

## SP After Mids:

27/11/2023

Background Process Skipped.

Select :-

```
int select ( int maxfd, fd_set *readset,  
            fd_set *writeset, fd_set *errorset,  
            struct timeval *t );
```

```
int main( ) { // infinite delay
    printf("Hello");
    Select(1, NULL, NULL, NULL, NULL); // block
}
will wait indefinitely
```

Now we create delay using select.

// 5 sec & 10usec delay

Now

```
int main() {
    struct timeval mytime; printf("Hello\n");
    mytime.tv_sec = 5; mytime.tv_usec = 10;
    int f = 0 Select(1, [NULL, NULL, NULL, &mytime]
    printf("Wired\n");
```

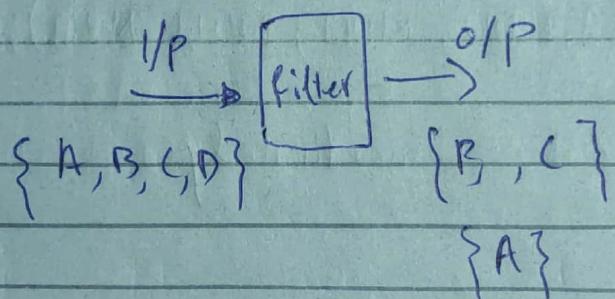
```

int main() {
    ; // monitor 2 files
    ;
    int s = select(maxfd + 1, &readset, NULL,
                   NULL, &mytime);

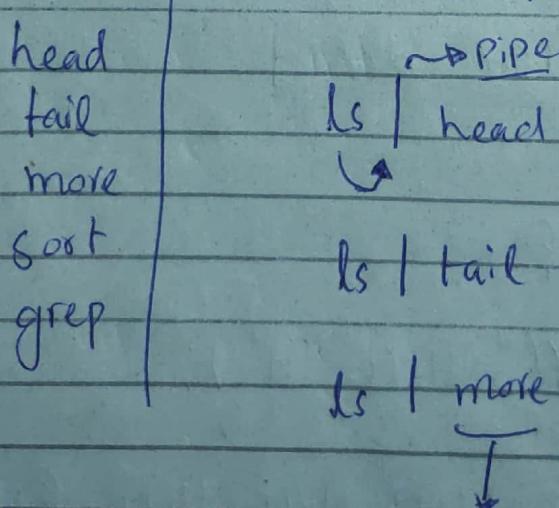
```

printf("Remaining time = %d lt %d usec  
 mytime.tv\_sec, mytime.tv\_nsec);

## Filters & Redirections -



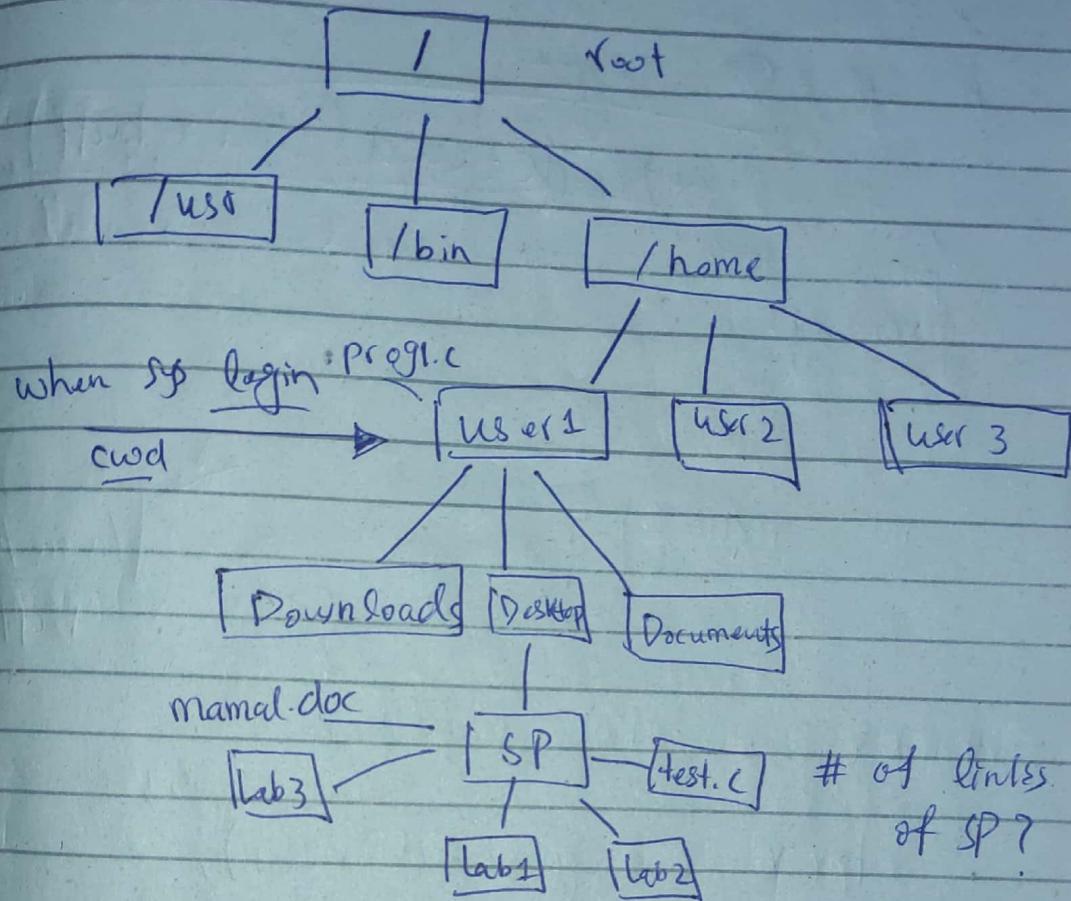
O/P is a subset of I/P



Space bar for more

## Chapter 5: Files & Directories

Directories → hierarchical



cd /home/usr1/Desktop/SP  
cd ./Desktop/SP  
cd .. /usr1/Desktop/SP ] relative  
cd Desktop/SP

cd /bin/.. /home/usr1/Desktop/ ] absolute

Prog1.c

main() {

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

const char \*

No need to allocate space for val.

A sys. call exist.

`char * getcwd (char * buff);`

for this we need to allocate memory.

int main() {

    ↓  
    [ /home/ksr ]  
    buff

~~char \*~~ = char buff [255];

    char \* x = getcwd (buff);

} Now we want to change our current dir.

error -1 ← int chdir ( const char \* path );

success 0 →

int main() {

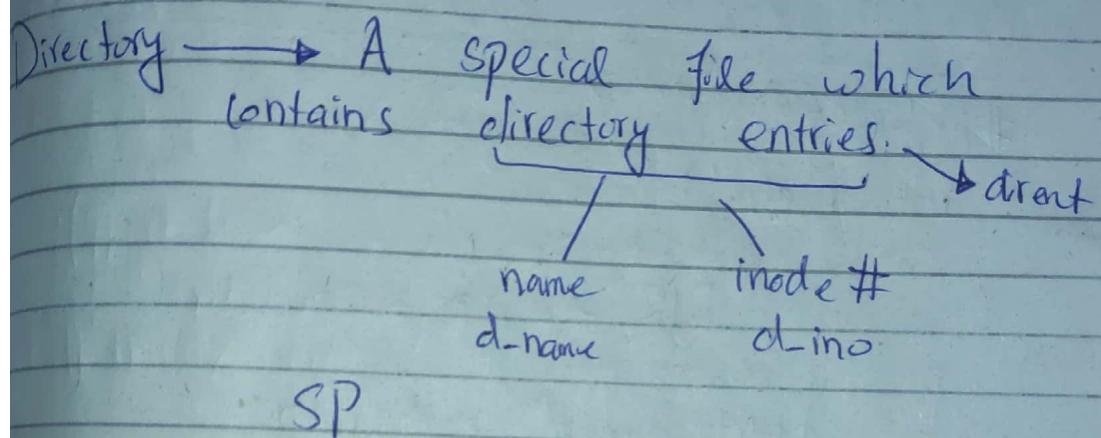
    char buff [255];

    char \* x = getcwd (buff);  
    printf ("My CWD = %s \n", buff);

Can be absolute/relative Path.

int xv = chdir("Desktop/SP");

④ x = getcwd(buff);  
printf("Path after chdir is: %s\n", buff);



name	inode #
test.l	87
mammal.doc	126
Lab1	500
Lab2	1023
Lab3	1110
..	1234
..	1552
NULL	

We want to open SP directory

ys (all).

DIR \* ~~char~~ opendir (const char \* dirname)

Now we want to read from  
SP

another Sys Call,

~~read~~  
struct dirent \* readdir ( DIR \* dirp );

#include <dirent.h>

int main () {

DIR \* dirp = opendir ("SP");

struct dirent \* direntp;

while (1) {

    direntp = readdir (dirp);

    if (direntp == NULL)

        break;

    printf ("Dirent name = %s it's, dirent  
inode = %d\n", direntp->d\_name,  
direntp->d\_ino);

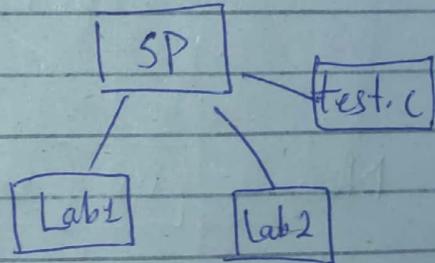
}

|

30 | 11 | 2023

Now to get other info,

a sys call exist.



{Syst start.h}

```
int stat( const char* name, struct stat *statbuff);
```

## Struct Stat

1) st\_ino  
 2) st\_mode  
 3) st\_nlink  
 4) st\_size  
 5) st\_atime  
 6) st\_ctime  
 7) st\_mtime  
 8) st\_mtime\_ns  
 9) st\_atime\_ns  
 10) st\_ctime\_ns

main()

DIR \*mydir = opendir (" . " );

struct dirent \*mydirent;

struct stat statbuff;

while ((mydirent = readdir (mydir)) != NULL)

printf (" Name = %s \t ", mydirent->d\_name)

stat (mydirent->d\_name, & statbuff);

printf (" Time = %s \t ", ctime (& statbuff.st\_atime))

printf (" Size = %d \t , gid = %d \t uid = %d  
Links = %d \t ", statbuff.st\_size, statbuff.  
st\_gid, statbuff.st\_uid, statbuff.st\_nlink)

Now

[int st-mode]

MACROS

Type of file

S\_ISDIR(m)

S\_ISREG(m)

S\_ISFIFO(m)

S\_ISOCKET(m)

Permissions

S\_IRUSR

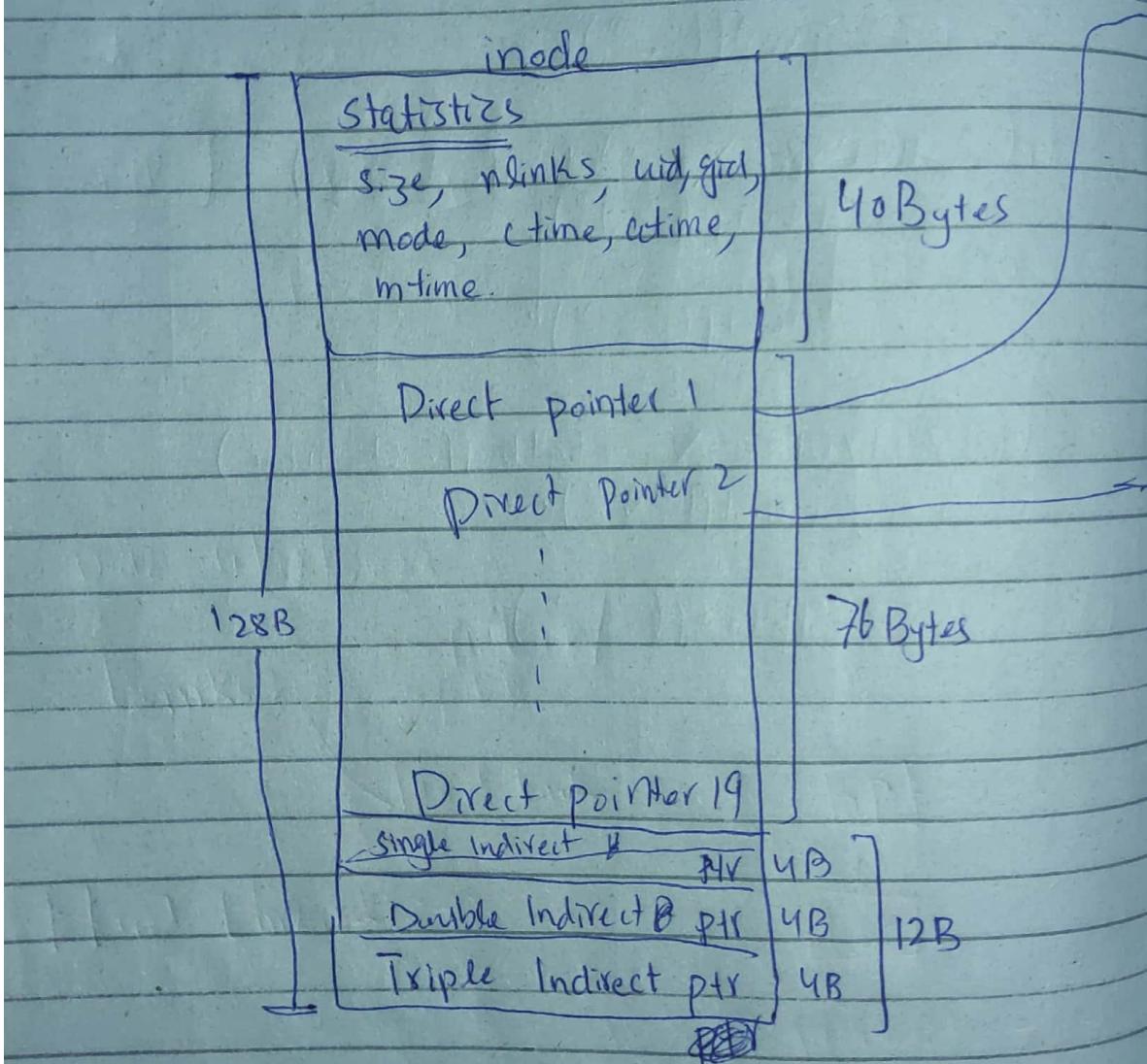
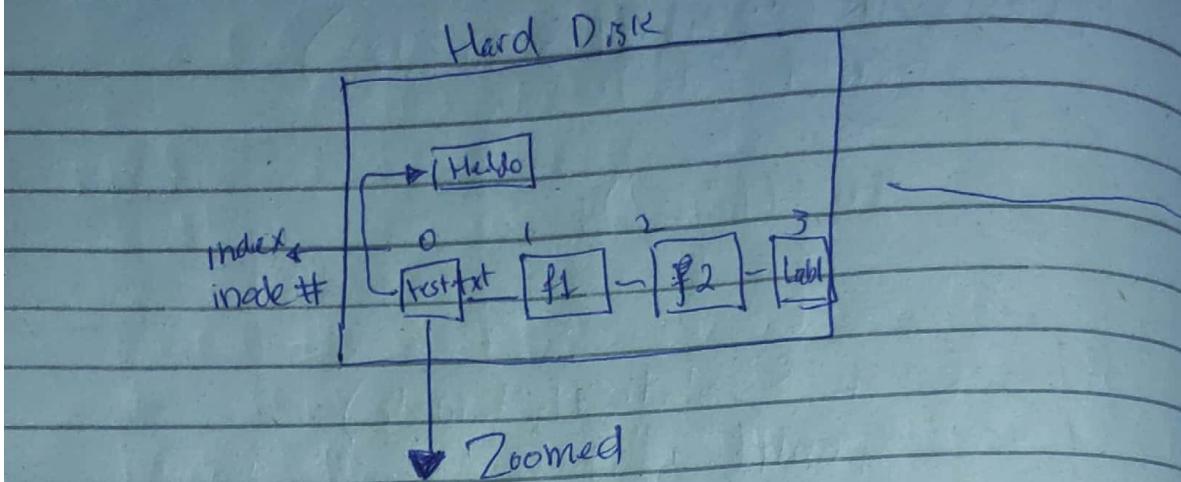
```
→ if (S_ISDIR (statbuff.st_mode))
    printf ("d");
else
    printf ("-");
```

```
if (S_IRUSR & statbuff.st_mode)
    printf ("r");
else
    printf ("-");
```

```
if (S_IWUSR & statbuff.st_mode)
    printf ("w");
else
    printf ("-");
```

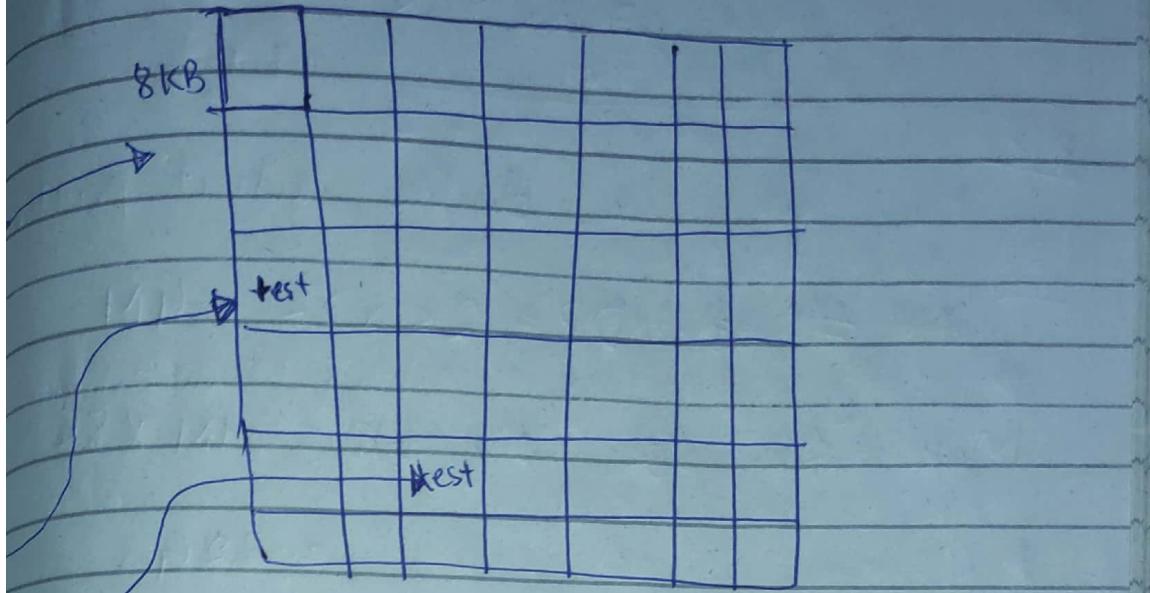
```
if (S_IXUSR & statbuff.st_mode)
    printf ("x");
else
    printf ("-");
```

```
}
```



$$\text{Total memory for DP} = 128 - 40 - 12 = 76$$

Total # of direct pointers =  $\frac{\text{T. available memory}}{\text{size of single}}$



$$= \frac{76B}{4B} = \underline{19 \text{ pointers}}$$

Max file using DPs = # of ptrs & size of  
1 block

$$= 19 \times 8KB = 152KB$$

SID

Total Memory available for DP = 8KB

$$\text{Total \# of DP} = \frac{\text{TAM}}{\text{Size of P}} = \frac{8KB - 2K}{4B} = 2048$$

Max file using SID = # of DPs \* Size of  
block

$$= 2K \times 8K = 16M$$

DIP

How we use double indirect pointer

$$\text{Total \# of DP} = 2K \times 2K = 4M$$

$$\text{Max file size using SID} = 4M \times 8K$$

$$/\! / \quad / \quad = 32G$$

TIP

$$\text{Total \# of DP} = 2K \times 2K \times 2K = 8G$$

$$\text{Max file size using TIP} = 64TB$$

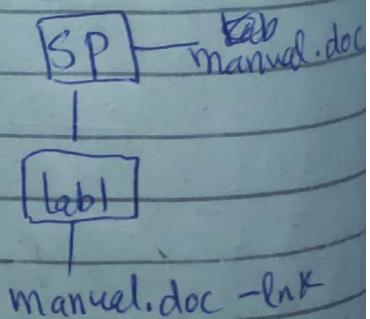
Date 4/12/2023

Links:

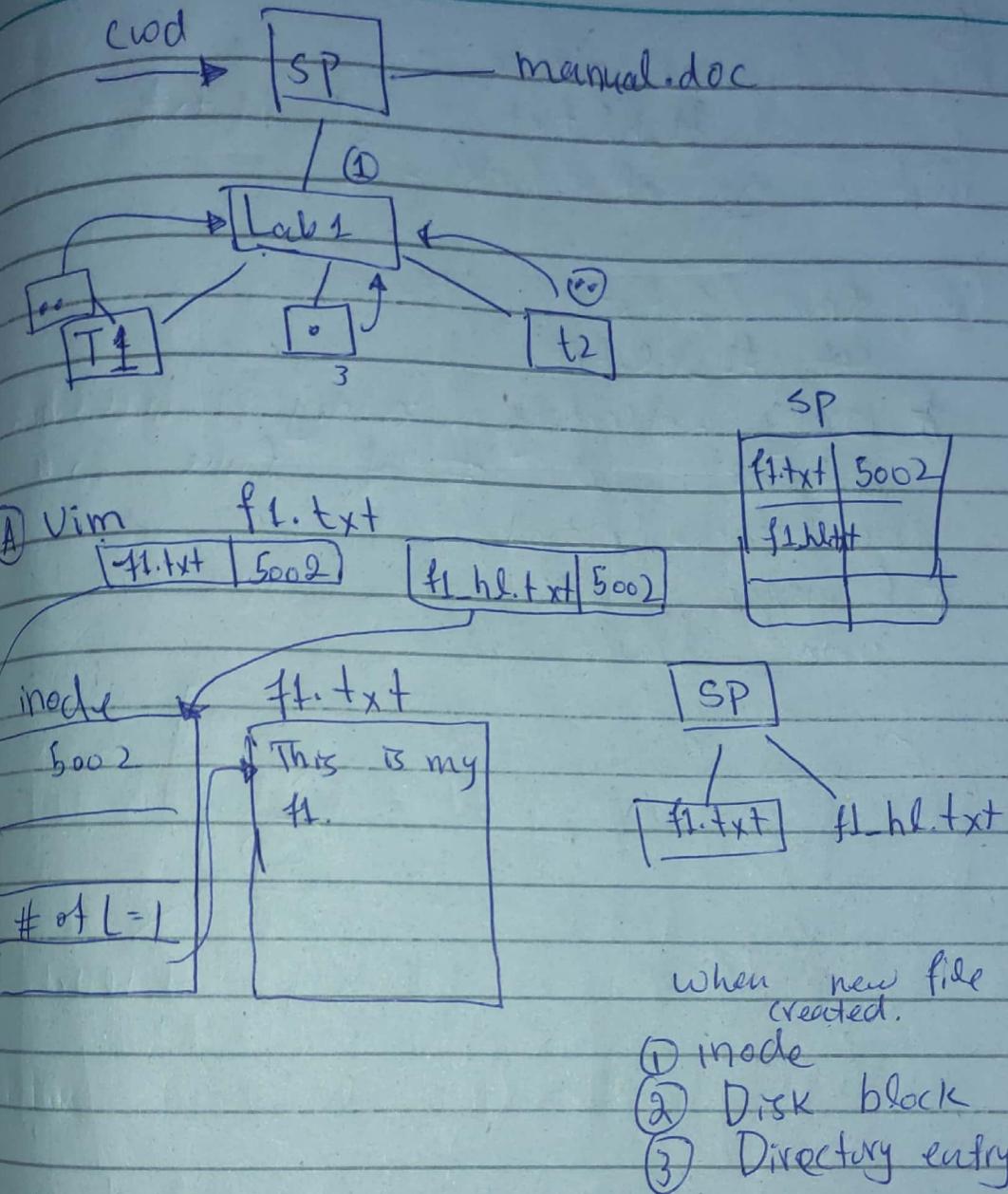
Soft Link /  
Symbolic

Hard Link

Hard Links:-



ls -l  
drwxr-- # links



### ② Link

Command: ln f1.txt f1\_hl.txt

Sys call: int link ( const char \*src,  
const char \*dst);

Now ln f1.txt f1\_hl.txt

When we create a hard link only  
directory entry is created in backend.  
and the links incremented.

(C)

Now if we write

cat f1\_hl.txt | cat f1.txt

and no. of links = 2

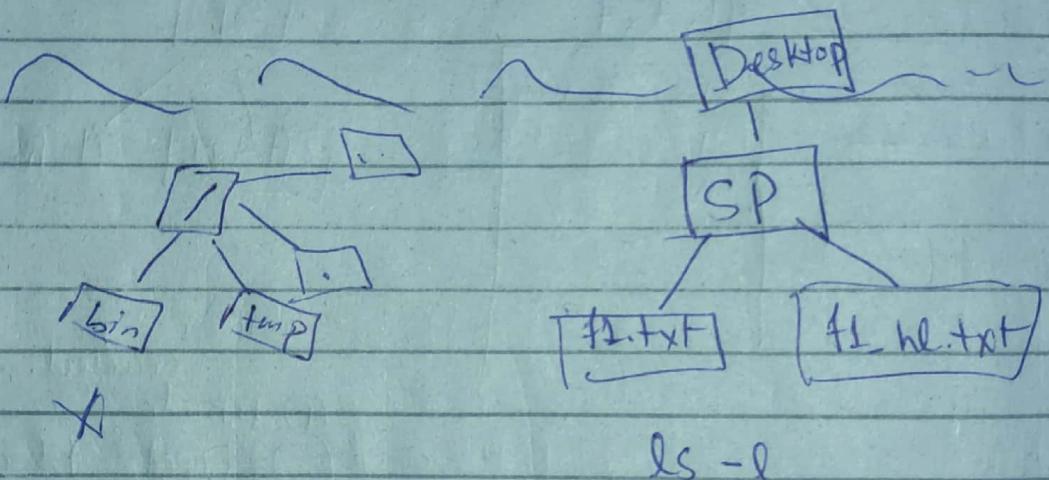
(D)

rm f1.txt

cat f1\_hl.txt

\* only directory entry will be deleted and # of links will be decremented.

\* Changes will be same in both links.



ls -l

# of Link	Name
2	f1.txt
2	f1_hl.txt

Soft Link :-

Command: ln -s f1.txt f1\_sl.txt

Sys Call : int SymLink (const char \*src, const char \*dst);

① vim f1.txt

② ln -s f1.txt  
f1\_sl.txt

③ cat f1\_sl.txt  
f1.txt

Both show  
Same context.

Now if we update  
the link by

④ vim f1\_sl.txt

⑤ vim f1.txt

⑥ cat f1\_sl.txt

file doesn't  
exist error.

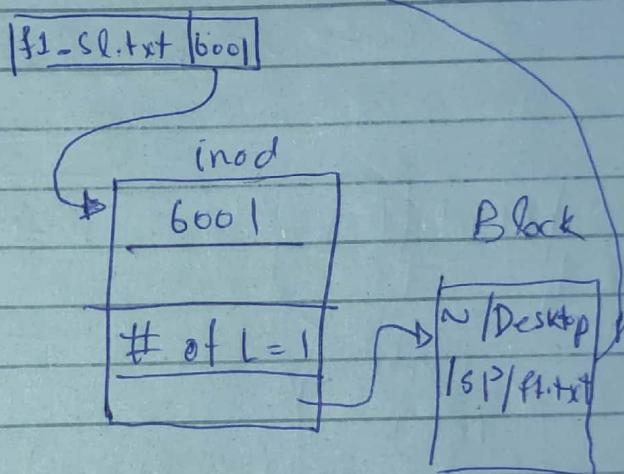
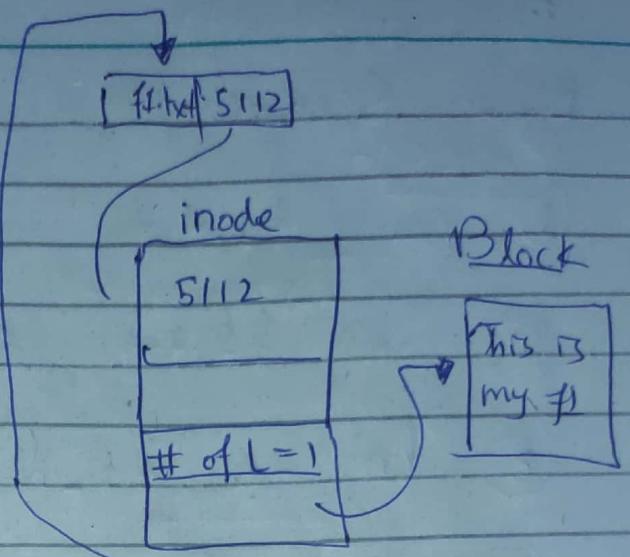
⑦ vim f1.txt

Hello  
World

cat f1\_sl.txt.

will print Hello World.

It means if we create another file  
with same name then we  
can still access it through  
soft link.



Now Scenario:

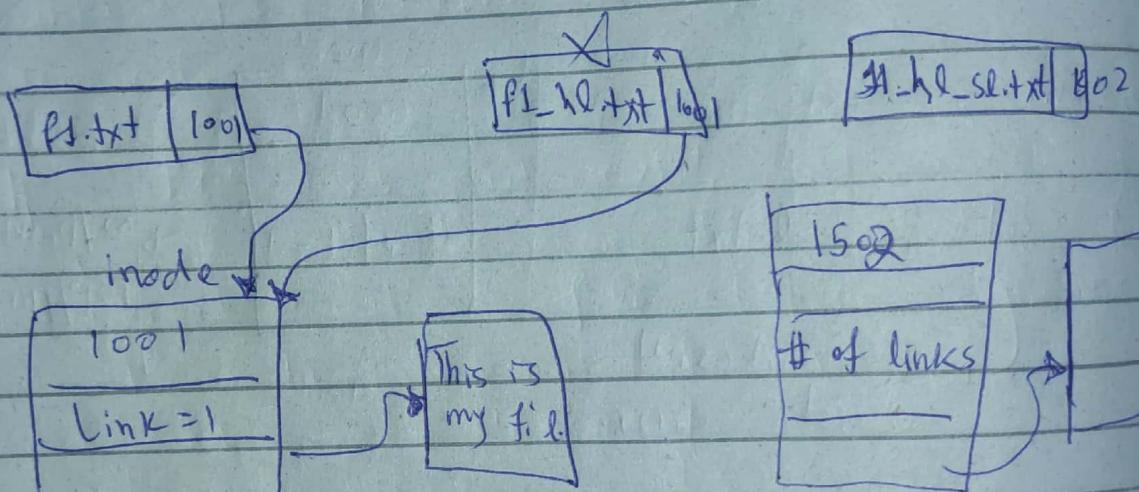
① vim f1.txt

② ln f1.txt f1-hl.txt

③ ln -s f1-hl.txt f1-hl-sl.txt

④ rm f1-hl.txt

⑤ cat f1-hl-sl.txt



⑥ vim f1.txt ("Hello World")

⑦ ln f1.txt f1-hl.txt

⑧ prog.c

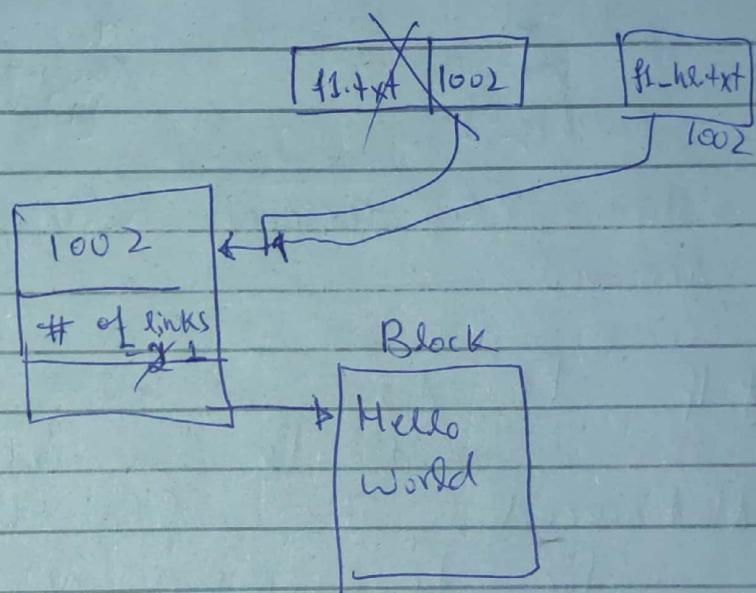
open f1.txt

read f1.txt into buff.

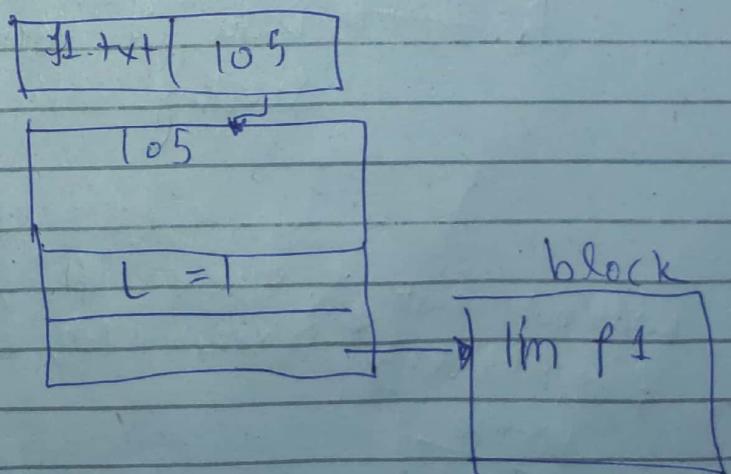
rm f1.txt

```
create f1.txt  
write ("I'm f1");  
close .
```

(4)

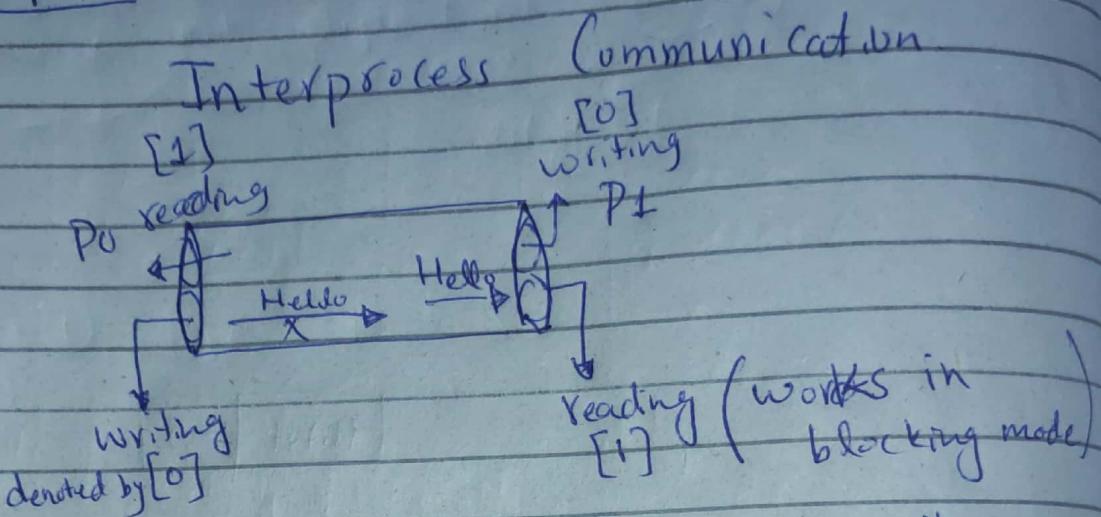


creating f1.txt



Date 7/12/2023

## Pipes:-



When P1 received hello, then it is removed.

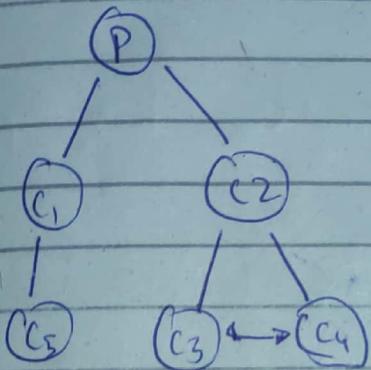
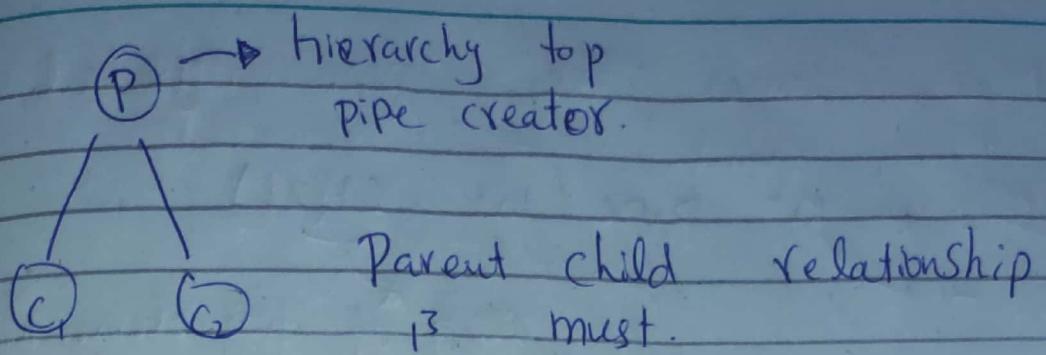
\* If P1 does not exist, then P0 can read its own data.

\* In Backend, it's file (Special file).  
For two reasons it is special

1 → Unnamed

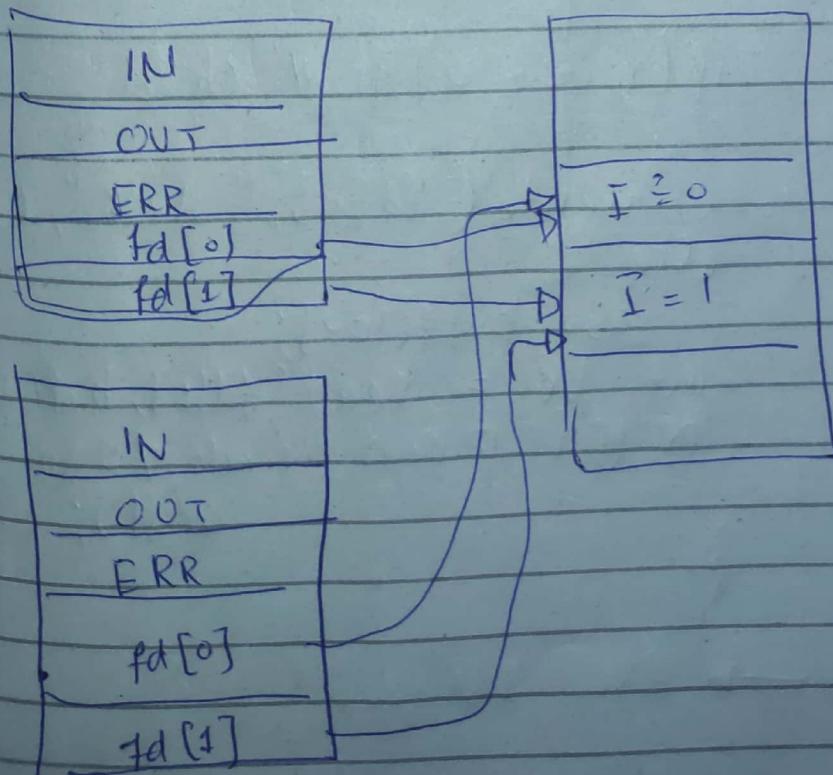
2 → Data is erased after reading.

will not wait if NONBLOCKING mode:  
open ("f1", O\_NONBLOCK)  
read (fd, buff, 100);



Parent

fd table



Now comes the pipe sys call.

success ← int pipe (int fd[2])  
exit -1 ← array of two elements.

Step ① Pipe creation:

```
int main() {
```

```
    int fd[2];
```

```
    int x = pipe(fd);
```

```
    int x = fork();
```

```
    if (x == 0) { // child
```

```
        write(fd[0], "Hello", 6);
```

```
    else {
```

```
        char buff[100];
```

```
        int br = read(fd[1], buff, 100);
```

```
        printf("Parent .read: %s\n", buff);
```

```
}
```

Rtp Server → file listing.

Error Case:- File Interrupt  
Resources Exhaust

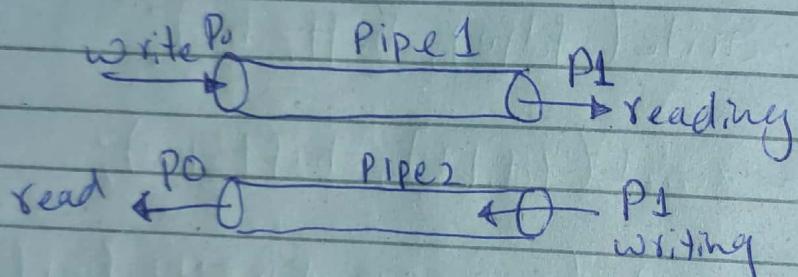
Two way communication:-

```
int main () {  
    int fd[2]; char buff[100];  
    Pipe(fd);  
    x = fork();  
    if (x == 0) {  
        br = read (STDIN_FILENO, buff, 100);  
        write (fd[0], buff, br);  
        sleep(1);  
        br = read (fd [1], buff, 100);  
        write (STDOUT_FILENO, buff, br);  
    }  
    else {  
        int br = read (fd[1], buff, 100);  
        write (STDOUT_FILENO, buff, br);  
        br = read (STDIN_FILENO, buff  
        write (fd[0], buff, br);  
    }  
}
```

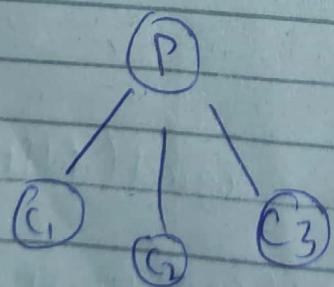
Whenever there is write and then read put sleep(2);

```
write(fd[0])  
sleep(2)  
read(fd[1])
```

Another Solution, make two pipes.



Multiple processes.



```
int main() {
    int fd[2];
    pipe(fd);
    int N = 3;
    for (i = 0; i < 3; i++) {
        x = fork();
        if (x == 0)
            break;
    }
    if (x > 0) {
        for (i = 0; i < N; i++)
            write(fd[i], "Hello", 6);
    } else {
        char buff[100];
        br = read(fd[i], buff, 6);
        printf("child read %s\n", buff);
    }
}
```

Date 11/12/2023

## Fifos / Named Pipes :-

- Pipes are unframed
- First In First Out

### → FIFO's

- Exist after process end
- Named file
- Data erased after read

### ② Syscall :-

```
int mkfifo (const char *filename, int perm);
```

-1: E

0: S

## Half Duplex Comm

→ Two Way Sequential Communication

Whenever client  
Server Model,  
make two programs.

Client

Server

seq

Receive  
get Q &

Receive rep  
&  
display \*

Respon

## Client.c

```
int main() {
    int char buff[255];
    int r = mkfifo("timefifo",
                    S_IRWXU);
    if (r == -1 && errno == EXIST) {
        perror("EXIST");
        return -1;
    }
    int fd = open("timefifo", O_RDONLY);
    if (fd == -1) {
        perror("Error");
        return -1;
    }
    int bw = write(fd, "Time?", 6);
    sleep(5);
    int br = read(fd, buff, 255);
    printf("current D&T\n= %s\n", buff);
}
```

## Server.c

```
int main() {
    char buff[100];
    int r = mkfifo("timefifo",
                    S_IRWXU);
    if (r == -1 && errno == EXIST) {
        perror("mkfifo failed");
        return -1;
    }
    int fd = open("timefifo", O_RDONLY);
    int br = read(fd, buff, 100);
    time_t t = time(NULL);
    char* time_s = ctime(&t);
    write(fd, time_s, strlen(time_s));
}
```

fifo solved the parent child problem

Make two fifos for avoiding write and read problem.

\* Select sys call ~~not~~ can be used for unknown sequence.

```

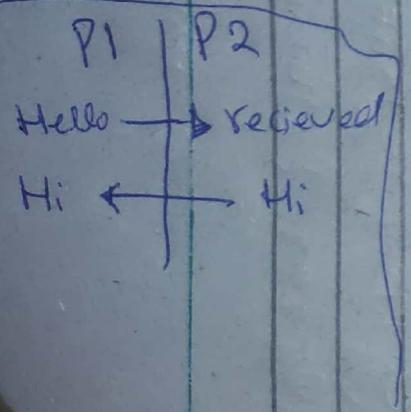
Prog. C main()
{
    int x = mkfifo ("chatfifo", S_IRWXU);
    // error

    int fd = open ("chatfifo", O_RDWR);
    fd_set readset;
    FD_ZERO(&readset);
    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("ops\n", buff);
}

```

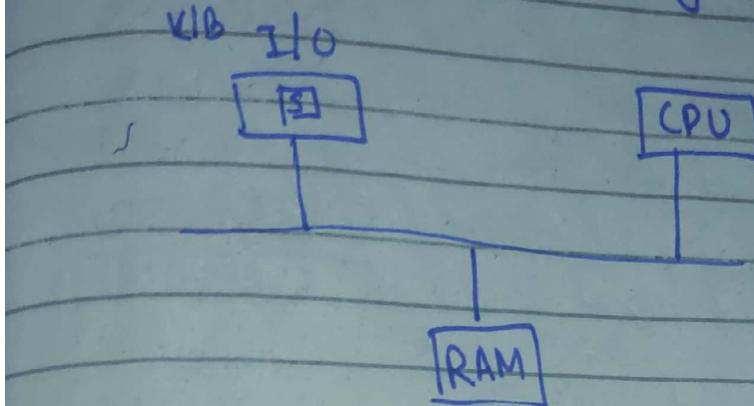
```

if (FD_ISSET(STDIN_FILENO, &readset))
    int br = read(STDIN_FILENO, buff,
                  255);
    write(fd, buff, br);
}

```

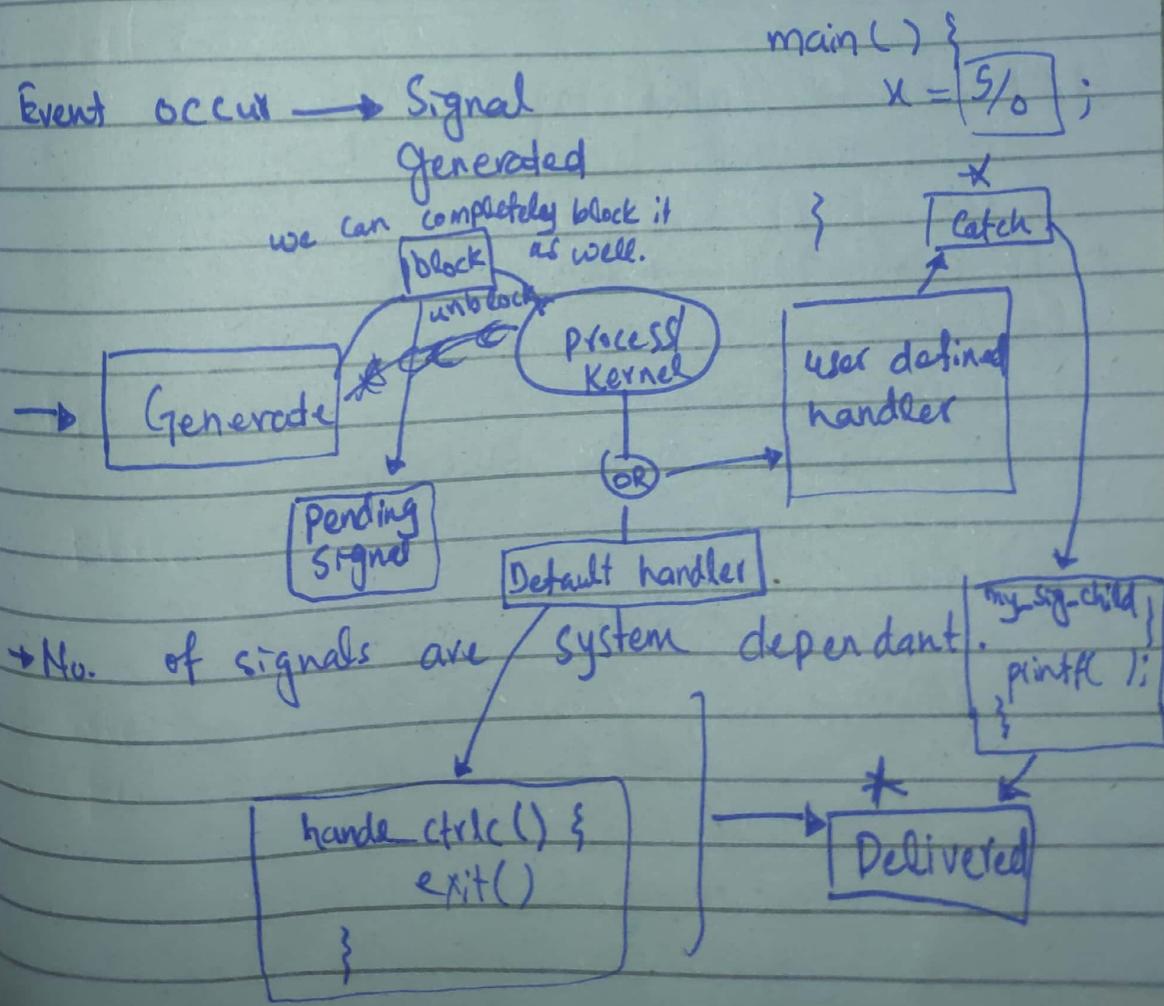
## CHAPTER ⑧: Signals

H.W events  
S.W events → Signal



we store it in buffer

Through signal we know that event has occurred.



List of available signals

\$ KILL - l

```
char a;  
cin >> a;
```

If user enters 1 then we can  
generate SIGUSR1

event  
not defined

- 1) SIGINT
- 2) SIGTERM
- 3) SIGKILL
- 4) SIGABRT
- 5) SIGCHLD
- 6) SIGUSR1
- 7) SIGUSR2
- 8) SIGSTOP
- 9) SIGCONT
- 10) SIGFPE
- ;
- 64) Default Action

pstty → for listing signals generated by terminal.

: ^C → SIGINT → Srg interrupt

wait() → wait for SIGCHLD

```
pausell;  
sigwait();  
sigsuspend();
```

# Signal Generation :-

By using Kill, we can generate signals.

\$ PS -all

1) init

2) ;

;

2005 PS

• /t1.0 &

Terminal  
has no control  
over it.

main () {

    while (1);

}

Now we use kill to terminate  
/t1.0

\$ kill -9 <sup>PID</sup>  
                ↓  
                SIGNO

int kill( int PID, int signo );

-1 : E

0 : S

int main () {

    kill ( 101, 9' SIGINT );

SIGKILL  
SIGTERM

SIGTERM

default behav.



Abnormal  
Termination.

```
kill(getppid(), SIGINT);  
kill(getpid(), SIGINT); // raise(SIGINT);
```

{

int raise ( int srgno ); → deliver a  
signal to itself.

main() {

```
x = fork();
```

```
if (x == 0) {
```

```
sleep(+);
```

```
exit(0);
```

```
kill(getppid(), SIGCHLD);
```

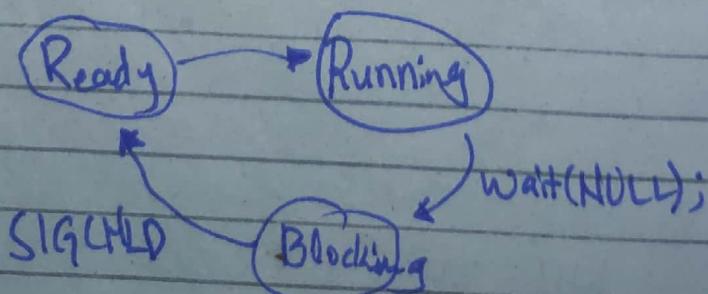
```
for(;;);
```

{

```
else {
```

```
wait(NULL);
```

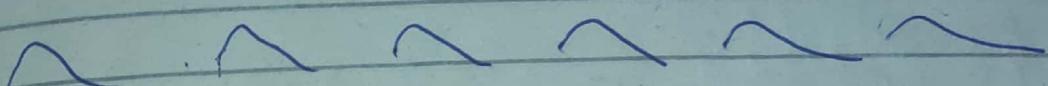
{}



→ Signal Lifetime → from Signal generation  
to  
Signal delivery

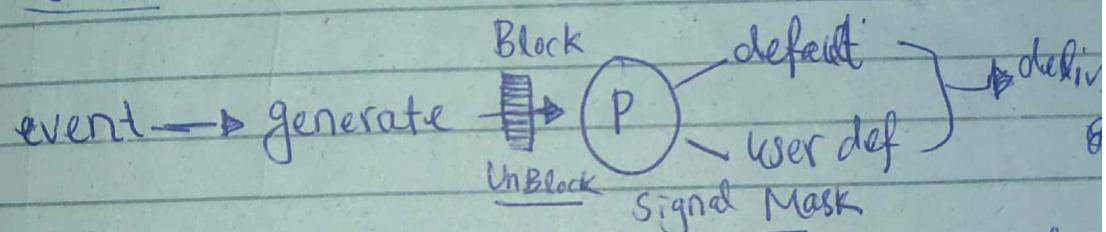
All signals have default action.

Pending state



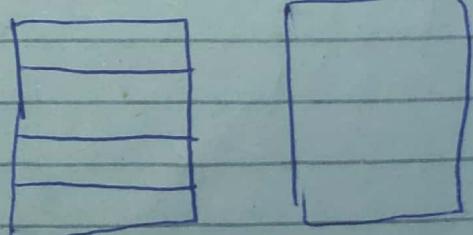
Date 18/12/2023

Signals :-



→ e.g. if we add SIGINT to it, then it will be blocked by sigset-t

→ Initially, it is fd table env empty.



① Block all signals except SIGINT.

`sigset_t` → integer.

- Step① fill `sigset_t`  
Step② Remove `SIGINT`  
from `sigset_t`

0	1
1	1
1	1
1	1
64	1

② Unblock all except `SIGINT`

Step③ Remove all

Step④ Add `SIGINT`

0
0
0
0
0
0

0
0
0
1
0
0

Step⑤ replace `sigset`

Now we need to replace our variable with Signal Mask of our process.

Now comes Sys calls.

For Step ①:-

int Sigfillset( sigset\_t \*myset);

For Step ③:-

int Sigemptyset( sigset\_t \*myset);

For Step ②:-

int Sigdelset( sigset\_t \*myset,  
-  
int signo);

For Step ④ : Adding signal

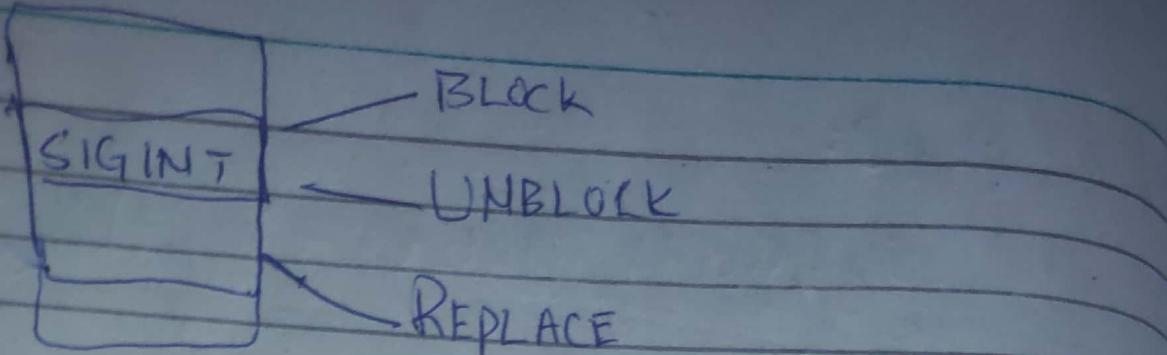
int Sigaddset( sigset\_t \*myset,  
int signo);

Now only remains the replace operation.

For step ⑤ :

int Sigprocmask( int how , sigset\_t \*set,  
sigset\_t \*set);

→ Blocking → SIG\_BLOCK  
→ UnBlocking → SIG\_UNBLOCK  
→ SET MASK



int main() {

`sigset(SIGINT, my_set);` → By default garbage  
`sigemptyset(&myset);` [0|0|0|0|0|0|0|0]

`sigaddset(&myset, SIGQUIT);`  
`sigaddset(&myset, SIGQUIT);`

[0|0|0|1|1|0|0]

`sigprocmask(SIG_SETMASK, &myset,`  
`&oldmask);`

`calc_GPA();`

M① [ `sigprocmask(SIG_SETMASK, &oldmask,`  
`NULL);` ]

M② [ `sigprocmask(SIG_UNBLOCK, &myset,`  
`NULL);`  
`for(;;);` ]

}

```
int kill ( int pid, int signo);  
int raise( signo);
```

int alarm ( int sec);  
(-1) → error  
Success → Remaining time of last → Starts a timer  
as it hits 0, then alarm.  
SIGALRM is generated.

```
int main ()  
{  
    alarm(5);  
    for(;;)  
}
```

*T = 5  
4  
3  
2  
1  
0  
SIGALRM*

*will return 0 in this case*

*default action  
process termination  
will occur*

```
int main() {
```

```
    alarm(60);
```

```
    for(i = 0; i < 1000; i++)  
        printf("Hello\n");
```

```
    int s = alarm(5);  
    printf("Rem time of prev alarm = %d\n",  
          s);
```

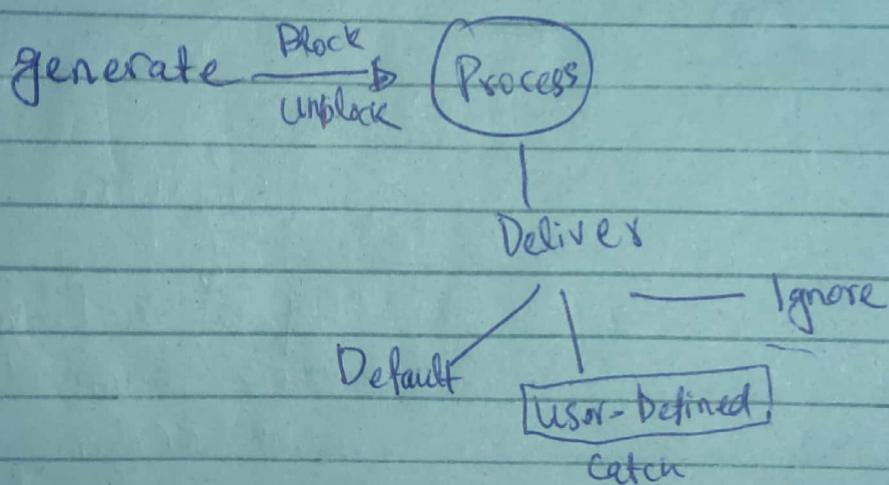
*will reset the timer*

Two alarms can be used to determine execution time of code.

Date 21/12/2023

### Catching Signals:-

↳ User Defined Handler called



Now we want to change the behaviour of interrupt signal no.

Always work  
and use exit(0) here

```
^C
void INTR_handler( int signal ){
    Count++;
    printf ("Sig interrupt received\n");
    return;
}
```

struct Sigaction {

```
int (*sig_IGN)(void); // SIG_IGN  
int (*sig_DFL)(void); // SIG_DFL  
int sa_flags; // user defined handler func address  
sigset_t sa_mask; // for blocking  
// additional signals during execution of user defined handler.
```

int count = 0;

```
main() {  
    struct Sigaction newact;  
    newact.sa_handler = INTR_handler;  
    newact.sa_flags = 0;
```

Sigemptyset(&newact.sa\_mask);

Sys call: sigaction(int signo, struct Sigaction \*newact, struct Sigaction \*oldact);  
for changing default handler

```
int Sigaction(int signo, struct Sigaction *newact,  
             struct Sigaction *oldact);
```

sigaction(SIGINT, &newact, &oldact);

Mr pause();  
always return (-1)

Unblock SIGINT

```
↑  
// int count = 0;  
// for(; ;)  
Pause(); // for( ; count < 10; )  
keeping CPU  
busy  
A  
(Busy Waiting)
```

③

}

```
printf("Process Normally Terminated\n");  
exit(0);
```

For restoring default behaviour;

① [ newact.sa\_handler = SIG\_DEF;  
sigaction(SIGINT, &newact, NULL);

② [ sigaction(SIGINT, &oldact, NULL);

Now solution for busy waiting is  
Pause sys call.

Pause() → wait for  
any signal

In order to wait for ten  
signals. use loop.

```
for( ; count < 10; )  
    Pause();
```

→ Atomically

sigwait() → unblocks a signal  
and waits for it.

sigsuspend() → wait for ~~added signals~~  
which are not present in signal set

sigwait() → wait until

int sigsuspend(const sigset\_t \*set);

int sigwait (const sigset\_t \*set, int signo);