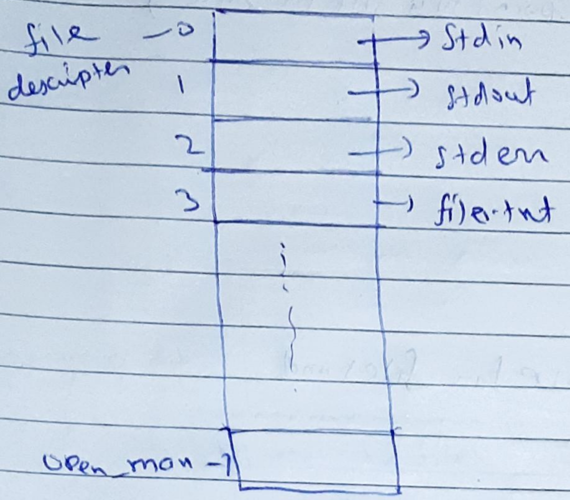
I/O Redirection

PP FDT :- Per process file descripter table
It keeps tracks all the files opened by a process. It is maintained by the process.

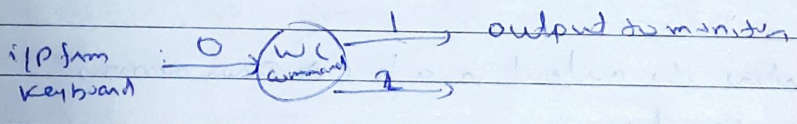


Open_max ~~is the no.~~ defines the maxi. no. of files a process can open. To know this value

```
$ getconf   OPEN_MAX
  o/p - 1024
```

By default 3 files opened by a process are stdin (keyboard), stdout (monitor), stderr (it also uses monitor to display error).

In a Linux or unix, everything is file. So, keyboard and monitor are also treated as file.



```
$ wc
```

=> ~~we~~ By default, wc takes i/p from keyboard using file descripter 0. But, ~~if~~ ~~we~~ to give the i/p to wc from a file, file descripter 0 ~~to~~ should be redirected to ~~a~~ the i/p file.

```
$ wc < /etc/passwd
```

o/p redirection :- here in place of showing the o/p ~~b~~ on monitor, it is redirected to a file. So, file descriptor ~~2 wil~~ will point to a file in place of monitor.

```
$ wc > abc
```

```
$ cat < file1 > file2
```

here, cat command takes i/p from file1 and ~~ep~~ print the o/p in file2.

```
$ cat >> file2      (for appending)
```

=> error redirection

```
$$ find / -name root
```

It will search all the files in the system which has 'root' as name. However, o/p will contain the error messages also such as 'permission denied'.
So, to store the output in a file & error messages in other file then

```
$ find / -name root 2> error.txt 1> output.txt
```

Here, 2 is file descriptor of stderr.

If we donot want to neither see the error messages nor store in a file then we can redirect the error messages to /dev/null. ~~of~~ this file is used to dump the data forever. It is like a blackhole.

```
$ find / -name 2>/dev/null 1~~0~~ root
```

for O/P ~~de~~redirection, > or 1> can be used
I/P " < or 0< " "
error - 2> is used always.

```
$ cat 0< file1 1> file2 2> file3
```

So, PPFDT will change from

0	→ stdin
1	→ stdout
2	→ stderr

to

0	→ file1
1	→ file2
2	→ file3

If file1 does not exists then error will be redirected to stderr not file3 bcoz initially shell will perform 0<file1 but it is failed then it will not set ~~2~~ and 1 and 2 to file2 and file3. So, 2 will remain attached to stderr. So, error will not be redirected to file3. It will be shown in the terminal itself.

```
$ cat 1>file4 2>file5 0< file6
```

Here, 1>file4 will be parsed 1st, so, ~~stdout~~ 1 and 2 will be set to file4 and file5 before encountering the error as 0< file6. So, now error will be ~~the~~ redirected to file5.