SP After Mids:- 27/11/2023

Background Process Skipped.

select :-

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

tv_sec tv_usec

```
int main( ) {  // infinit delay
    printf ("Hello");
    select (1, NULL, NULL, NULL, NULL); //block
}
```

will wait indefinitely

Now we create delay using select.

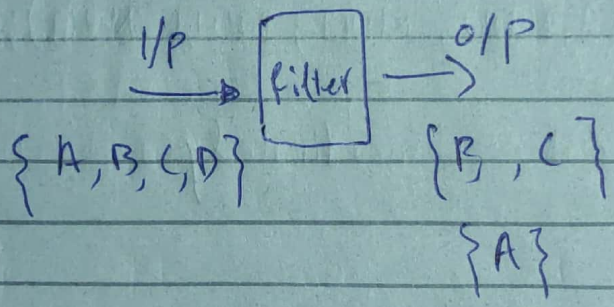
// 5 sec & 10 μsec delay

Now

```
int main () {
Struct timeval mytime;  printf("Hello \n");
mytime.tv_sec = 5;  mytime.tv_usec = 10;
int = 0  Select(1, NULL, NULL, NULL, &mytime);
    printf("world \n");
}
```

```
int main(){
    .
    .
    .
    // monitor 2 files
    .
    .
    .
int r= select(maxfd+1, &readset, NULL,
               NULL, &mytime);
printf("Remaining time = %d \t %d usec",
        mytime.tv_sec, mytime.tv_usec);
```

Filters & Redirection:-



O/P is a subset of i/p

head
tail
more
sort
grep

pipe

ls | head

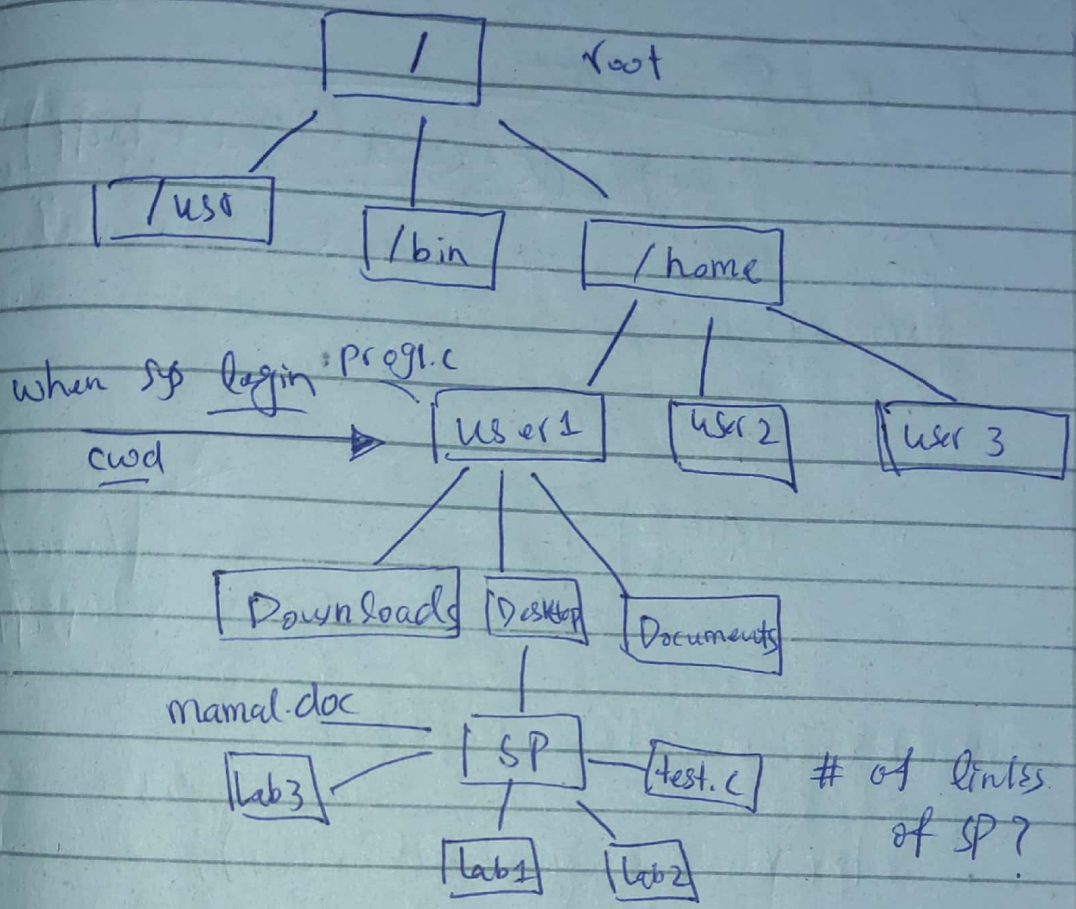
ls | tail

ls | more

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

Progl.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.

size

```
char * getcwd(char * buff);
```

for this we need to allocate memory.

```
int main(){
```

x → /home/kesri
buff

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

size

char * x = getcwd(buff,);

}

Now we want to change our current dir.

```
int chdir(const char* path);
```

error -1

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

Directory → A special file which contains directory entries. → dirent

- name → d_name
- inode # → d_ino

SP

name	inode #
test.c	57
manual.doc	126
Lab1	500
Lab2	1023
Lab3	1110
.	1234
..	1552
NULL	

Now we want to open SP directory

Call.

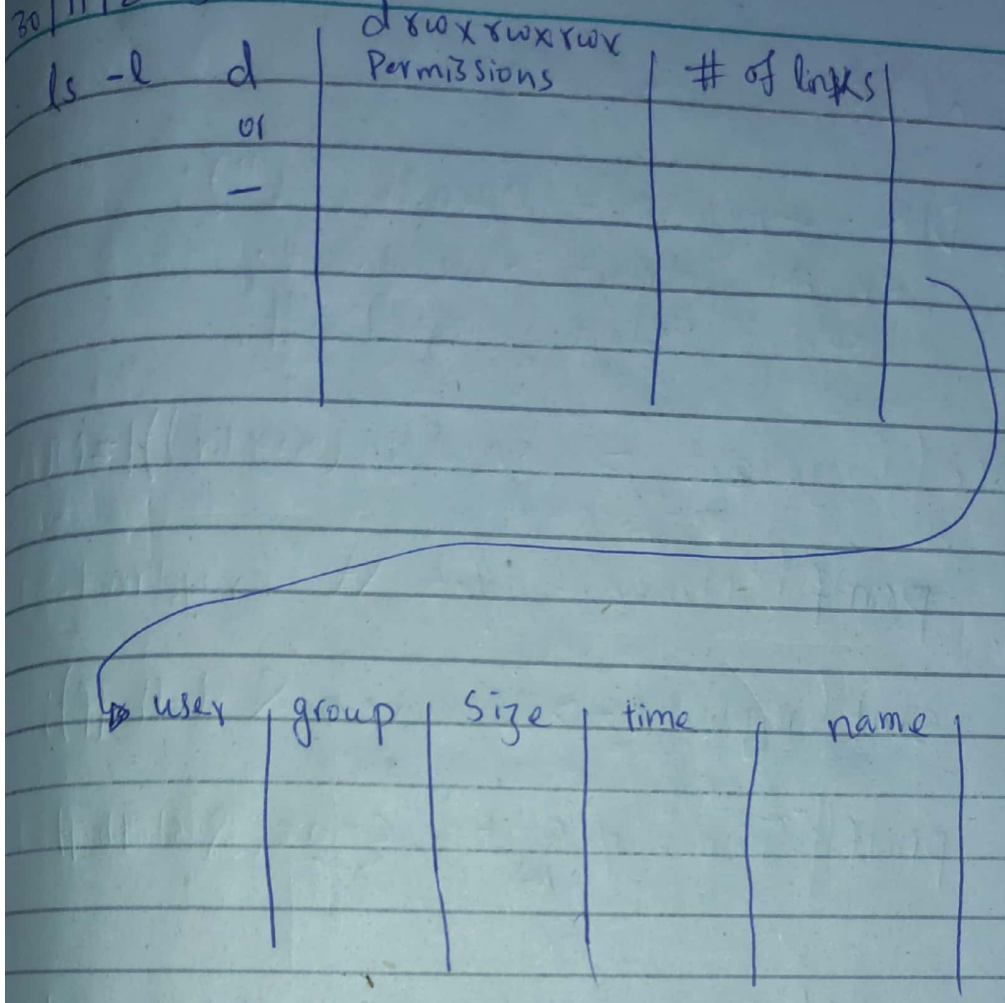
DIR * ~~char *~~ opendir (const char *dirname)

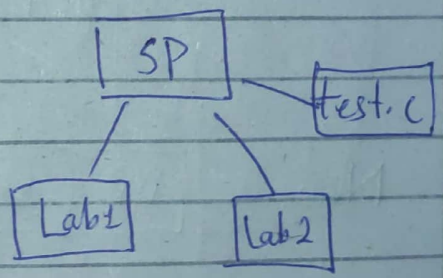
Now we want to read from SP another sys call,

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.

<sys/stat.h>

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

Struct Stat

1) int st_dev
2) int st_ino
3) int st_mode
4) int st_nlink
5) int st_uid
6) int st_gid
7) time st_atime
8) time st_ctime
9) time st_mtime
10) time st_atime

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

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

    printf("Name = %s \t", mydirent->d_na

    stat(mydirent->d_name, &statbuff);

    printf("Time = %s \t", ctime(&statbuff.st_a

    printf("Size = %d \t, gid = %d \t uid = %d
        Links = %d \t", statbuff.st_size, Statbu
        st_grd, statbuff.st_uid, Statbuff.st_nlin
```

Now `int st_mode`

MACROS

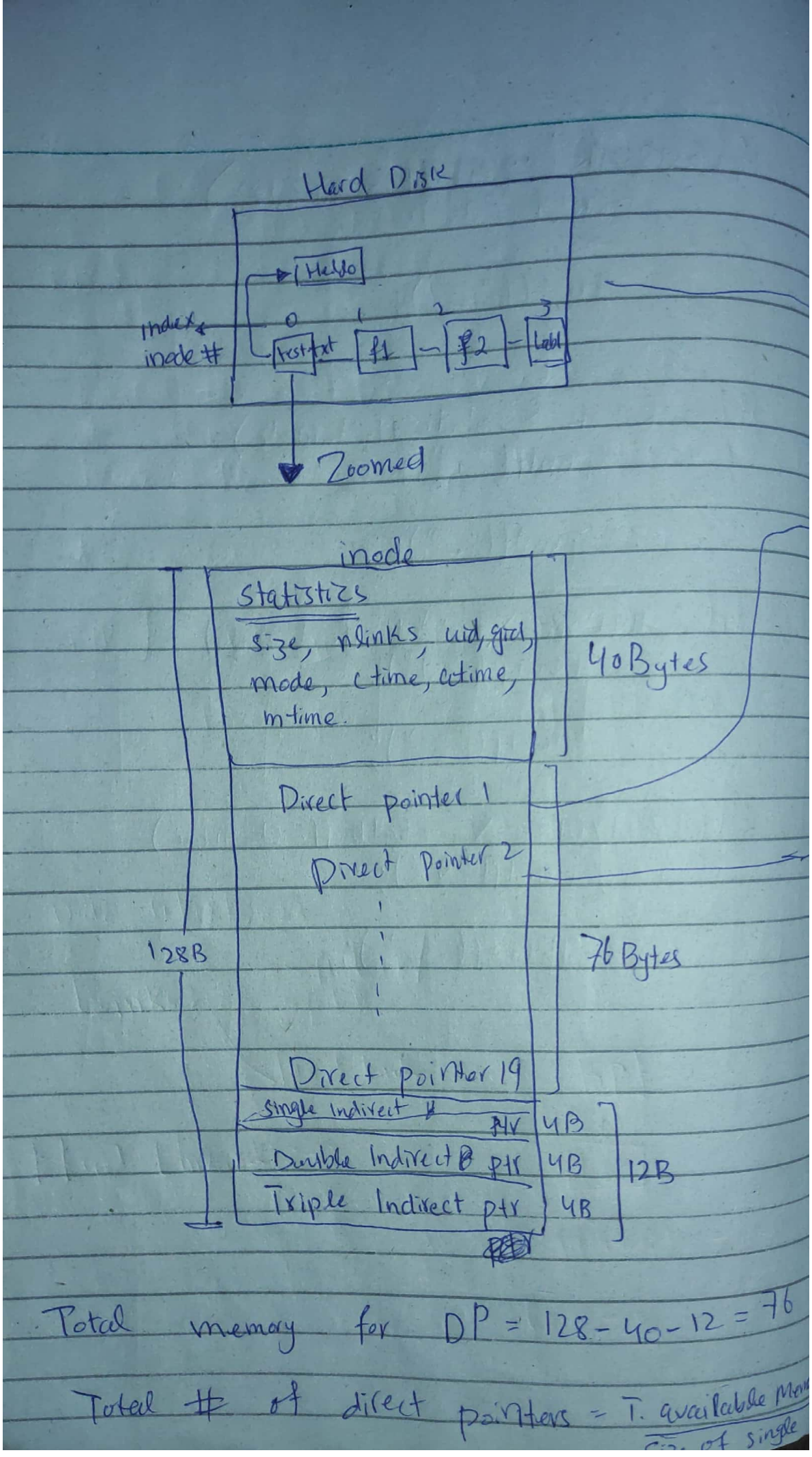
Type of file

S_ISDIR(m)
S_ISREG(m)
S_ISFIFO(m)
S_ISSOCKET(m)

Permissions

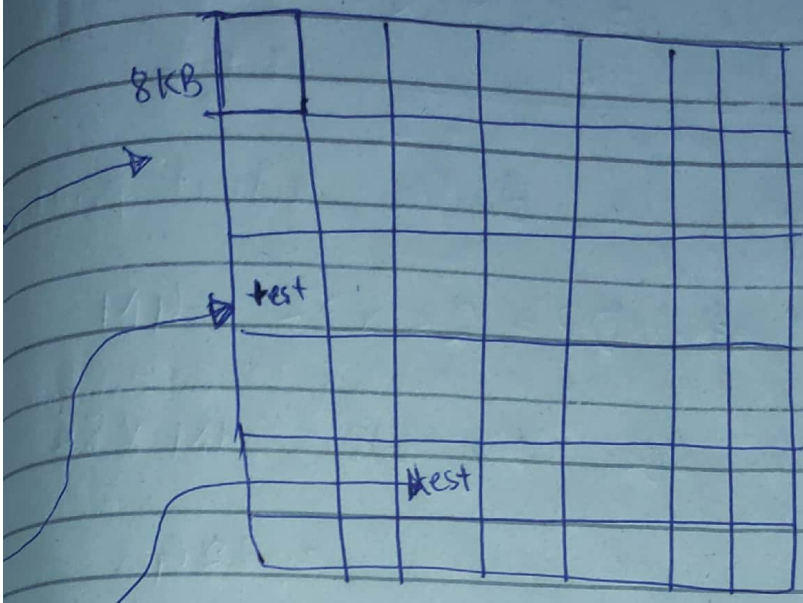
S_IRUSR

```
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("_");
```



Total memory for DP = 128 - 40 - 12 = 76

Total # of direct pointers = T. available Mem / size of single



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

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

$= 19 \times 8KB = 152KB$

SID

Total Memory available for DP = 8KB

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

Max file using SID = # of ptrs * Size of block

$= 2K * 8K = 16M$

DIP

How we use double indirect pointer

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

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

// // // = 32G

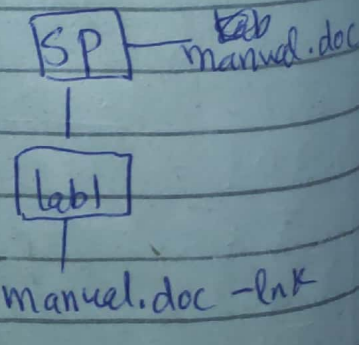
TIP Total # of DP = $2K * 2K * 2K = 8G$

Max file size using TIP = 64 TB

Date 4/12/2023

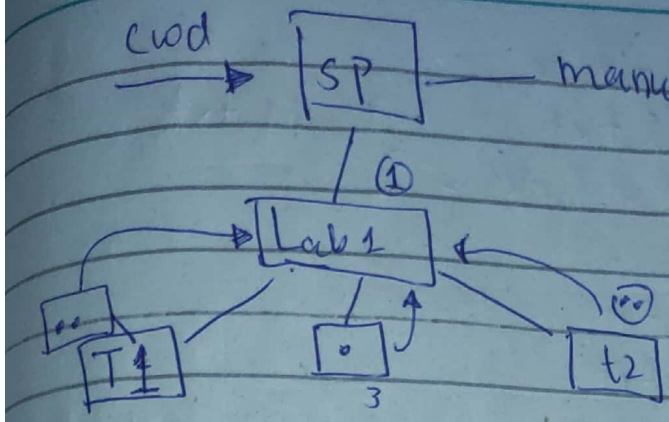
Links: — Soft Link / Symbolic
— Hard Link

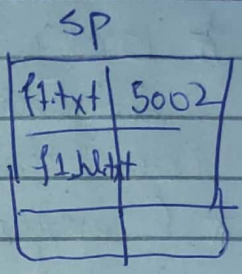
Hard Links:-



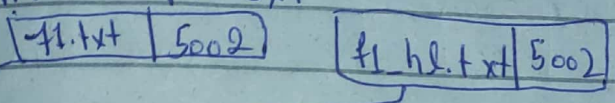
ls -l H.L

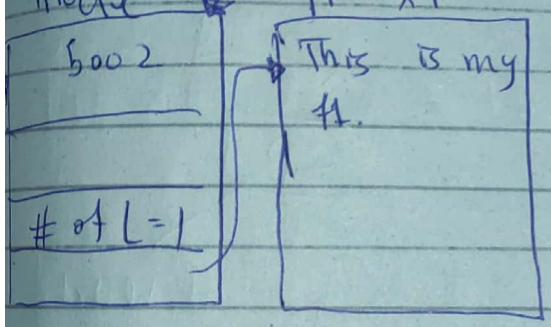
drwr--- # links

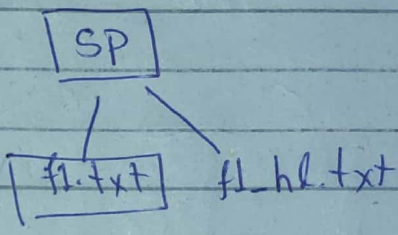




(A) Vim f1.txt







when new file created.

① inode
② Disk block
③ Directory entry

(B) Link

Command: ln f1.txt f1_hl.txt

Syscall: 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 ... links incremented.

(C)

Now if we write

cat f1_hl.txt | cat f1.txt

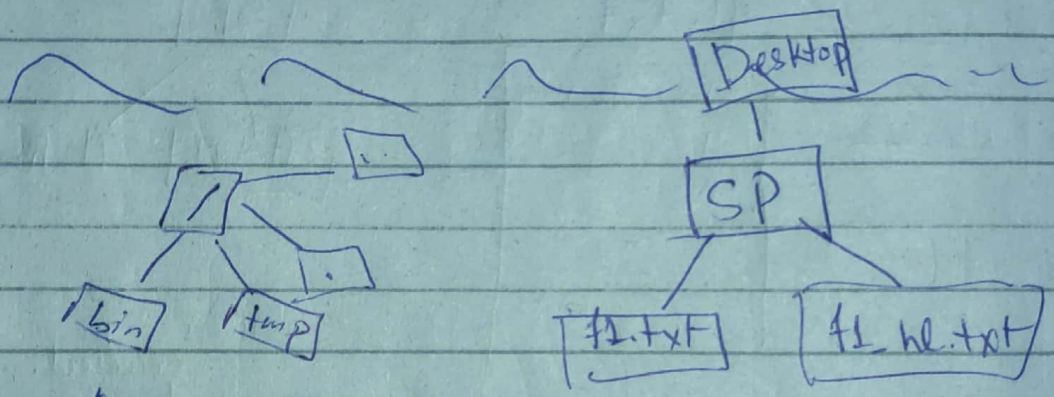
and no. of links = 2

(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 rm f1.txt

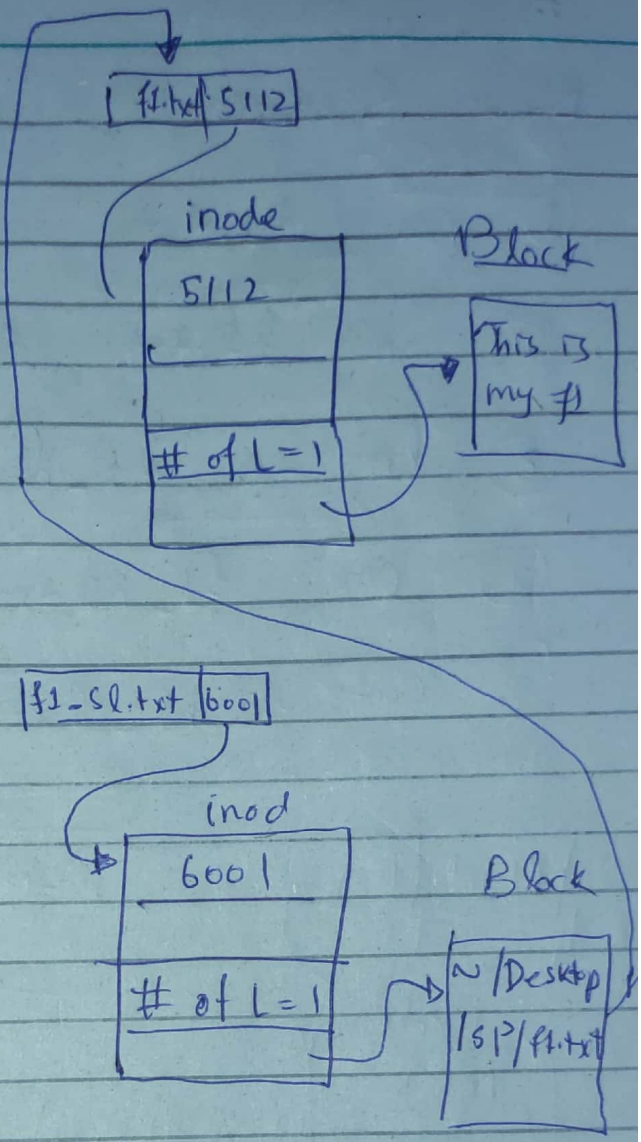
⑥ cat f1_sl.txt

↳ file doesn't exist error.

⑦ vim f1.txt

hello world

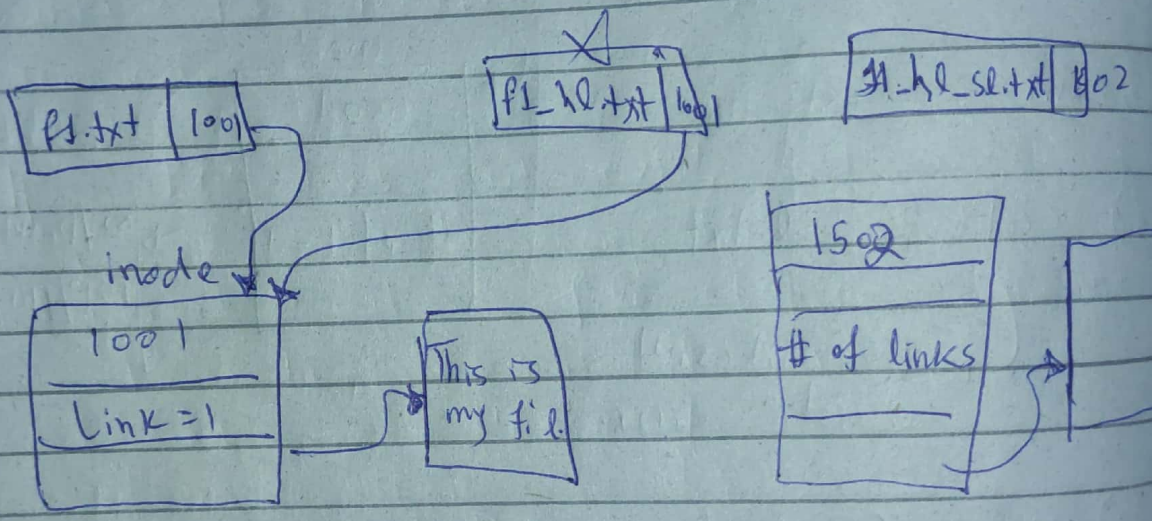
cat f1_sl.txt.
will print hello world.



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

New 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

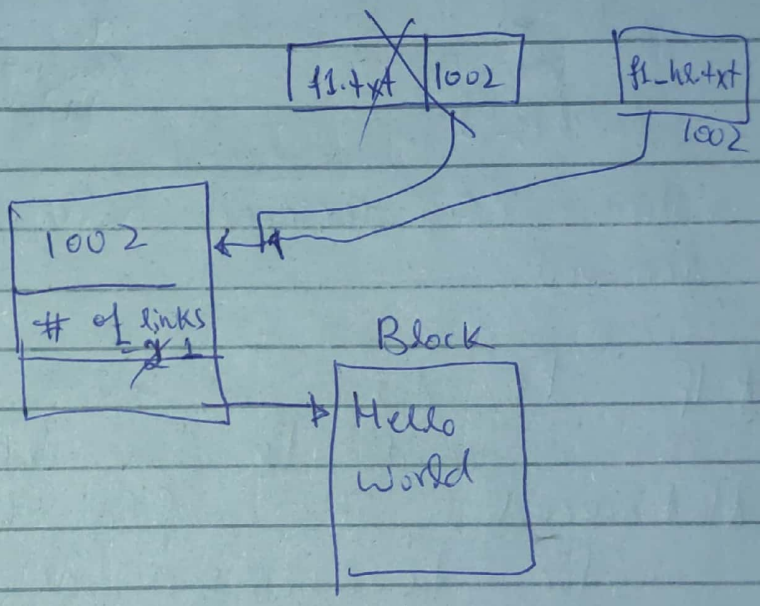


1. vim f1.txt ("Hello World")
2. ln f1.txt f1_hl.txt
3. prog.c

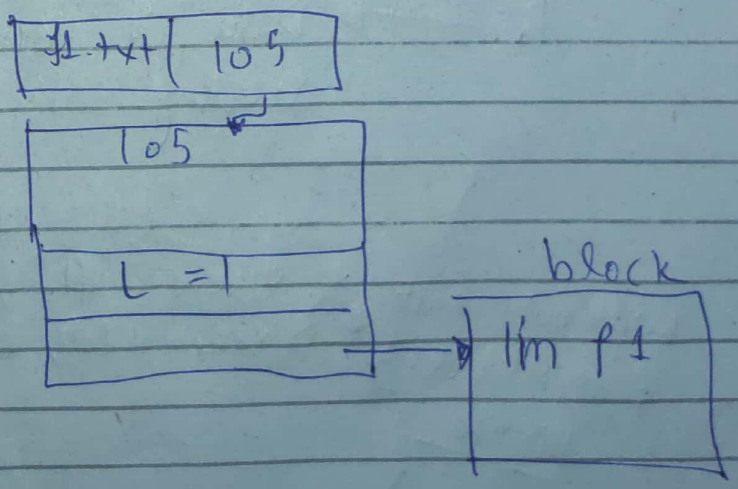
open f1.txt
read f1.txt into buff.
rm f1.txt

create f1.txt
write ("Im f1");
close.

(4)



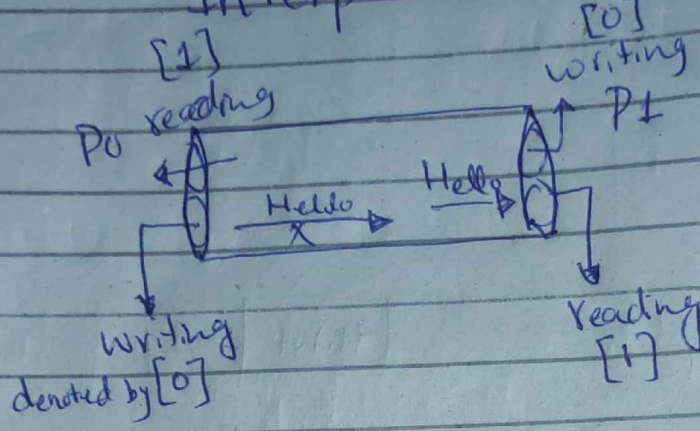
creating f1.txt



Date 7/12/2023

Pipes:-

Interprocess Communication

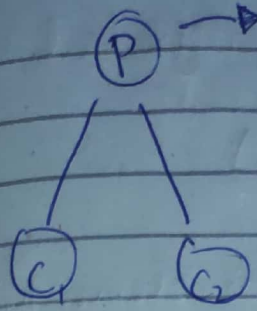


When P1 recieved hello, then it is removed

* If P1 does not exist, then Po can read it's own data.
* In Backend, it's file. (Special file).
  For two reasons it is special
  1 ⟶ Unnamed
  2 ⟶ Data is erased after reading.

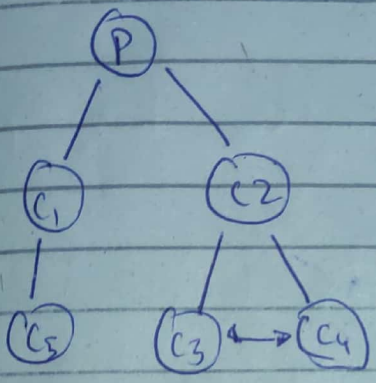
Will not wait if NONBlocking mode.

```
open("f1", O_NONBLOCK)
read(fd, buff, 100);
```



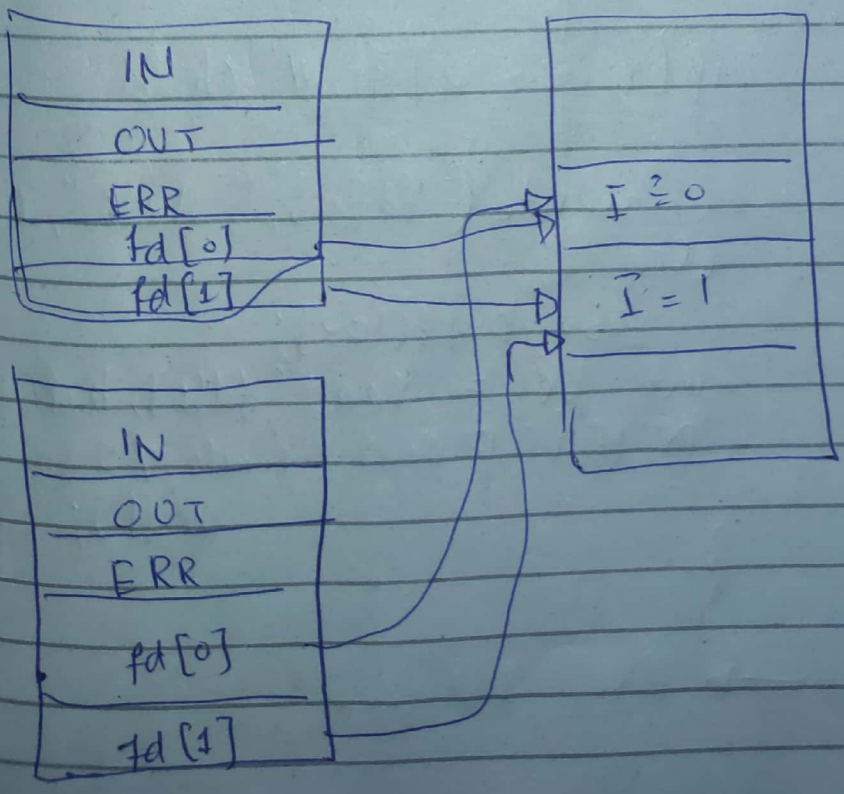
→ hierarchy top
pipe creator.

Parent child relationship must.



Parent

fd table



Now comes the pipe sys call.

int pipe(int fd[2])

success 0 ← error -1

fd[2] ← array of two elements.

Step ① Pipe creation:

```
int main(){
        int fd[2];
        int x = pipe(fd);          } Step ①
        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);
        }
```

Ftp Server ⟶ file listing.

Error Case:- File Interrupt
Resources Exhaust

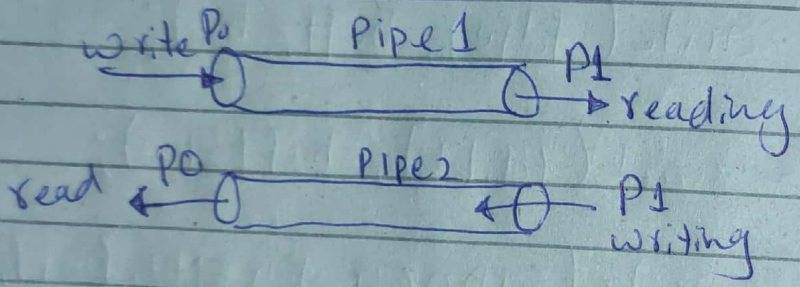
Two way Communication:-

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

        write(STDOUT_FILENO, buff, br);

    }

    else {

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

        writ (STDOUT_FILENO, buff ,br);

        br = read (STDIN_FILENO, buff

        write ( fd[0], buff, br );

    }
```

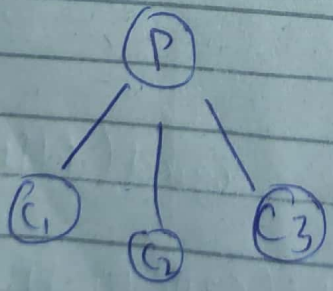
whenever there is write and then read put sleep(2);

```
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[0], "Hello", 6);
    }
    else{
        char buff[100];
        br = read(fd[1], buff, 6);
        printf("child read %s\n", buff);
    }
```

Date 11/12/2023

Fifos / Named Pipes :-

- Pipes are unnamed
- First In First Out

- FIFO's
  - Exist after process end
  - Named file
  - Data erased after read

* Syscall :-

```
int mkfifo(const char *fifoname, int perm);
-1: E
0: S
```

Half Duplex Comm

- Two way Communication Sequential

Whenever client Server Model, make two programs.

Client	Server
seq →	recieve get O&
recieve rep & display ←	respo

Client.c

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

Server.c

```
int main(){ char buff[100];
    int r = mkfifo("timefifo", S_IRWXU);
    if(r==-1 && errno!=Exxist){
        perror("mkfifo failed\n");
        return -1;
    }
    int fd = open("timefifo", O_RDWR);
    block here
    int br = read(fd, buff, 100);
    time_t t = time(NULL);
    char* time_str = ctime(&t);
    write(fd, time_str, strlen(time_str)+1);
```

fifo solved the parent child problem

Make two fifos for avoiding write and read problem.

* Select sys call ~~must~~ can be used for unknown sequence.

Prog. 6

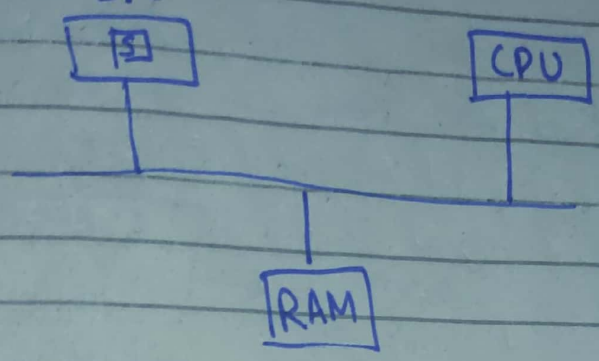
```
main(){
    int x = mkfifo("chatfifo", S_IRWXU);
    // error

int fd = open("chatfifo", O_RDWR);
  fd_set readset;
  FD_ZERO(&readyset);
  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);
}
}
```

P1	P2
Hello →	→ recieved
Hi ←	← Hi

CHAPTER (8): Signals

H.W events
S.W events → Signal

KB I/O

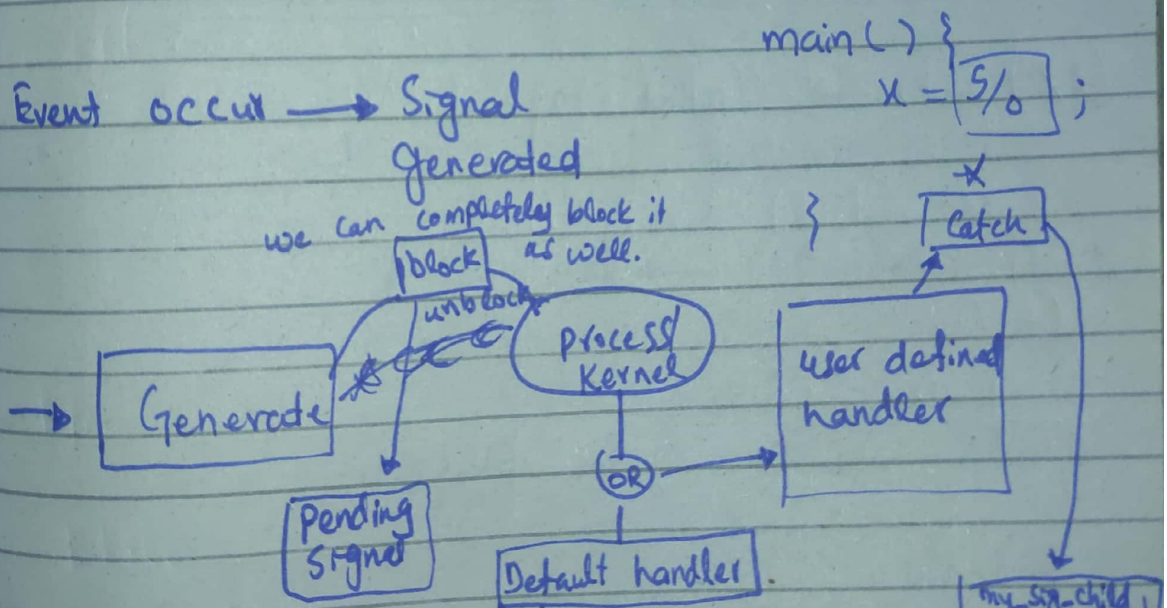


we store it in buffer

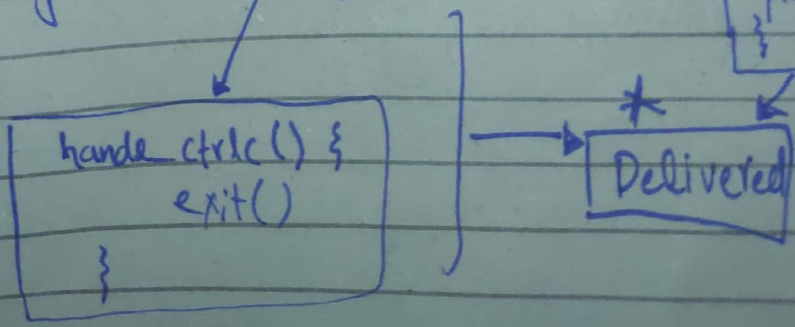
Through signal we know that event has occurred.

Event occur → Signal Generated

we can completely block it as well.



→ No. of signals are system dependant.



List of available signals

$kill -l

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

Default Action

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

If user enter 1 then we can generate SIGUSR1

event not defined

$stty → for listing signals generated by terminal.

: ^C → SIGINT → Sig interrupt

wait() → wait for SIGCHLD

```
pause();
sigwait();
Sig suspend();
```

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 -9 101
```

PID

SIGNO

```
int kill(int PID, int signo);
```

-1 : E
0 : S

```
int main() {
    kill(101, 9/SIGINT);
}
```

SIGKILL
SIGTERM

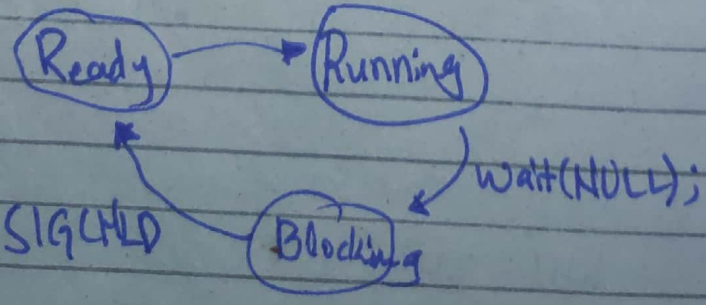
default behaviour ⇓ Abnormal Termination.

SIGTERM

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

`int raise(int signo);` ⟶ deliver a signal to itself.

```
main(){
    x = fork();
    if(x==0){
        sleep(1);
        exit(0);
        kill(getppid(), SIGCHLD);
        for(;;);
    }
    else {
        wait(NULL);
    }
}
```



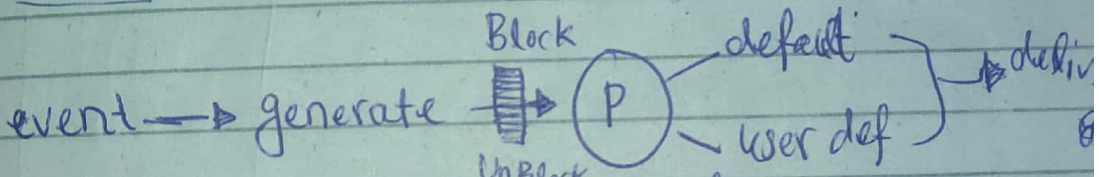
→ Signal Lifetime → from Signal generation to Signal delivery

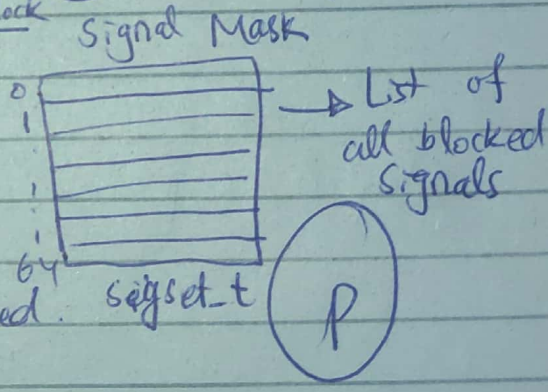
All signals have default action.

pending state

Date 18/12/2023

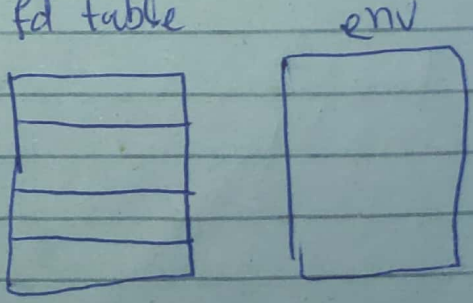
Signals :-





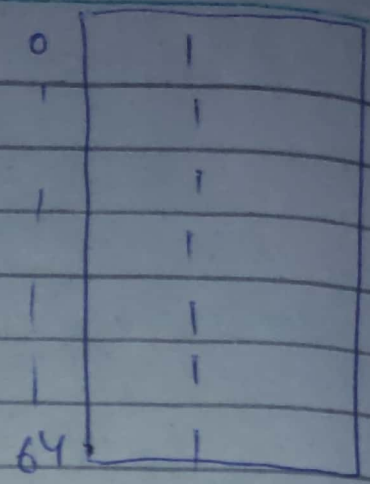
→ e.g if we add SIGINT to it, then it will be blocked.

→ Initially, it is empty.



① Block all signals except SIGINT.

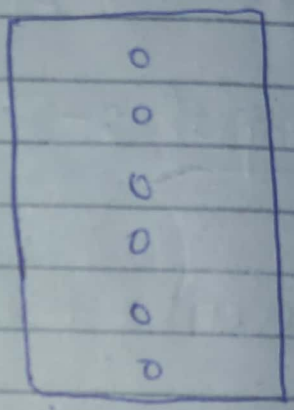
sigset_t ⟶ integer.



Step① fill sigset_t
Step② Remove SIGINT from sigset_t

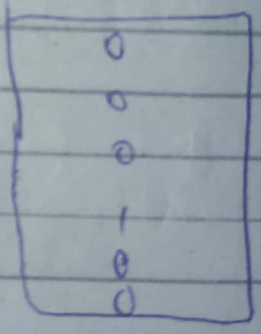
② Unblock all except SIGINT

Step① Remove all



Step② Add SIGINT

Step③ 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);

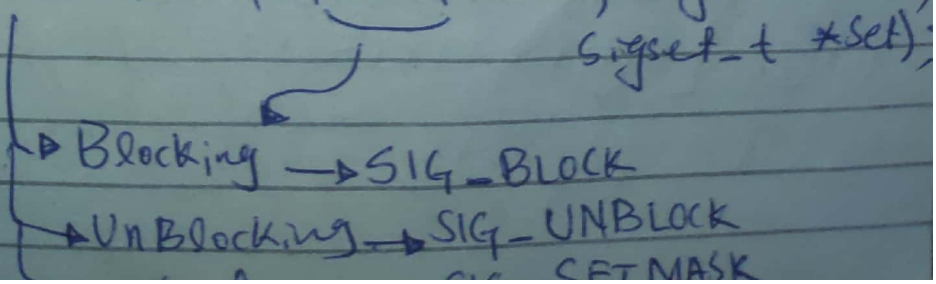
for step ④: Adding signal

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);




```
int main(){
    sigset_t myset;      → By default garbage
    sigset_t oldmask;
    Sigemptyset(&myset);
    Sigaddset(&myset, SIGINT);
    Sigaddset(&myset, SIGQUIT);
    Sigprocmask(SIG_SETMASK, &myset,
                &oldmask);
    Calc_GPA();
```

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

M② // [sigprocmask(SIG_UNBLOCK, &myset, N

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

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

int alarm ( int sec);
```

(-1) → error

Sucess → Remaining time of last alarm.

→ Starts a timer as it hits 0, then SIGALRM is generated.

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

will return 0 in this case

T = 5 4 3 2 1 0

SIGALRM → default action process termination will occur

```
int main() {
    alarm(60);
    for(i=0; i<1000; i++)
        printf("Hello\n");

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

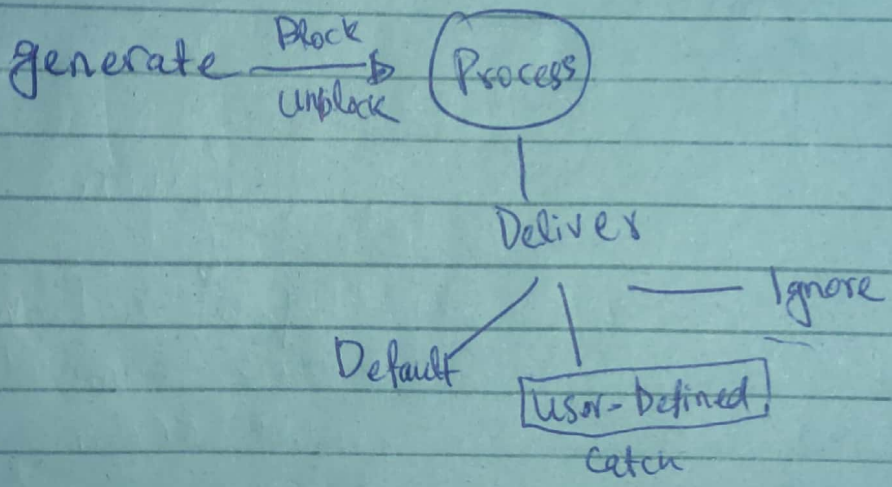
→ will reset the timer

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 void

^C

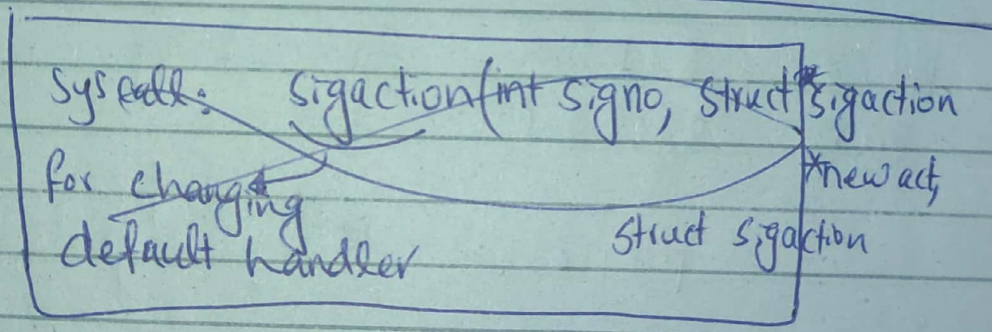
```
void INTR_handler(int signo){
    Count++;
    printf("Sig interrupt recieved\n");
    return;
}
```

can't use exit() here

```
struct sigaction {
int ((void (*) sa_handler) (int);   // SIG_IGN, SIG_DFL, user defined handler func address
    int sa_flags;                   // 0
    sigset_t sa_mask;               // for blocking additional signals during execution of User defined handler.
}
```

```
int count = 0;
main() {                                  → Block SIGINT here
    struct sigaction newact;
    newact.sa_handler = INTR_handler
    newact.sa_flags = 0;
```

sigemptyset(&newact.sa_mask);



int sigaction(int signo, struct sigaction *newact, struct sigaction *newact);

sigaction(SIGINT, &newact, &oldact); (NULL)

pause();
always return (-1)

Unblock SIGINT

// int count = 0;
// for(;;);
pause(); // for(; count<10;)

keeping CPU Busy
(Busy Waiting)

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 for loop.

for(; count<10;)
pause();

→ Atomically

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

sigsuspend() → wait for ~~added signals~~ (those signals which are not present in sigmask)

sigwait() → ~~wait~~ old+a

int sigsuspend (const sigset_t * Set);

int sigwait (const sigset_t *set, int signo);

→ Atomically

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

sigsuspend() → wait for those signals which are not present in sigmask

sigwait() → ~~wait~~ old+a

int sigsuspend(const sigset_t * Set);

int sigwait(const sigset_t *set, int signo);

Date 28|12|2023

Waiting for Signals:-

main(){
// Block all signals
→ ?

→ ? SIG CHLD Atomic
// Unblock signal
Pause();

}

```
int sigsuspend(const sigset_t *newmask);

flag = 0;

main(){

    sigset_t newmask
    sigfillset(&newmask);

    sigprocmask(SIG_BLOCK, &newmask, NULL);
                SIG_SETMASK
    // Block all signals.

    sigdelset(SIGINT, &newmask);

    // Now SIGINT is removed
       from set

    // we can call sigsuspend now
indefinite block  if(flag==0){
        sigsuspend(&newmask);
    }
    // Process Termination will be
    // Abnormal. For normal termination
    // use handler.
}

void sig_handler(int signo){
    flag=1;
    return;
}
```

Now we want to wait for our desired signal only.

```
main(){

  sigset_t newmask, oldmask;

  sigprocmask(SIG_SETMASK, NULL, &oldmask);
  // Got old/default mask.
  sigprocmask
  sigaddset(SIGINT, &oldmask);

  sigprocmask(SIG_BLOCK, &oldmask, NULL);

  sigdelset(SIGINT, &oldmask);

  if(flag==0)
     sigsuspend(&oldmask);

}
```

int sigwait(const sigset_t, * int *signo);

```
main(){
    sigset_t mask;
    sigemptyset(&mask);
    sigaddset(SIGINT, &mask);
    sigaddset(SIGQUIT, &mask);
    Sigprocmask(SIG_BLOCK, &mask, NULL);
    int sig;
    sigwait(&mask, &sig);
    if(sig == SIGINT){
        printf("Sig int received\n");
    else
        printf("Sig quit received\n");
}
```

Four Ways

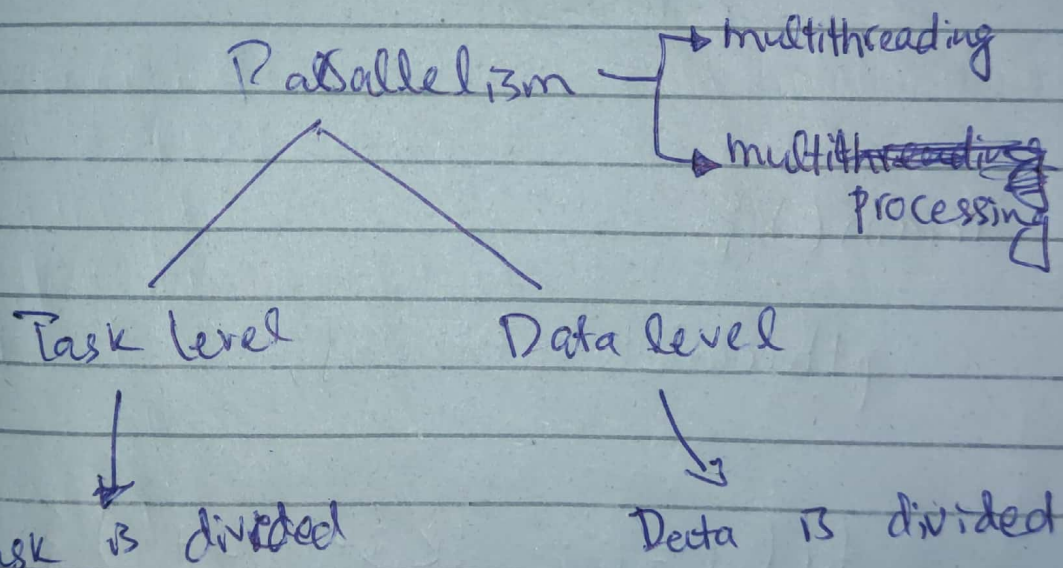
1. `for(;;)` // infinite loop
2. `pause()`
3. `Sigsuspend();` → pause() + Non block

④ sigwait();

label sigset jump	goto siglongjump

Date 08/01/2024

Threads:- Chapter No. 12



Overheads of multiprocessing:-

- Memory overhead
- Process Creation
- Context Switch.

→ Divide our program to multiple flows.

→ Data level parallelism

```
void * threadfun (void *x) {
    printf("Thread ID = %d \n",
           pthread_self());
}
```

→ implicit pthread_exit

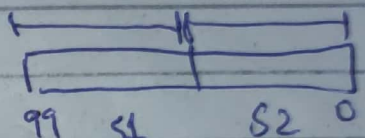
```
main() {
    pthread_t tid[3];
    for( i=0; i<3; i++ ) {
        pthread_create(&tid[i], NULL,
                       threadfun, NULL);
    }
    pthread_exit(NULL);
    exit(0) // Cascaded Termination
}
```

T2 50 99

Now if we want to wait for all threads.

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

Now we want to implement data level parallelism.




```
int sum = 0;  int array[100];
main(){
~~int sum~~;
pthread_t tid[2];

    for( int i=0; i<2; i++){
        pthread_create(&tid[i], NULL, sumArray,
                       (void*)&i);

for(i=0; i<2; i++)
    pthread_~~creat~~join (tid[i], NULL);

Printf(" Sum of all = %d \n", sum);

exit(0);

}
```

```
void * sumArray (void *arg){
    int index = *((int *)arg);
    int start = index *
    .
    .
    .
    int local_sum = 0
    for (i = start; i < end; i++){
        local_sum += Array[i];
    sum = sum + local_sum;
}
```