File Management

(i) File types

Regular
Device
Directory
Pipe

API's

↳ applications could be ported to different versions of OS

↳ Or OS that comply with POSIX standard

Source files, Binary files, script files
Read or written by normal programs

Directory files contain regular files
Represented as tree ds

Pipe files - could be used to read or write concurrently by two process for IPC

Device files provide interface to low level devices

Character device files provide acess to keyboard, display and printer

Block device files → disk files

Regular Files
→ Text file
→ Binary file
→ Text files → printed on screen or created and modified by text editor
→ Binary files → generated by programs or compilers
↳ Read by programs only

→ Executable files → execution rights are set

Directory files
→ File folder that contain other files
→ Created by mkdir command
$ mkdir <dirname>

Scanned with CamScanner

Unix maps all devices as files
The device types are classified as
(i) Major device type
(ii) Minor device type

Each of major device type is called as major device number

Devices are supported by device drivers
Device drivers provides interface between application programs and low level devices

Major device number:-
→ Index into kernel table of device driver addresses
→ Used to invoke the particular device driver

Minor device number:
→ Indicates one of the device instances for a given device
→ tty number, disk partition etc
→ Argument to the device driver function

Block device files

→ Written/Read one block at a time

→ Hard-disk drives, pen-drives, solid state drive

→ Block devices are accessed as block of bytes

```
ls -l /dev/sda0
     o brw-rw-rw- /dev/sda
         ↓
       block
```

Here we map the sectors into blocks of system

Creating a block device

→ `mknod /dev/dname b <major dev number> <minor dev number)`

Character device files.

→ Written or read one character at a time

→ Console terminals, keyboard. line printers serial ports.

→ /dev/tty <x> device file associated with the terminal

tty file → teletypes

- ls -l /dev /tty1

```
crw - rw - rw  /dev/tty1
↓
character
```

- $tty
  ↳ file associated is displayed.

```
# include <stdio.h>   // print1.c
main(){
    printf ("Hello, world on tty \n");
}
```

```
#include <stdio.h>   // print2.c
main(){
 FILE *fp= fopen(" dev /tty1 ", "w");
  fprintf (fp, "Hello, world on tty\n");
}
```

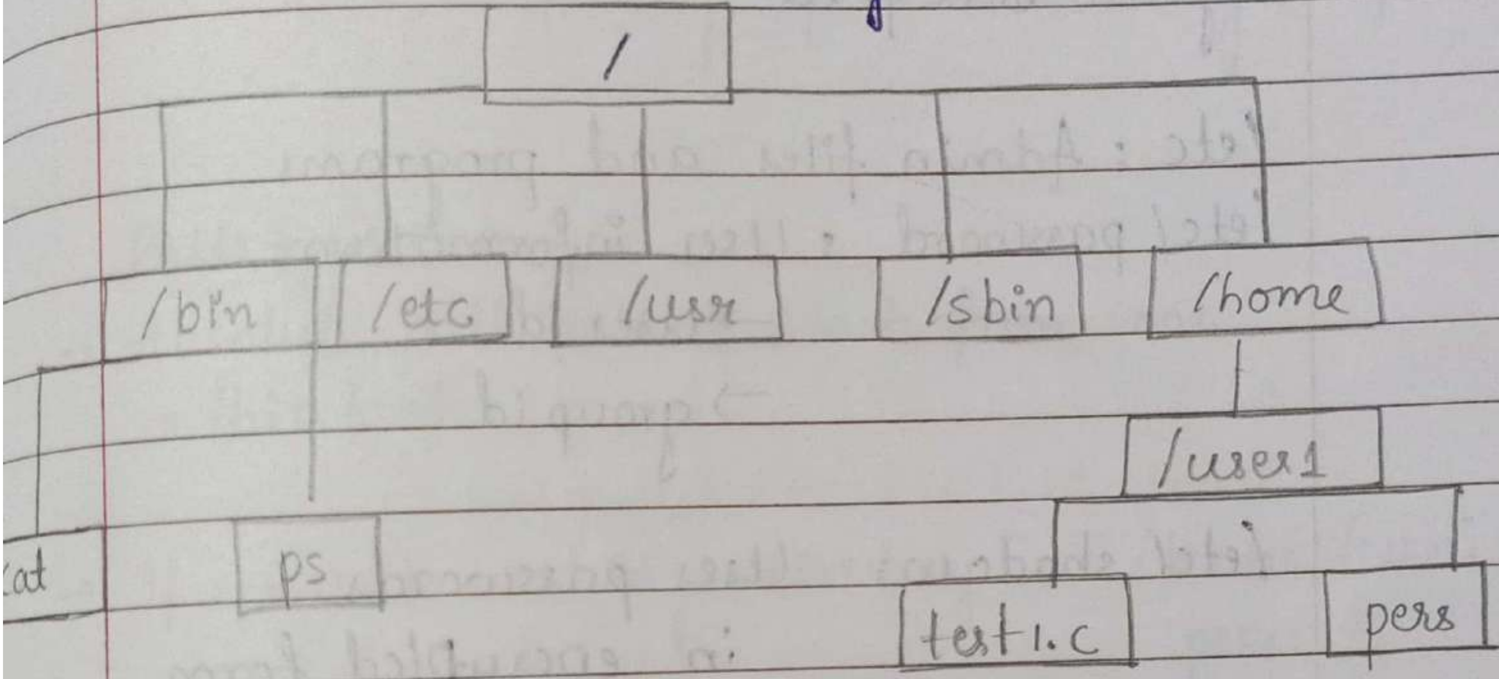
```
→ mknod / dev/dname  c <major dev number>
                       <minor dev number>
mknod /dev / tty5  c 4 5
```

By super user

DATE: / /

```
→ ls -l /dev/tty*
→ ls -l /dev/sda*
```

File hierarchy



root directory leafnode

It starts with root directory. It is inverted tree structure with root at the top and multiple directories at next level. These are typically system level directories. It contains system level files and utilities relevant to

Cat - print file on screen

ps- print process status

system administration.

File Attributes:-

File type → Type of file (c, b, p, link).

Acess permission → Acess permissions for owner, groups and others

Hardlink count → No of hard links to a file (incremente by ln command)

User ID (UID) → User ID of file owner

Group ID (GID) → Group ID of file owner

File Size → File size in bytes

Last acess time → Last time file was accessed

Last modify time → Time when file was modified for any of acess permission, UID, GID or hard link count

Inode number → System inode no. of file

File System ID → FILE System ID where the file is stored

Inode takes keeps track of all file

DATE: / /

File Attributes that can't be changed:-

File Inode number

File system ID - it is where physical data for file is present

File type

Major and minor device no's

The file access permissions can be changed by chmod command

```
$chmod 777 <filename>
  # change mode to 777.
```

```
$ ls-l <filename>
 -rwxrwxrwx
```

```
ln -s → softlink
```

```
* ls -i <dirname>
```

For inode listing

-OR-

```
ls -i <filename>
```

Directory files contain a link between filename and actual file using inode number

lseek - for random access in a file
fcntl - for specifying certain acess to the file
fstat, stat - for getting file details

0 → input
1 → output
2 → error

rc = reference count, how many process are accessing file

File API's:

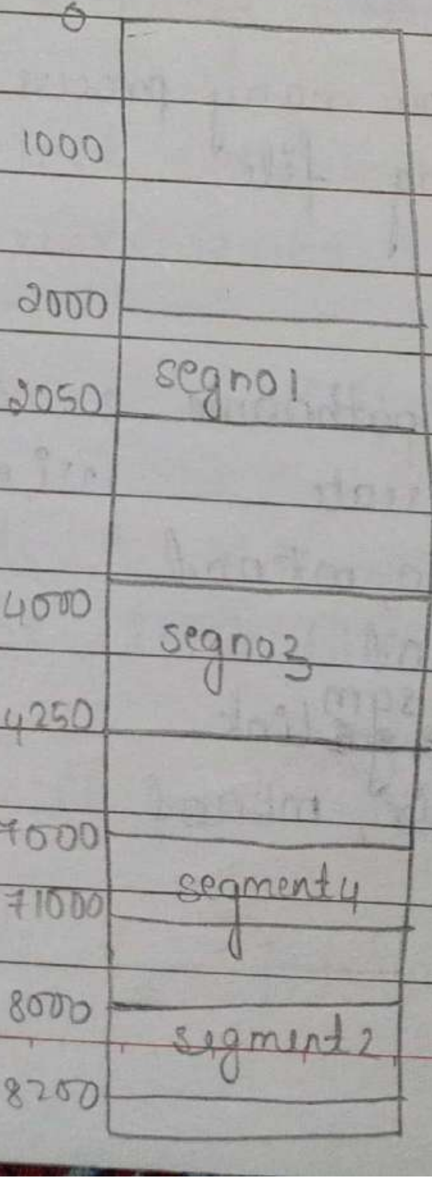
→ * File is identified by a pathname		command
Regular	open, create	vi, emac, pico
FIFO	mkfifo, mknod	mkfifo
device	mknod	mknod
symbolic files	symlink	ln-s
directory	mkdir, mknod	mkdir

Segment

segment no = 3   offset = 70
S = 3   d = 70.

Segment table

Segno	base	limit	Range
1	2000	50	2000-2050
2	8000	200	8000-8200
3	4000	250	4000-4250
4	7000	[illegible]00	7000-7100



offset less than limit

Physical address = base address of segno + offset (if less than limit)

TLB (time calculation) (Paging with TLB)

→ memory acess time = 100ms.

→ TLB access = 20ms.

→ hit ratio = 70%.

→ Effective memory acess time = hit ratio x(100 + 20)+ (1 - hit ratio) * (100 + 100 + 20)

For given set of programs the logical addres is generated from CPU and is converted to physical address (address binding) using some specified technique then we refer these physical address to fetch the Statements from memory and bring them to CPU.

= 0.7 (20 + 100) + 0.3 (20 + 100 + 100)

Page fault -> when referred page is not in main memory so it must be brought from secondary memory.

FIFO => Page fault rate = $\frac{\text{No. of page faults (5) (+)}}{\text{No of references (7)}}$

LRU => Least recently used

(Reference strings are page no referred by program)

2 5 3 4 7 2

2	2	2	3	3	2
	5	5	5	7	7
		3	4	4	4
+	+	+	+	+	+

Page fault rate = $\frac{7}{7} = \underline{\underline{1}}$