SP L# 01

SELECT FUNCTION

Monitors files ↑

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

- maxfd → # ready files
- t → tv_sec, tv_usec

```
int main(){                          // to block a program
   printf("Hello");
   Select( 1, NULL, NULL, NULL, NULL);// blocks the pgm.
}                                     (infinite delay)
```

wont print Hello becaz "Hello" in buffer

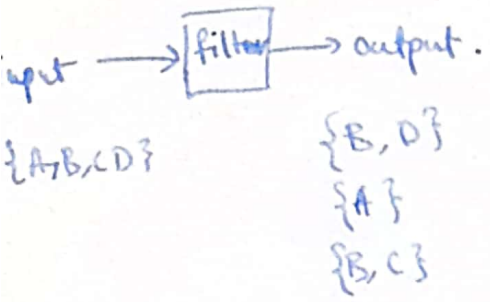
```
//5 sec & 10 sec delay
int main(){
   printf(Hello);
int r=select( 1, NULL, NULL, NULL, &mytime);
}
```

```
struct timeval mytime;
mytime.tv_sec = 5; mytime.tv_usec = 10;
```

Timer: 5:10, 5:9, 5:8, ..., 1:00, 0:99, ..., 0:0

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

FILTERS & REDIRECTION :



head, tail, more, sort, grep

>> ls
↓
lists all the commands

>> ls | head

>> ls | tail

>> ls | more

>> ls | sort | grep "f1"

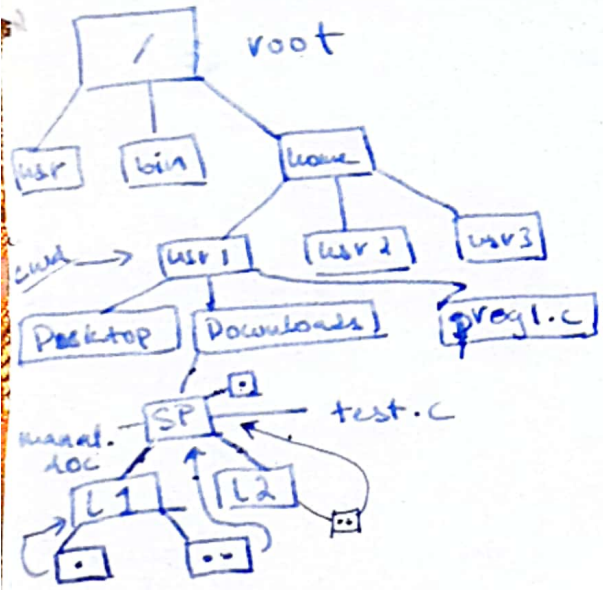
error → -1
success → 0

int chdir(const char *path)
char *getcwd(char *buff)
↓
sys calls.

FILES & DIRECTORIES

CHAPTER 5

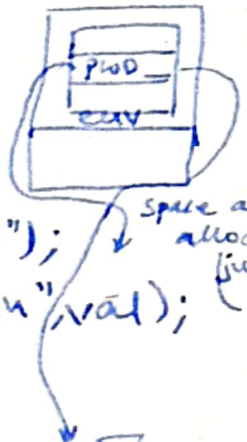
DIRECTORIES :



No. of links of SP ⇒ 4

Prog 1.c

```
int main(){
  char *val = getenv("PWD");
  printf("my CWD = %s \n", val);
  char buff[255]
  char *r = getcwd(buff);
  printf("My CWD = %s \n", buff);
  int rv = chdir("../Downloads/SP")
  r = getcwd(buff);
  printf(My CWD (as above));
```



Absolute (starts from root) → cd /home/usr1/Downloads/SP

Relative (starts from dots & double dots)
→ cd ./Downloads/SP
→ cd .. /usr1

int

SP L#2

text.c — SP ← CWD

② ③

DIRECTORY → contains directory entries → dirent

SP

name → name, inode #

SP

name	inode #
test.c	125
manual.doc	300
lab1	628
lab2	1100
.	1541
..	1862
NULL	

index pointer

Steps

1. open prog1.c
2. read fd size.
3. print
4. repeat 2 unless upon you reach end of file

files

Implementing LS command

Sys Call

DIR * opendir(const char *dirname)

struct dirent *readdir(DIR *dirp);

LS command

```
int main(){
    DIR *dirp = opendir("SP");          // assuming we are on downloads
    struct dirent *direntp;
    while(1){
        direntp = readdir(dirp)
        if(direntp == NULL)
            break;
        printf("Dirent name - %s if Dirent inode = %d \n}
                direntp -> d_name, direntp -> d_ino);
```

SP Lt#2

text.c — SP
lab1 lab2

DIRECTORIES:

ls -l ⇒

	mode	# of links	user	group	size	time	name
d	rwx rwx r wx	2	student	student	6 4 8	12 Feb	Lab2
-	r - - - - w - - x	1	student	student	1024 B	13 Feb	text.c

int

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

error -1 success 0

<sys/stat.h>

-, d, f, s

atime -> access time
mtime -> modify time
ctime -> create time

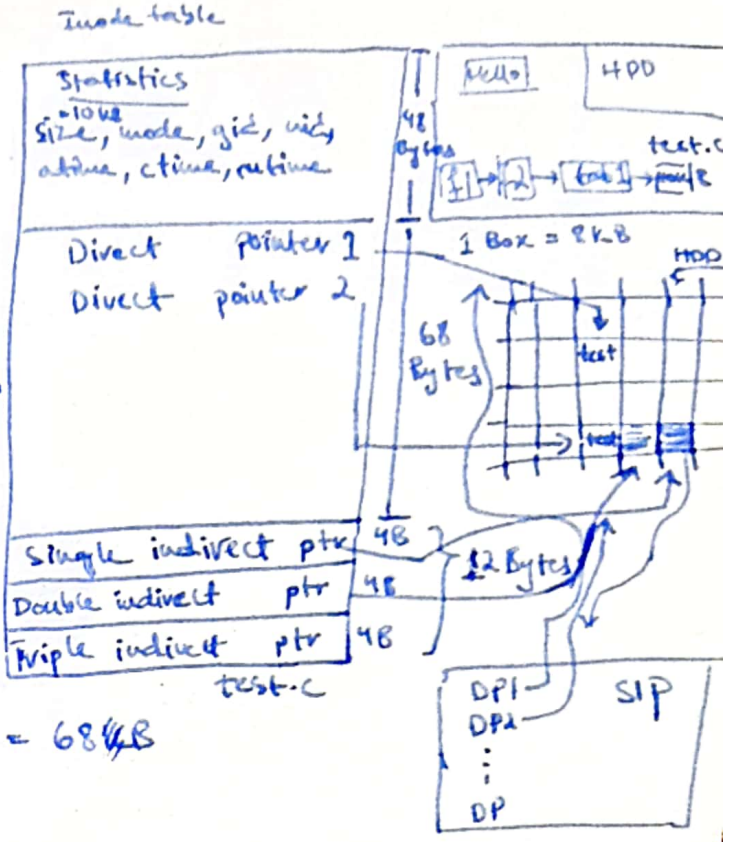
steps
1. open current working directory.
2. read until reaches NULL
3. print the information

struct stat
1) int st_dev
2) int st_ino
3) int st_size
4) int st_mode
5) int st_nlink
6) int st_uid
7) int st_guid
8) time ctime
9) time mtime
10) time atime

```
int main(){
    DIR *mydir = opendir(".");
    struct dirent *mydirent;
    struct stat statbuff;
    while( (mydirent = readdir(mydir)) != NULL){
        print("Name = %s \t", mydirent->d_name);
        stat(mydirent->d_name, &statbuff)        print info of this
        printf("Time = %s \t", ctime(&statbuff.st_atime));
        printf("Size = %d \t", IDs = %d \t %d \t", statbuff.st_size,
                statbuff.st_gid, statbuff.st_uid);
        printf("Links = %d \t", statbuff.st_nlinks);
        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("-");

return 0;
}
```

S_IRUSR → checks bitwise mode & shows permissions.



Total memory available for direct pointers = 68 KB

size of single pointer = 4B

Total # of direct pointers = $\frac{\text{Available memory}}{\text{size of ptr}} = \frac{68}{4} = 17$ >

Max file size using DP (Direct pointers) = # of DP * size of Block

= 17 * 8 KB ⇒ 136 KB

Max total (file size) = 136KB + 16MB + 32GB + 64TB =

Single indirect pointer (SIP)

Total memory available for DPs = 8KB

Total # of DP = $\frac{M}{S} = \frac{8KB}{4B} \Rightarrow 2K = 2048$

Max file size using SIP = # of Ptr * size of Block

(Same formula) = 2K * 8KB = 16 MB

Double indirect pointer (DIP)

Total No. of DIP = 2K * 2K = 4M

Max file size using DIP ⇒ 4M * 8KB = 32GB

Triple indirect pointer:

Total # of DIP = 2K * 2K * 2K

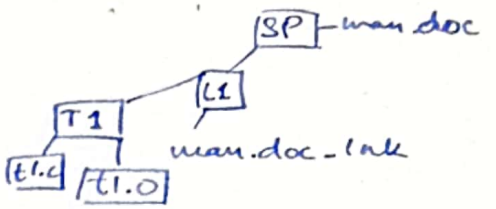
= 8G

Max file size using TIP

= 8G * 8KB

⇒ 64TB

SP # 03



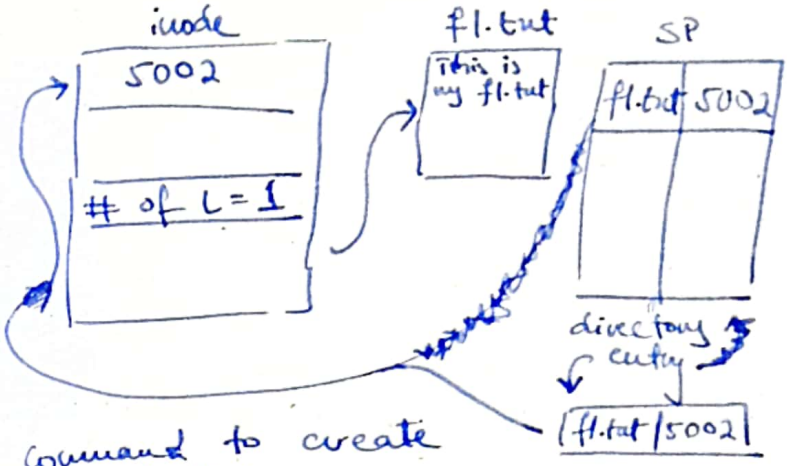
inode table for every file

Links:

Two types of links → hard link
↳ soft link / symbolic link

HARD LINKS:

when we do ls -l ⇒ output → -d xyz-xyz--- # of links ... these are hard links of original files.



⇒ Vim fl.txt gedit fl.txt
when a file is created an inode table is also created.
directory entry is also created.

- ① inode is created (pointed to
- ② directory entry created.
- ③ Space on disk allocated (Block)

Command to create hard link:

ln fl.txt fl_hl.txt

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

0 → success
-1 → error

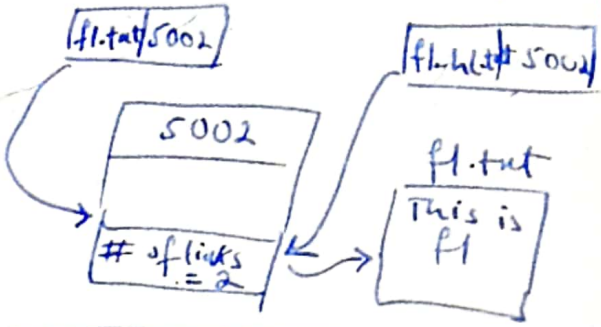
Steps :

① Vim fl.txt
② ln fl.txt fl_hl.txt
③ cat fl_hl.txt
cat fl.txt
④ rm fl.txt (deleted the original file)
cat fl_hl.txt → after this # of links = 1

ls -l	# of links
fl.txt	2
fl_hl.txt	2

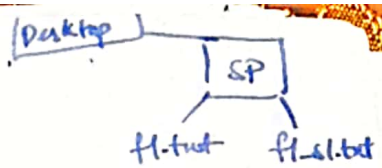
when hard link is created only directory entry is created

fl.txt	5002
fl_hl.txt	5002



rm command only removes the directory entry & decreases the no. of links.

Soft links (symbolic links)



Fln -s fl.txt fl_sl.txt

int symlink (const char *src, const char *dst)

0 → success

-1 → error

Steps:

① Vim fl.txt

② ln -s fl.txt fl_sl.txt

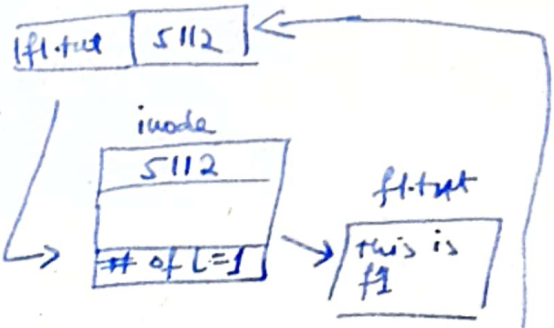
③ cat fl.txt
cat fl_sl.txt
→ same answer from both!

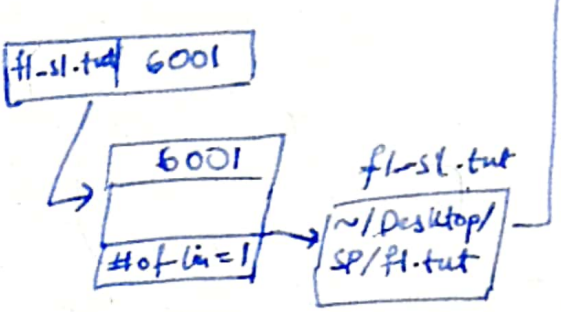
④ Vim fl_sl.txt → when we update it from here the original file is also updated.

⑤ rm fl.txt → (removes the directory entry only)

→ # of L=0 → after this

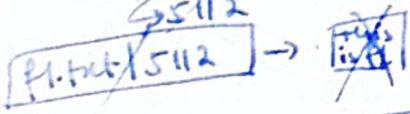
All is lost in the space of everything.



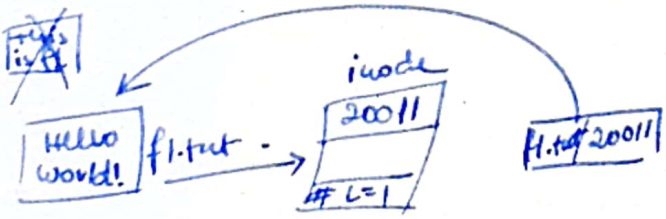


Same as copy but the file contains the path of original file.

deletes → inode → 5112



⑥ Vim fl.txt



Same thing but new file

⑦ cat fl_sl.txt

output ⇒ Hello world!

— x — x — x — x — x — — x — x — x — x — Final x — x — x —

No output → error } till step ⑤

— x — x — x — x — x — — x — x — x — x —

① Vim fl.txt

② ln fl.txt fl_hl.txt

③ ln -s fl_hl.txt fl_hl_sl.txt

④ rm fl_hl.txt

⑤ cat fl_hl_sl.txt

⑥ cat fl.txt fl_hl.txt

Check camera roll

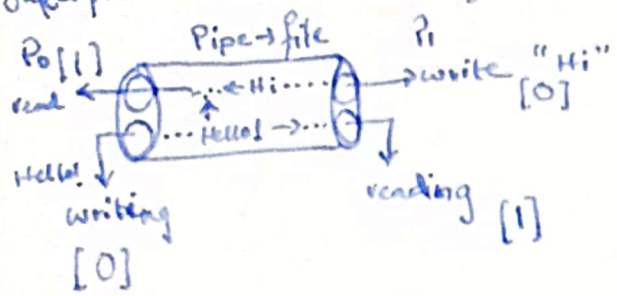
4th Dec 2023

must for second (eg:

SP L#04

PIPES:

Interprocess communication:



int pipe(int fd[2]);

0 → success
-1 → error

Steps:

① create pipe
②

Both the process can read & write at the same time. they can also self read their written msg too

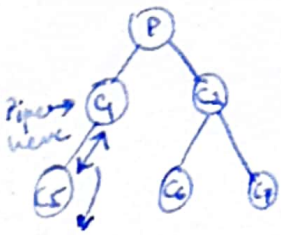
[Pipe] → alias → [file] → unnamed
→ data erased after reading.

differences from a usual file

- opened in blocking mode

For interprocess communication their must be parent child relationship.

communication b/w C1, P & C2



all child can access the read & write Pipes.

communication b/w both only.

```
int main(){
   int fd[2];                                   } step ①
   int r = pipe(fd);
                                  if (r==-1)
                                     perror("Pipe failed");
   int x = fork();                   return -1;
   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);
   }
```

include error checking

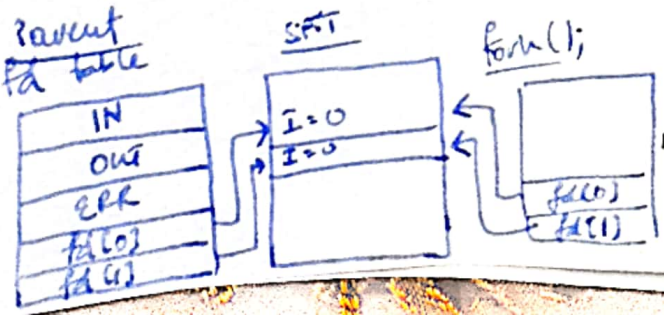
Two way communicating using pipe:

```
int main(){
   int fd[2];
   pipe(fd);
   x = fork();
   if(x==0){
      br = read(STDIN_FILENO, buff, 100);
      write(fd[0], buff, br);   // sent to pipe
      br=read(fd[1], buff, 100); sleep(2);
      write(STDOUT_FILENO, buff, br);
   else{
      int br = read(fd[1], buff, 100);
      write(STDIN_FILENO, buff, br);
      write(fd[0], buff, br);
```

need content switching here

block →

block →



Parent → child

user I/P

receive

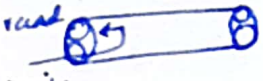
display

user I/P write → receive & display.

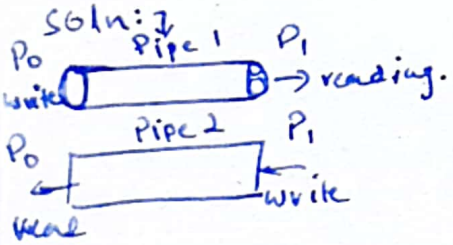
Two problems with the above program

① block

② in child code, its reading & writing continuously. writing & reading

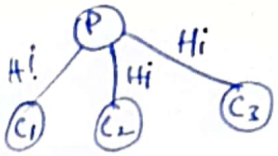


Solution:

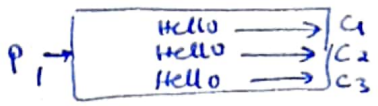



```
int main(){
  char buff[100];
  int fd[2];
  pipe(fd):
  int p2[2];
  pipe(p2);
```

Same as camera roll.




```
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)          // parent
    for(i=0; i<N; i++){
      write(fd[0], "Hello", 6);

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

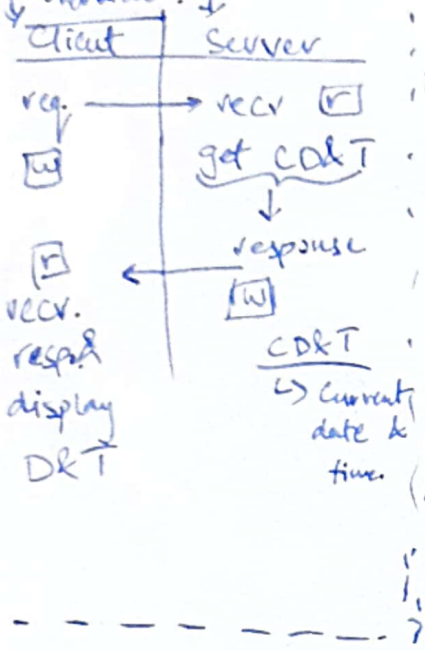


Fifos / Named pipes

SP Lt#05

Half duplex communication:

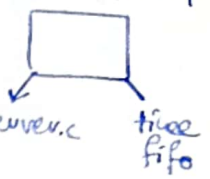
2 way communication



int mkfifo (const char *fifoname, int perm)

error → -1

success → 0



client.c

```
int main() {
int r = mkfifo("timefifo", S_IRWXG);
if (r == -1 && errno != EEXIST) {
perror("failed");
return -1; }
int fd = open("timefifo", O_RDWR);
//error check
int bw = write(fd, "Time", 6);
sleep(5);
}
```

server.c

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

request

Block until client sends the request.

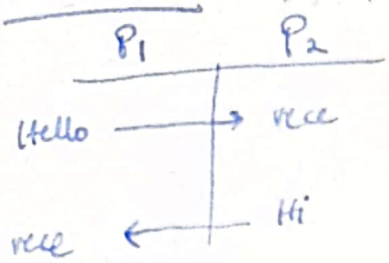
int br = read(fd, buff, 255);
printf("Current DkT = %s \n", buff);

sending to client

Both program are blocked after write line check the code in camera roll. for the one without time based thingy. (time bound.).

To tackle this problem we used fifo 1 for the first one & fifo 2 for the second one. (No need for sleep).

Model alm:



Prog.c:

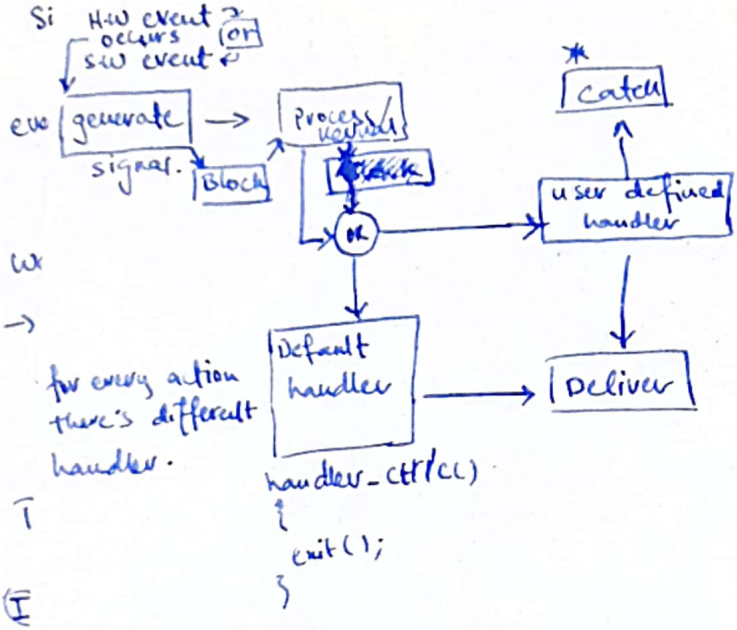
```
int main(){
  int r=mkfifo("chatfifo", S_IRWXU);
  //error checking.

  int fd=open("Chatfifo", O_RDWR);
we don't know the pl will enter the data first or the p2 so we
using select function.
  while(1){
  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){
     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);
     if (strcmp(buff, "Goodbye"))
        break;
  }
  }
  unlink ("chatfifo");
}
```

struct timeval myt;
myt.tv_sec =60;
myt.tv_usec = 0;

same thing here!

SP L#06

SIC CHAPTER 8 SIGNALS: <signal.h>



for every action there's different handler.

SIGNAL GENERATION:

user defined:

$ kill -9 101
(signo, PID)

what the above does is kill the big process 101

-9 is the kill signal.

suppose:
we did ps -all
1) init
:
101) t1.0
:

t1.0
main(){
while(1);
}

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

-1 → error
0 → success

```
int main(){
    kill(101, +9/SIGINT);
    kill(getppid(), SIGINT);
    kill(getpid(), SIGINT);
    int raise(int signo);
}
```

SIGKILL
SIGABRT
SIGTERM

default behaviour → process termination abnormal.

parent kill.

own kill.

deliver the signal to its own.

Signal mask.

>> $ kill -L
gives us the list of available signal
a total of 64 signals.
signo

$ stty ->

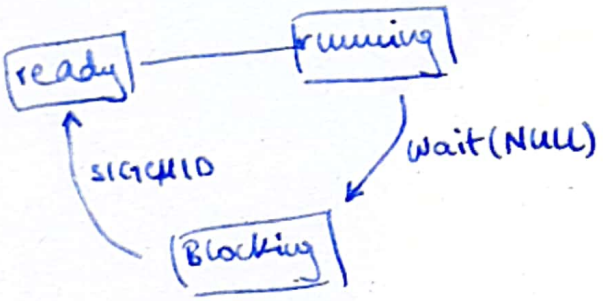
Block box is for when we want to generate a signal but we do not want it delivered.

To wait for specific signals

```
wait(NULL);
Pause( );
Sigwait( );
Sigsuspend( );
```

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

we terminated parent but child stuck in a forever loop if we hadn't added the kill line both would have been stuck forever. so child saved parent from forever loop.



wait(NULL) waits for the signal from the child

SP L#7

SIGNALS:

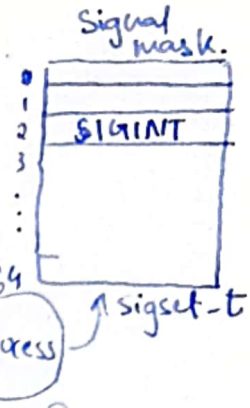
Signal ends from delivery to sent.

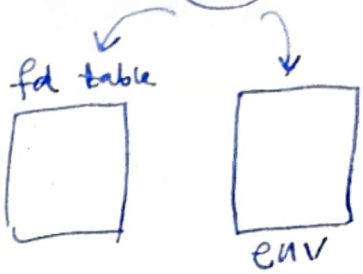
event → generate → Block → P → default / user def → deliver

to block

Want the signal/sent to block state.

→ In todays lecture
- → How to block
- → How to avoid block.





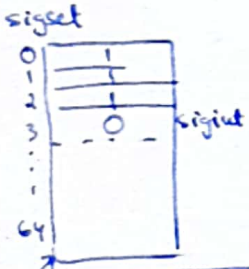
if the signal mask contains any signal the signal will be blocked.

To avoid blocking remove it from signal mask.

Task 1

To ways to do task 1

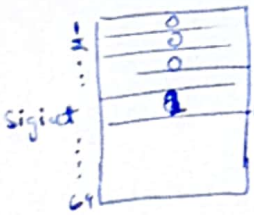
(I) Block all signals except SIGINT.



Steps
1. Fill SIGSET
2. remove sigint

one (1) here means active

(II) unblock all except sigint



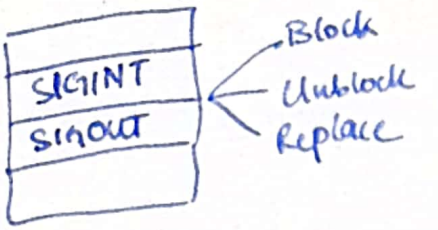
Steps
3. remove all
4. add sigint.

Step (3)

for both replace our made sigset & mask it of the signal mask of the process

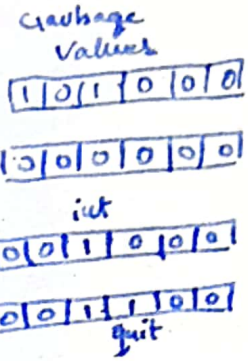
```
(1)  Sigfillset(sigset_t *myset);
     1 valid not(+1)
     add all the signals & return them.

(2)  sigemptyset(sigset_t *myset);
     empty's the signal set.

(3)  sigdelset(sigset_t *myset, int signo);

(4)  sigaddset(sigset_t *myset, int signo);

(5)  Sigprocmask(int how, sigset_t *myset
                 , sigset_t *old mask);
```

int HOW



SIG-BLOCK ①
SIG-UNBLOCK ②
SIG-SETMASK ③

```
int main(){
    sigset_t myset, oldmask;
    sigemptyset(&myset);
    sigaddset(&myset, SIGINT);
    sigaddset(&myset, SIGQUIT);
    sigprocmask(SIG-SETMASK, &myset, &oldmask)
    calculate-GPA();
    sigprocmask(SIG-SETMASK, &oldmask, NULL);
    // sigprocmask(SIG-SETMASK, &myset, NULL);
    for(;;);
```



this is for (II)
this is for (I)

we wanted to block some signals until the GPA was calculated but we want the old state back.

To generate signal use kill.
self generation of signal.

int kill(int pid, int signo);
int raise(int signo);
int alarm(int sec);

starts a timer, when the timer hits 0, the signal sigalarm is generated.

T = 5 4 3 2 1

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

← SIGALARM → default action of this is process termination

abnormal termination.

SP L#08

The timer is running & the program is processing too.

int alarm(int sec);

-1 → error

success → remaining time of last alarm.

0 → if no previous alarm

```
int main(){
    int alarm(60);
// will return zero.
    for(i=0; i<1000; i++){
        printf("Hello \n");
    }
    int r = alarm(5);
    printf("Remain time = %d \n", r);
```

CATCHING SIGNALS

CAMERA ROLL

21st Dec

SIG_DFL

```
main(){ → Block. sigint.
    struct sigaction newact;
    newact.sa_handler = INTR_handler;
    newact.sa_flags = 0;

    sigemptyset(&newact.sa_mask);   // to block any additional signals.
```

int count = 0; ⇐ Globle variable.

---

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

0 → success
-1 → error.

The signal we want to replace.

replace with our/user defined function action

to save old action or use NULL

Sigaction(SIGINT, &newact, &Oldset/NULL);

// coz waiting for signal.

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

// now making changes that if the user enters "ctrl c" 10 times the process should end — "add count ++ in the handler" code

// Busy waiting (for loop)

```
printf("process normally terminated");
exit(0);
}
```

→ if no for loop, the program execution will be so fast that it wont allow the user to enter 'ctrl c'.

→ CPU cycles are wasted due to for loop.

→ to not waste the cycles of the computer we use pause();

Latest Latest

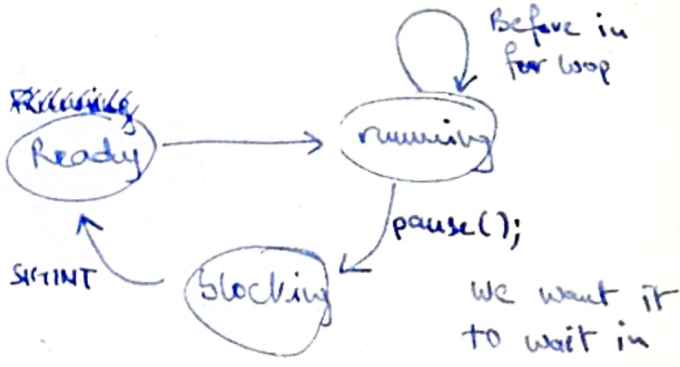
```
for(;count<10;){
    pause();
}
```

← counts the no of times pause is called.

→ Now instead of for loop use pause(); function.

→ check latest camera roll.

→ No matter what the signal the program will end.



SPL#09

→ No matter the signal type the program will end.

problem ① for loop

Sol ① use pause

problem ② if the signal arrives before pause() the parent will keep waiting. (Block)

Sol ② Block SIGINT in the starting. & unblock before pause → or any signal

problem ③ what if the signal arrives before pause() & the signal SIGINT is not catched. the pause() wont work.

Sol ③ Sigwait()
we want it to act it atomically.

if written before it the sigal _____

① for (;;);

② pause();

③ sigsuspend(); pause() + unblock

④ Sigwait();

Waiting for Signals: int flag=0 → Globle variable.

int suspend (const sigset_t *newmask);
↳ changes the mask of the program
↳ Blocks too! ↳ what we

```
main(){
    sigset_t newmask;
    sigfillset(&newmask);
    sigprocmask(SIG_BLOCK, &newmask, NULL);
    // ↑ to block all the signals
        // we can also use
        // the SIG_SETMASK too will give the same output.

    sigdelset(SIGINT, &newmask);          // New set that does not
                                          // include sigint.
    if (flag==0){
    sigsuspend(&newmask);
    }
}   // abnormal termination due to
    no signal handler.   // declare one from previous class.
```

```
void sighandler(int signo){
    flag=1                    // from cgpt know what this is for.
    return;

}
```

SP L#10

```
int main(){
    sigset_t  newmask, oldmask;
    sigprocmask(SIG_SETMASK, NULL, &oldmask);
    sigaddset(SIGINT, &oldmask);
    Sigprocmask(SIG_BLOCK, &oldmask, NULL);
    sigdelset(SIGINT, &oldmask)
    if(flag==0)
        sigsuspend(&oldmask);
} // handler.
```

include handler for this program

any signal other than sigchld will trigger the signal Process mask


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

It will wait for the signal that we put into the argument of.

```
int main(){  //handler
    sigset_t  mask();
    sigemptyset(&mask);
    sigaddset(SIGINT, &mask);
    sigaddset(SIGQUIT, &mask);
    Sigprocmask(SIG_BLOCK, &mask);
    int sig;
    sigwait(&mask, &sig);
```

→ jo signal ko bahir nikale.

```
if(sig==SIG_INT)
    printf("Sig int received");

else
    printf("Sig quit received");
```

SP L#11

CHAPTER 11

THREADS

Parallelism

- Task level
- Data level

parallelism can be achieved using multithreading & multiprocessing.

multithreading → parallelism

multiprocessing → parallelism
↳ overhead

overheading in multiprocessing is due to parent & child both have seperate image processes.
Communication b/w child & parent will also create overhead.

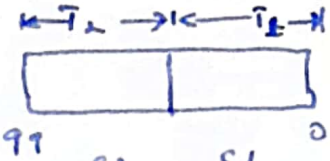
Now we will opt for multithreading.

t3 t2 t1 ← all accessing it

```
void *threadfun(void *x){
    printf("Thread ID = %d \n",
           pthread_self());
    pthread_exit(NULL);
    // exit(0)
```

pthread_exit(NULL) → exit the process.

```
int main(){

    pthread_t tid[3];
    for(i=0; i<3; i++){
        pthread_create(&tid[i], NULL,
                       threadfunc, NULL);

    // exit(0);                    // process termination
    //pthread_exit(NULL)           // master dies child runs.

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



DATA level parallelism.

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

convert into ↓ void pointer.

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

```
for (i=0; i<2; i++){
    pthread_join(tid[i], NULL);
}
printf("Sum of all = %d \n", sum);
exit(0);
}
```

```
Void *sumArray(void * arg){
  int index = *(int*)arg;
  int start = index * 100 / num_thread;
  int end = start + 49;
```

OR 2

```
        OR  start + ((100/num_threads) - 1);
```

	start	end
t1	0	49
t2	50	99

```
  int local_sum;
  for(int i=start; i< end; i++){
      local sum += Array
      Sum = Sum + local sum;
      pthread_exit(NULL);
```

For task level parallelism look in camera roll.