

IT253 - OPERATING SYSTEMS

ASSIGNMENT 2

Name: **Sachin Prasanna**

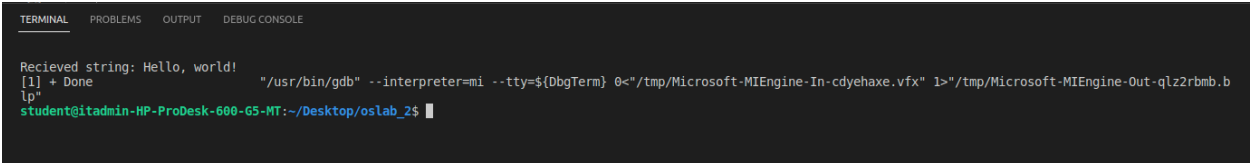
Roll No.: **211IT058**

2)

Code Written:

```
C exercise2.c > main(void)
1  #include <stdio.h>
2  #include <unistd.h>
3  #include <sys/types.h>
4  #include <string.h>
5  #include <stdlib.h>
6
7  int main (void)
8  {
9      int fd[2], nbytes;
10     pid_t childpid;
11     char string[] = "Hello, world!\n";
12     char readbuffer[80];
13
14     pipe(fd);
15
16     if((childpid = fork()) == -1){
17         perror("fork");
18         exit(1);
19     }
20
21     if(childpid == 0){
22         close (fd[0]);
23         write(fd[1], string, (strlen(string) + 1));
24         exit(0);
25     }
26
27     else{
28
29         close (fd[1]);
30
31         nbytes = read(fd[0], readbuffer, sizeof(readbuffer));
32         printf("Recieved string: %s", readbuffer);
33     }
34
35     return (0);
36 }
```

Output:



```
TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE

Recieved string: Hello, world!
[1] + Done
lp"
student@itadmin-HP-ProDesk-600-G5-MT:~/Desktop/oslab_2$
```

Observation: The result, as expected, is the string **“Hello, world!”**. It is a basic program where the child process is sending a message to the parent process. **One pipe is created and communication happens in this pipe, where the information goes through the kernel as well.** Since the child is sending data to the parent, **the child closes its fd[0], which is for read operation.** Then, it **writes the message to be sent using the write() system call.**

Since the parent is receiving or reading the data sent by the child, its **fd[1] is closed**, which is used for write operation. Thereafter, **it reads the message sent by the read() system call** and the read value is outputted to the console.

3)

Code Written:

PTO

```

C exercise3.c > main(void)
1  #include <stdio.h>
2  #include <unistd.h>
3  #include <sys/types.h>
4  #include <string.h>
5  #include <stdlib.h>
6
7  int main (void)
8  {
9      int fd[2], nbytes;
10     pid_t childpid;
11     char string[] = "Hello, world!\n";
12     char readbuffer[80];
13
14     pipe(fd);
15
16     if((childpid = fork()) == -1){
17         perror("fork");
18         exit(1);
19     }
20
21     if(childpid == 0){
22
23         close (fd[1]);
24         nbytes = read(fd[0], readbuffer, sizeof(readbuffer));
25         printf("Recieved string: %s", readbuffer);
26     }
27
28     else{
29         close (fd[0]);
30         write(fd[1], string, (strlen(string) + 1));
31         exit(0);
32     }
33
34     return (0);
35 }

```

Output:

```

TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE

Recieved string: Hello, world!
[1] + Done          "/usr/bin/gdb" --interpreter=mi --tty=${DbgTerm} 0<"/tmp/Microsoft-MIEngine-In-4knkjs3tf.ziq" 1>"/tmp/Microsoft-MIEngine-Out-urcuqild.1
20"
student@itadmin-HP-ProDesk-600-G5-MT:~/Desktop/oslab_2$

```

Observation: Code written in Question 2 is modified by simply **interchanging the roles of parent and child**. In this case, the child receives or reads the data, so its fd[1] is closed. The parent is sending or writing the data, so its fd[0] is closed. The **code for reading data is written under the child process and the code for writing data is written under the parent process**.

4)

Text File:

```
test.txt
1 CCN ASSIGNMENT 2
```

PTO

Code:

```
1  #include <stdio.h>
2  #include <unistd.h>
3  #include <sys/types.h>
4  #include <string.h>
5  #include <stdlib.h>
6
7  int main (void) {
8      int fd[2], fd2[2], nbytes;
9      char filename[80];
10     char readbuffer[80];
11     char msgRecieved[80];
12     char finalMsg[80];
13     pid_t childpid;
14
15     //Two pipelines are needed to solve this, and hence are defined below.
16     pipe(fd);
17     pipe(fd2);
18
19     if((childpid = fork()) == -1){
20         perror("fork");
21         exit(1);
22     }
23
24     /*FD(0) is to READ
25      FD(1) is to WRITE*/
26
27     if(childpid == 0){
28         /*CLIENT AND CHILD PROCESS*/
29
30         close(fd[0]);
31
32         printf("Enter File name: ");
33         scanf("%s", filename);
34
35         write(fd[1], filename, (strlen(filename) + 1));
36         close(fd[1]);
```

```
37
38         nbytes = read(fd2[0], finalMsg, sizeof(finalMsg));
39         printf("Recieved string: %s", finalMsg);
40         close (fd2[0]);
41
42         exit(0);
43     }
44
45     else{
46         /*SERVER AND PARENT PROCESS*/
47         close (fd[1]);
48
49         nbytes = read(fd[0], readbuffer, sizeof(readbuffer));
50         printf("Received file name: %s",readbuffer);
51
52         FILE* fp;
53         fp = fopen(readbuffer,"r");
54
55         close(fd2[0]);
56
57         if(fp == NULL){
58             //fp is NULL when the file does not exist
59             char errorMsg[40] = "ERROR! NO FILE IN THAT NAME";
60             write(fd2[1], errorMsg, (strlen(errorMsg)+1));
61         }
62
63         else{
64             fgets(msgRecieved,80,fp);
65             fclose(fp);
66             write(fd2[1], msgRecieved, (strlen(msgRecieved)+1));
67         }
68         close(fd2[1]);
69     }
70     return (0);
71 }
```

Output (When file is present):

```
TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE

Enter File name: test.txt
Received file name: test.txt
Recieved string: CCN ASSIGNMENT 2

[1] + Done
student@itadmin-HP-ProDesk-600-G5-MT:~/Desktop/oslab_2$ "
```

Output (When file is not present):

```
TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE

Enter File name: hello
Received file name: hello
Recieved string: ERROR! NO FILE IN THAT NAME

[1] + Done
student@itadmin-HP-ProDesk-600-G5-MT:~/Desktop/oslab_2$ "
```

Explanation of code: The program begins by declaring the necessary variables and creating the two pipelines using the pipe() function. Then, a child process is forked off the parent process, and the child process is responsible for getting the file name from the user and sending it to the parent process (which acts as the server in this case).

Child Process (Client):

The child process first **closes the read end of the first pipeline** (fd[0]), since it does not need to read anything from it. It then asks the user to enter a file name using printf(), and waits for the user to enter the file name using scanf(). The file name is stored in the filename variable, **which is then written to the write end of the first pipeline (fd[1]) using write()**.

The child process then closes the write end of the first pipeline, since it has finished writing to it. It waits to receive a string from the server process using the read() function, which **reads from the read end of the second pipeline (fd2[0])**. The string received from the server is stored in the finalMsg variable,

which is then displayed to the user using printf(). Finally, the child process closes the read end of the second pipeline and exits.

Parent Process (Server):

In the parent process (which acts as the server), the **write end of the first pipeline (fd[1]) is closed, since the parent process does not need to write anything to it.** The parent process waits to receive the file name from the client process using read(), **which reads from the read end of the first pipeline (fd[0]).** The file name is stored in the readbuffer variable, which is used to open the file using fopen().

The read end of the second pipeline (fd2[0]) is then closed, since the parent process does not need to read anything from it. **If the file could not be opened (i.e., fp is NULL), the parent process writes an error message** to the write end of the second pipeline (fd2[1]) using write(). **If the file could be opened, the parent process reads the content of the file using fgets(),** stores it in the msgReceived variable, closes the file using fclose(), and writes the string to the write end of the second pipeline using write(). Finally, the parent process closes the write end of the second pipeline and returns 0 and the program is completed.

Observations: If a valid file name is entered, the program will read the contents of the file and display them on the screen as depicted in the screenshots above. However, if the filename entered does not exist, an error message will be displayed.
