CSC3150 Homework1 Report

Song Wenxin 120090625

1. Overview

This homework mainly contains two tasks. In the first task, the program will run a process in the user mode. First, the program will fork a child process to execute the test program. The parent process will wait until child process terminates. After the child process is finished, the parent process will receive the SIGCHLD signal by wait() function. Then the program will print the termination or the error message in the terminal. The main flow chart can be expressed as Figure 1. In the second task, we need to implement my_fork(), my_wait(), my_exec() functions and so on by ourselves. It first creates a kernel thread and then run my_fork() function. In the my_fork() function, it will execue the test program. Within this test program, it will raise signal. The parent process will wait until child process terminates. After the child process is finished. The signal will be caught by the my_wait() function. Similar to task 1, related message will be printed out in kernel log. The main flow chart can be expressed as figure 2.

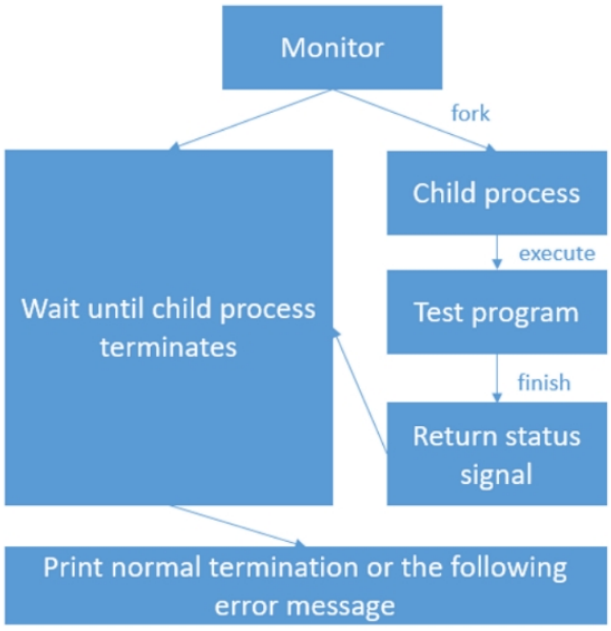

Figure 1

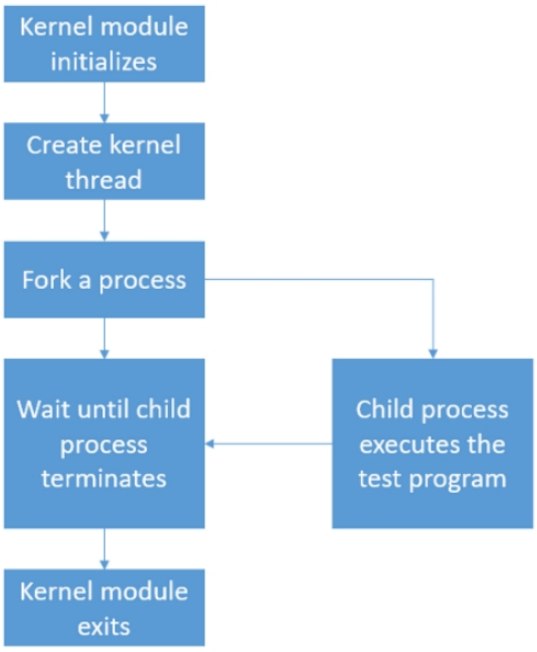

Figure 2

2. Environment Set up

We will use the following environment setting:

Linux Distribution: `Ubuntu 5.4.0-6ubuntu1~16.04.12`

Linux Kernel Version:

```
root@csc3150:/# uname -r
5.10.99
```

GCC Version:

```
gcc version 5.4.0 20160609
```

This homework requires us to use linux kernel with version newer or equal to 5.10.x. However, the initial

version does not satisfy the requirements. Thus before we compile the kernel, we first need to download a newer version. First, we need to clear the previous setting. Then we need to decompress the folder and copy the original config file to the decompressed folder. Now we can start reconfiguration. Then, before building the image, we should go to some files to export some functions, such as “kernel_clone” (fork.c) , “do_execve” (exec.c), “getname_kernel” (namei.c) and “do_wait”(exit.c), we need to find their corresponding files in the decompressed file and export them by using EXPORT_SYMBOL. Or we can not use them in our program. After some building work, installing work and rebooting to load new kernel, we now have the kernel with the version of 5.10.99.

3. **Design and Explanation**

3.1 Program 1

3.1.1 Basic Ideas

In the first task, the program will run a process in the user mode. First, the program will fork a child process to execute the test program. The parent process will wait until child process terminates. After the child process is finished, the parent process will receive the SIGCHLD signal by wait() function. Then the program will print the termination or the error message in the terminal.

3.1.2 Implementation

Firstly, it will call the fork() to create a child process. Then, if the pid is equal to 0, then it means it is the child process. Then it will first set the arguments for function execve() and then call it to execute the test program. If pid is equal to -1, it means that the fork process is in trouble. Then it will exit with signal 1. Else, the it is in parent process and it will wait until the child process terminates by the waitpid() function. It will also print the output to the terminal.

3.1.3 Program output for Program 1

1. abort

```
root@csc3150:/home/vagrant/csc3150/source/program1# ./program1 ./abort
Process start to fork
I'm the Parent Process, my pid = 6189
I'm the Child Process, my pid = 6190
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGABRT program

Parent process receives SIGCHLD signal
child process get SIGABRT signal
```

2. alarm

```
root@csc3150:/home/vagrant/csc3150/source/program1# ./program1 ./alarm
Process start to fork
I'm the Parent Process, my pid = 6255
I'm the Child Process, my pid = 6256
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGALRM program

Parent process receives SIGCHLD signal
child process get SIGALRM signal
```

3. bus

```
root@csc3150:/home/vagrant/csc3150/source/program1# ./program1 ./bus
Process start to fork
I'm the Parent Process, my pid = 6345
I'm the Child Process, my pid = 6346
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGBUS program

Parent process receives SIGCHLD signal
child process get SIGBUS signal
```

4. floating

```
root@csc3150:/home/vagrant/csc3150/source/program1# ./program1 ./floating
Process start to fork
I'm the Parent Process, my pid = 6453
I'm the Child Process, my pid = 6454
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGFPE program

Parent process receives SIGCHLD signal
child process get SIGFPE signal
```

5. hangup

```
root@csc3150:/home/vagrant/csc3150/source/program1# ./program1 ./hangup
Process start to fork
I'm the Parent Process, my pid = 6498
I'm the Child Process, my pid = 6499
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGHUP program

Parent process receives SIGCHLD signal
child process get SIGHUP signal
```

6. Illegal_instr

```
root@csc3150:/home/vagrant/csc3150/source/program1# ./program1 ./illegal_instr
Process start to fork
I'm the Parent Process, my pid = 6661
I'm the Child Process, my pid = 6662
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGILL program

Parent process receives SIGCHLD signal
child process get SIGILL signal
```

7. interrupt

```
root@csc3150:/home/vagrant/csc3150/source/program1# ./program1 ./interrupt
Process start to fork
I'm the Parent Process, my pid = 6716
I'm the Child Process, my pid = 6717
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGINT program

Parent process receives SIGCHLD signal
child process get SIGINT signal
```

8. kill

```
root@csc3150:/home/vagrant/csc3150/source/program1# ./program1 ./kill
Process start to fork
I'm the Parent Process, my pid = 6760
I'm the Child Process, my pid = 6761
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGKILL program

Parent process receives SIGCHLD signal
child process get SIGKILL signal
```

9. normal

```
root@csc3150:/home/vagrant/csc3150/source/program1# ./program1 ./normal
Process start to fork
I'm the Parent Process, my pid = 6846
I'm the Child Process, my pid = 6847
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the normal program

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

10. pipe

```
root@csc3150:/home/vagrant/csc3150/source/program1# ./program1 ./pipe
Process start to fork
I'm the Parent Process, my pid = 6890
I'm the Child Process, my pid = 6891
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGPIPE program

Parent process receives SIGCHLD signal
child process get SIGPIPE signal
```

11. quit

```
root@csc3150:/home/vagrant/csc3150/source/program1# ./program1 ./quit
Process start to fork
I'm the Parent Process, my pid = 6934
I'm the Child Process, my pid = 6935
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGQUIT program

Parent process receives SIGCHLD signal
child process get SIGQUIT signal
```

12. segment_fault

```
root@csc3150:/home/vagrant/csc3150/source/program1# ./program1 ./segment_fault
Process start to fork
I'm the Parent Process, my pid = 6979
I'm the Child Process, my pid = 6980
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGSEGV program

Parent process receives SIGCHLD signal
child process get SIGSEGV signal
```

13. stop

```
root@csc3150:/home/vagrant/csc3150/source/program1# ./program1 ./stop
Process start to fork
I'm the Parent Process, my pid = 7024
I'm the Child Process, my pid = 7025
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGSTOP program

Parent process receives SIGCHLD signal
child process get SIGSTOP signal
```

14. terminate

```
root@csc3150:/home/vagrant/csc3150/source/program1# ./program1 ./terminate
Process start to fork
I'm the Parent Process, my pid = 7101
I'm the Child Process, my pid = 7102
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGTERM program

Parent process receives SIGCHLD signal
child process get SIGTERM signal
```

15. trap

```
root@csc3150:/home/vagrant/csc3150/source/program1# ./program1 ./trap
Process start to fork
I'm the Parent Process, my pid = 7166
I'm the Child Process, my pid = 7167
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGTRAP program

Parent process receives SIGCHLD signal
child process get SIGTRAP signal
```

3.2 Program 2

3.2.1 Basic Ideas

First, it creates a kernel thread. In the thread, it will call my_fork() function. In the my_fork() function, it will fork a child process and execue the test program. Within this test program, it will raise signal. The parent process will wait until child process terminates. After the child process is finished, the signal will be caught by the my_wait() function. Related message will be printed out in kernel log.

3.2.2 Function Specification

void my_print(int status): It will print out related information in the kernel log based on different status. Or example, if the status is 14, it will print out "[program2] : get SIGALARM signal".

int my_exec(void): It will first set the path of the test program. Then it call the getname(kernel) to get the file name. Later, it call the do_execve() to run the test program.

int my_wait(pid_t pid): First, it will creat an instance for struct wait_opts. Then it will set the value for each attributes of the instance. Then it will pass this instance to function do_wait() as an parameter. It will return the signal of the child process.

int my_fork(void *argc): First, it will create a kernel_clone_args instance and initialize the attributes of it. The function my_exec() is one of its attribute. Then it will pass it to kernel_clone() as a parameter. Later, it will call my_wait() to wait until the child process terminates. At last, it will call my_print() to output related information to the kernel log.

static int __init program2_init(void): The whole program starts from here. It will create a kernel thread and call the my_fork() function.

3.2.3 Implementation

First, the program will run the module_init(). It will create a thread an run my_fork(). my_fork() will create a kernel_clone_args instance and initialize the attributes of it. The function my_exec() is one of its attribute. Then it will pass it to kernel_clone() as a parameter. Later, it will call my_wait() to wait until the child process terminates. At last, it will call my_print() to output related information to the kernel log.

3.2.4 Program output for Program 2

1. abort

```
[31481.327149] [program2] : module_init {Song Wenxin} {120090625}
[31481.346089] [program2] : module_init create kthread start
[31481.346366] [program2] : module_init kthread start
[31481.367020] [program2] : The child process has pid = 31982
[31481.401300] [program2] : This is the parent process, pid = 31981
[31481.421876] [program2] : child process
[31481.486818] [program2] : get SIGABRT signal
[31481.501897] [program2] : child process has abort error
[31481.519740] [program2] : The return signal is 6
[31482.965270] [program2] : module_exit./6
```

2. alarm

```
[31647.889746] [program2] : module_init {Song Wenxin} {120090625}
[31647.914122] [program2] : module_init create kthread start
[31647.915001] [program2] : module_init kthread start
[31647.938613] [program2] : The child process has pid = 748
[31647.970661] [program2] : This is the parent process, pid = 747
[31647.986478] [program2] : child process
[31649.964708] [program2] : get SIGALARM signal
[31649.976036] [program2] : child process has alarm error
[31649.991338] [program2] : The return signal is 14
[31652.506994] [program2] : module_exit./6
```

3. bus

```
[31427.798399] [program2] : module_init {Song Wenxin} {120090625}
[31427.821599] [program2] : module_init create kthread start
[31427.821894] [program2] : module_init kthread start
[31427.840091] [program2] : The child process has pid = 31556
[31427.876860] [program2] : This is the parent process, pid = 31555
[31427.897372] [program2] : child process
[31427.955122] [program2] : get SIGBUS signal
[31427.969093] [program2] : child process has bus error
[31427.985684] [program2] : The return signal is 7
[31428.993539] [program2] : module_exit./6
```

4. floating

```
[31393.702996] [program2] : module_init {Song Wenxin} {120090625}
[31393.723047] [program2] : module_init create kthread start
[31393.723534] [program2] : module_init kthread start
[31393.743513] [program2] : The child process has pid = 31155
[31393.776599] [program2] : This is the parent process, pid = 31154
[31393.796311] [program2] : child process
[31393.854667] [program2] : get SIGFPE signal
[31393.869660] [program2] : child process has float error
[31393.890884] [program2] : The return signal is 8
[31395.238148] [program2] : module_exit./6
```

5. hangup

```
[30317.514715] [program2] : module_init {Song Wenxin} {120090625}
[30317.533557] [program2] : module_init create kthread start
[30317.534302] [program2] : module_init kthread start
[30317.554247] [program2] : The child process has pid = 23362
[30317.583509] [program2] : This is the parent process, pid = 23361
[30317.599499] [program2] : child process
[30317.609793] [program2] : get SIGHUP signal
[30317.621617] [program2] : child process is hung up
[30317.634325] [program2] : The return signal is 1
[30318.959586] [program2] : module_exit./6
```

6. illegal_instr

```
[30383.766588] [program2] : module_init {Song Wenxin} {120090625}
[30383.785747] [program2] : module_init create kthread start
[30383.786340] [program2] : module_init kthread start
[30383.804859] [program2] : The child process has pid = 23799
[30383.840830] [program2] : This is the parent process, pid = 23798
[30383.860904] [program2] : child process
[30383.927608] [program2] : get SIGILL signal
[30383.942392] [program2] : child process has illegal instruction error
[30383.963557] [program2] : The return signal is 4
[30389.669380] [program2] : module_exit./6
```

7. interrupt

```
[30413.695628] [program2] : module_init {Song Wenxin} {120090625}
[30413.722714] [program2] : module_init create kthread start
[30413.723092] [program2] : module_init kthread start
[30413.740746] [program2] : The child process has pid = 24303
[30413.775349] [program2] : This is the parent process, pid = 24302
[30413.794529] [program2] : child process
[30413.806865] [program2] : get SIGINT signal
[30413.820611] [program2] : terminal interrupt
[30413.838911] [program2] : The return signal is 2
[30415.125347] [program2] : module_exit./6
```

8. kill

```
[ 3460.493540] [program2] : Module_init {Song Wenxin} {120090625}
[ 3460.510031] [program2] : Module_init create kthread start
[ 3460.524921] [program2] : Module_init kthread start
[ 3460.537448] [program2] : The child process has pid = 22908
[ 3460.537448] [program2] : This is the parent process, pid =  22907
[ 3460.556322] [program2] : child process
[ 3460.629011] [program2] : get SIGKILL signal
[ 3460.644691] [program2] : child process terminated
[ 3460.644691] [program2] : The return signal is 9
[ 3462.411408] [program2] : Module_exit
```

9. normal

```
[30462.697817] [program2] : module_init {Song Wenxin} {120090625}
[30462.718281] [program2] : module_init create kthread start
[30462.718926] [program2] : module_init kthread start
[30462.736787] [program2] : The child process has pid = 25156
[30462.769977] [program2] : This is the parent process, pid = 25155
[30462.789304] [program2] : child process
[30462.802034] [program2] : child process exit normally
[30462.819756] [program2] : The return signal is 0
[30464.145005] [program2] : module_exit./6
```

10. pipe

```
[30492.016861] [program2] : module_init {Song Wenxin} {120090625}
[30492.040189] [program2] : module_init create kthread start
[30492.041092] [program2] : module_init kthread start
[30492.059516] [program2] : The child process has pid = 25580
[30492.088353] [program2] : This is the parent process, pid = 25579
[30492.105872] [program2] : child process
[30492.116244] [program2] : get SIGPIPE signal
[30492.127853] [program2] : child process has pipe error
[30492.143202] [program2] : The return signal is 13
[30493.141557] [program2] : module_exit./6
```

11. quit

```
[30510.016435] [program2] : module_init {Song Wenxin} {120090625}
[30510.035502] [program2] : module_init create kthread start
[30510.036191] [program2] : module_init kthread start
[30510.056143] [program2] : The child process has pid = 25969
[30510.086416] [program2] : This is the parent process, pid = 25967
[30510.102645] [program2] : child process
[30510.167945] [program2] : get SIGQUIT signal
[30510.178960] [program2] : terminal quit
[30510.189408] [program2] : The return signal is 3
[30511.229154] [program2] : module_exit./6
```

12. segment_fault

```
[30533.198646] [program2] : module_init {Song Wenxin} {120090625}
[30533.225679] [program2] : module_init create kthread start
[30533.226075] [program2] : module_init kthread start
[30533.245096] [program2] : The child process has pid = 26359
[30533.279260] [program2] : This is the parent process, pid = 26358
[30533.298919] [program2] : child process
[30533.369826] [program2] : get SIGSEGV signal
[30533.383485] [program2] : child process has segmentation fault error
[30533.404781] [program2] : The return signal is 11
[30534.502398] [program2] : module_exit./6
```

13. stop

```
[31020.444740] [program2] : module_init {Song Wenxin} {120090625}
[31020.475028] [program2] : module_init create kthread start
[31020.475608] [program2] : module_init kthread start
[31020.492339] [program2] : The child process has pid = 30335
[31020.527643] [program2] : This is the parent process, pid = 30334
[31020.547258] [program2] : child process
[31020.559873] [program2] : get SIGSTOP signal
[31020.574065] [program2] : child process stop
[31020.588400] [program2] : The return signal is 19
[31021.829354] [program2] : module_exit./6
```

14. terminate

```
[30591.981124] [program2] : module_init {Song Wenxin} {120090625}
[30592.004326] [program2] : module_init create kthread start
[30592.005032] [program2] : module_init kthread start
[30592.023619] [program2] : The child process has pid = 27116
[30592.057319] [program2] : This is the parent process, pid = 27115
[30592.077176] [program2] : child process
[30592.090121] [program2] : get SIGTERM signal
[30592.104224] [program2] : child process terminated
[30592.119194] [program2] : The return signal is 15
[30593.068966] [program2] : module_exit./6
```

15. trap

```
[30623.191245] [program2] : module_init {Song Wenxin} {120090625}
[30623.210515] [program2] : module_init create kthread start
[30623.211004] [program2] : module_init kthread start
[30623.232398] [program2] : The child process has pid = 27539
[30623.269263] [program2] : This is the parent process, pid = 27538
[30623.287840] [program2] : child process
[30623.344867] [program2] : get SIGTRAP signal
[30623.359053] [program2] : child process has trap error
[30623.375973] [program2] : The return signal is 5
[30625.268693] [program2] : module_exit./6
```

4. Feelings and Experiences

From this homework, I learn how to compile the kernel. When I am compiling the kernel, I meet a lot of

problems. And I gradually learn to think of and solve the problem by myself. Only if I can not find it on Google, I will ask others for help.

Second, I learnt how to fork a child process in user mode and kernel mode. I also have a general understanding for different return signals, such as SIGTOP, SIGABRT and SIGBUS.

Signals.

Last but not least, I also learn how to use many linux commands. For example, modifying the fork,c to export the function by “a” and “:wq” as well as using “sudo su” to get to the root account. This homework makes me realize that linux is so beautiful and I would like to learn more about it.