Examples:-
* race conds , val++, val--
* prevent race conds using mutex
* prevent concurrent execution of for loop, using mutex
* using unnamed semaphore for race conds, concurrent examples
---
* named semaphore

producer - consumer scenario
* a process/thread will add data -- producer
* a process/thread will remove data -- consumer
* common buffer / data source

* either producer or consumer, only can access common data at a time (shared resource) -- mutual exclusion - R1
* consumer should block if buffer empty - R2
* producer should block if buffer full - R3

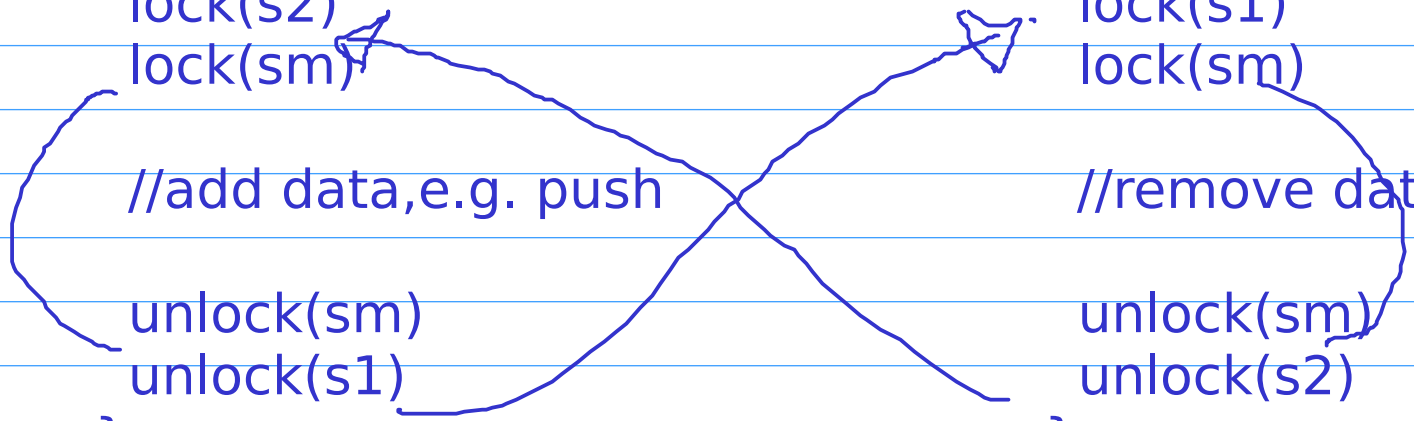
sm -- semaphore, init to 1 / mutex -- mutual exclusion
s1 -- semaphore, init to 0 - R2 (cons block if buffer is emp
s2 -- semaphore, ival? size of buffer - R3 (prod block if buffer is full)

Producer

```
while(1)  {
  lock(s2)
  lock(sm)

  //add data,e.g. push

  unlock(sm)
  unlock(s1)
}
```

Consumer

```
while(1)   {
  lock(s1)
  lock(sm)

  //remove data, e.g. pop

  unlock(sm)
  unlock(s2)
}
```



Bounded Buffer Problem
Unbounded Buffer Problem -- producer never blocks

s2 val -- free slots in buffer
s1 val -- filled elements in buffer

Producer

```
while(1)  {
  lock(s2)
  lock(sm)
  //.....
  unlock(sm)
  unlock(s1)
```

Consumer

```
while(1)   {
  lock(sm)
  lock(s1)        //?? : dead lock
  //....
```

Avoid - Apply mutual exclusion, after resolving dependencies

Spinlock
Condition variables

Spinlock vs Semaphore
==> if cond is not met, Semaphore will block the process
==> if cond is not met, Spinlock will hold proces in a busy loop (spinning)

Spinlock:- flag=0;

lock:-
    while(flag);
    flag=1;
unlock:-
    flag=0;

P1 | P2

lock(s1) | lock(s1)

//critical | //critical

unlock(s1) | unlock(s1)

Atomic instructions:-
XCHG or SWP

flag=0
r1 <--1
while(XCHG(r1,flag)); //perfect soln

H/w level atomic execution
* disable switching
* data bus locking to avoid
  variable access by other CPUs

If critical section is small + SMP
    ==> spinlocks
if critical section is big (or) no SMP
    ==> semaphores

File System Management:-

File System? Organization of files & dirs, content in files

create, modify, delete, rename, copy, move files

File Systems Impl -->

Real/Physical Systems:-
    FAT32, NTFS, ext2, ext3, ext4, XFS, ZFS, HFS, reiserfs, cramfs
Pseudo Filesystems:-
    procfs, sysfs, devfs, debugfs, tmpfs
    { /proc /sys /dev /sys/kernel/debug, /var/tmp }

File System -- Logical Disk/Storage -- Physical
                        (SATA, IDE/PATA, SSD, USB Storage)
FS : Blocks, Disk: Sectors

Virtual File System(VFS) -- gateway for all other file systems(common bridge

uspsace read --> vfs read --> fat32 read --> sata read
                                  --> xfs read --> ssd read

format ?? making new file system
mounting

5 parts on disk:- /dev/sda1, /dev/sda5, /dev/sda6, /dev/sda7, /dev/sda8
C: D: E: F: G:
Pen drive with no part:- /dev/sdb
(or) pen drive with two parts:- /dev/sdb1, /dev/sdb2

/ -- root partition (sda7)

/dev/sda1 -- /mnt/c mount /dev/sda1 /mnt/c
/dev/sda5 -- /mnt/d
/dev/sda6 -- /mnt/e
/dev/sda8 -- /mnt/g

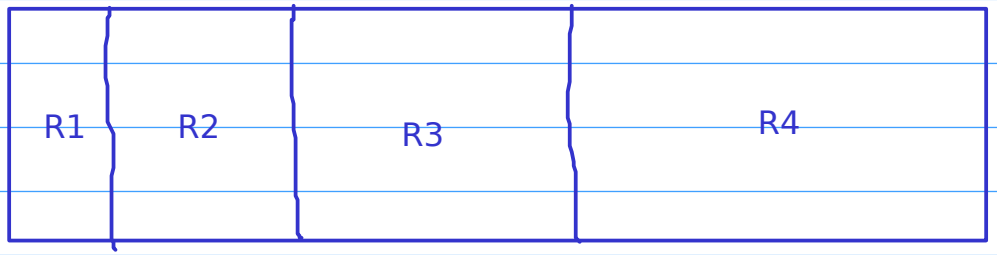
/dev/sdb -- /mnt/usb mount /dev/sdb /mnt/usb

umount /mnt/c
umount /mnt/c

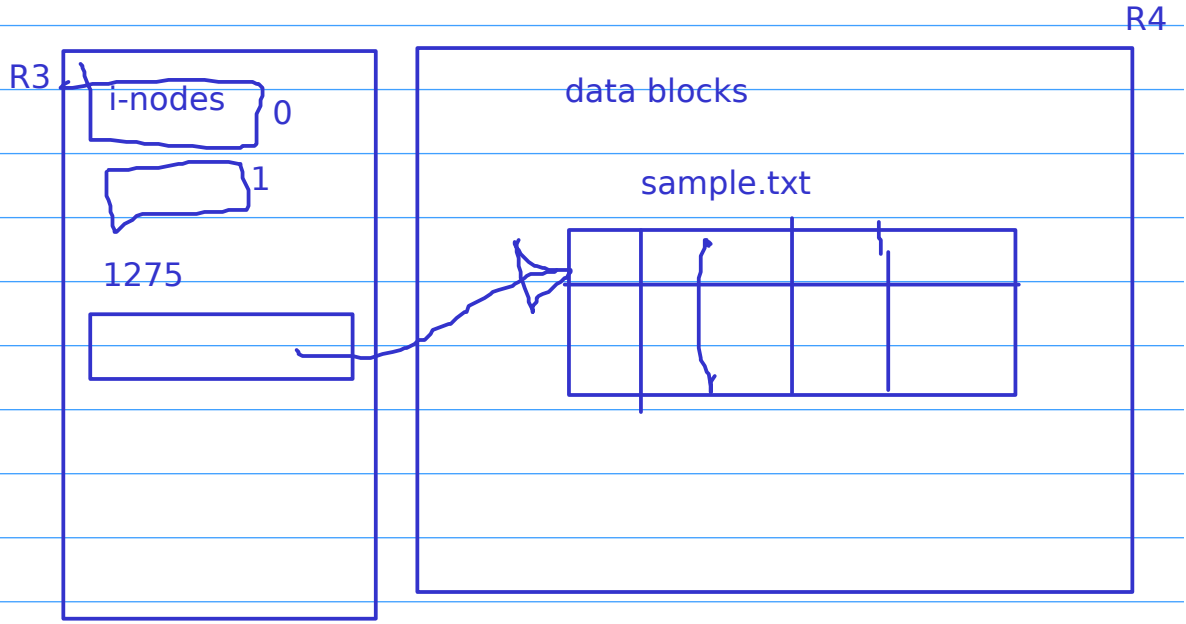
Automount ==> /media/xxx

FS Layout:-

R1 -- boot sector/block
R2 -- super block
R3 -- inode block
R4 -- data blocks



Block size, e.g. 4 KB
File of 40 KB



i-node:-
data structure/descriptor
holding file attributes
file descriptor

ls -i ==> inode numbers
ls -li ==> inode numbers + file details

File Attributes Stored in i-node:-
* block info (of file)
* permissions
* type of file (regular, dir, symlink, pipe/fifo, char spl, block spl)
* user, group
* file size
* timestamps - mtime, atime, ctime

ls -l command (or) lstat example

file name vs inode number mapping ==> directory

/home/rajesh/os/test/sample.txt

inode no. of / ==> 2
    ls -lid /

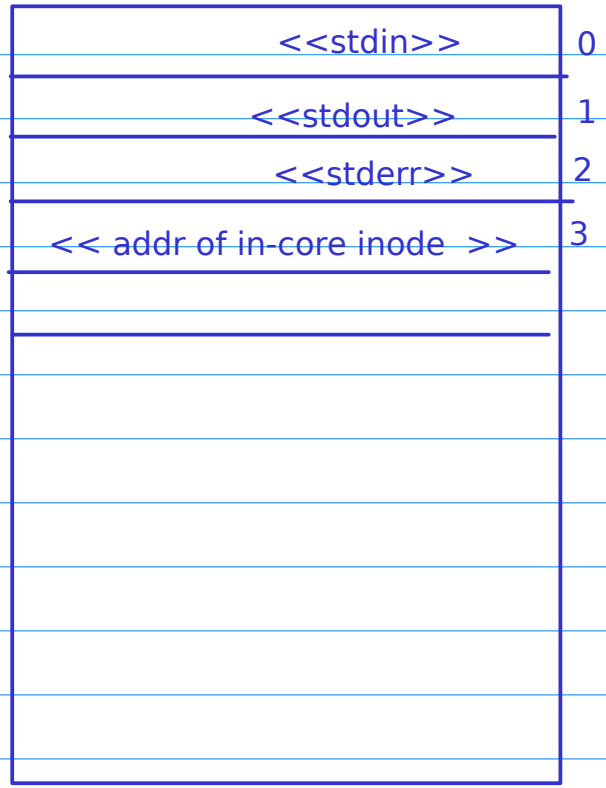
Self Study:-
    hard & soft links, ln command
    lstat system call
    chmod, chown, umask -- permissions & ownership

file handling -- open, read, write, close

write:-

```
fd=open("sample.txt",O_WRONLY);                 //O_WRONLY|O_CREAT, 0666
char msg[]="Hello Linux";
nbytes=write(fd, msg, len);
close(fd);
-------------------------
//alpha.txt:-      ABCD..XYZ0123..9          //abcd...xyz
fd=open("sample.txt",O_RDONLY);
char buf[128];
int maxlen=128;
nbytes=read(fd,buf,maxlen);          //nbytes:36
//print buf
close(fd);
-------------------------
int buflen=10;
read(fd, buf, buflen);                      //AB..IJ,     offset:10, ret:10
read(fd, buf, buflen);                      //KL..ST,   offset:20          ret:10
read(fd, buf, buflen);                      //UV...0123, offset:30       ret:10
read(fd, buf, buflen);                      //456789, offset:36  ret:6
```

```
read(fd, buf, buflen);
//??, ret will be zero
-------------------------------------------
while(1) {
   nbytes=read(fd,buf, buflen);
   if(nbytes<0) perror("read");
   if(nbytes==0) break;
   write(1, buf, nbytes);
}
-------------------------------------------
char ch;
read(fd, &ch, 1);
----------------------------
fd = open("alpha.txt",O_RDONLY);

target inode will be copied to
memory, addr of memory
inode is kept in open fd table

nbytes=read(fd, buf, len);

close(fd);
```



ulimit -n # max files

alpha.txt ==> 36 bytes

fd = open("alpha.txt", O_RDONLY);

```
nbytes=read(fd,buf, 10);                //off:10, AB...IJ
lseek(fd, 15, SEEK_SET);        //off:15
nbytes=read(fd,buf,6);          //buf:PQRSTU, off:21
lseek(fd, -8, SEEK_CUR);        //off:13
nbytes=read(fd,buf,5);          //buf:NOPQR, off:18
lseek(fd,-12, SEEK_END);                //off:24    (36-12)
```

SEEK_SET
SEEK_CUR
SEEK_END

---

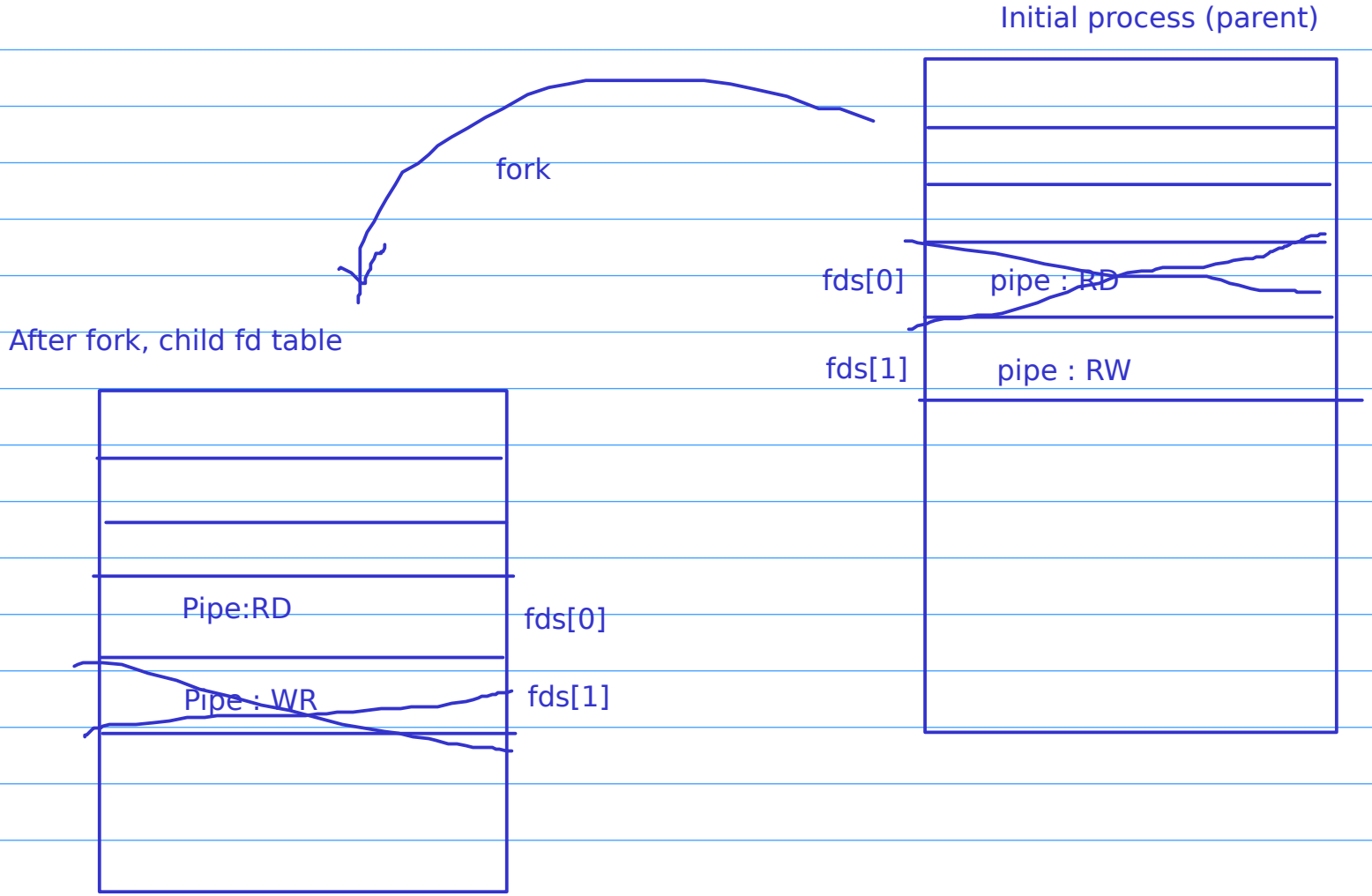
pipe / fifo:-

* used for IPC (data exchange)
* pseudo file
* one process can write (one end), othe process can read(other end)
* stream of data, FIFO order
* reading an empty pipe ==> block process

unnamed pipe (pipe) -- no file name, memory inode exists
                     -- applicable for related processes only (parent-child)

named pipe (FIFOs) -- file name applicable, disk inode exists
                     -- but data blocks in memory
                     -- applicable for any two processes

```
int fds[2];
ret=pipe(fds);  //fd[0] -- read, fds[1] -- write
```



TODO:-
Named Pipes (FIFOs)

Further Topics:-

Message Queues
Virtual Memory
Shared Memory
Scheduling