LAB-05- I/O REDIRECTION

- Learning Outcomes
  - I/O Redirection on Terminal using & operator.
  - File Handling in C.
  - I/O Redirection using dup, dup2 system call.

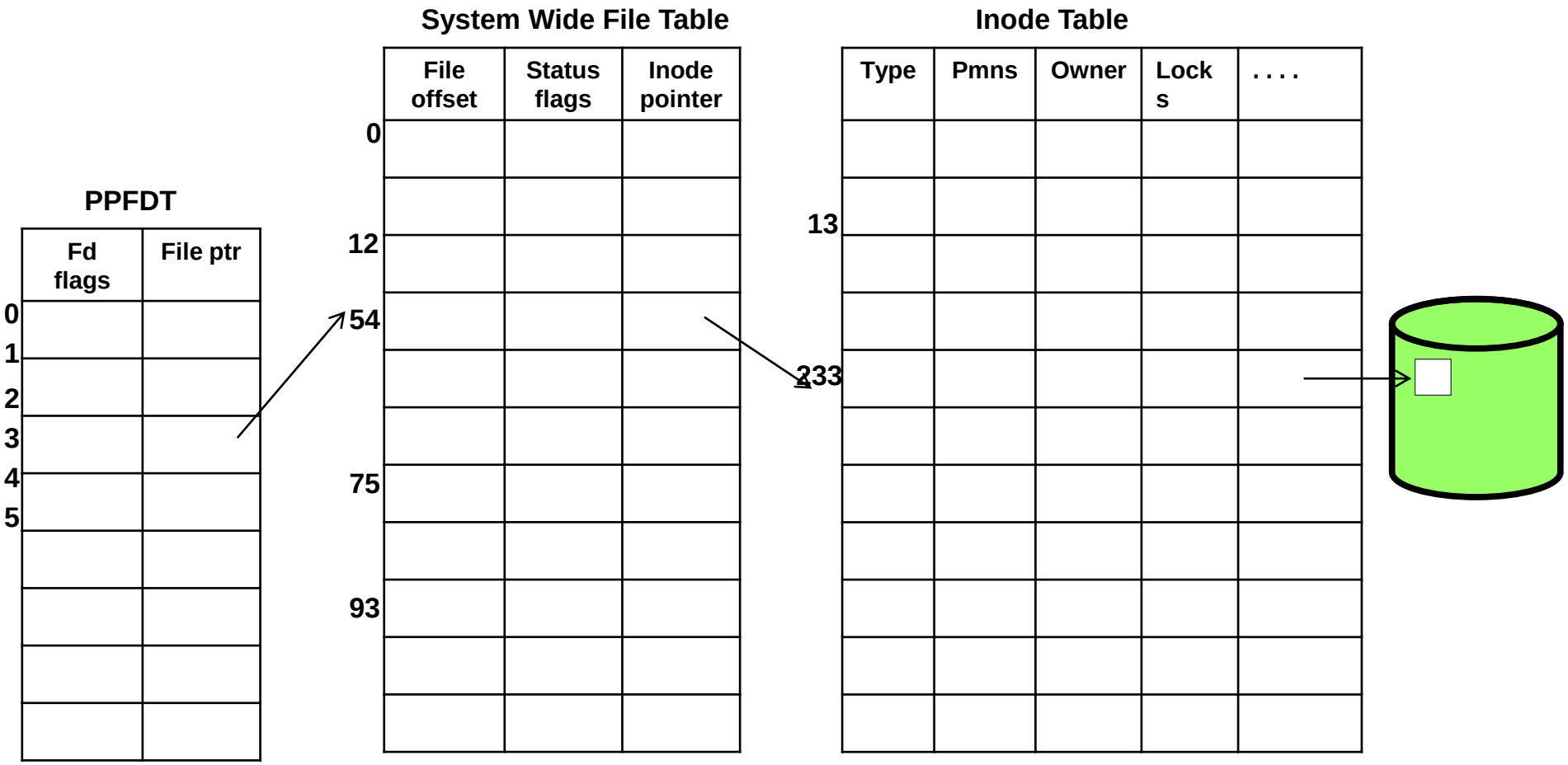
Connection of an Opened File

File Management in Linux

Following are the four key system calls for performing file I/O (programming languages and software packages typically employ these calls indirectly via I/O libraries):

- **fd = open(pathname, flags, mode)** opens the file identified by pathname, returning a file descriptor used to refer to the open file in subsequent calls. If the file doesn't exist, open() may create it, depending on the settings of the flags bit. The flags argument also specifies whether the file is to be opened for reading, writing, or both. The mode argument specifies the permissions to be placed on the file if it is created by this call. If the open() call is not being used to create a file, this argument is ignored and can be omitted
- **numread = read(fd, buffer, count)** reads at most count bytes from the open file referred to by fd and stores them in buffer. The read() call returns the number of bytes actually read. On eof, read() returns 0.
- **numwritten = write(fd, buffer, count)** writes up to count bytes from buffer to the open file referred to by fd. The write() call returns the number of bytes actually written, which may be less than count
- **status = close(fd)** is called after all I/O has been completed, in order to release the file descriptor fd and its associated kernel resources

File Descriptor to File Contents



File Descriptor	Purpose	POSIX Name	stdio Stream
0	Standard input	STDIN_FILENO	stdin
1	Standard output	STDOUT_FILENO	stdout
2	Standard error	STDERR_FILENO	stderr

File Descriptor to File Contents

- Each process in UNIX has an associated PPFDT, whose size is equal to the number of files that a process can open simultaneously
- File descriptor is an integer returned by **open()** system call, and is used as an index in the PPFDT
- File descriptor is used in **read()**, **write()** and **close()** system call
- Kernel uses this descriptor to index the PPFDT, which contain a pointer to another table called **System Wide File Table**
- In the **System Wide File Table**, other than some information there is another pointer to a table called **Inode Table**
- The inode table contains a unique Inode to every unique file on disk

Standard Descriptors in UNIX / Linux

- Three files are automatically opened for every process to read its input from and to send its output and error messages to.
- These files are called standard files:
  - 0 – **Standard Input (stdin).** Default input to a program is from the user terminal (keyboard), if no file name is given.
  - 1 – **Standard Output (stdout).** A simple program's output normally goes to the user terminal (monitor), if no file name is given.
  - 2 – **Standard Error (stderr).** Default output of error messages from a program normally goes to the user terminal, if no file name is given.
- These numbers are called File Descriptors – System calls use them to refer to files.

Examples - File Handling

```
int main(){
  char buff[256];
  read(0, buff, 255);
  write(1, buff, 255);
  return 0;
}
```

```
int main(){
  char buff[256];
  while(1){
     int n = read(0, buff, 255);
     write(1, buff, n);
  }
  return 0;
}
```

Example - File Handling

```
int main(){
  char buff[2000];
  int fd = open ("/etc/passwd", O_RDONLY);
  int n;
  for(;;){
     n = read(fd, buff, 1000);
     if (n <= 0){
          close(fd);
          exit(-n);
     }
     write(1, buff, n);
  }
  return 0;
}
```

Example - File Handling

```
int main(){
  int n;
  char buff[1024];
  int
  fd=open("file.txt",O_CREAT|O_TRUNC|O_RDWR,0666);
  for(;;){
     n = read(0, buff, 1023);
     if (n <= 0){
          printf("Error in reading kb.\n");
          exit(-n);
     }
     write(fd, buff, n);
  }
  close(fd);
  return 0;
}
```

I/O Redirection

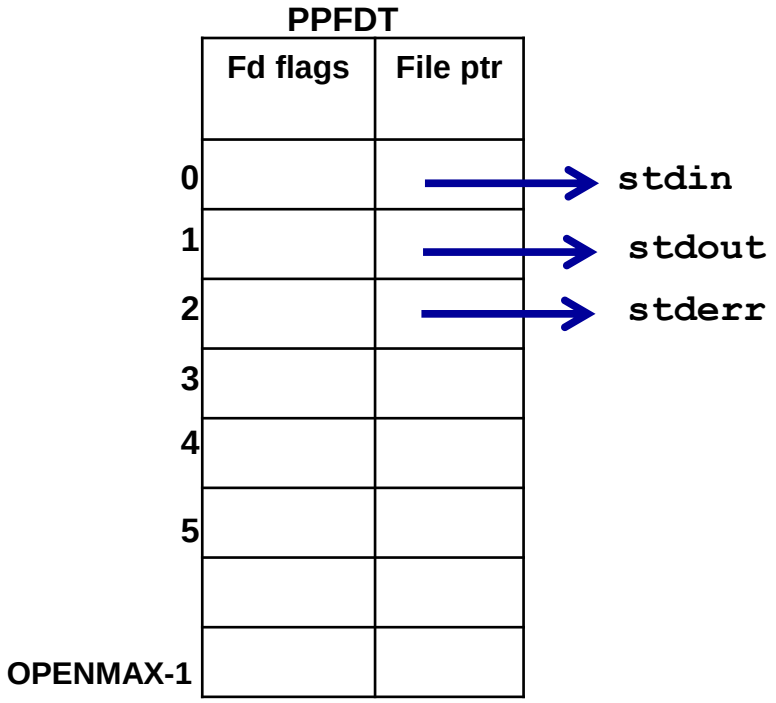
Stdin and Stdout for Commands

Example 1:

```
$ cat
This is GR8
This is GR8
<CTRL + D>
$
```

Example 2:

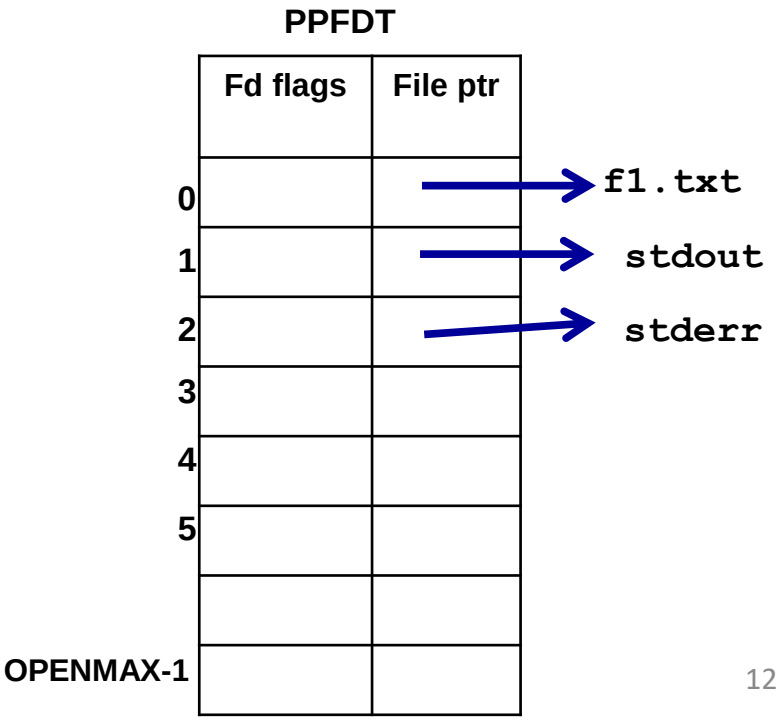
```
$ sort
rauf
arif
kamal
<CTRL+D>
arif
kamal
rauf
$
```



Redirecting Input of a Command (0<)

- By default, **cat** and **sort** commands takes their input form the standard input, i.e. key board. We can detach the key board from stdin and attach some file to it; i.e. **cat** command will now read input from this file and not from the key board

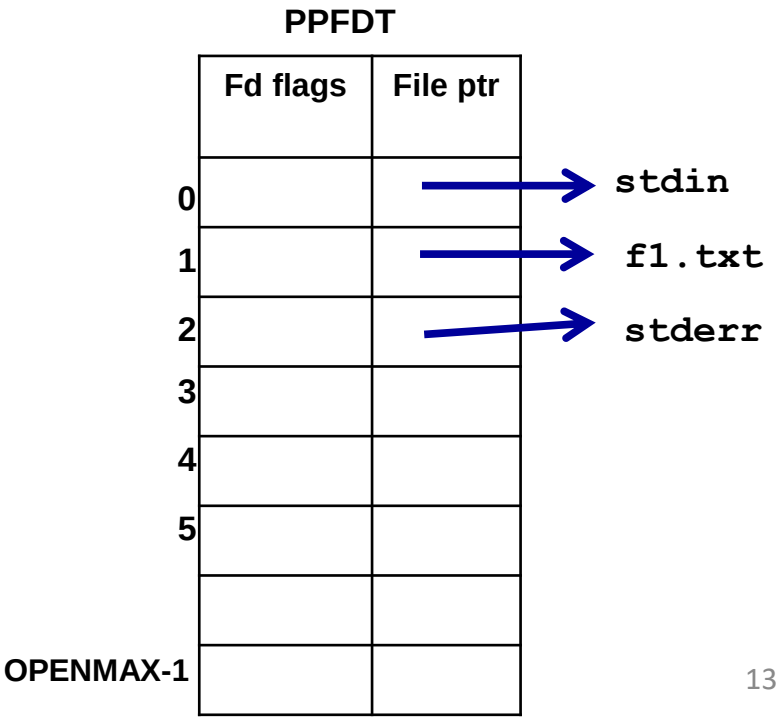
```
# cat  0< f1.txt
# sort 0< f1.txt
```



Redirecting Output of a Command (1>)

Similarly, by default **cat** and **sort** commands sends their outputs to user terminal. We can detach the display screen from stdout and attach a file to it; i.e. **cat** command will now write its output to this file and not to the display screen

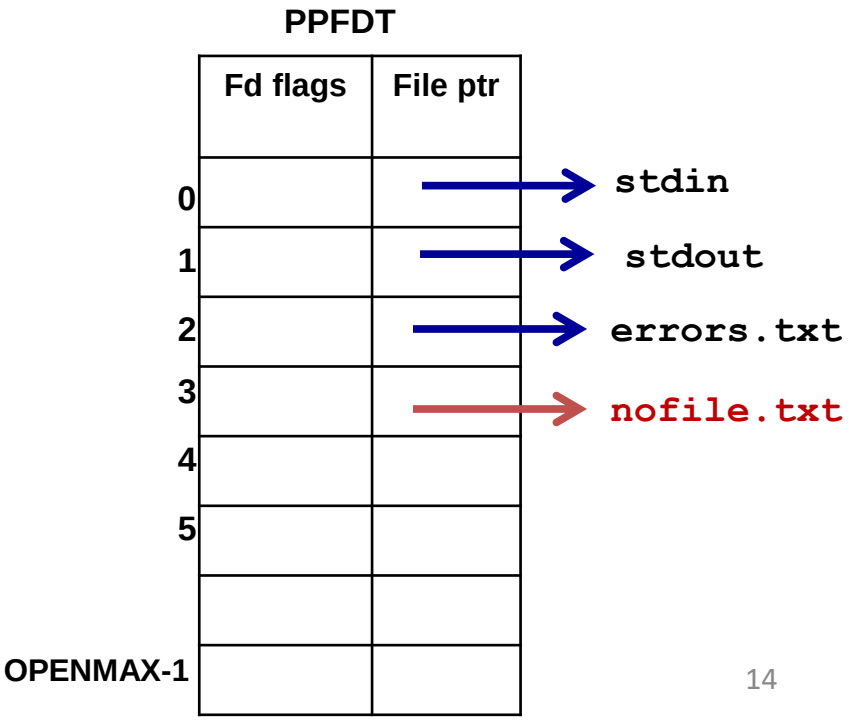
```
# cat 1> f1.txt
```



Redirecting Error of a Command (2>)

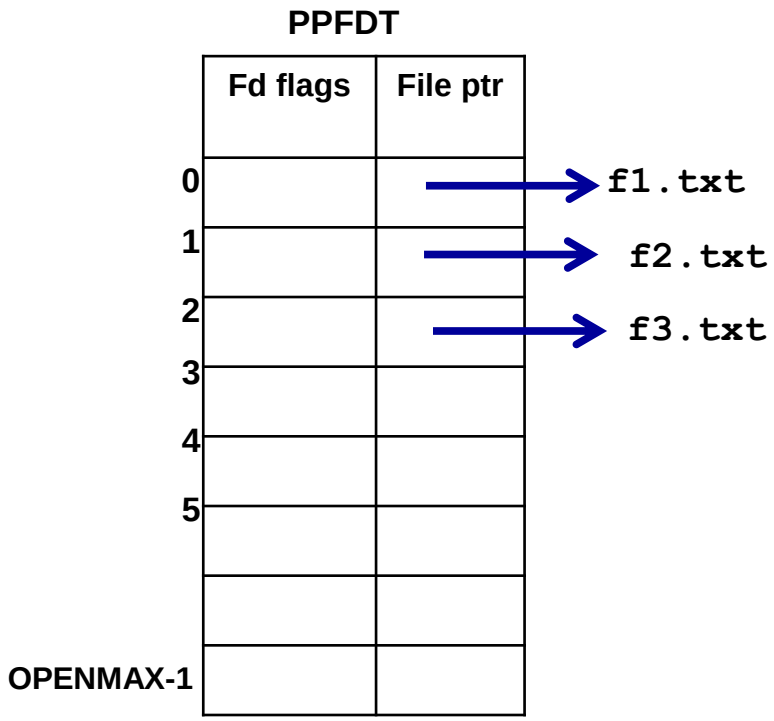
Similarly, by default all commands send their error messages on stderr, which is also connected to the VDU. We can detach the VDU from the error stream and can connect it to a file instead. This is called error redirection

```
# cat nofile.txt 2> errors.txt
```



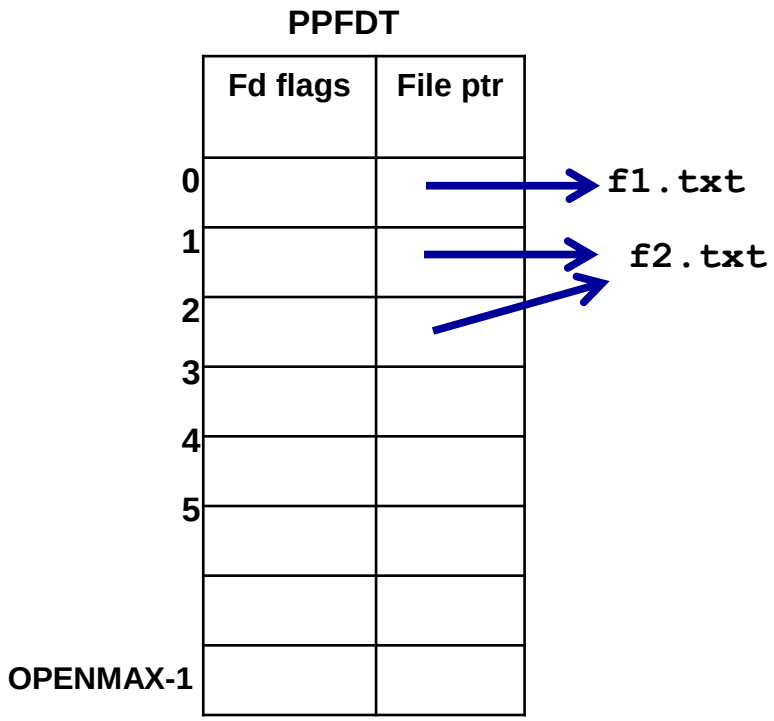
Redirecting Input, Output and Error

```
$ cat 0< f1.txt 1> f2.txt 2> f3.txt
```



Duplicating a File Descriptor

```
$ cat 0< f1.txt 1> f2.txt 2>&1
```



Tasks

In-Lab Task

Draw PPFDT of following Commands

- Differentiate between following two commands (if file2 do not exist)

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

- Differentiate between following two commands (if lab1 do not exist)

```
$ cat lab1.txt 1> output.txt 2> error.txt
$ cat 0< lab1.txt 1> output.txt 2> error.txt
```

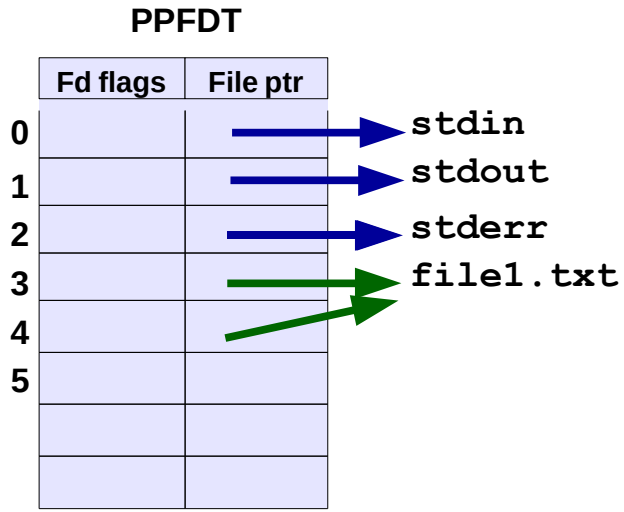
- **Concatenate three files into one.**
- **Execute the command `ls` in such a manner that the output stored in a file "true" while the error is saved in a file "false".**

I/O Redirection using dup ( )

dup() System call

```
int dup(int oldfd);
```

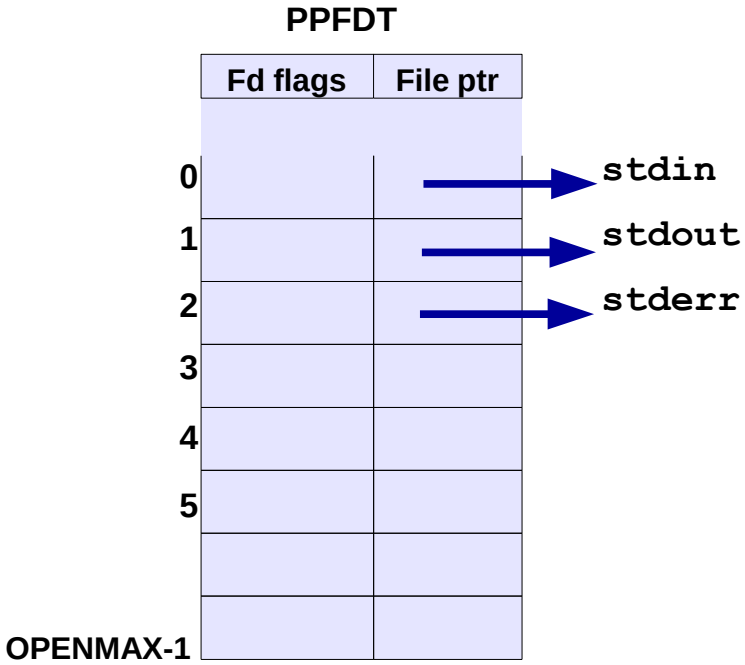
- The dup() call takes oldfd, an open file descriptor, and returns a new descriptor that refers to the same open file description
- The old and the new descriptor both point to the same entry in the SWFT. After a successful return from these function , old and new fd's can be used interchangeably
- The new descriptor is guaranteed to be the lowest unused file descriptor.



Example: dup.c

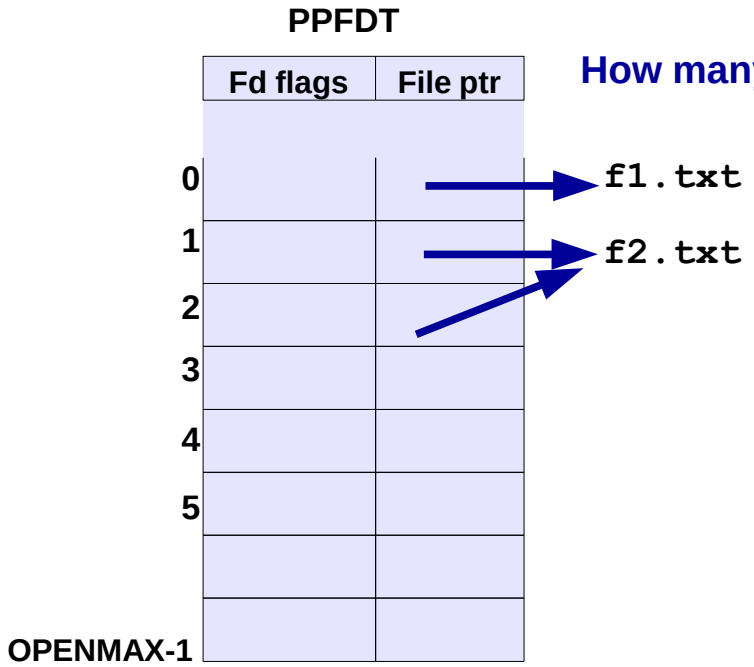
Facts about I/O Redirection on the Shell

```
$ cat
```



Facts about I/O Redirection on the Shell

```
$ cat 0< f1.txt 1> f2.txt 2>&1
```



100$ Q:

How many command line arguments are passed to the `cat` program?

Example: `listargs.c`

```
$ ./a.out 0< /etc/passwd 1> /dev/tty 2> errfile
```

`dup()` System call

```
int dup(int oldfd);
```

- We know that `dup()` call guarantees that the new descriptor returned is the lowest unused file descriptor
- If we run the following LOCs, the `open` call will return 3, the `dup` call will return the lowest unused descriptor which will be zero. So finally descriptor zero points to the opened file instead of `stdin`

```
fd = open(...);
close(0);
newfd = dup(fd);
```

- To make the above code simpler, and to ensure we always get the file descriptor we want, we can use **`dup2()`**

dup2() System call

```
int dup2(int oldfd, int newfd);
```

- The dup2() system call makes a duplicate of the file descriptor given in oldfd using the descriptor number supplied in newfd
- If the file descriptor specified in newfd is already open, dup2() closes it first
- We can simplify the preceding calls to close(0) and dup(fd) on previous slide to the following:

  ```
  dup2(fd, 0);
  ```

- A successful dup2() call returns the number of the duplicate descriptor (i.e., the value passed in newfd)
- If oldfd is a valid file descriptor, and oldfd and newfd have the same value, then dup2() does nothing—newfd is not closed, and dup2() returns the newfd

`dup3()` System call

```
int dup3(int oldfd, int newfd, int flags);
```

- The `dup3()` system call performs the same task as `dup2()`, but adds an additional argument, flags, that is a bit mask that modifies the behavior of the system call
- At the time of this writing, `dup3()` supports one flag, O_CLOEXEC, which causes the kernel to enable the close-on-exec flag (FD_CLOEXEC) for the new file descriptor
- The `dup3()` system call is new in Linux 2.6.27, and is Linux-specific

Examples: Input Redirection

Method 1: close-open (stdinredir1.c)

```
close(0);
fd = open("/etc/passwd", O_RDONLY);
```

Method 2: open-close-dup-close (stdinredir2.c)

```
fd = open("/etc/passwd", O_RDONLY);
close(0);
newfd = dup(fd);
close(fd);
```

Method 3: open-dup2-close (stdinredir3.c)

```
fd = open("/etc/passwd", O_RDONLY);
newfd = dup2(fd, 0);
close(fd);
```

In Lab Task

- Write a program using dup() system call that duplicated file descriptor 0,1,2,3 to a single file.
- Do the above task using dup2().