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 $fork()$ function to create a child process to execute the test program. I use a $waitpid(pid, \&status, WUNTRACED);$ to make the parent process waits for getting the signal from child process. I use $execve(arg[0], arg, NULL);$ 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:

```
switch (WTERMSIG(status))
{
        case 1 :
        printf("child proccess get SIGHUP signal\n");
        printf("child process is hang up\n");
        break;
        case 2 :
        printf("child proccess get SIGINT signal\n");
        printf("child process is interrupted\n");
        printf("interrupt\n");
        break;
        case 3 :
        printf("child proccess get SIGQUIT signal\n");
        printf("child process is quited and terminated\n");
        break;
        case 4 :
        printf("child proccess get SIGILL signal\n");
        printf("child process occurs an illegal error\n");
        break;
```

Because the pid of child process is 0 relative to its parent and that of parent process is positive, therefore, I use an $if-else\ structure$ to distinguish the child process and the parent process which is shown as follows:

```
        //child process
        if (pid == 0) {

        int i;
        char *arg[argc];

        for(i=0;i<argc-1;i++){
                arg[i]=argv[i+1];
        }
        arg[argc-1]=NULL;
printf("I'm the parent process,my pid = %d\n", getppid());
printf("I'm the child process,my pid = %d\n", getpid());
        printf("Child process start to execute the program:\n");
/* execute test program */
        execve(arg[0],arg,NULL);

        printf("Continue to run original child process!\n");

        perror("execve");
        exit(EXIT_FAILURE);
}
        //parent process
        else{
        waitpid(pid, &status, WUNTRACED);
```

(2) 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:

```
[10/11/20]seed@VM:~/.../program1$ ./program1 ./normal
Process start to fork
I'm the parent process,my pid = 5772
I'm the child process,my pid = 5773
Child process start to execute the program:
------------CHILD PROCESS START------------
This is the normal program

------------CHILD PROCESS END------------
Parent process revieving the SIGCHLD signal.
Normal termination with EXIT STATUS = 0
```

The output of abort:

```
[10/11/20]seed@VM:~/.../program1$ ./program1 ./abort
Process start to fork
I'm the parent process,my pid = 5779
I'm the child process,my pid = 5780
Child process start to execute the program:
------------CHILD PROCESS START------------
This is the SIGABRT program

Parent process revieving the SIGCHLD signal.
child proccess get SIGABORT signal
child process is abort by abort signal
CHILD EXCUTION FAILED!
```

The output of stop:

```
[10/11/20]seed@VM:~/.../program1$ ./program1 ./stop
Process start to fork
I'm the parent process,my pid = 5784
I'm the child process,my pid = 5785
Child process start to execute the program:
------------CHILD PROCESS START------------
This is the SIGSTOP program

Parent process revieving the SIGCHLD signal.
Child proccess get SIGSTOP signal
child process stopped
CHILD PROCESS STOPPED
```

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:

```
[10/11/20]seed@VM:~/.../linux-4.10.14$ grep do_execve Module.symvers
0x0a750c43      do_execve       vmlinux EXPORT_SYMBOL
[10/11/20]seed@VM:~/.../linux-4.10.14$ grep _do_fork Module.symvers
0xeac5b58b      _do_fork        vmlinux EXPORT_SYMBOL
[10/11/20]seed@VM:~/.../linux-4.10.14$ grep getname Module.symvers
0x7bbcd4b9      inet_getname    vmlinux EXPORT_SYMBOL
0x3de351aa      sock_no_getname vmlinux EXPORT_SYMBOL
0xa1faca03      getname vmlinux EXPORT_SYMBOL
0xde520091      inet6_getname   vmlinux EXPORT_SYMBOL
[10/11/20]seed@VM:~/.../linux-4.10.14$ grep do_wait Module.symvers
0xf37409c9      do_wait vmlinux EXPORT_SYMBOL
```

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 my_exec, my_fork and my_wait functions.

The my_exec is used to execute the test program in child process in kernel module. The my_wait is used to make the parent process stop until the child process finish or get signal from it. The my_fork 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).

(2) The steps to execute the program2 and output:

Hints: check the path which is shown as follows:

```
const char path[] = "/home/seed/work/assignemt/source/program2/test";
```

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):

```
[11565.206810] module_init
[11565.206811] module_init create kthread start
[11565.208880] module_init kthread starts
[11565.208932] The child process has pid = 18627
[11565.208933] This is the parent process,pid = 18626
[11565.216199] child process
[11565.218428] get SIGBUS signal
[11565.218429] child process has a bus error
[11565.218430] The return signal is 7
[11596.488850] module_exit
```

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:

```
//clear all the contents in the tmp.txt file
int tmp = open("tmp.txt", O_WRONLY | O_TRUNC);
close(tmp);
```

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 *zero_pid* 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:

```
if (zero_pid == 0){
        char *arg[argc];
        for(int j=0;j<=argc-i;j=j+1){
                arg[j]=argv[j+i-1];
        }
        for(int k = 1; k < i; k=k+1){
                arg[argc-k]=NULL;
        }
        execve(arg[0],arg,NULL);
}
```

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.

```
int tmp_2 = open("tmp.txt",O_RDWR|O_CREAT|O_APPEND,0666);
char buf[999];
sprintf(buf, "%d\n", getpid());
write(tmp_2, buf, strlen(buf));

if (pid > 0){
        wait(&status);
}
close(tmp_2);
```

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

(3) 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:

```
[10/11/20]seed@VM:~/.../bonus$ ./myfork hangup normal8 trap
------------CHILD PROCESS START------------
This is the SIGTRAP program

This is normal8 program
------------CHILD PROCESS START------------
This is the SIGHUP program

the process tree:18906->18908->18909->18910
```

4 The environment of running my program:

(1) The version of OS:

```
[10/11/20]seed@VM:~/.../bonus$ uname -a
Linux VM 4.10.14 #2 SMP Sat Oct 10 04:25:40 EDT 2020 i686 i686 i686 GNU/Linux
```

(2) The version of kernel:

```
[10/11/20]seed@VM:~/.../bonus$ uname -r
4.10.14
```

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.