Final term.

Day:_____ Date:_____

int select (------ , struct timeval *t

tv_sec tv_usec

struct timeval myt;

myt.tv_sec=5; myt.tv_usec=10

select (------ &myt) // for time delay only.

FILTERS: head, tail, more, grep, sort.

input redirect

pipe

ls |head ls |tail man ls - print manual of ls

output redirect

cat f1.txt |more → shows first page.

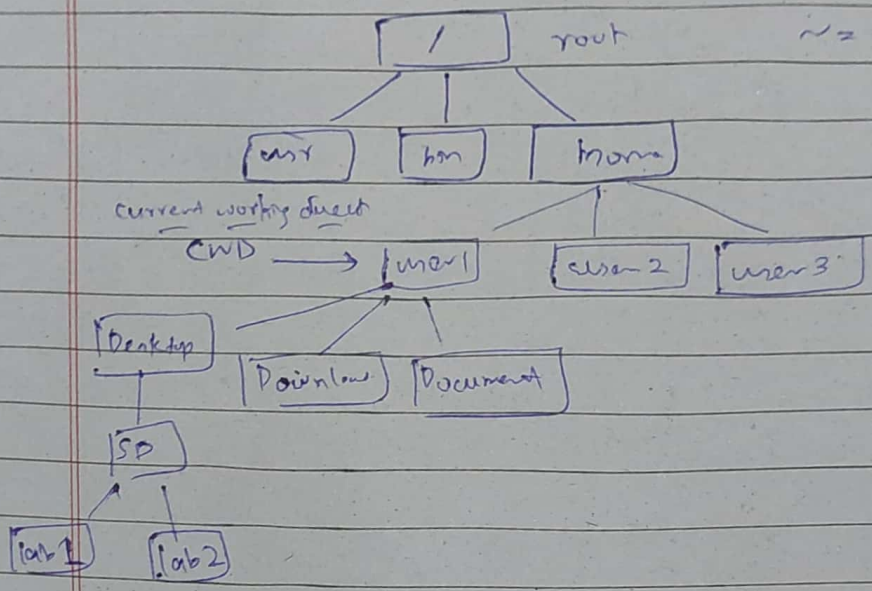
cat f1.txt |grep "hello" → see no of hello

ls |sort |head → sort ls & print head only.

ls |grep "f1" → print only f1. file

Day:_____ Date:_____

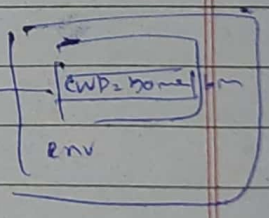
Chapter 5:
Files & Directories.



cd Desktop/SP. } relative add.
cd ./Desktop/SP. } CWD user1.
cd ../user1/Desktop/SP. }
cd /home/usr1/Desktop/SP } Absolute add.

/ → root → absolute add.
. or .. , space → relative add

print current directory & process id




```
main (){
    char *val;
    val = getenv("PWD");
    printf("My CWD = %s \n", val);
```

getcwd() → for current direct knowing

```
char buff[255];
char *ret = getcwd(buff);
printf("CWD = %s \n", buff);
```

ret pointing /home/usr1

*ret, buff both pointing same location

nyl

Day:_____ Date:_____

int chdir(const char *path);

success = 0 error = -1

chdir("./Desktop/SP");
ret = getcwd(buff);
printf("curr dir after chdir : %s\n", buff);
}

Directory = special file → contains directory entries.

dirent

↳ Name
↳ inode no.

SP.

lab1.	123
lab2	512
test.c	611
manual dir	300
. (current directory)	550
.. (parent directory)	1123
NULL	

DIR * opendir(const char *path);
struct dirent *readdir(DIR *dirp) → return dirent pointer

d_name d_ino.

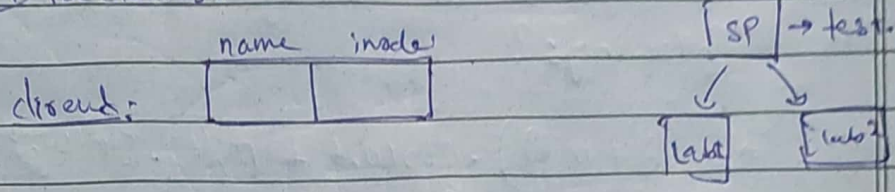
int main(){
DIR *dirp = opendir("SP");
struct dirent *direntp;
direntp.

while((direntp = readdir(dirp)) != NULL)
printf("name %s inode %d", direntp->d_name, direntp->d_ino);

Date 30/1/23 CW M T W T F S

SYSTEMS PROGRAMMING.

Directories:

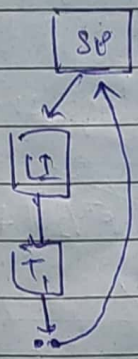


ls -l implementation.

drwxrwxrwx directory all perms allowed.
-xwxrwxr-- file.

- directory has minimum of 2 links.

So



so SP will have .., ., and T1 links

- ls -l returns directory related stats. and we can fetch this data using "stat" system call.

SYNOPSIS:- struct

- int stat (const char* path, struct stat* statbuf

in

#include <sys/stat.h>

#include ~~two~~ <time.h>
has ctime() to get string rep of stat_ctime datatype.

Date ______ M T W T F S

Structure of "stat" struct.

```
main() {
    DIR* myDir = opendir(".");
    struct dirent *mydirent;
    struct stat statbuff;

while((mydirent = readdir(mydir)) != null) {

    printf("Name=%s\n", mydirent->d_name);
    stat(mydirent->d_name, &statbuf);

    printf("Time = %s\t", ctime(&statbuff.st_atime));

    printf("Size=%d\t, gid = %d\t, links = %d\t",
      statbuff.st_size, statbuffer.st_gid,
      st_gid, statbuff.st_nlinks);

// find out type of file

if(S_ISDIR(statbuff.st_mode))
    printf("d");
else
    printf("_");
// Permission for User
if (S_IRUSR & statbuf.st_mode)
}
    //got it

}
for oth grp, and OTH
    Same
```

struct Stat:

0 int std_dev
1 int st_ino
2 int st_mode
3 int st_size
4 int st_nlinks,
5 int st_uid,
6 int st_gid,
time st_ctime,
time st_mtime,
time st_atime

info given by "mode"

Type of file
S_ISDIR(m) → mode
S_ISREG(m)

mode is 32 bit int where each bit represents different information.

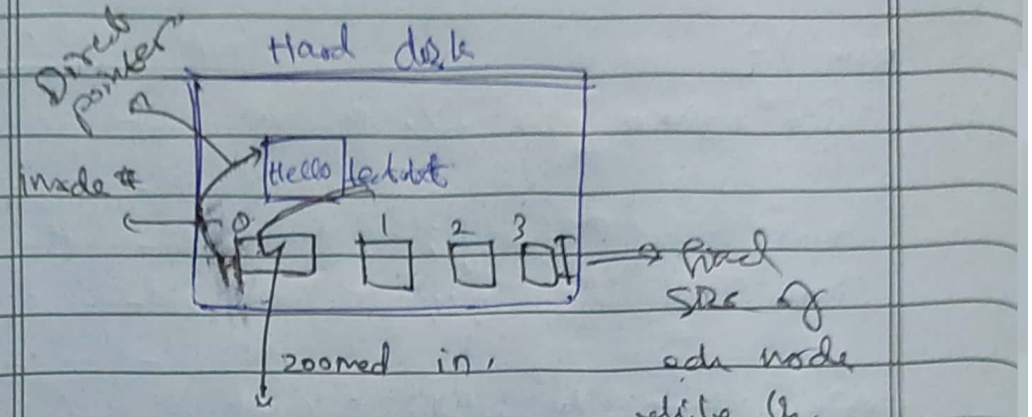
use bit wise "and" &

AND of all -1

Castelli

Date M T W T F S

Permission Retrieval from the Disk



Find size of each node while the last is extensible.

name can be accessed via the direct of the file Directory entry

Disk is managed in blocks in Unix.

and assume size of FB

⇒ Run size accessed R 16 KB effectively.

Castelli

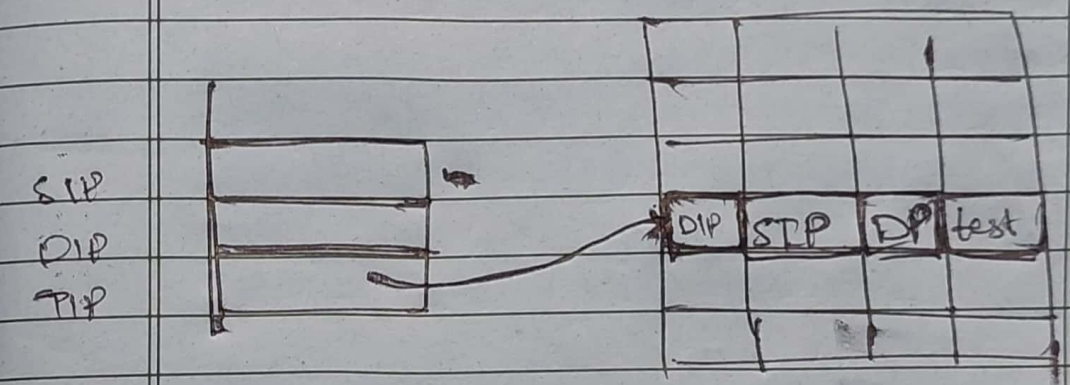
Date M T W T F S

memory for DP => 128 - 40 = 88 => 7GB
so total
available pointers are : (19)

So max file size can be 19 x 8 => 152 KB.

=> conclusion: optimize and choose b/w
=> its Block size and DP size.

OR Enter • 2048 • or Double indirect pointer.

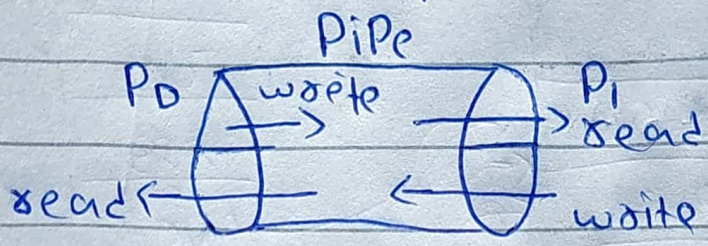


Reduce each level as you go up.

•

Lec #4

Pipes: Inter-process Communication:



- Pipes are used for inter Process Communication.
- if P0 writes Data the P1 will read & Vise Versa.
- if theres only one leg of P0 So it read & write on itself.

- Pipe is a special type of file.
- Un-named
- file is Destroyed after no process have opeaned it.
- Data is erased from write, when the other Process read

Pipe is two way communication.

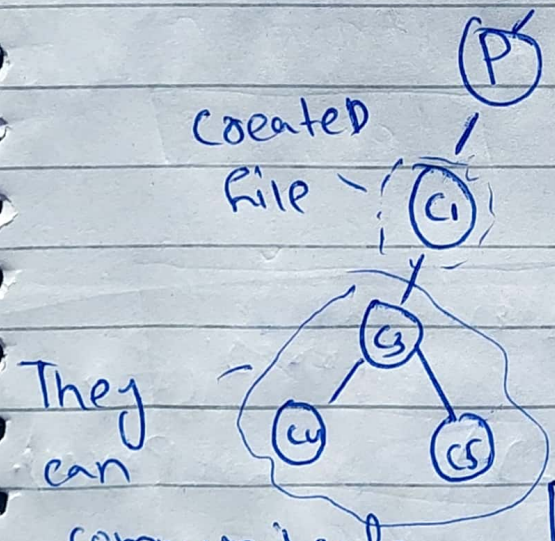
Data erased after it is read.

unnamed → Destroyed if no Process have opened it for w/R.

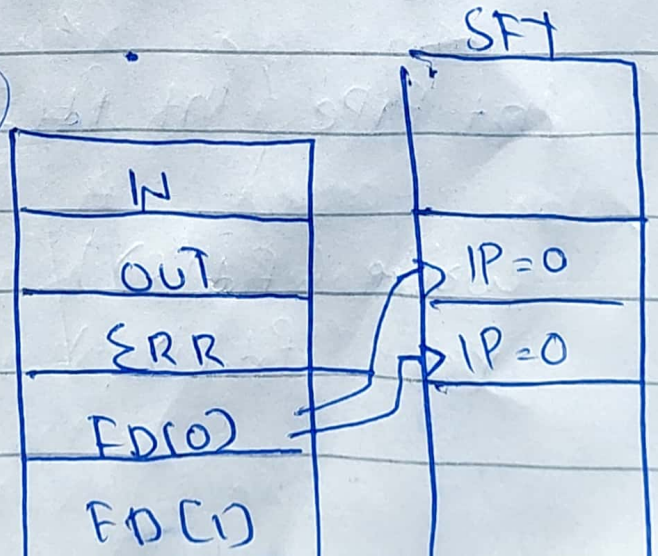
Special file

- limitations :-

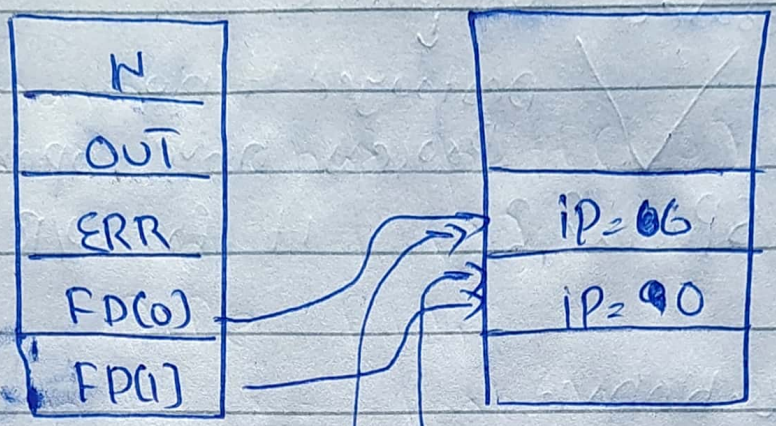
if the process which has created the pipe. So the Parent of that file could not communicate. (because of inheritance) Cannot communicate.



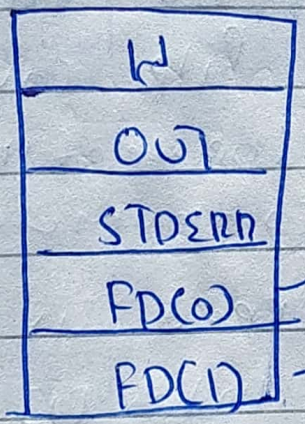
- read is in index [1], & write is in index [0].



The Process 1 where stop writing
So other Process Process will write
from next SFT



fork();



- special file always open in blocking mode

Program

2 element array (two Process).

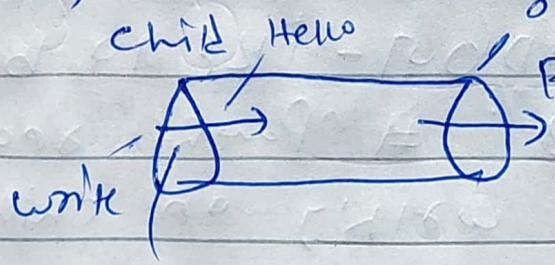
int Pipe (int fd(2))

write (fd(0)...)

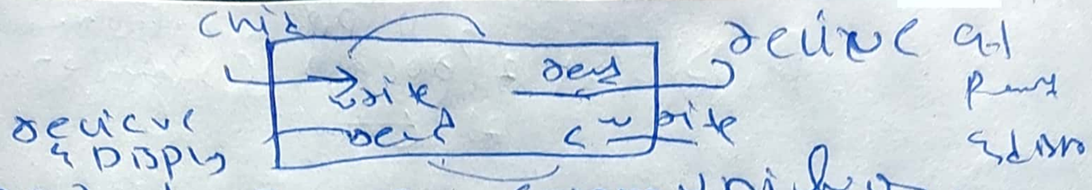
read (fd(1)...)

Steps: 1) Process creation
2) ~~Exit~~

```
int main()
{                                  Taking 2 element array.
1)      int fd[2];
        int r = Pipe(fd);
        if (r==-1) { Perror("pipe Failed");
          return -1;

2)      int x = fork();
        if (x==0)
          write(fd(0), "Hello", 6);
        else {
              char buff[100]
        br = read (fd(1), buff, 100);
          Printf("Parent recieved
                 %s\n", buff);
        }
```



Destroy ~~<~~ after read.

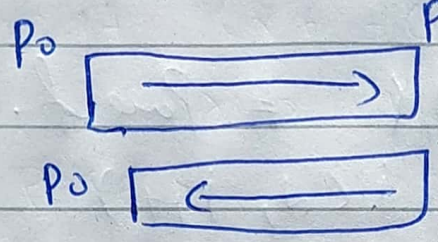


now two way communication

```
main ()
{       char buff(100);
      int  fd(2];
     Pipe (fd);
   int  x = fork();
   if  (x==0) {   // child
      int br = read (STDIN-FILE NO
                         buff, 100);
  cs    write (fd(0), buff, br);
          sleep(2)
# for  Recieving  at  child
        br = read  (fd (1), buff, 100),
     write ( STDOUT - FILE NO, buff, br)
  }        Display  at child.         ;
else
    #  Reaciving  at  Parent.
     int br = read (fd (1), buff, 100)
Disp write ( STDOUT - FILE NO, buff, br);
 br  =          # Parent read
 & now  waiting  on  child.
   br = read  (STDIN -FILE NO, buff,
                     100 );
     write ( fd (0), buff, bd );
}
```

we want To Do content Switch So the child writy Data should not read it itself. (sleep (2).

. if we want that this problem should neva come So we will make two pipes (because we dont know how much time it should sleep).



3 way communication.

Print Hello 3 times

So (6) ⟷ 2

⟷ 2

⟷ 2

26

```
main()
{
    int fd[2];
    Pipe(fd);
    int d = 3;
for (i=0; i<d; i++){
    x = fork();
    if (x==0)
        break;
}
    if (x > 0){  # Parent
    for (i=0; i<d; i++)
    write(fd[0], "Hello", 6);
    }     # sleep(2)
else {
        char * buff(100);
    br = read(fd[1], buff, 100);
    Printf("Child recieved
        %s\n", buff);
```

In this Parent write Hello 3 times & 3 child read then, sleep & 6 (byte) for no end.

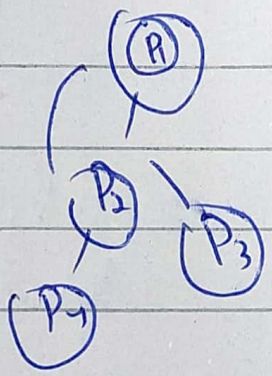
after mids.

11/12/23 SPlecH5

(FIFO)

filess / named pipes:- (First in First out).

- FIFO are named files.
- in FIFO pipes you can communicate b/w two ~~Parent~~ independent (Parent & child)



- your can commont b/w all.

- FIFO are not destroyed automatically. but we have to do it manually.

Steps:-

1) make / create FIFO
Sys call: int mkfifo (error = -1)
(Success = 0)

mkfifo (const char *name, int Permission);

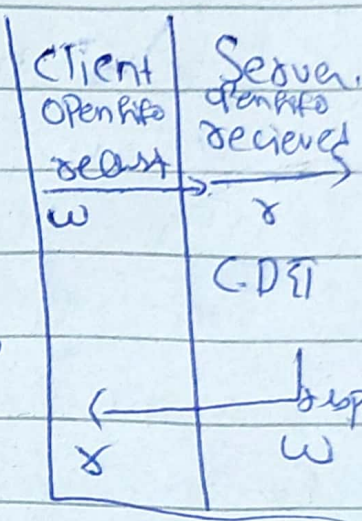
2) open,
3) R, W, 4) Closed.

• Data erase in when Process terminate

EESIT= Check the Reson of error others).

#Client

```
int main()
{
1) int x = mkfifo("Time fifo", S_IRWXU);
if (x==-1 && errno!=EEXIST)
{ Perror("fifo creation failed\n");
  return -1;
```



3)

6)

11

```
2) int fd = open("time fifo", O_RDWR);
   // error checking.

3) int bw = write(fd, "Time ?", 6);
Server-  sleep(10);
   int main()
{  int x = mkfifo("time fifo", S_IRWXU);
   int fd = open("time fifo", O_RDWR);
```

ends // if client Dip not run so it fail

```
   char buff[255];
4) int br = read(fd, buff, 255);
```

5) get time in user readable.

---

3) time_t t = time(NULL);
char *timestr = ctime(&t);
// user readable

6) int bw = write(fd, timestr, strlen(timestr)+1);

// at client beare no space here.

int br = read(fd, buff, 255);

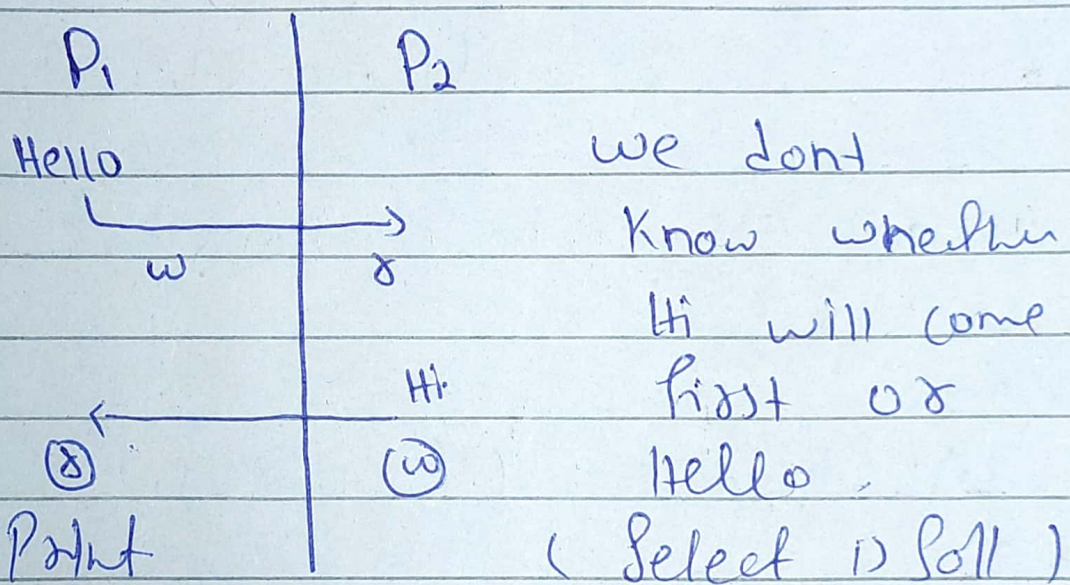
Printf("Current DST = %s\n", buff);

}

- but issue can come if it (client) read & write fd on its own so blocks will occur.
- So apply Sleep.
- but we dont know how much time to sleep

So make two Diffent files.
See Pics for Sol! (FIFO)!

Full Duplex Communication



Programming :-

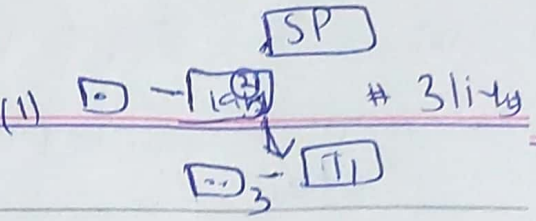
```
main()
{
int x = mkfifo ("chat fifo", S-IRWXU);
    // error checks.

fd = open ("chat fifo", O-RDWR);
        // error checks.
```

it is for Process P1 So for 2 Run 2 times.

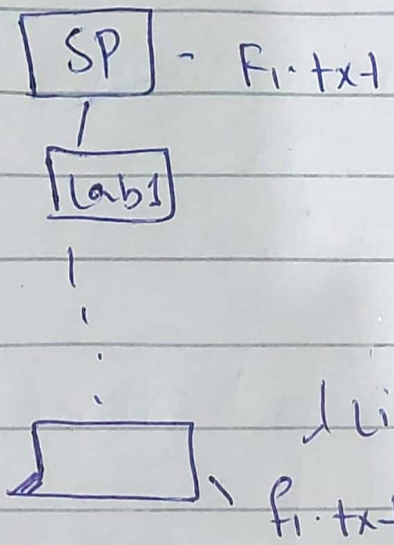
// Select.

```
fd_set readset;
FD_ZERO (readset); while(1){
FD_SET (fd, &readset);
FD_SET (STDIN_FILENO, &readset);

int maxfd = (fd > STDIN_FILENO) ? fd :
                    STDIN_FILENO);
int n = Select (maxfd+1, &readset,
        NULL, NULL, NULL);

if (FD_ISSET (fd, &readset))
{
    int br = read (fd, buff, 255);
    Printf ("%s\n", buff);
}
if (FD_ISSET (STDIN_FILENO, &readset))
{
    int br = read (STDIN_FILENO, buff,
                    255);
    write (Fd, buff, br);
} , unlink ("fifo1");
```

no of links of file = 1)



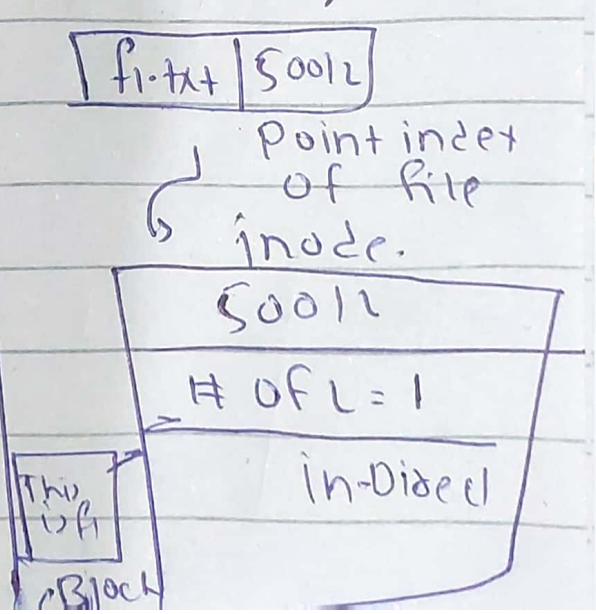
- **Links:-** (shortest path to file)



- we will make the link ; new So we can easily access it.
- Two types:-

Hard Link, Soft links (Symbolic link).

- Hard Link:-
- File & Directory - creation is hard link (vim f1.txt).
- ln command for hard link.



3) rm f1.txt.

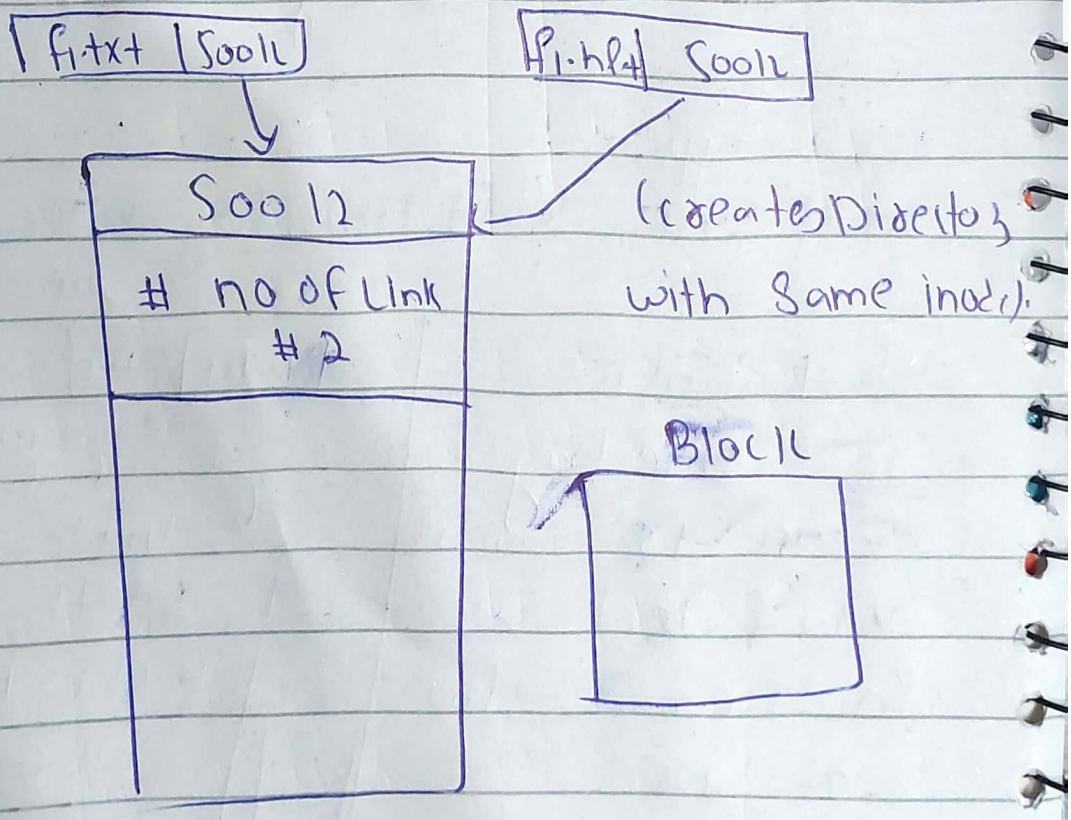
4) cat f1.hl.txt
cat f1.txt.

(2) ln.txt f1-hl.txt

- To make link in programmy the command is ln & Syscall is: int link.

int link (cost char* src, const char* dst)

0 Success)
-1 error. (failed).



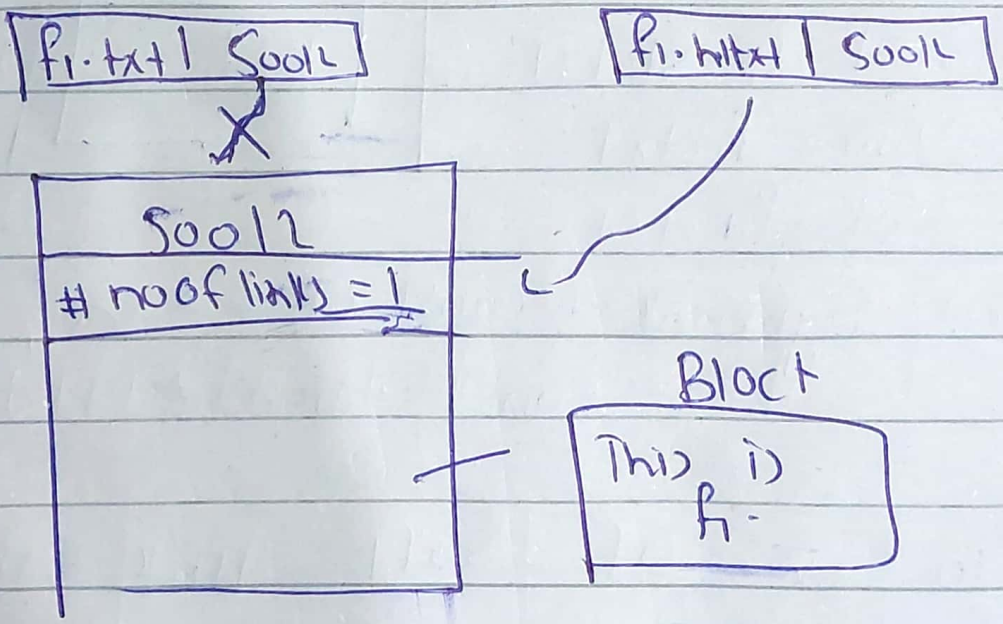
So f1.txt no of links = 2
f1-hl.txt 2

So whenever hard link created the Directory entry is created & no of links incremented.

. if rm f1.txt.
Remove Directory entry & Decrement no of links.

. But if all the file are removed So inode & Block are removed as well.

rm f1.txt.

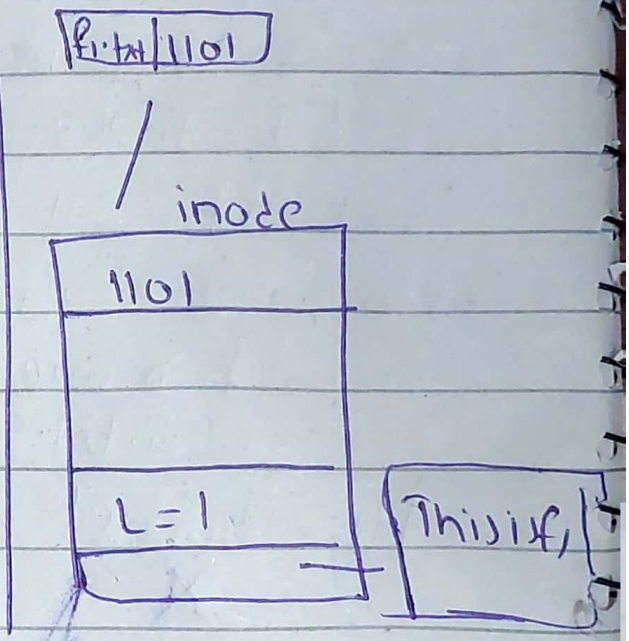


cat f1.txt.
cat f1.hl.txt

if we update the file & hard link so the changes will be done on ~~inode~~ Block.

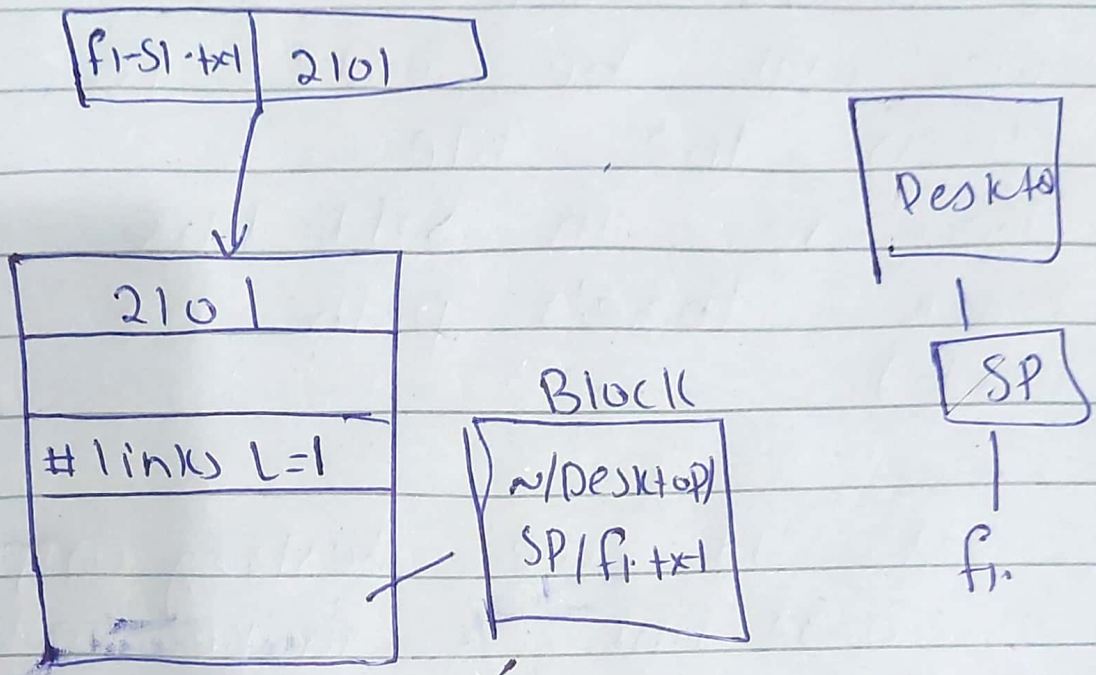
SOFT link:

① Vim f1.txt
② ln -s f1.txt, f1-sl.txt for creating Soft link
③ rm. f1.txt
④ cat f1-sl.txt.
⑤ Vim f1.txt
⑥ cat f1.txt



Int Symlink (const char * src,
const char *dst)

In case of Soft link another file will be made with other diffent inode.



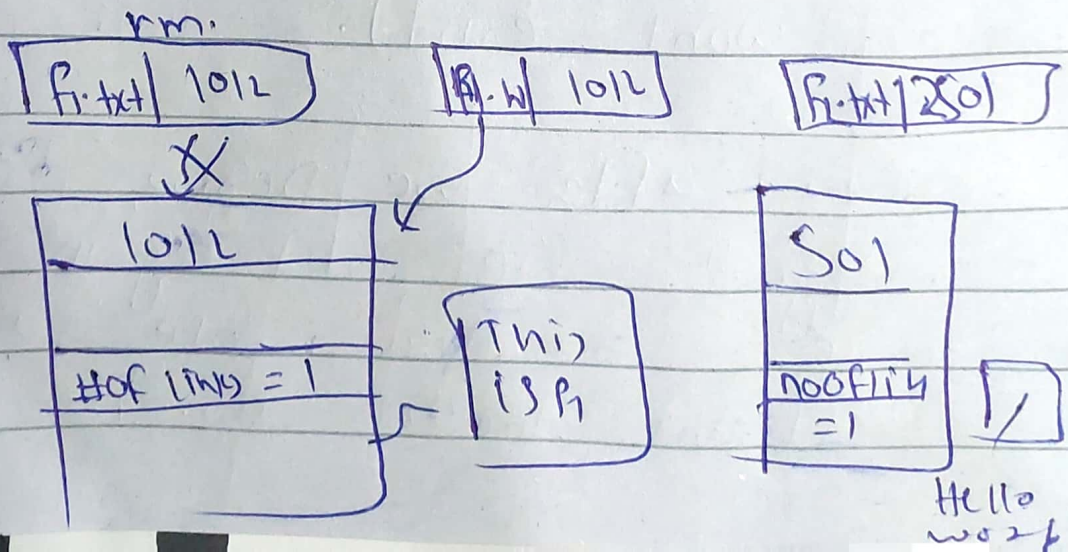
in Block there will be Path of the file.

. if we remove the orignl file So we wont be able to access the linked file (because the links # 0 file wont exist).

. Now afte we Destroy the orignal file but again we make another file with diffent inode no. but

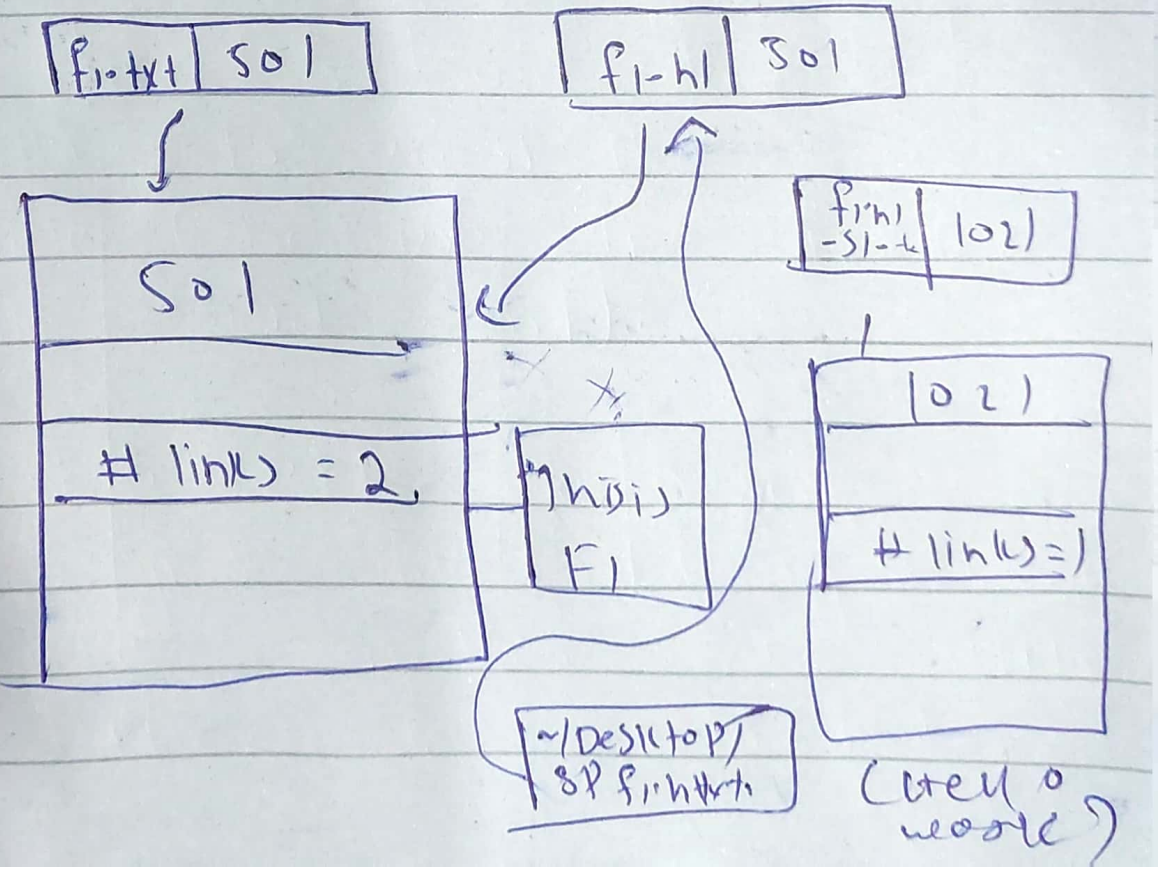
Path is same so we will be able to access the file. (Because in soft link Block there's path no inode).

- But if we destroy origin in hard link & make another file with diffent inode So when we access the h. link we will get this is f1. & when we will access file So we will get Hello world.



Hard link of Soft link

1. vim f1.txt
2. ln f1.txt f1-hl.txt
3. ln -s f1.hltxt
   f1.hlstxt
4. rm f1-hl.txt
5. cat f1-hl st.txt
6. ln f1.txt f1-hl1txt f1-hl-f1-txt



Links: < Hard Link / Soft / Symbolic links.

int
int

Soft Link

① vim f1.txt ("I'm f1").

② ln f1.txt f1_hl.txt

f1.tx

③ prog.c
↳ open f1
read f1 into (buff = I'm f1")

open/trunc | rm f1.txt
open/create f1.txt

write Hello into f1

close.

SP

f1.txt	501
f1_hl.txt	501
f1_hl_sl.txt	1021
.	408
..	302

#

④ cat f1.txt
cat f1_hl.txt

Chapter 8 Signals

Signal: notification of an event.

H.W event } Signal
S.F event } generate

- we can manually also can generate Signal.

- To find list of availabe Signals (with no & name).

Kill -l

Signo used for Signal generation.

1) SIGINT
2) SIGABRT
3) SIGKILL
4) SIGTERM
5) SIGSTOP
etc.

- Signo is Signal number
- with Signal name.

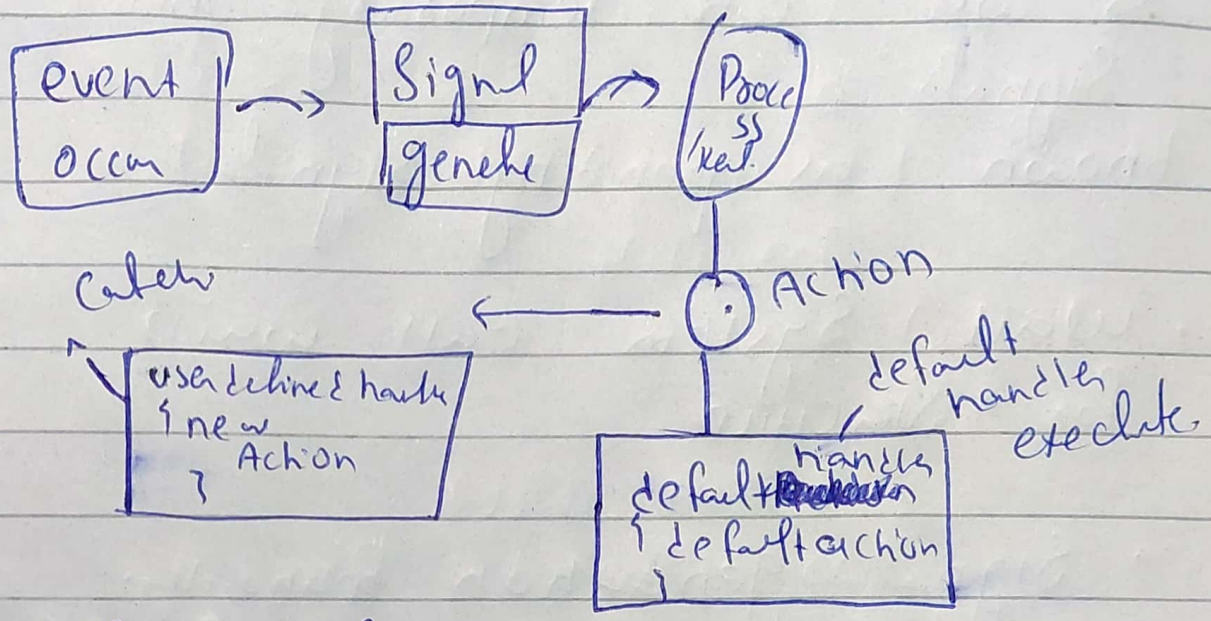
} Occur when event occur.

6) SIGUSR1
7) SIGUSR

} Occur without an event. event is not defined.

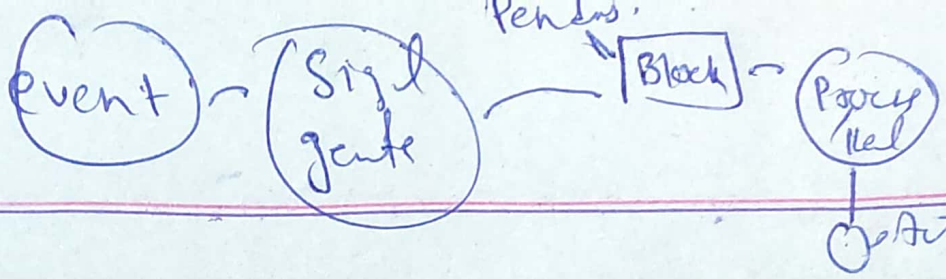
- for every signal. signal handler is define.

SIGUSR1 } is left for user
SIGUSR2 } to define it for any event.



after default action we can say signal is delieved.

- We can chaye default action by making user define handler.
- when handler do the action we can say signal is delicnd.



- if we dont want gented Signal to go to process/Kernel So we can temporary block Signals
known as pending Signals

- wait() System call waits for SIGCHILD.

- Signal generation manually:
./P1.0 & -B.G
(Back ground Process, without controlling terminal)
(no ctrl-c or Something work on it.)
./P1.0 &
main()
{ for(i;;);
}
(we want to Kill this Process manually.

STEPS:
1) Signal generation
$Ps -aux
1) init
2) login
3) user
.
.
55 ./P1.0
!
2056 Bs

where we want to deliver
it (Process no).

Signal no — Signal generate. if we don't know p. no

$ kill -1 655 or -S SIGINT 55

$ kill -S SIGINT 35

int kill (PID, int Signo);

0 : S

-1 : ERR

main ()
{

kill (55, SIGINT (001)); — P1.0 get terminated abnormally

kill (getPPID(), SIGINT) - Batch Parent terminate.

kill (get pid(); SIG ABRT); terminate my own process abnormally!

(Signal to itself).

or

int raise (int Signo);

• This sys call give Signal to itself.

```
int main()
{
    int x = fork()
if (x == 0) {                    SIGCHILD generated
                                 Send to Parent.
    Kill (getppid(), SIGCHILD);
      for (ii);
}
   else
        wait (NULL);
    } So Parent Process
wait will over.
```

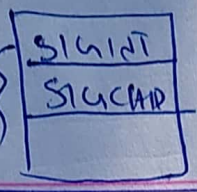
. if Parent do Cascaded termination
Terminate & terminate all of
its child as well.

```
main()
{  int x = fork();
   if (x == 0) {
       for(i,i); (infinite loop)
else {
       Printf ("Hello\n"));
     Kill (x, SIGINT);
  } }
```

1) In first there is infinite loop the Parent should wait forever, But as we manually generated an Signal to the Parent (SIGCHILD) So Parent will not wait forever.

2) In 2 there's an infinite loop again but shll as we make an Signal and Passed (x) (child) and will taminate. (Parent & its child too).

SIGINT & SIGCHLD will be blocked.



Process mask.

Leet-

- Each Process has associated Signal mask (containing Signal) (This mask contain blocking) Signal

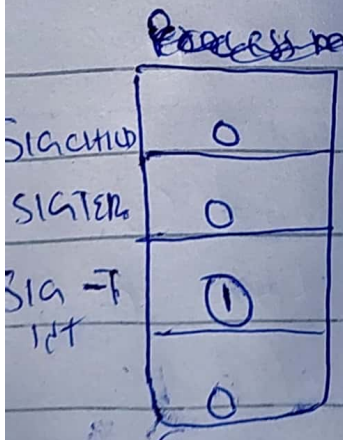
Scenarios for Blocking Sigal:- (SIGINT).

① unblock all signal except SIGINT

② Block all signal except SIGINT.

Scenario (1)

SigSet-t (to set the index we want to Block.)

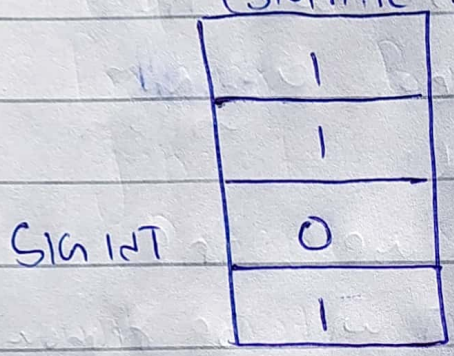


SIGNAL SET

① Block all Sigal

② Add SIGINT

Scenario (2) :-



(3) (B) Fill with all Sig.

(4) (A) Remove SIG INT.

STEP (5)

(5) Now Replace SIGNAL SET with SIGNAL / Process mask.

1) Sigemptyset (Sigset-t, *myset);
(Removiy).
(To Remove all set)

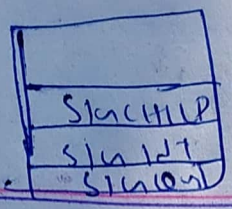
3) Sigfillset (Sigset-t * myset);
(To fill all set)

(2) SIGADDSET (Sigset-t *myset, int Signo
(for add specific sig) // SIG INT);

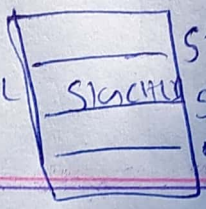
(4) Sigdelset (Sigset-t *myset, int Sig
// SIG INT);

(5) SigProcmask (, Sigset-t *myset
, Sigset-t *OLD mask);

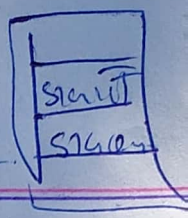
Sig-Block



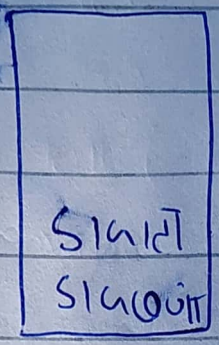
Sig-unBlock



Sig-SET MASK



Block Unblock Replace



→ Block: in block the Sig mask will not be Replaced but additions will be

- unblock:

In this case the SIGINT, SIGQUIT will be removed but SIGCHILD will be present

Replace:

in this case the Sig mask will completely Replaced with new

Block interrupt signal system

① have sigset

② Remove all

③ add two sigs

④ Replace.

```
main()
{
```

① Sigset_t myset, oldset;

② sigemptyset(&myset);

③ sigaddset(&myset, SIGINT);

④ sigaddset(&myset, SIGTERM);

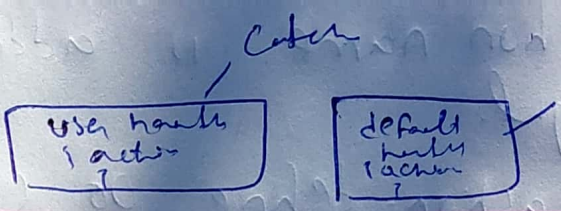
⑤ Sigprocmask(SIG_SETMASK, &myset, &oldset);

Now we want OLD Process Back. after our work

Sigprocmask(SIG_SETMASK, &oldset, NULL);

int alarm will genete
Sigal after time end
when time = 0, Proces
SIGALSI- terminate.

int kill (Pid-t, Signo);
int raise (int signo);
int alarm (int sec);
Error = -1, Success = Remaing
int main() time.
{
alarm(5);
for (i;i);
Printf ("Hello \n");
}
Process run for 5 sec
SigAlarm will generate
an Sigal after 5 sec
& then Proces end.



- Catching Signals:-

```
Void interrupt handle (int signo)
{   Printf ("Interrupt Signal Received \n");
    return;
}
Struct Sigaction {
    Void (* Sa-handler) (int);
    int Sa-flags ; // 0
    Sigset-t Sa-mask; // add additional
}
```

SIG-IGN
SIG-DFL
user defined handler address

Data type (Define the Signal Sets. (which we want to Block during execution).

Signal to the Block Signal during the exec of Signal handler.

- function name is address of signal.

int Sigaction: define or change action

int test=0; (int Signo, struct Sigaction *newaction, struct Sigaction, oldact)

int main()
{

struct Sigaction newact;
newact.sa_flag=0;

Sigemptyset (newact.sa_mask); (no additional Sign added)

newact.sa_handler= interrupt_handler;

// int Sigaction (int Signo, struct Sigaction *newact, struct Sigaction *oldact);

Sigaction(SIGINT, &newact, &oldact); (newbehaviour)
Signal whose behaviour we want to change

sleep(5); // wait for signal.
for(;;);
Printf("Process normally terminated \n")
exit(0);

Ready → Running

Sol. Blocked → Pause()

Wait in for loop is actually wasting CPU cycle.

So we will use pause sys call instead.

from,-

```
sleep(5)
Pause(); // waits for signal.
```

{

for this Block all signal below int main().

{ unblock signal above Pause.

Sigsuspend(...);

atomically execute.

- Waiting for Signals.
- SigSuspend Do the blocking & unblocking simultaniously.

```
int SigSuspend (const Sigset-t* mask);
main()
{
        // block all signal
    Sigset-t mask;
      Sig fillSet (mask);
  Sigprocmask (Sig-Block, &mask, NULL);

  Sigdelset( SIGINT, &mask);

  Sig Suspend ( &mask) // unblock with new mask without SigInt
{       Printf ("Normally teminate");
// block + Pause all except SIGINT.
above main
    int Sig Suspend (const Sigset-t *mask
        void handler (int signo)
        {
            return;
        }
```

main()

Same behaviour for all signal except SIGTSTP. Block.

```
{
    Sigset_t mask, oldmask;
    Sigprocmask (SIG_BLOCK, NULL, &oldmask);
    Sigaddset (SIGINT, &oldmask);

    Sigprocmask (SIG_SETmask,

    Sigdelset (SIGINT, &oldmask);

    if (test == 0);
    Sigsuspend (&oldmask);
        Printf ("Terminating \n");
}
```

Wait Signal Call;

```
int Sigwait (const Sigset * waitfor,
             int * Sig);
```

shows us number which put us out of waiting).

```
main()
{
    Sigset_t waitfor;
    Sigemptyset (&waitfor);
    Sigaddset (SIGINT, &waitfor);
    Sigaddset (SIGQUIT, &waitfor);

    Sigprocmask (SIG_BLOCK, &waitfor, NULL);

    Sigwait (&waitfor, &Sig);
      if (Sig == SIGINT)
    Printf ("Interrupt recieved\n");
    else
        Print ("Sigquit recieved\n");
    }
```

Threads:- (light weight Process)

- flow of execution.
- we Do Parallalism

Parallelism
/ \
Data level Task level
(Data is Divided) (Tasks are Divide)

- for Parallalism we has two methods.
  - Multiprocessing
  - Multi threading

- Multiprocessing come with it's overhead (Memory overhead) (More (memory is utilized).
  - Context Switching overhead.
  - Resources more Required.

- Pthread System call Return errno in case of error not -1.

- Process creation overhead.

- Multi threading:-
  - It is better than Multiprocessing.
  - no extra shared memory is done.
  - Inter process communication is easy.

Program:- (3 threads Print their id & then terminate):

```
Void* thread-func (void* arg)
{
    Print f("my ID = %ld\n",
        Pthread-Self());
```

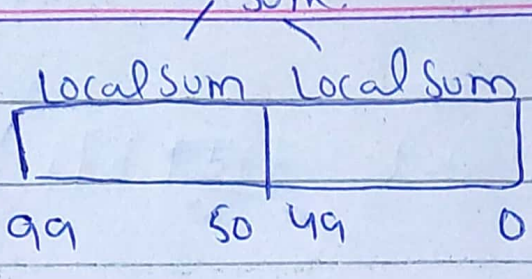
Pthread-exit (NULL); => Thread terminate

```
main()  Pthread_t  tid[3];
{
    for (i=0, i<3, i++)
Pthread_create(&tid[i], NULL,
        thread func, NULL);

for (i=0, i<3, i++)
    pthread_join(tid[i], NULL);
```

(wait for thread)

in Data level function

Global ID 1



	start	end
T0	0	49
T1	50	99

```
Void * Sum array (void* arg)
{       // type casting
    int index = *((int*)arg);
    int start = index * 100/num;
int end = index start + 49 or 100/num -1)
```

```
int local_sum = 0;
for (i = start; i<end; i++);
    local-sum+ = Array[i];
G- Sum + = local-sum;
```

```
int G- Sum = 0;
int Array[100];

int main()
{     // initilize array
   Pthread-t tid[2];
    for (i=0, i<2, i++)

Pthread-coeate (&tid[i], NULL, SumArry
   (void*)&i);

for (i=0 ; i<2; i++);
   Pthread- join (tid[i], NULL);

   Print f ("Sum of all elemet
      = %.d\n", sum);
   exit(0);
}
```