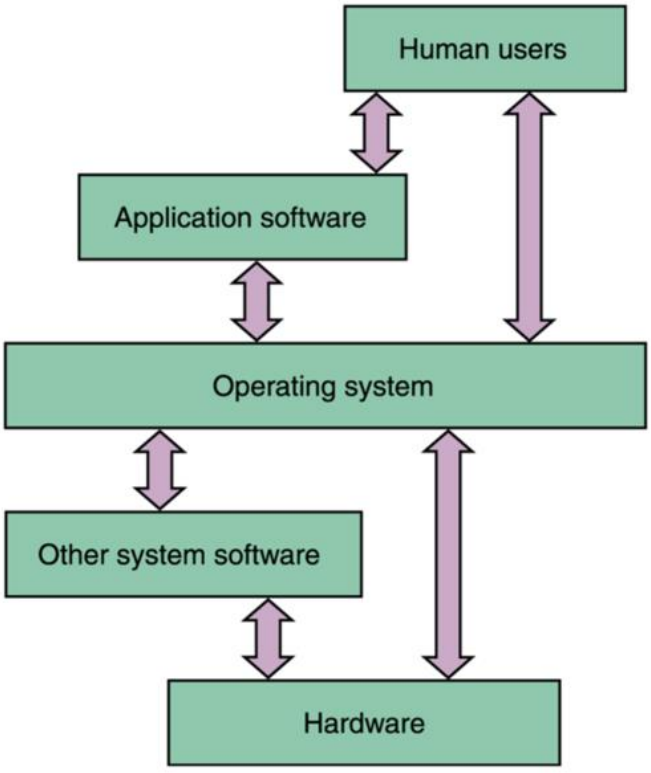
Computer system can be divided into four components:

• **Hardware** – provides basic computing resources CPU, memory, I/O devices

• **Operating system** - controls and coordinates use of hardware among various applications and users

• **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

• **Users** - People, machines, other computers



Classification of Operating Systems

Operating systems can be classified as follows:

- **Multi-user:** is the one that concede two or more users to use their programs at the same time. Some of O.S permits hundreds or even thousands of users simultaneously.
- **Single-User:** just allows one user to use the programs at one time.
- **Multiprocessor:** Supports running the same program in more than one CPU.
- **Multitasking**: Allows multiple programs running at the same time.
- **Single-tasking**: Allows different parts of a single program running at any one time.
- **Real time:** Responds to input instantly. Operating systems such as DOS and UNIX, do not work in real time.

Files and file systems

The file system contains Files and Directories. A file is a basic unit of storage for data. Represented by (-) in Command-line Interface (CLI). Linux uses directories to hold information about other files. Also, known as folders in Windows. Represented by (d) in the CLI.

Properties of a File System

- Files are stored on disk or other storage and do not disappear when a user logs off thus its non-volatile. Examples are magnetic disks, optical disks and tapes.
- Files have names and are associated with access permission that permits controlled sharing.
- Every File has a logical location where they are located for storage and retrieval.

Objective of File management System

- It provides I/O support for a variety of storage device types.
- Minimizes the chances of losing or destroying of the data
- Helps OS to standardized I/O interface routines for user processes.
- It provides I/O support for multiple users in a multiuser systems environment.

OVERALL TO REMEMBER ABOUT FILES

- A file is a collection of correlated information which is recorded on secondary or non-volatile storage like magnetic disks, optical disks, and tapes.
- A File Structure needs to be predefined format in such a way that an operating system understands it.

- File type refers to the ability of the operating system to differentiate different types of files like text files, binary, and source files.
- Indexed Sequential Access method is based on simple sequential access
- In Sequential Access method records are accessed in a certain pre-defined sequence
- The random-access method is also called direct random access
- Three types of space allocation methods are:
  - Linked Allocation
  - Indexed Allocation
  - Contiguous Allocation
- Information about files is maintained by Directories

Special files on Unix systems

Used to represent a real physical device such as a printer, tape drive or terminal, used for Input/Output (I/O) operations. Device or special files are used for device Input/Output(I/O) on UNIX and Linux systems. They appear in a file system just like an ordinary file or a directory. On UNIX systems there are two flavors of special files for each device, character special files and block special files:

- When a character special file is used for device Input/Output(I/O), data is transferred one character at a time. This type of access is called **raw device access**.
- When a block special file is used for device Input/Output(I/O), data is transferred in large fixed-size blocks. This type of access is called **block device access**.

For terminal devices, it's one character at a time. For disk devices though, raw access means reading or writing in whole chunks of data – blocks, which are native to your disk.

- In long-format output of ls -l, character special files are marked by the "c" symbol.
- In long-format output of ls -l, block special files are marked by the "b" symbol.

REDIRECTION

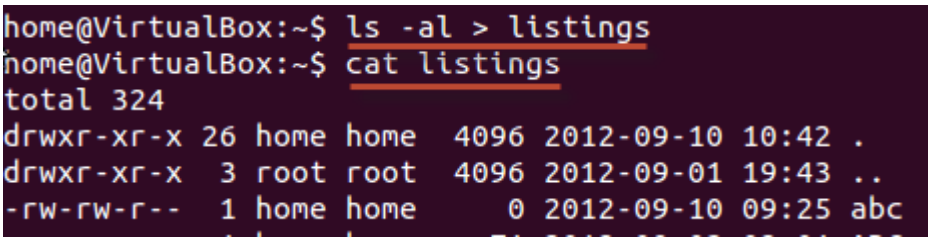
Redirection is a feature in Linux such that when executing a command, you can change the standard input/output devices. The basic workflow of any Linux command is that it takes an input and give an output.

- The standard input (stdin) device is the keyboard.

- The standard output (stdout) device is the screen.

Output Redirection – The '>' symbol is used for output (STDOUT) redirection.
Input Redirection – The '<' symbol is used for input(STDIN) redirection.

- Here the output of command **ls -al** is re-directed to file "listings" instead of your screen.



- If you do not want a file to be overwritten but want to add more content to an existing file, then you should use '>>' operator.

PIPES

UNIX allows you to link commands together using a pipe. The pipe acts a temporary file which only exists to hold data from one command until it is read by another. A Unix pipe provides a one-way flow of data. The output or result of the first command sequence is used as the input to the second command sequence. To make a pipe, put a vertical bar (|) on the command line between two commands. For example: **who | wc -l**

In long-format output of ls –l, named pipes are marked by the “p” symbol.

- Ordinary pipes – cannot be accessed from outside the process that created it. Typically, a parent process creates a pipe and uses it to communicate with a child process that it created.
- Named pipes – can be accessed without a parent-child relationship.

Process Management

- 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**.
- Process needs resources to accomplish its task
  - CPU, memory, I/O, files

    - Initialization data
- Process termination requires reclaim of any reusable resources
- Single-threaded process has one **program counter** specifying location of next instruction to execute
    - Process executes instructions sequentially, one at a time, until completion
- Multi-threaded process has one program counter per thread
- Typically, system has many processes, some user, some operating system running concurrently on one or more CPUs
- Concurrency by multiplexing the CPUs among the processes / threads
- **Activities -** The operating system is responsible for the following activities in connection with process management:
    - Creating and deleting both user and system processes
    - Suspending and resuming processes
    - Providing mechanisms for process synchronization
    - Providing mechanisms for process communication
    - Providing mechanisms for deadlock handling
- **Process State:** As a process executes, it changes state
    - new: The process is being created
    - running: Instructions are being executed
    - waiting: The process is waiting for some event to occur
    - ready: The process is waiting to be assigned to a processor
    - terminated: The process has finished execution

SIGNALS

A signal is an event generated by the UNIX and Linux systems in response to some condition. Upon receipt of a signal, a process may take action. A signal is just like an interrupt; when it is generated at the user level, a call is made to the kernel of the OS, which then acts accordingly.

There are two types of signals:

- **Maskable:** signals which can be changed or ignored by the user (e.g., Ctrl+C).
- **Non-Maskable**: signals which cannot be changed or ignored by the user. These typically occur when the user is signaled for non-recoverable hardware errors.

Examples:

Signal	Description
SIGHUP	Hang-up detected on controlling terminal or death of controlling process.
SIGINT	Issued if the user sends an interrupt signal (Ctrl + C).
SIGQUIT	Issued if the user sends a quit signal (Ctrl + D).

Task scheduling

Whenever using a UNIX-based operating system, certain tasks are to be performed repeatedly. Running them manually every single time is time-consuming and overall inefficient. To solve this issue, UNIX comes with its built-in task schedulers. These task schedulers act like a smart alarm clock. When the alarm goes off, the operating system will run the predefined task.

In the case of Linux, it comes with two basic but powerful tools: **Cron** daemon (default task scheduler) and **at** (more suitable for one-time task scheduling).

Cron

The cron daemon is responsible for running a lot of jobs at specific times. These tasks are generally run in the background at scheduled times. It offers great flexibility irrespective of the task, irrespective of the interval (hour, week, month, year, or whatever).

Cron keeps track of its actions using the crontab file. The crontab file is a script that contains all the necessary information to run all the cron jobs.

at

While cron is the primary way of task scheduling, at offers the ability to run a command/script at a specific time or at a fixed interval, note that at will run the target job once whereas cron would re-run the job at the interval. The at tool is less popular compared to cron, but it's relatively easier to use. You can use certain keywords like midnight or teatime (4 P.M.).