The School of Information and Software Engineering in UESTC



**Lab Report Practical 2**

Student ID 2016220301035

Student Name Xiaojie Li

Course Name Operating System

Theory Teacher Jia Chen

Lab Teacher Jia Chen

1. **Lab name：**

Process communication with pipe

1. **Class Hours:**

4 hours

1. **Experimental Objectives**
2. be familiar with application development under Linux
3. be familiar with Linux process control primitives
4. master the use of the process communication mechanism pipe in Linux.
5. master the parent process and the child process synchronization in Linux.
6. **Experimental Contents**

In Linux using the system call fork () to create two sub-processes, using the system call pipe () to build a pipe, each sub-processes writing a sentence to the pipe:

Child process 1 is sending a message!

Child process 2 is sending a message!

The parent process reads message from the two sub-processes through pipe, and displayed it on the screen. Then terminate the two sub-processes.

Claim:

(1). The parent process firstly receives the message from sub-process P1, then receives the message from sub-process P2

(2) To implement it in Linux.

1. **Experimental Principle**

First create two sub-processes, pay attention to the use of fork () function which creates a process in Linux. Due to the synchronization between parent and two child processes, using waitpid () function to implement that the parent process reads data and print it after the child process finished. Only after the child process writing data to the pipe, the parent process can be able to open pipe.

Since the fork function allows the child process to complete copy of the entire address space of the parent process, so the child process owns read and write ports of the pipe. So it’s better to turn off unused port. Since " The parent process firstly receives the message from sub-process P1, then receives the message from sub-process P2". So there must be two synchronization problems. The first one is between parent and child process which requires the parent can read after child writing, the second one is between child and child process which requires child process P1 write first, then the P2 write.

1. **Experimental steps:**

Step 1: Create a pipe.

Step 2: Create a child process, P1, write "Child process 1 is sending a message!" to the pipe, and it should be synchronization with the parent process.

Step 3: Create the child process 2, write to the pipe "Child process 2 is sending a message!" And it should be synchronization with the parent process.

Step 4: The parent process reads data from the pipe and prints it.

1. **Experimental Results and Analysis:**

Code to create a pipe, a child process P1 and Child process P2, letting the parent process read their data from pipe and prints it out.

The Code and results are as follows:

|  |
| --- |
| Codes:  #include<signal.h>  #include<unistd.h>  #include<stdio.h>  #define sum 50  char \*message1 = "Child process1 is sending a message!";  char \*message2 = "Child process2 is sending a message!";  int main()  {  int p1,p2;  int a[2];  char b[50];  pipe(a);  while ((p1=fork())== -1);  if(p1==0){  close(a[0]);  write(a[1],message1,sum);  //sleep(3);  return 0;  }else{  waitpid(p1,NULL,0);  while ((p2=fork())== -1);  if(p2==0){  close(a[0]);  write(a[1],message2,sum);  return 0;  }else{  waitpid(p2,NULL,0);  close(a[1]);  read(a[0],b,sum);  printf("%s\n",b);  read(a[0],b,sum);  printf("%s\n",b);  return 0;  }  }  } |
| Results:  IMG_20180602_104223 |

**Table 1** The implementation of pipe communication

1. **Experimental Summary:**

In this experiment, the communication between two processes are implemented on pipe. Besides, I have a better understand of the Communication Pipeline and the asynchronous data communication.

**Report Score：**

**Tutor：**