LABORATORY 6

–

OPERATING SYSTEMS

RUFINO GARCIA SANCHEZ

EXERCISE 1

- **Write the program which creates two processes: :**
    - **a parent reads data from the pipe,**
    - **a child writes to the pipe**

TASK 1

The code of the task 1 is this:

```c
#include <unistd.h>
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>
#include <string.h>

int main() {
int ret_val;
int pfd[2];
char buff[32];
char string1[]="String for pipe I/O";

ret_val = pipe(pfd);                    /* Create pipe */
if (ret_val != 0) {                  /* Test for success */
  printf("Unable to create a pipe; errno=%d\n",errno);

  exit(1);                           /* Print error message and exit */
}
if (fork() == 0) {
   /* child program */
   close(pfd[0]); /* close the read end */
   ret_val = write(pfd[1],string1,strlen(string1)); /*Write to pipe*/
   if (ret_val != strlen(string1)) {
      printf("Write did not return expected value\n");
      exit(2);                         /* Print error message and exit */
   }
}
else {
   /* parent program */
   close(pfd[1]); /* close the write end of pipe */
   ret_val = read(pfd[0],buff,strlen(string1)); /* Read from pipe */
   if (ret_val != strlen(string1)) {
      printf("Read did not return expected value\n");
      exit(3);                         /* Print error message and exit */
   }
   printf("parent read %s from the child program\n",buff);
}
exit(0);
}
```

Consider the following situation (described below) and explain what happens in these cases:

When writing to the full pipe:

```
int main() {
int ret_val;
int pfd[2];
char buff[32];
char string1[]="String for pipe I/O";

ret_val = pipe(pfd);                    /* Create pipe */
if (ret_val != 0) {                 /* Test for success */
  printf("Unable to create a pipe; errno=%d\n",errno);

  exit(1);                          /* Print error message and exit */
}
if (fork() == 0) {
   /* child program */
   close(pfd[0]); /* close the read end */
   ret_val = write(pfd[1],string1,strlen(string1)-1); /*Write to pipe*/
   if (ret_val != strlen(string1)) {
      printf("Write did not return expected value\n");
      exit(2);                          /* Print error message and exit */
   }
}
else {
   /* parent program */
   close(pfd[1]); /* close the write end of pipe */
   ret_val = read(pfd[0],buff,strlen(string1)); /* Read from pipe */
   if (ret_val != strlen(string1)) {
      printf("Read did not return expected value\n");
      exit(3);                          /* Print error message and exit */
   }
   printf("parent read %s from the child program\n",buff);
}
exit(0);
}
```

And this happens when we execute the program:

```
Read did not return expected value
Write did not return expected value
```

When reading from the empty pipe:

```
int main() {
int ret_val;
int pfd[2];
char buff[32];
char string1[]=""; /* Nothing in the string */

ret_val = pipe(pfd);                    /* Create pipe */
if (ret_val != 0) {                 /* Test for success */
  printf("Unable to create a pipe; errno=%d\n",errno);

  exit(1);                          /* Print error message and exit */
}
if (fork() == 0) {
   /* child program */
   close(pfd[0]); /* close the read end */
   ret_val = write(pfd[1],string1,strlen(string1)); /*Write to pipe*/
   if (ret_val != strlen(string1)) {
      printf("Write did not return expected value\n");
      exit(2);                          /* Print error message and exit */
   }
}
else {
   /* parent program */
   close(pfd[1]); /* close the write end of pipe */
   ret_val = read(pfd[0],buff,strlen(string1)); /* Read from pipe */
   if (ret_val != strlen(string1)) {
      printf("Read did not return expected value\n");
      exit(3);                          /* Print error message and exit */
   }
   printf("parent read %s from the child program\n",buff);
}
exit(0);
}
```

And this happens when we run and execute the code:

```
parent read □□□ from the child program
```

When writing to the pipe when the reader had been closed:

```
int main() {
int ret_val;
int pfd[2];
char buff[32];
char string1[]="Hi, now the reader is close"; /* Nothing in the string */

ret_val = pipe(pfd);                  /* Create pipe */
if (ret_val != 0) {              /* Test for success */
  printf("Unable to create a pipe; errno=%d\n",errno);

  exit(1);                           /* Print error message and exit */
}
if (fork() == 0) {
   /* child program */
   close(pfd[0]); /* close the read end */
   ret_val = write(pfd[1],string1,strlen(string1)); /*Write to pipe*/
   if (ret_val != strlen(string1)) {
      printf("Write did not return expected value\n");
      exit(2);                         /* Print error message and exit */
   }
}
else {
   /* parent program */
   close(pfd[0]); /* close the write end of pipe */
   ret_val = read(pfd[0],buff,strlen(string1)); /* Read from pipe */
   if (ret_val != strlen(string1)) {
      printf("Read did not return expected value\n");
      exit(3);                         /* Print error message and exit */
   }
   printf("parent read %s from the child program\n",buff);
}
exit(0);
}
```

And this is what happens:

```
Read did not return expected value
```

When reading from the pipe with writer had been closed:

```
int main() {
int ret_val;
int pfd[2];
char buff[32];
char string1[]="Hi, now the reader is close"; /* Nothing in the string */

ret_val = pipe(pfd);                  /* Create pipe */
if (ret_val != 0) {              /* Test for success */
  printf("Unable to create a pipe; errno=%d\n",errno);

  exit(1);                           /* Print error message and exit */
}
if (fork() == 0) {
   /* child program */
   close(pfd[1]); /* close the read end */
   ret_val = write(pfd[1],string1,strlen(string1)); /*Write to pipe*/
   if (ret_val != strlen(string1)) {
      printf("Write did not return expected value\n");
      exit(2);                         /* Print error message and exit */
   }
}
else {
   /* parent program */
   close(pfd[1]); /* close the write end of pipe */
   ret_val = read(pfd[0],buff,strlen(string1)); /* Read from pipe */
   if (ret_val != strlen(string1)) {
      printf("Read did not return expected value\n");
      exit(3);                         /* Print error message and exit */
   }
   printf("parent read %s from the child program\n",buff);
}
exit(0);
}
```

This is what happens when we run and execute the code:

```
Read did not return expected value
Write did not return expected value
```

TASK 2

The code of this program is in the next image:

```c
#include <unistd.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <errno.h>
#include <fcntl.h>
#include <stdlib.h>
 #include <stdio.h>
 #include <string.h>
int main(){
char string1[]="String for pipe I/O now we are trying to go over the buffer";
char buff[32];

    if ((mkfifo("pipe1",0777)) != 0) {
        printf("Unable to create a fifo; errno=%d\n",errno);
        exit(1);                        /* Print error message and return */
    }



    if (fork() == 0) {
        /* child program */
        int fd= open("pipe1",O_WRONLY);
        if(fd==-1){
            return 1;
        }
        write(fd,string1,strlen(string1)); /*Write to pipe*/
        close(fd);
    }
    else {
        /* parent program */
        int fd= open("pipe1",O_RDONLY);
            if(fd==-1){
                return 1;
            }
        read(fd,buff,strlen(string1)); /* Read from pipe */
        printf("parent read %s from the child program\n",buff);
        close(fd);
    }


}
```

In this case I have obtained exactly the same results as the first task. In this part of the exercise I have reached the same goals as the first part so I will introduce my conclusion to achieve the principal and final goal of the exercise that is work with pipes.

I have obtained and I have reached the conclusions that I'm going to explain in the next paragraphs:

First of all I have to introduce that all this information comes from the lectures and also from the man of Linux that is based in unix.

When writing to the full pipe:

If a process attempts to write to a full pipe (see below), then write(2) blocks untilsufficient data has been read from the pipe to allow the write to complete.

When reading from the empty pipe:

If a process attempts to read from an empty pipe, then read(2) will block until data is available.

When writing to the pipe when the reader had been closed:

If all file descriptors referring to the write end of a pipe have been closed, then an attempt to read(2) from the pipe will see end-of-File.

When reading from the pipe with writer had been closed:

If all file descriptors referring to the read end of a pipe have been closed, then a write(2) will cause a SIGPIPE signal to be generated for the calling process.