Systems Programming

Book: Unix Systems Programming: Communication Concurrency & Threads (Author K. A. Robins & Steven Robins)

Contents:

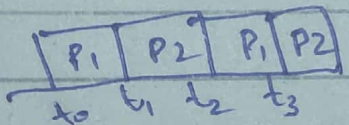
1. ——— self study → Reading Assignment
2. Program, Process, threads
3. Processes
4. Unix I/O (standard I/O device) (Regular files)
5. Files & Directories
6. Special files (fifos / pipes / sockets)
8. Signals
9. Times & Timers
13. Threads / 15. IPC using shared Memory.

Midterm → 5.1

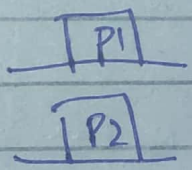
represent memory Address of func

```
main() {
    scanf();  ——> sys call
```

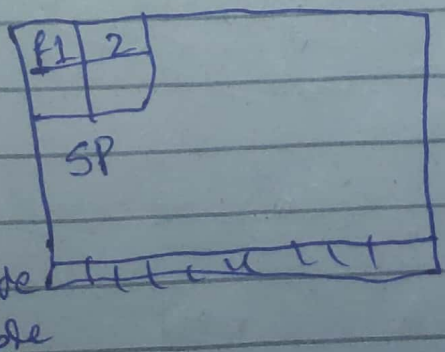
Concurrency



Parallelism



HDD



Signal:- notification of event.

M. Term

CLO's :-

CLO 1: error handling / file handling / signal handling.

F. Term

CLO 2: parallel processing (multiprocessing / multithreading)

F. term CLO 3: Interprocess Communication for real world.

Grading Criteria :-

Mid : 30 %
final : 40 %
Sessional : 30 %
Project / Attendance / Quizzes (3-5) / Assignment (3)

Oct 5, 2023

Programs, Process & Threads

Program → Process

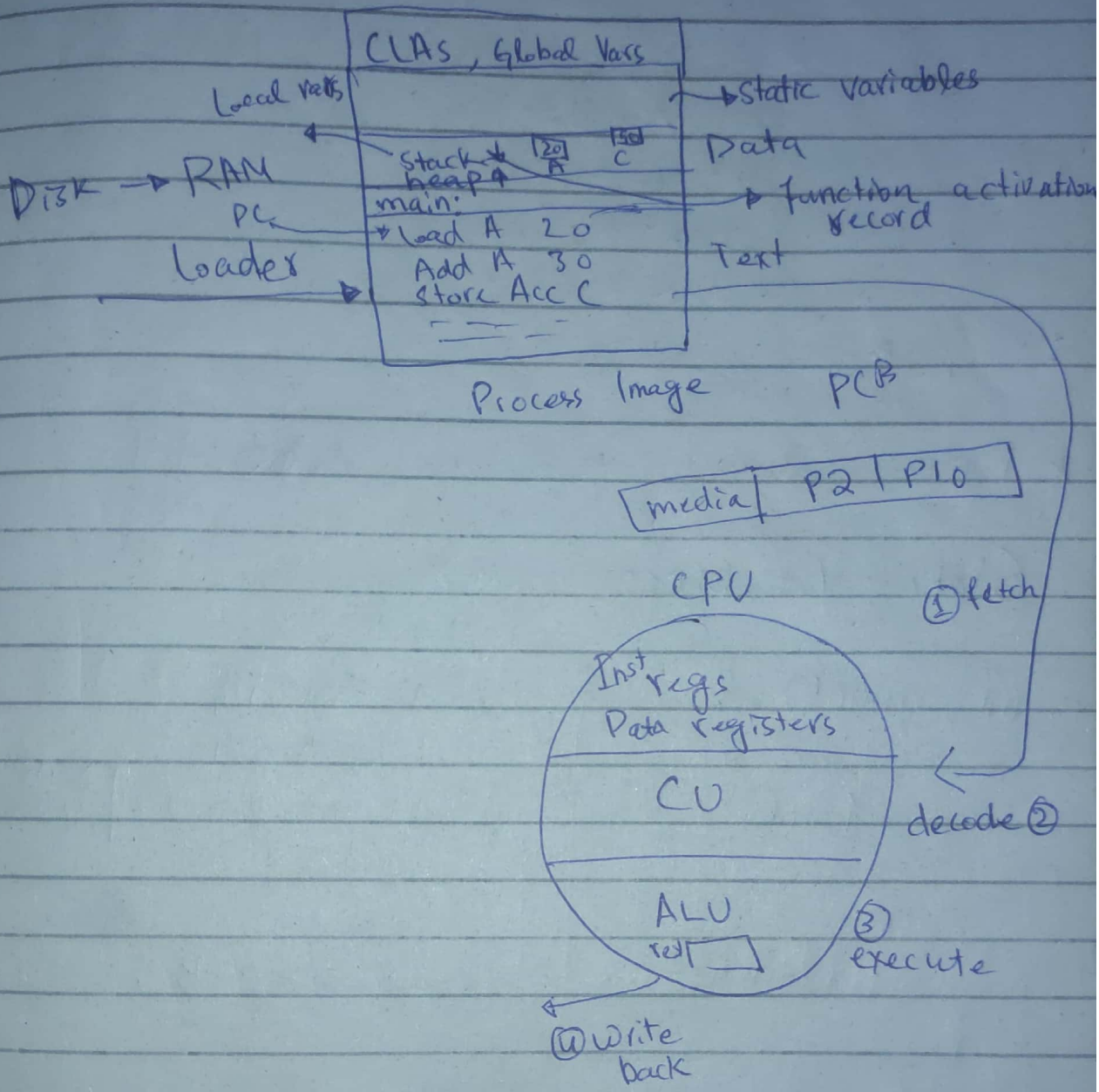
```
#include < ___ >
#define COUNT 30
```
→ pre-compiler, preprocessor directives
→ do not compile,

```
int main(int argc, char * argv[]) {
    int A = 20;
    int C = A + COUNT
    printf("%d \n", C);
    return (0);
}
```
starting point
→ exit status returned to parent

gcc p1.c -o p1.o

p1.c → disk

p1.o → disk



Variables → Scope, Usage, Lifetime

```
int main ( ) {
    int A = 20;  int C = A + COUNT;
    fun(A);  ⟶ stack will be divided
```

Function stack will be destroyed when it returns.

```
fun(int A){
    int X = 30;
    int y = 40;
    X++; Y++;
    cout<< X<<Y;
}
```

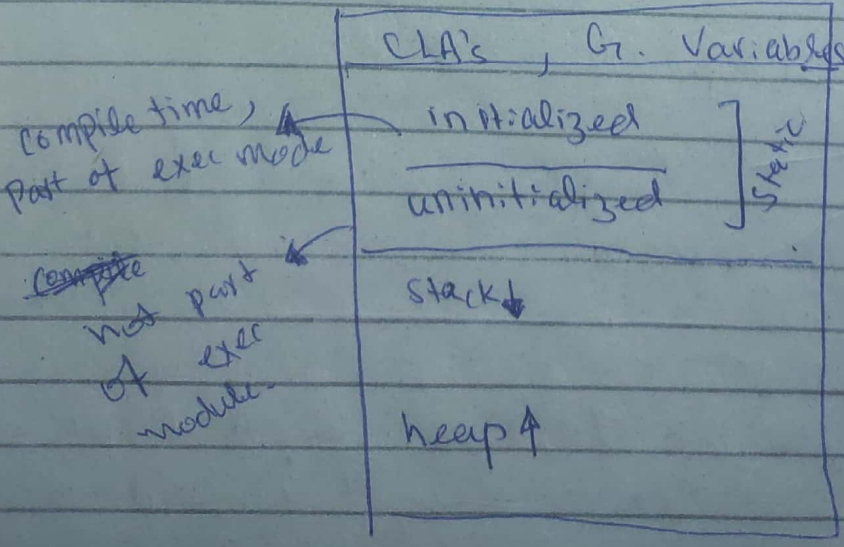
* Dynamic Memory must be deallocated

Static Variables:

```
int main() {
    func();
    func();

    return 0;
}
```

```
void fun(){
    int A = 10;
    static int B = 20;   // dec + init — execute only once
    A++;
    B++;
    cout << A << B;
}
```



PCB → A linked list.

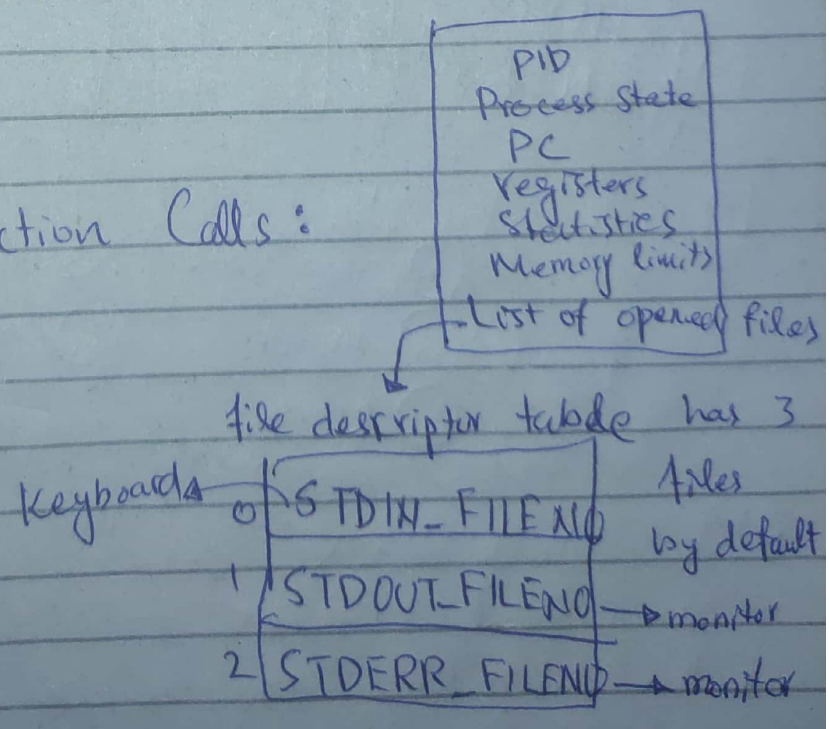
Locals vars are destroyed while static aren't.

```
void fun() {
    static int c;
    c = 10;
    c++;
    cout << c;
```

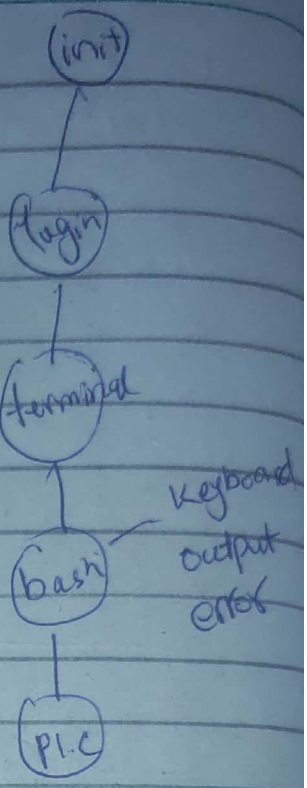
`Static int A[100];`	`static int A[100] = {1, 2, 3};`
400 bytes	400 + 100 * 4 + 7 = 807

Oct 6, 2023

Library function Calls:



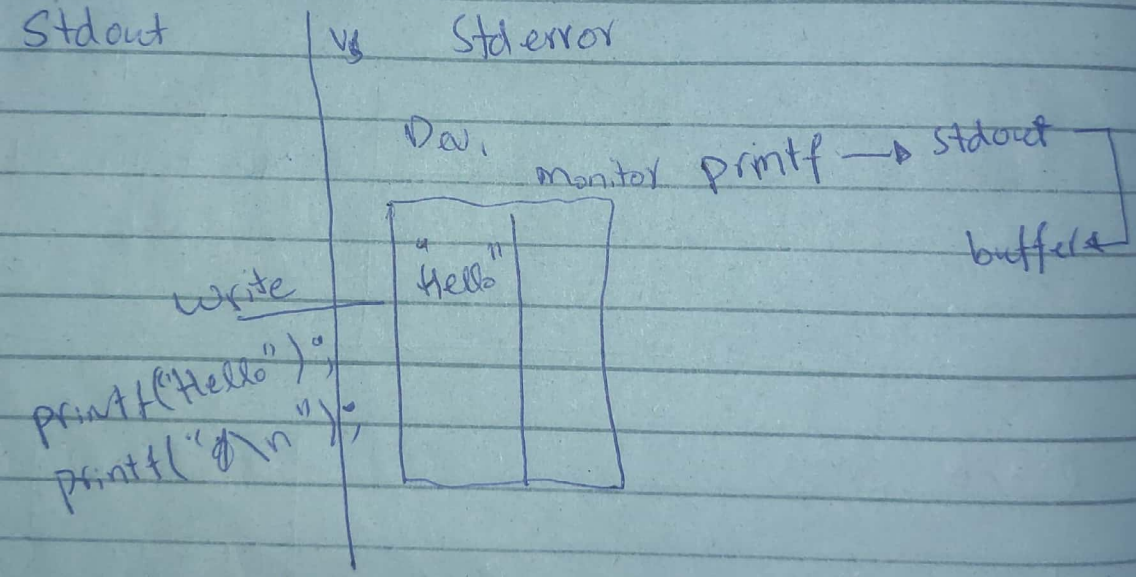
→ Child process will have same PC, registers and list of opened files.



Stdin, Stdout, Stderr are opened by default

* Why two entries for monitor.

Stdout vs Stderror



① new line ③ fflush
② input ④ buffer full
⑤ process terminate

→ We deal with device buffer in order to save cycles.

→ In errors case, we want to let user know immediately. So stderror has no buffer.

→ buffer is memory in RAM.

```
main() {
    printf("Hello1");  → stdout
    printf("Hello2");  → stderr
}      perror
```

Output: Hello2 success
Hello1

filename string

```
fprintf(stdout, "Hello1");
fprintf(stderr, "Hello2");
```

Library function Calls :-

① Detect error
② Error Reason

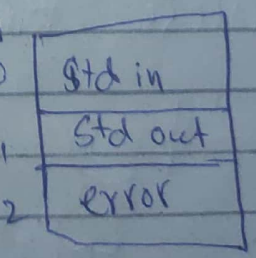
return_type name (args);

int

int close(int fd);

-1 : error
0 : success

error



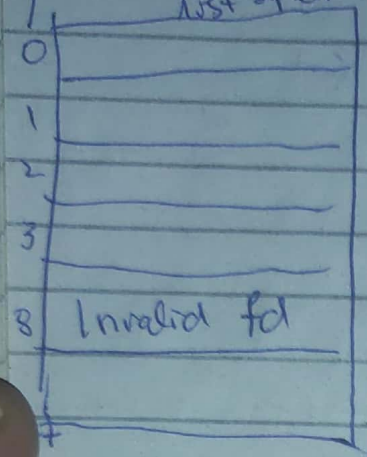
② set errno

-1 ① Return failure

```
int main()
{
    int ret = close(4);
    if (ret == -1) { // error
        perror("error"); printf("Error = %d \n", errno);
        fprintf(stderr, "Failed to close : %s", strerror(errno));
        return -1
    }
```

errno

errno.h
List of errors



$ man

char * strerror(int errno);

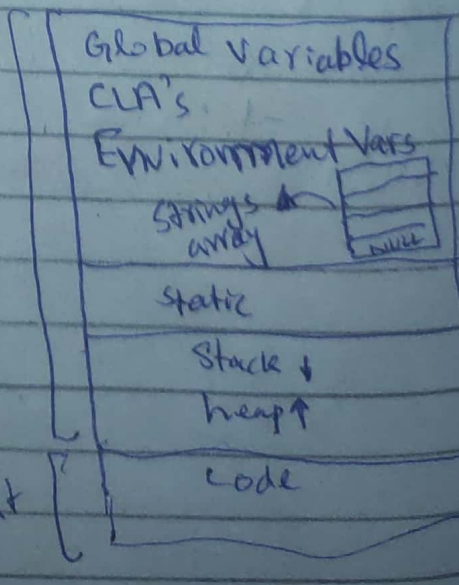
close(1) → will close stdout and printf will fail.

Date 9 Oct, 2023

Process Environment

— $ env

↳ List Environment Vars



ls → pwd

↓

PATH

HOME = /home/student

Var name — Value

echo $HOME

Environment vars are also inherited by child process.

extern char ** environ

↳ Let compiler know that we are using it from another source file.

→ it will not be declared as a new var.

```
main() {
    for( int i = 0; environ[i] != NULL; i++)
        printf("%s\n", environ[i]);
}
```

```
char * getenv(const char * varname);
```

```
main(){
    char * valofpath;
    valofpath = getenv("PATH");
    printf("PATH = %s\n", valofpath);
```

Normal Process Termination:

Exit handlers

stty

1. Explicit exit → different (same)
2. Implicit exit (return statement of main)
3. _exit() / _Exit() same

---

int atexit(void *fun(void))

0: success
non-zero: failure

```
int main(){
    atexit(fun1); atexit(fun2);
① printf("In main \n");
    exit(0);
}

_exit(0); // does not call
          // exit handlers
```

```
void fun1(void){
③ printf("Process
Terminating Normally
\n");
}

void fun2(void){
② printf("All sts
executed\n");
}
```

```
int main() {
    atexit(fun1);    atexit(fun2);
    int ret = close(4);
    if (ret == -1) {
        fprintf(stderr, "Failed close sc");
        exit(-1);
    }
    return 0;
}
```

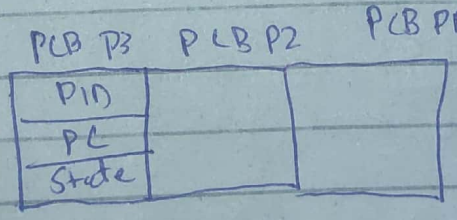
→ String Tokenization Skipped.

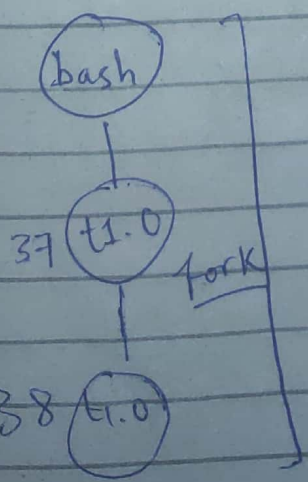
→ Chap 2 Quiz in next class.

Chapter (3) SP

16/10/2023

Processes:





ps -al
↳ list of processes

pstree

init

for making a group,
sudo groupadd SectionC
sudo usermod -aG SectionC Hamza
Append 3 14

1st Column
ID 2nd Column state
D → uninterruptible sleep
S → interruptible sleep, wait
R
Z → Zombie
T → Signals

---

int getpid(); root root
int getppid(); 0 0
int getuid();
int getgid();

Effective ID's :-

user: int geteuid();
group: int getegid();

---

logged in as Hamza

```
sudo groupadd myfriends
```

Let's test it:

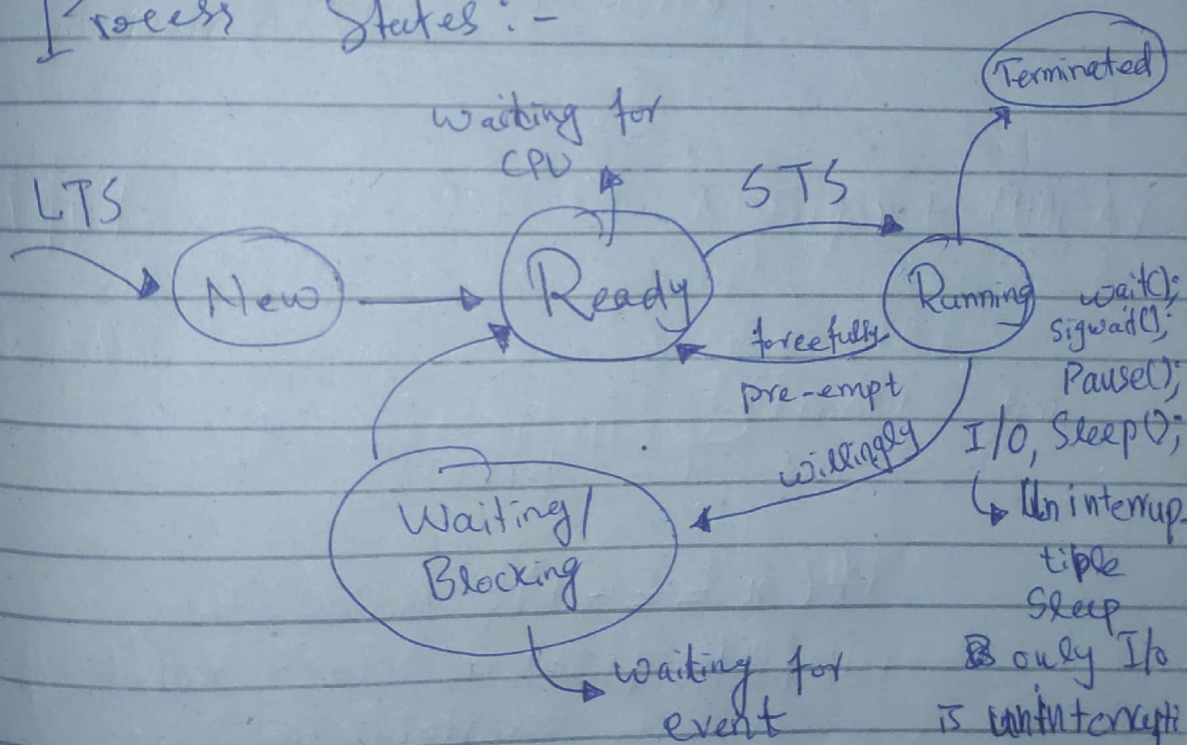
```
main() {
    printf("My ID = %d, My GID = %d \n",
            getuid(), getgid());
}
```

./t10 → My ID = 14, GID = 4

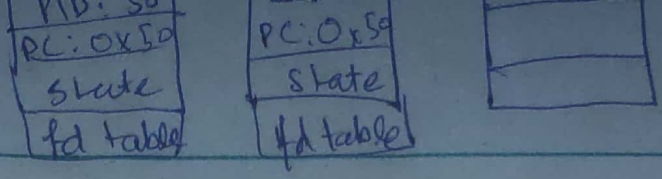
sudo ./t10 → My ID = 0, GID = 0

Effective ID's
↳ Temporarily Changed

Process States :-



When child process terminates, parent recieve SIGCHLD signal.

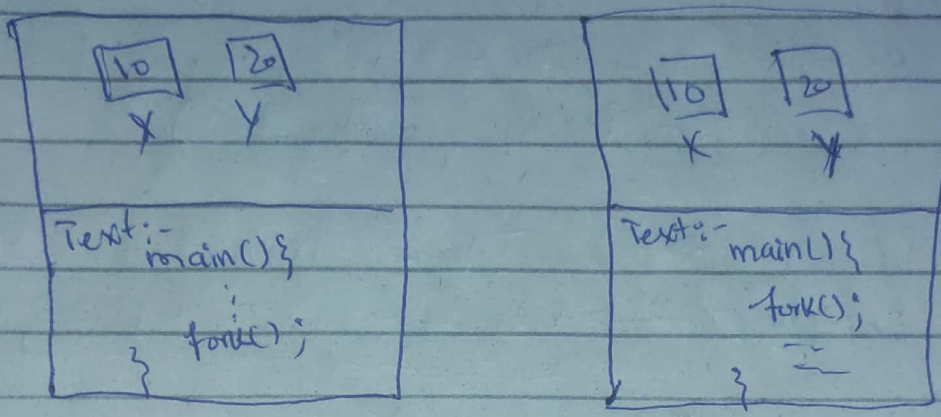


① no resources
② limit exceed

Multiprocessing:

int fork();

-1 : error
0 :
>0 :



Parent 37

Child 38

→ List of all errors [HW]

errno: EAGAIN → print int
↳ corresponding String

EAGAIN → represent the errno.

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

    if(x == -1){ // failure
        fprintf(stderr, "fork Failed %s\n",
                strerror(errno));
        exit(-1);
    }
printf("\n %d, my x = %d\n",
        getpid(), x);
return 0;
}
```

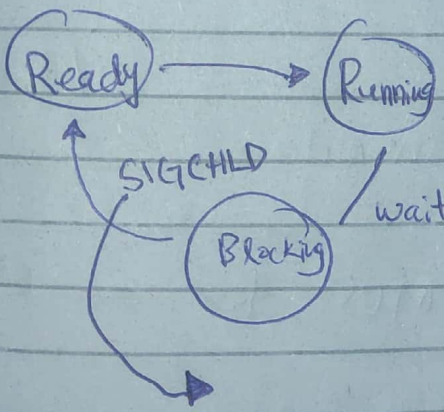
Date 19 Oct 2023

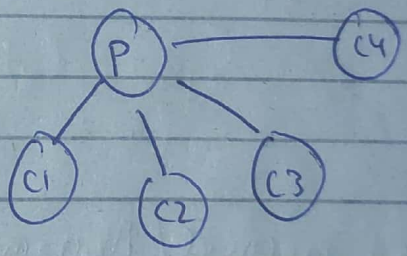
Processes:-

int fork()
error : -1 , errno
Success: 0
>0 : parent
child ID

int wait (int *status)
↳ child reasons of termination.

error : -1 , errno
1. ECHLD — no child exist.
2. EINTR
3. EINVALID — only in waitpid
Success: >0
child ID





int waitpid (int pid, int *s, int option)

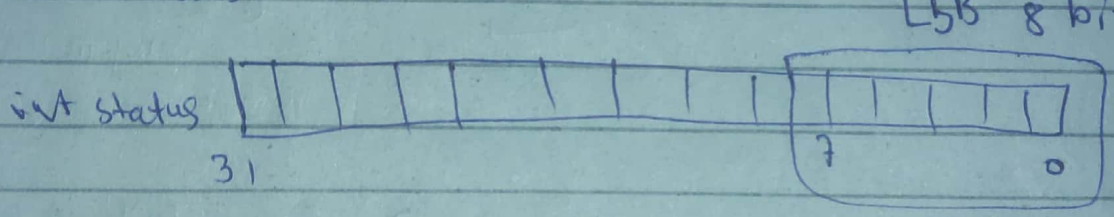
Process Termination
- exit
- return
- exit
- Exit

Normal → Exit status returned to parent in wait argument.

ABNORMAL → terminated completely
→ Suspend → stopping sig

```
int main(){
    int x = fork();
  if(x==0){
        printf("I'm child");
        exit(4); // return 4
  }
    int status;
    int ret = wait(&status);
```

int status



For normal/Abnormal termination,
WIFEXITED
↳ WEXITSTATUS for ex value

Now for abnormal termination,
WIFSIGNALED
↳ for find termating signal
WTERMSIGNAL.

For last scenario, WIFSTOPPED
↳ WSTOPSIG

```
if (WIFEXITED(status))
    printf("Child exited normally with
            status = %d\n", WEXITSTATUS(status));

if (WIFSIGNALED(status))
    printf("Terminating Signal = %d\n",
            WTERMSIG(status));

if (WIFSTOPPED(status))
    printf("Stoping Signal = %d\n", WSTOPSIG
            (status));
```

```
int main() {

    int N = 20;
    int cid [N];

for (int i=0; i<N; i++) {
    cid[i] = fork();
if(cid[i] == 0) {
    printf("I'm child");
    exit(4);

}
elseif (cid[i] == -1) {
    printf(stderr, "Fork failed: %s\n"
            strerror(errno));

return -1;
```

int status (8)

int ret = waitpid(cid[3], &status, 1);

-exit(4); → does not empty stdout buffer.

printf("Before for loop\n"); will be printed 6 times.

Date 26/10/2023

Execute System Call:

int execl, int execlp (p → path)
int execv, int execvp
int execle, int execve

```
int main() {
    execlp("ls", "ls", NULL);
    printf("Hello");
}
```

→ will not print

error: -1

success: X → will not return.

errno: E2BIG
EINVALID
EACCESS
}

Contains built in commonds path — Searches in PATH

Complete Path → from which command.

```
int main
    exec("/bin/ls",
```

PATH = /bin : /usr/bin : /lib : /usr/lib ...
① ② ③ ④ ⑤

which t1.0 ⟶ Command not found

---

int execl(const char *path, const char *filename,
    const char *arg0, const char *arg1, ----
    --- const char *argN, NULL);

int execlp(const char *filename, const char *
    arg0, ------ const char *argN, NULL
    );

./t1.0 1 2

```
int main(int argc, char *argv[]){
    if (argc < 3) {
        printf("Invalid Args");
        return -1;
    }
    int sum;
```

```
Sum = atoi(argv[1]) + atoi(argv[2]);

printf("I'm %s, Sum = %d \n", argv[0],
                Sum);
    return Sum;
}
```

t2.c

```
int main(int argc, char *argv[]) {

    int x = fork();
    if (x == 0) // child
        execl("./t1.0", "t1.0", "1", "2",
                argv[1] argv[2]);
    else
    int y = wait(&S);
    printf("Child process returned Sum
            = %d\n", WEXITSTATUS(S));
}
```

./t2.0 1 2

```
int execv(const char* file(Path), const
        char *argv[])

    int x = fork();
    if(x == 0) // child
{
    execvp("./t1.0", &argv[1]);
```

address of 1st element

t2.c

```
int main( int argc, char *argv[])
    int x = fork()
```

t2.c

```
int main ( int argc, char *argv[]){
    char *newenv[] = { "VAR1 = Batch23",
                       "VAR2 = Sect C",
                       NULL};
    int x = fork();
    if (x == 0){
        // print old env
        execve("./t1.0", &argv[1], newenv);
    }
    wait(NULL);
}
```

Date 30th Oct, 2023

Q1 Why not make a function which creates a child and replace the image itself?

Q2 why not use ~~error~~ try catch for error handling?

Background ... Skip

Date 30th Oct, 2023 SP

File Handling :-

I/O Devices are treated as files.

cp src dst
cp f1.txt f2.txt
Copies contents of f1 into f2

1. open source file for (reading.)
2. Open dst file for (writing)./created dst file
3. Read from src
4. Write to dst
5. Close all

open flags

```
int open(const char *path, int oflags,...);
```

optional arg

int for permission

error :- -1
Success :> 0

```
int main(){
    int perm = S_IRWXU|S_IRGRP|S_IWGRP|S_IROTH;
    int fd1 = open("f1.txt", O_RDONLY);
    // error checking is must
    if(fd1 == -1){ perror("failed to open
        // for reading\n"); return -1;}
    // int fd2 = open("f2.txt", O_WRONLY
```

starts se writing, erase full content.

End mei writing start karo.

Mutually Exclusive → at a time one can be used.

oflags
- O_RDONLY
- O_WRONLY
- O_RDWR
- O_CREAT → If filed does not exist.
- O_EXCL → error. If already exist.

permissions.
- S_IRUSR → for reading
- S_IWUSR
- S_IXUSR
- S_IRWXU

Bitwise OR if f2 does not exist

```
$ O_WRONLY | O_CREAT, S_IRWXU | S_IRGRP |
                S_IROTH);
```

O_CREAT must have third argument for permission.

// cp will overwrite f2.

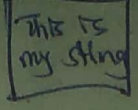
```
int fd2 = open("f2.txt", O_WRONLY | O_CREAT |
                O_TRUNC
// error checking.
```

// O_EXCL, O_TRUNC and O_APPEND are also mutually exclusive.

```
// Read from src.
   char buff[100];

// Here comes read() to the
// rescue.

// int read(int fd, char *buff,
        int size of buffer);
```



bytes returned

```
int br = read(fd1, buff, 100);
```

read // error checking.

success: 0

0 > 0

if we used read again, then it will return 0.

↳ End of file / Empty file.

// Now write

// int write(int fd, const char * buff
int length of buff).

strlen(buff) OR br

```
int bw = write(fd2, buff, br);
```

If 100 is ...

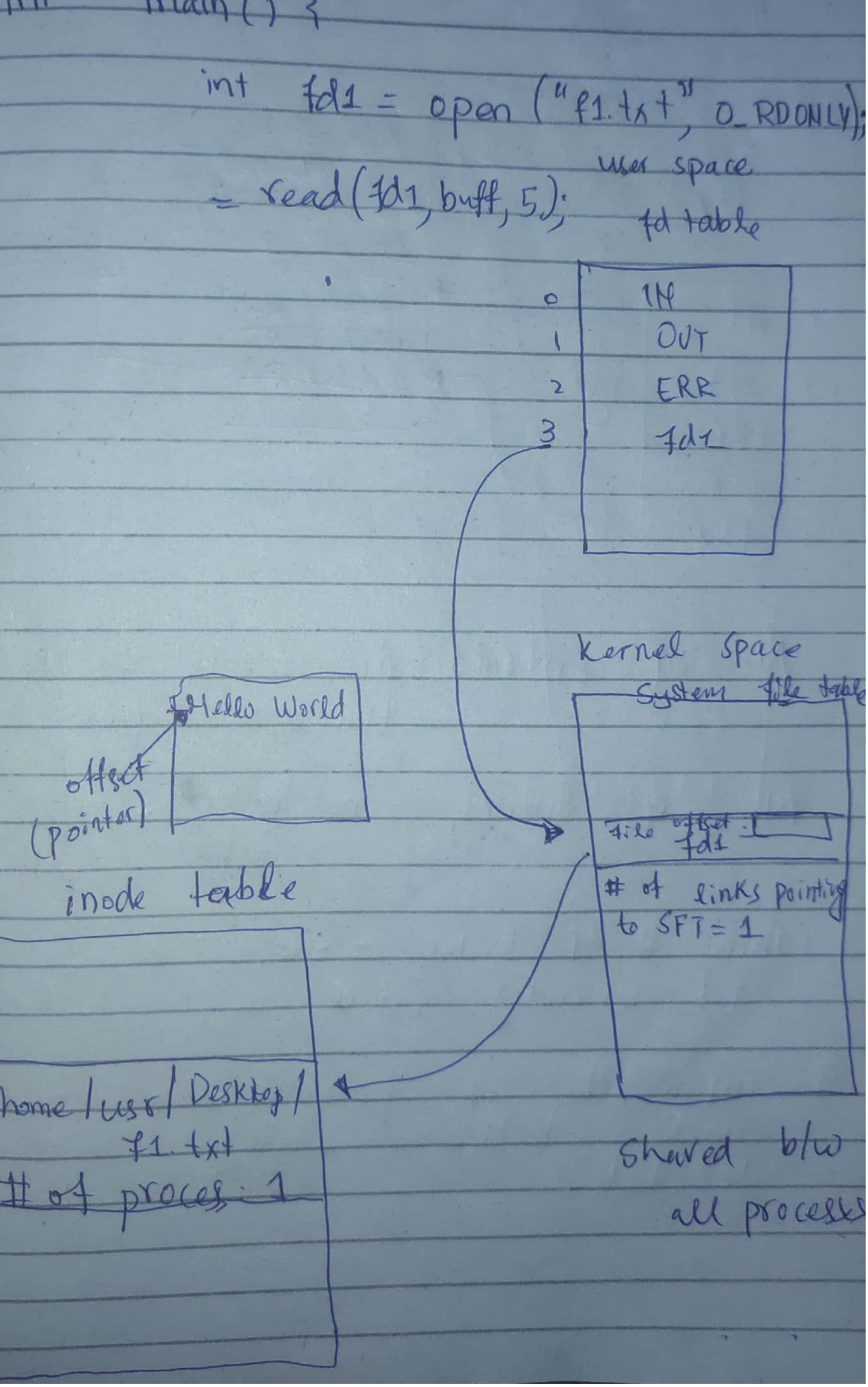
then it will ... garbage values.

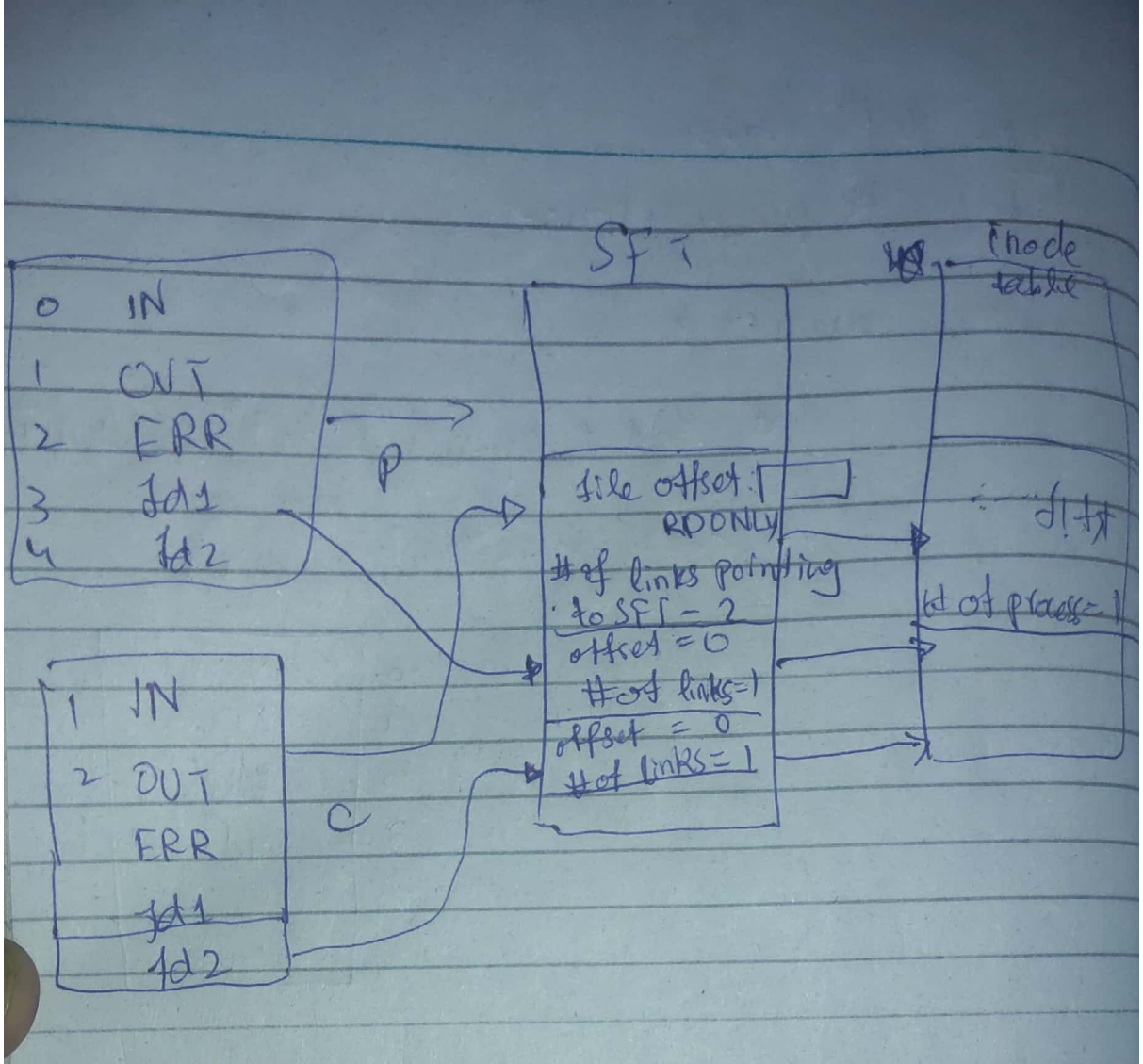
```
close(fd1);
close(fd2);
}
```

Date 6/11/2023

File Representation:-

```
int main() {
    int fd1 = open("f1.txt", O_RDONLY);
    = read(fd1, buff, 5);
```





No. of Entries in SFT = No. of times open is called

No. of Entries in inode table = No. of actual files

* When # of links == 0, then SFT entry will be deleted.

* When # of process == 0 then inode table entry will be deleted.

```
int main() {
        char buff[10];          → offset is shared
        int fd1 = open("f1.txt", O_RDONLY);
        int x = fork();

If (x == 0) {
        int fd2 = open("f2.txt", O_RDONLY);
        int br = read(fd2, buff, 5);
        printf(buff);

}
else {
        int fd2 = open( "      " );
        int br = read( "      " );
        printf( "      " );
}
```

Filters & Redirection :-

ls > f1.txt

① Cat → No Redirection

(will read from STDIN, write to STDOUT by default)

IN
f1.txt
STDERR

② Cat f1.txt → Input Redirection

IN
OUT
ERR

read from f1.txt, write to STDOUT

STDIN
f1.txt
ERR

(3) cat > f1.txt

↳ read from STDIN, write to f1.txt

↳ output redirection

(4) cat f1.txt > f2.txt

↳ IN/OUT redirection

read from f1.txt, write to f2.txt.

---

int dup2(int fd1, int fd2);

↳ for changing fd entries

IN
OUT
ERR
f1.txt

fd

dup2(fd1, STDIN_FILENO);

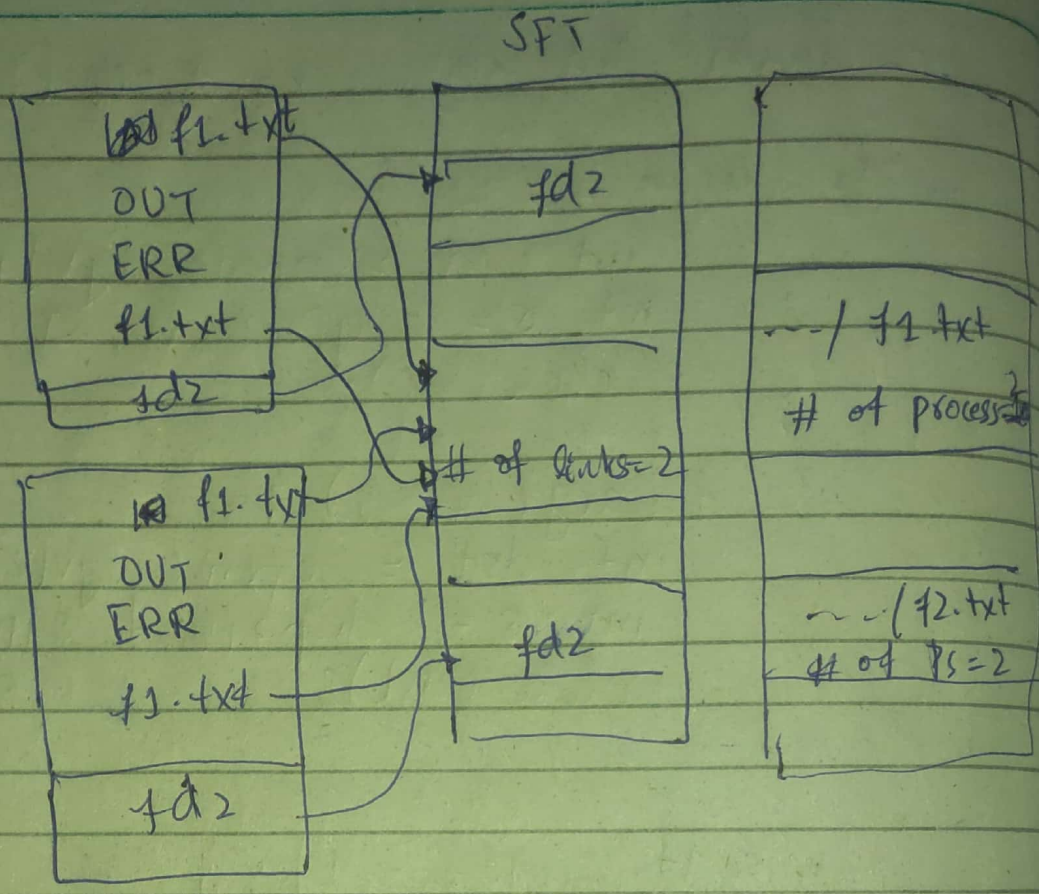
close(fd1);

fd

[0]	f1.txt
	OUT
	ERR
fd	fd1 txt

```
int main( int argc, char * argv[]){
    if (argc == 2){
        int src = open(argv[1], O_RDONLY);
        int r = dup2 (src, STDIN_FILENO);
        close(src);
    }
    elseif (argc == 3){
        int dst = open(argv[2], O_WRONLY);
        int r = dup2(dst, STDOUT_FILENO);
        close (dst);
    }

    else if (argc == 4){
        int src = open(argv[1], O_RDONLY);
        int dst = open(argv[3], 1 );
        dup2 (src, STDIN, FILENO);
        dup2(dst, STDOUT, 1 );
        close ( src);
        close (dst);
    }
}
read ( STDIN_FILENO
write
```




```
int main( int argc, char *argv[])
{
    int fd = open("f1.txt", O_RDONLY
    dup2(fd, STDIN_FILENO);
    int x = fork();
    int fd2 = open("f2.txt", O_RDON
    if (x == 0)
        close(fd);
}
```