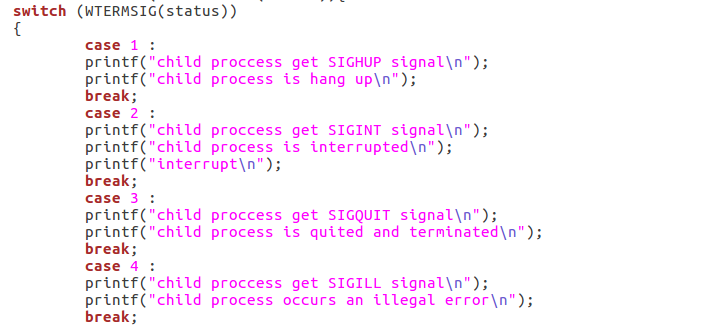
**Report**

**Li Huayue 118010138**

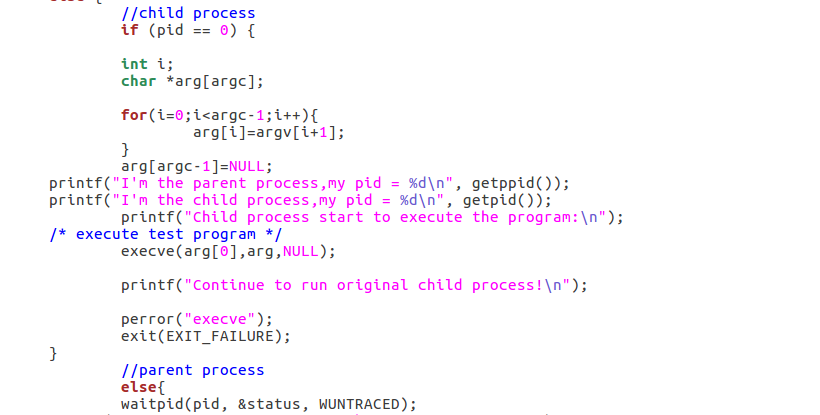
**1 Program1:**

1. The thought of design:

I get my idea of program1 from the note of tut1. I use a function to create a child process to execute the test program. I use a to make the parent process waits for getting the signal from child process. I use to connect the test programs and program1 file and execute test programs in child process. To get and print the information like signal in the terminal, I use a switch structure which is shown as follow:



Because the of child process is 0 relative to its parent and that of parent process is positive, therefore, I use an to distinguish the child process and the parent process which is shown as follows:



1. The steps to execute the program1 and output:

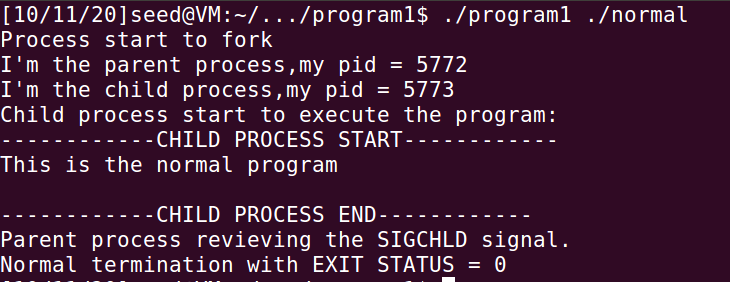
The steps to execute my program1 is shown as follows(use normal, abort and stop as examples):

Step1: enter “make” in the terminal to compile all the files in the folder

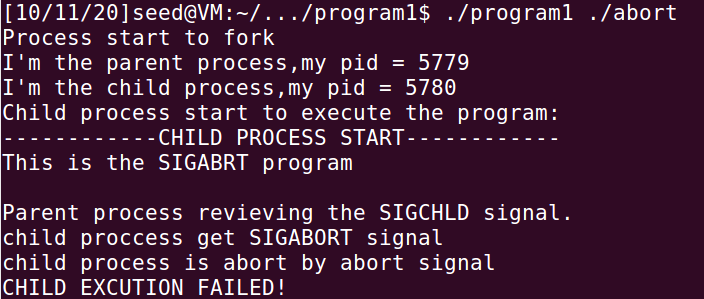
Step2: enter “./program1 ./normal” (use normal as an example), then you will see the output in the terminal.

Output:

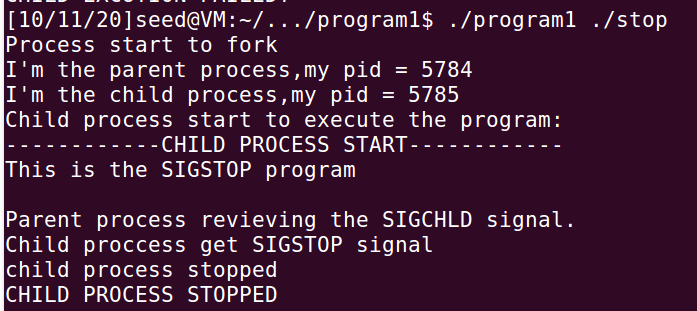
The output of normal:



The output of abort:



The output of stop:



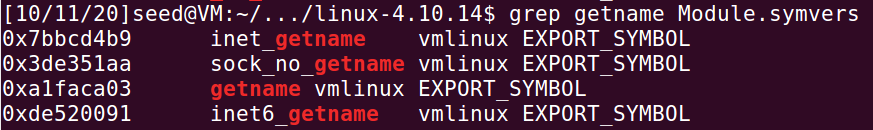
**2 Program2:**

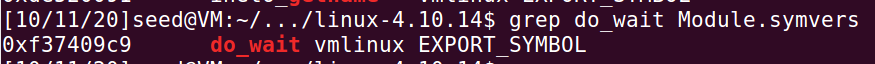
1. The thought of design:

I get the idea of program2 from the note of tut3. I first change the content of four parts in the kernel which is shown as follows:









Then I compile and install the kernel following the steps in the tut note. I use these four functions in my code by changing lightly of the tut code provided. Then I design the , and functions.

The is used to execute the test program in child process in kernel module. The is used to make the parent process stop until the child process finish or get signal from it. The is used to create the child process in kernel module (In this part, we cannot use pid is equal to 0 or not to distinguish the child process and parent process which is different from program1).

1. The steps to execute the program2 and output:

Hints: check the path which is shown as follows:



1 use “gcc test.c -o test” to compile the test file

2 enter “make” in the terminal

3 enter “sudo su”

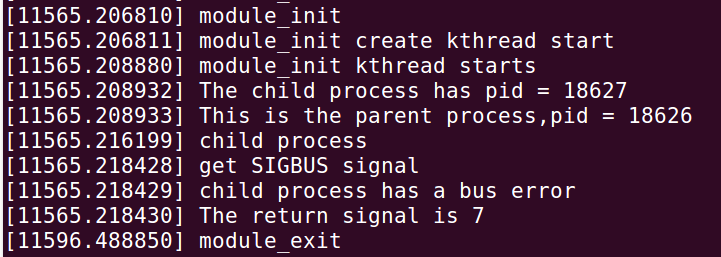
4 enter “insmod program2.ko”

5 enter “dmesg”

Hints: If there already exists program2.ko, you need to rmmod program2.ko and then follow the steps from 4 to 5 (or the output may be strange).

Then you will see all the output

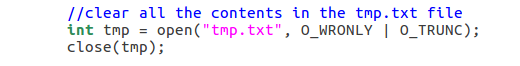
Output (use SIGBUS as an example):



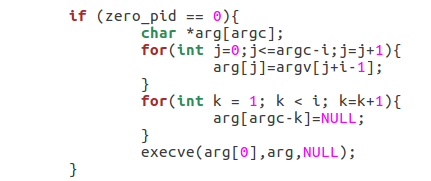
**3 Program3:**

1. The thought of design:

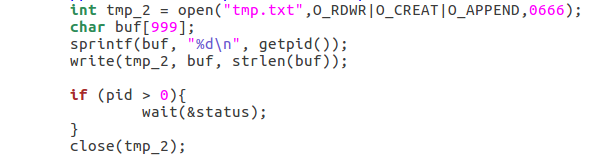
In this program, I use the file stream to solve this problem. Whenever I run the program, I first clean all the contents in the tmp.txt file:



Then I use a special “for” structure to make the child process “still in” the for loop while the parent process can go down to execute the following code. I use a to store the special pid which belongs to the process which is not only child but also parent. They will go to the following code and execute the test program in the child process:



I use a buf connected with the tmp.txt file to store all the pids of each process and finally print it in the terminal.



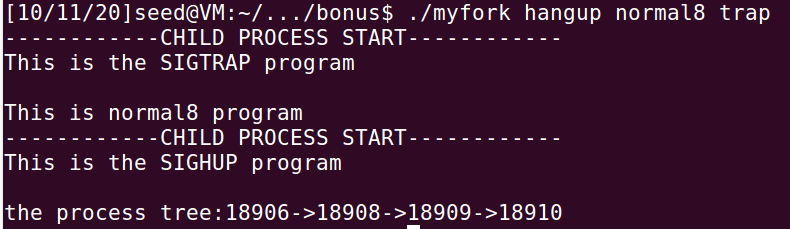
Finally, I will print the process tree in the terminal by reading the tmp.txt file:

1. The steps to execute the program2 and output:

1 enter ”make” in the terminal

2 enter “./myfork hangup normal8 trap” in the terminal (use hangup normal8 trap as an example)

Output:



**4 The environment of running my program:**

1. The version of OS:



1. The version of kernel:



**5 What I have learned from the tasks:**

I have learned a lot about the environment of Linux and kernel. The setup and compile process is difficult and troublesome but I have learned a lot from the process. The tasks are difficult for me because this is the first time for me to use Linux to run program but I have tried my best and overcome a lot of problems. I have learned a lot from this project.