

Lab – File Descriptor Table

Christian Khoury

1 Exploring Redirections

- create a file, call it *text1* and write anything in it
- What happens when you run “cat text1 > text2” ? look at the contents of text2
- write a small program that does the following :
 1. open “text1” in read mode (man 2 open)
 2. open “text2” in write mode
 3. redirect standard output to text2 (man 2 dup2)
 4. create a loop that reads text1 and writes it on standard output

Depict what happens in the file descriptor table upon running the preceding code.

2 Pipes & the file descriptor table

A pipe is used in the following example :

```
ps aux | more
```

It enables two different processes to exchange information using file descriptors.

1. What kind of interaction is there between these two functions (ps and more) ?

Hereafter, you'll find a small example using pipes ! Understand its content especially how pipes are created and used.

```
#include <sys/wait.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
int main(int argc, char *argv[]) {
```

```

int
pipefd[2];
pid_t cpid;
char buf;

if (argc != 2) {
    fprintf(stderr, "Usage: %s <string>\n", argv[0]);
    exit(EXIT_FAILURE);
}

if (pipe(pipefd) == -1) {
    perror("pipe");
    exit(EXIT_FAILURE);
}

cpid = fork(); if
(cpid == -1) {
    perror("fork");
    exit(EXIT_FAILURE)
;
}

if (cpid == 0) { /* Child reads from pipe */
    close(pipefd[1]); /* Close unused write end */

    while (read(pipefd[0], &buf, 1) > 0)
        write(STDOUT_FILENO, &buf, 1);

    write(STDOUT_FILENO, "\n", 1);
    close(pipefd[0]);
    _exit(EXIT_SUCCESS);
} else { /* Parent writes argv[1] to pipe */
    close(pipefd[0]); /* Close unused read end */
    write(pipefd[1], argv[1], strlen(argv[1]));
    close(pipefd[1]); /* Reader will see EOF */
    wait(NULL); /* Wait for child */
    exit(EXIT_SUCCESS);
}
}

```

Now, it's time to apply what you've learnt ! Code the following operation : "ps aux | more" using pipes.

- Child runs the **more** operation. **more** is a pager which reads information from the **standard input = STDIN_FILENO** and displays it on the standard output, therefore
 1. close the pipe write end
 2. make the standard input to be the read end ! (use **dup2** for this purpose)
 3. execute "more"
- Parent runs the **ps aux** operation which uses the **standard output = STDOUT_FILENO** for display.
 1. close the pipe read end
 2. execute "ps aux"

Depict what happens in the file descriptor table upon running the preceding code.

3 Non-Blocking Calls

```
#include <stdio.h>
#include <unistd.h>
#include <errno.h>
#include <sys/types.h>
#include <fcntl.h>

int main() {
    int i; char
    buf[100];

    fcntl(STDIN_FILENO, F_SETFL, O_NONBLOCK);

    for (i = 0; i < 10; i++)
    { int nb;

        nb = read(STDIN_FILENO, buf, 100);
        printf("nwrites = %d\terror = %d\n", nb,
            errno);
    }
}
```

- What happens when the « read » call is executed with or without the « fcntl » function call ?