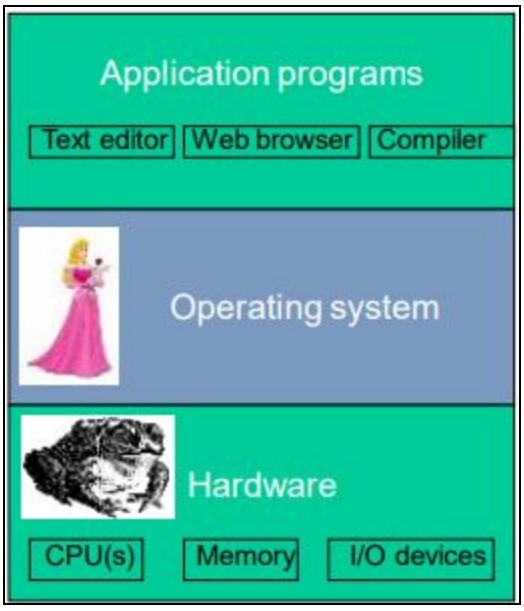
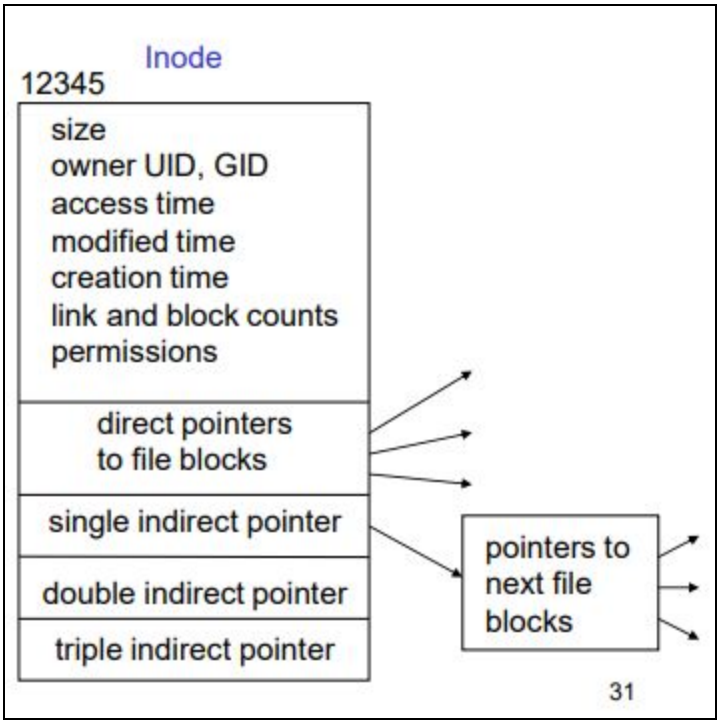
Week 1 Notes

1. **Operating Systems (OS):**
    - An **operating system** is the software layer between user applications and hardware.
    - It serves as a resource manager.
      I.e. It allows the proper use of resources like hardware, software and data.
    - It also serves as a control program (protection).
      I.e. It controls execution of user programs to prevent errors and improper use of the computer.
    - Turns ugly hardware into beautiful abstractions (file and directories).



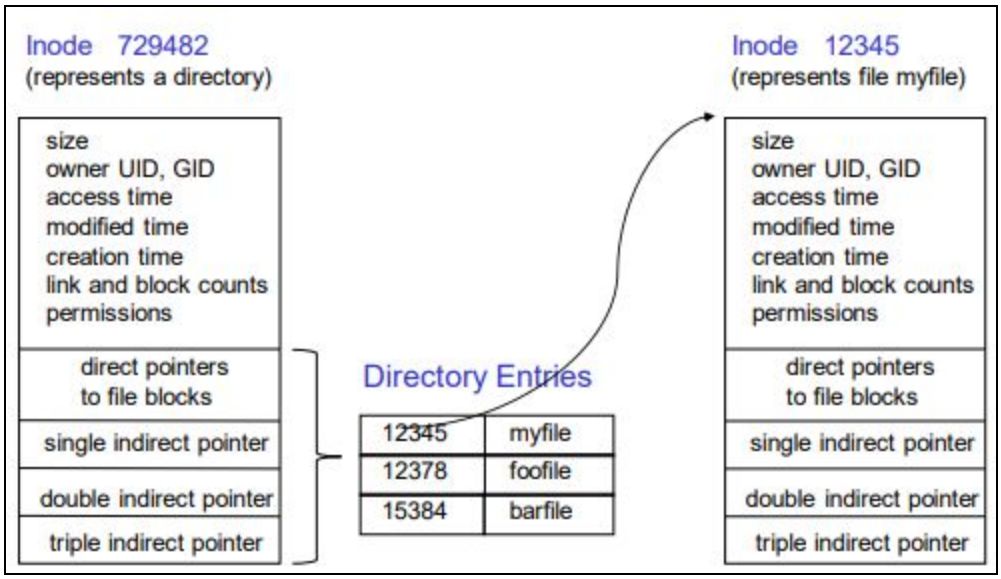
    - You can use Unix commands like cd, mv, cat, ls, etc to navigate directories and files.
    - You can use the man command to see what a command does.
2. **Files:**
    - A **file** is a name collection of data with some attributes:
        - Name
        - Owner (User and Group)
        - Size
        - Permissions
        - Time of creation
        - Last Access
        - Last Modification
        - Location on disk
    - To get info on a file in Unix, we can use:
        1. ls -l
        2. stat

- A data structure called **inode** stores the info of a file, including disk blocks which contain the file's data.



- A file is identified by its **inode number**.

3. **Directories:**
   - A **directory** is a collection of files and sub-directories.
   - In Unix, every directory is a file.
   - The **root**, denoted by /, is a special directory.
   - A directory entry maps a file name to an inode.



4. **Directory Hierarchy:**
    - The directory hierarchy is an **acyclic graph** because of **links**.
5. **Links:**
    - Sharing of files can be implemented by creating a new directory entry called a **link**, which is a **pointer** to another file or directory.
    - There are 2 types of links:
        1. **Hard Link:**
            - Created with ln <target> <name of link>
            - The second directory entry is identical to the first and shares the same inode number. Furthermore, both are files.
            - If the first directory entry is deleted, the second one is still there.
        2. **Soft Link:**
            - Created with ln -s <target> <name of link>
            - The second directory entry points to a small file containing the path of the first directory entry.
            - The second directory entry is a link, and has a different inode number than the first.
            - If the first directory entry is deleted, then we have a **dangling reference**.
6. **Permissions:**
    - **File permissions** have the follow setup: ----------
    - After the first dash, the next 3 dashes are for the owner. Then, the following 3 dashes are for the groups. Lastly, the final 3 dashes are for others.
      I.e. -(---)(---)(---)
      The red dashes are for the owner.
      The blue dashes are for the groups.
      The orange dashes are for others.
    - **Directory permissions** have the follow setup: d----------
    - After the d, the first 3 dashes are for the owner. Then, the following 3 dashes are for the groups. Lastly, the final 3 dashes are for others.
      I.e. d(---)(---)(---)
      The red dashes are for the owner.
      The blue dashes are for the groups.
      The orange dashes are for others.
    - Each entries specify 3 permissions, **read**, **write** and **execute**.

	read	write	execute
Denoted by	r	w	x
File Permission	Gives authority to open and read a file.	Gives authority to modify the contents of a file.	Gives authority to execute a file.
Directory Permission	Gives authority to run ls on a directory.	Gives authority to add, remove and rename files in a directory.	Gives authority to cd into a directory.

- Example:
  drw-rw-r-- means that:
  - The owner has read and write permissions, but not execute permissions on the directory
  - The group has read and write permissions, but not execute permissions on the directory.
  - Others have read permissions on the directory.
- You can use the command chmod to change permissions.
- Examples:
  1. chmod u+x fname will give the user who owns fname execute permissions.
  2. chmod g+r fname will give all users in group read permissions.
  3. chmod a+rwx will give all users all permissions.

7. <u>Shell:</u>

- A **shell** is a commandline interpreter.
- It is the interface between the user and an OS.
- The shell is a program that:
  1. Waits for input commands.
  2. Analyzes commands.
  3. Determines what actions are to be performed.
  4. Performs the actions.
- Shells can execute all the Unix commands, do I/O redirection, pipelining of commands, filtering output of commands, job control, shell programming and more.

8. <u>Input and Output Redirection:</u>

- Programs read from standard input (keyboard), write the results on standard output (screen) and write errors to standard error (screen).
- You can redirect input, output and errors using the following:
  1. > filename redirects the output to the file.
     I.e. It replaces the file's original text with the output.
  2. >> filename appends the output to the file.

3. < input file redirects input

- **Standard output (stdout)** is denoted by 1.
- **Standard error (stderr)** is denoted by 2.
- Examples
    1. ls > output.txt will overwrite the contents of output.txt with the output of ls.
    2. ls >> output.txt will append the output of ls to the contents of output.txt.
    3. ls -z 2>output.txt will overwrite the contents of output.txt with an error message, because ls -z is not a valid command.

9. Pipelines:

- Use | to send the output of one command to the input of another command.

10. Filters:

- A **filter** reads from standard input, processes the input and writes to standard output.
- Some useful filters
    - wc: count words, lines, characters
    - grep: filter lines that do or do not match a pattern
    - uniq: remove repeated lines
    - sort: sorts input
    - head: output only the first lines of the provided input
    - tail: output only the last lines of the provided input
    - sed: a stream editor to perform text transformations

11. Job Control:

- A **job/process** is a program in execution. Use ps to view processes.
- **Foreground job** has control of the terminal.
- **Background job** runs concurrently with the shell in the background.
- To run a program in the background, append & to the name of the program.
- At any point, a program can be suspended. Hit <ctrl> z to suspend the current foreground job.
- The command jobs gives you a list of jobs and each is associated with a number.
    - fg [num] puts job num in the foreground.
    - bg [num] puts job num in the background.
    - kill %num kills job num.