CSC3150
Operating System

Report for Assignment1

120090466

He XingJie

9th October, 2022

The Chinese University of Hong Kong, Shenzhen

Table of Contents

1 Design

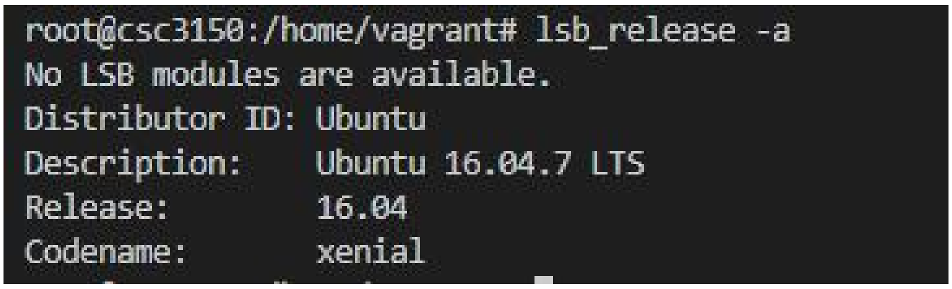
Assignment 1 requires us to do some programming about process in user mode and kernel mode. We need to set up the environment for the virtual machine, write the code to deal with process operations, compile the kernel and modify its source code.

- Task 1
  Task 1 aims to fork a process, execute the test program and print out the signal information. Since it is based on user mode, we can use the API like fork(), execve() and waitpid() conveniently. Following the guide o tutorial, I fork a process first, then for the child process, it will print out the information, get the name of file to be executed, and execute the file; for the parent process, it will wait until the child process has terminated. Receiving the status returned by child process, it will judge which signal it is(normal/sig/stop). Here I use switch-case to do the classification of the signal. The whole procedure of finishing task 1is clear and easy.
- Task 2
  Task 2 is similar with task 1 in procedure. The difference is that it is written in kernel mode, which means the implementation of fork, execute and wait is more underlying or basic. First a kernel thread is created, running the my_fork() function. My_fork function is used to fork a process(my_exec()) to execute the test program. In the my_exec() function, the executable file name is specified, and the kernel API function do_execve() is responsible for running the test file. This is what child process will do. For the parent process, it will wait for its child process to be terminated due to my_wait() function. In my_wait() function, the actual waiting action is implemented by the do_wait() function. The information of the child process's status is stored in wo.wo_start variable. Just like task 1, I used this variable's value to classify child process's signals, and then print out the corresponding information to kernel log. Finally, exit the module. (See more details in my code)
  As for the part beyong the code, we need to export the symbol of do_wait(), kernel_clone(), do_execve() and getname_kernel() functions to make use of them since they belong to the kernel API. After the modification, recompilation of the kernel is needed. The updated modules need to be installed. The difficulty of task 2 distributes a little to the environment setup part.
- Bonus
  In this task, the goal is to implement pstree function in linux.
- Some details
  1. The flag of wo (struct wait_opts) should be set to WEXITED | WUNTRACED to deal with the stop signal.

2. The classification of stop signal in task 2 requires wo_start to right shift 8 bits (>>8) or its return value is 4991 (because of the macro definition and bottom-line design).

2 Environment

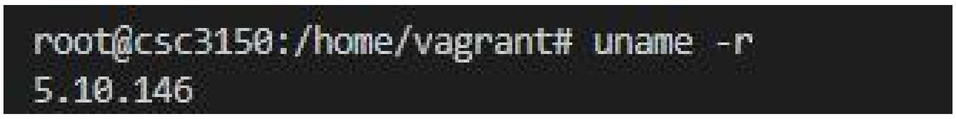
OS version: Linux Ubuntu 16.04



```
root@csc3150:/home/vagrant# lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 16.04.7 LTS
Release:        16.04
Codename:       xenial
```

Figure1 OS version

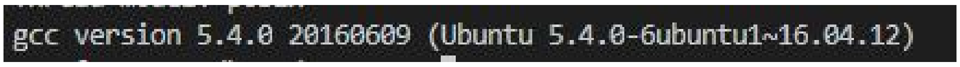
Kernel Version: 5.10.146



```
root@csc3150:/home/vagrant# uname -r
5.10.146
```

Figure2 Kernel version

GCC Version: 5.4.0



```
gcc version 5.4.0 20160609 (Ubuntu 5.4.0-6ubuntu1~16.04.12)
```

Figure3 GCC version

3 Excution

This part shows the execution steps of the program.

- Task 1
  1. Cd to the directory of program1 (This directory contains all the source codes and makefile)
  2. Type "make" in the terminal. (use "make clean" to clean and rebuild)
  3. Type "./program1 ./Test" (Test is the filename of the file to be executed. E.g. "./program1 ./normal"
- Task 2

  Before the execution, certain operations to kernel need to be done.
  1. Update the kernel source code (add EPORT_SYMBOL() for 4 functions

invoked)

2. Compile the kernel: sudo su; cd to kernel file; make bzImage; make modules; make modules_install; make install;
3. Reboot
4. Start to execute. Cd to the directory of program2 (This directory contains all the source codes and makefile)
5. Compile the test program: "gcc -o test test.c"
6. Type "make" ("make clean" can clean the rebuild files)
7. Type "sudo insmod program2.ko" to install the module.
8. Type "sudo rmmod program2" to remove the module.
9. Type "dmesg | grep program2" to check the kernel log.

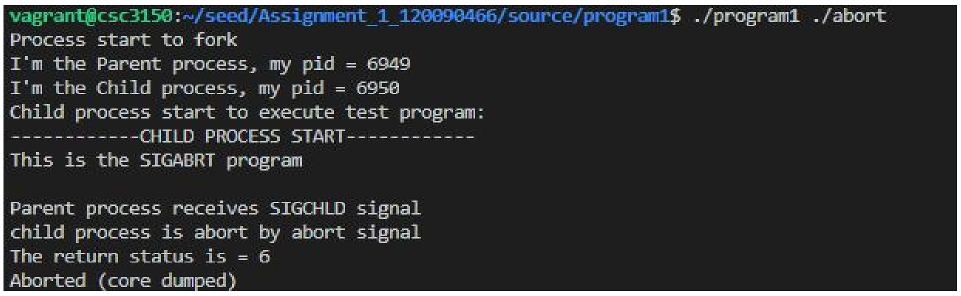
➢ Bonus
1. Cd to the directory of bonus.
2. Compile it with "gcc -o pstree pstree.c"
3. Type "./pstree"

4 Output

4.1 Task 1

Output is showed below:

1. Abort signal

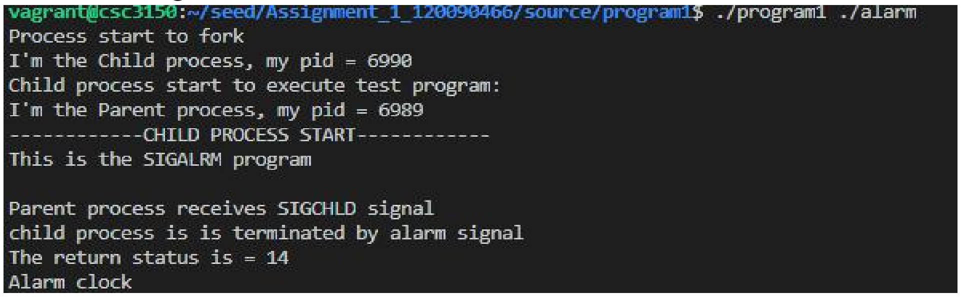


```
vagrant@csc3150:~/seed/Assignment_1_120090466/source/program1$ ./program1 ./abort
Process start to fork
I'm the Parent process, my pid = 6949
I'm the Child process, my pid = 6950
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGABRT program

Parent process receives SIGCHLD signal
child process is abort by abort signal
The return status is = 6
Aborted (core dumped)
```

Figure4 The output of testing abort.c

2. Alarm signal

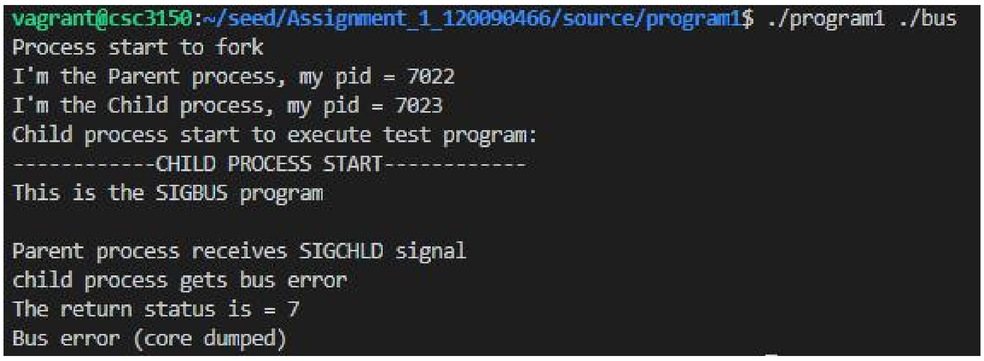


```
vagrant@csc3150:~/seed/Assignment_1_120090466/source/program1$ ./program1 ./alarm
Process start to fork
I'm the Child process, my pid = 6990
Child process start to execute test program:
I'm the Parent process, my pid = 6989
------------CHILD PROCESS START------------
This is the SIGALRM program

Parent process receives SIGCHLD signal
child process is is terminated by alarm signal
The return status is = 14
Alarm clock
```

Figure5 The output of testing alarm.c

3. Bus signal

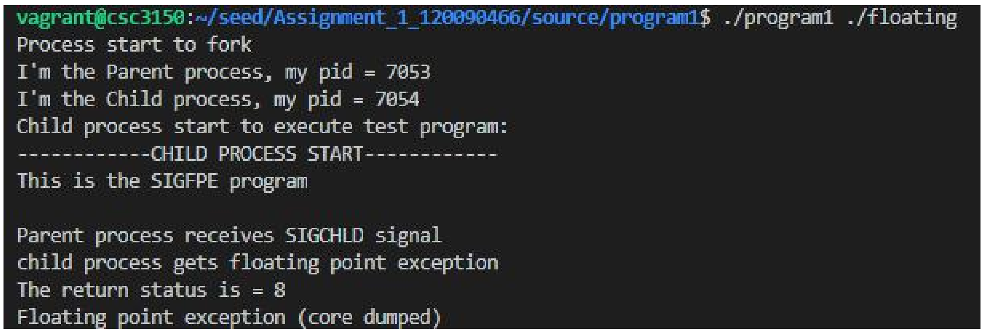


```
vagrant@csc3150:~/seed/Assignment_1_120090466/source/program1$ ./program1 ./bus
Process start to fork
I'm the Parent process, my pid = 7022
I'm the Child process, my pid = 7023
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGBUS program

Parent process receives SIGCHLD signal
child process gets bus error
The return status is = 7
Bus error (core dumped)
```

Figure6 The output of testing bus.c

4. Floating signal

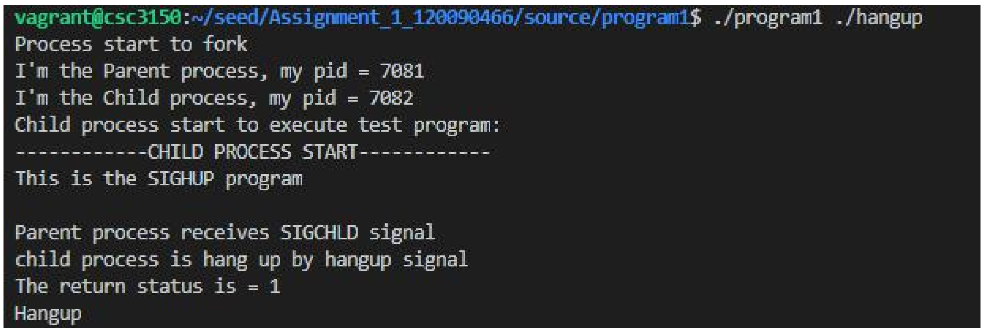


```
vagrant@csc3150:~/seed/Assignment_1_120090466/source/program1$ ./program1 ./floating
Process start to fork
I'm the Parent process, my pid = 7053
I'm the Child process, my pid = 7054
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGFPE program

Parent process receives SIGCHLD signal
child process gets floating point exception
The return status is = 8
Floating point exception (core dumped)
```

Figure7 The output of testing floating.c

5. Hangup signal

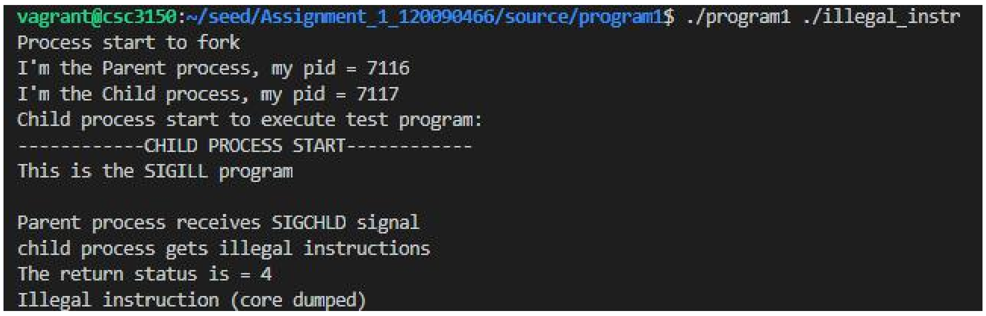


```
vagrant@csc3150:~/seed/Assignment_1_120090466/source/program1$ ./program1 ./hangup
Process start to fork
I'm the Parent process, my pid = 7081
I'm the Child process, my pid = 7082
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGHUP program

Parent process receives SIGCHLD signal
child process is hang up by hangup signal
The return status is = 1
Hangup
```

Figure8 The output of testing hangup.c

6. Illegal_instr siganl

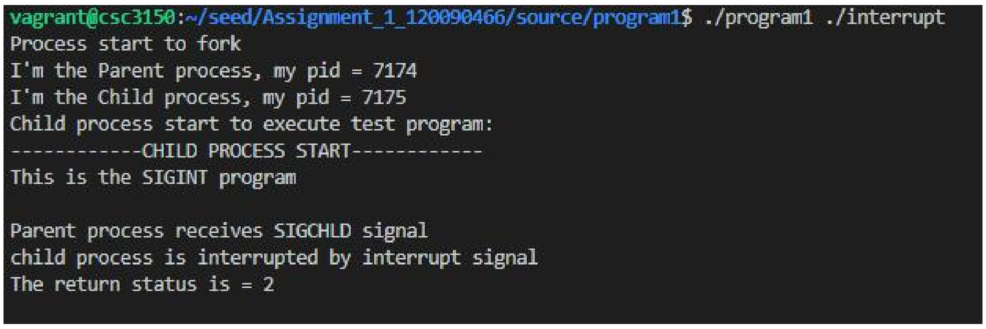


```
vagrant@csc3150:~/seed/Assignment_1_120090466/source/program1$ ./program1 ./illegal_instr
Process start to fork
I'm the Parent process, my pid = 7116
I'm the Child process, my pid = 7117
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGILL program

Parent process receives SIGCHLD signal
child process gets illegal instructions
The return status is = 4
Illegal instruction (core dumped)
```

Figure9 The output of testing illegal_instr.c

7. Interrupt siganl

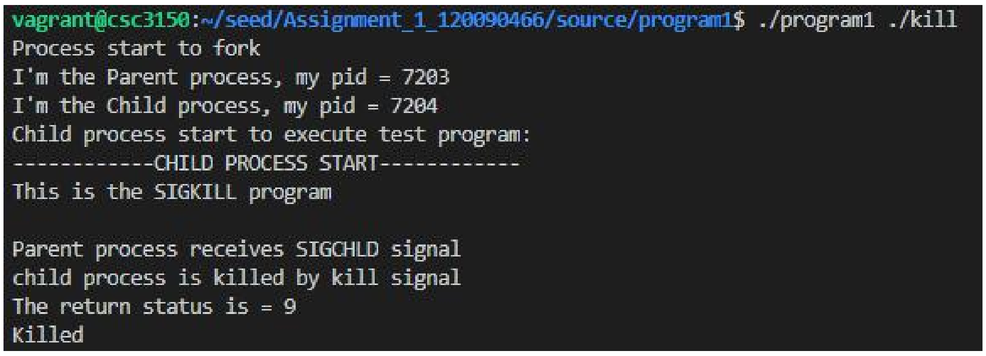


```
vagrant@csc3150:~/seed/Assignment_1_120090466/source/program1$ ./program1 ./interrupt
Process start to fork
I'm the Parent process, my pid = 7174
I'm the Child process, my pid = 7175
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGINT program

Parent process receives SIGCHLD signal
child process is interrupted by interrupt signal
The return status is = 2
```

Figure10 The output of testing interrupt.c

8. Kill signal

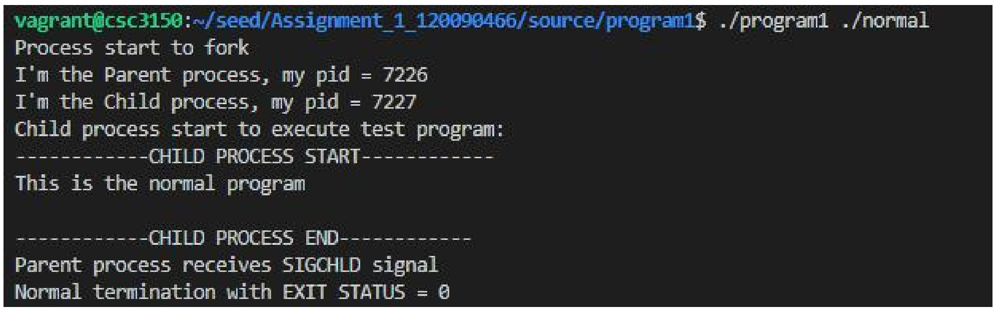


```
vagrant@csc3150:~/seed/Assignment_1_120090466/source/program1$ ./program1 ./kill
Process start to fork
I'm the Parent process, my pid = 7203
I'm the Child process, my pid = 7204
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGKILL program

Parent process receives SIGCHLD signal
child process is killed by kill signal
The return status is = 9
Killed
```

Figure11 The output of testing kill.c

9. Normal execution

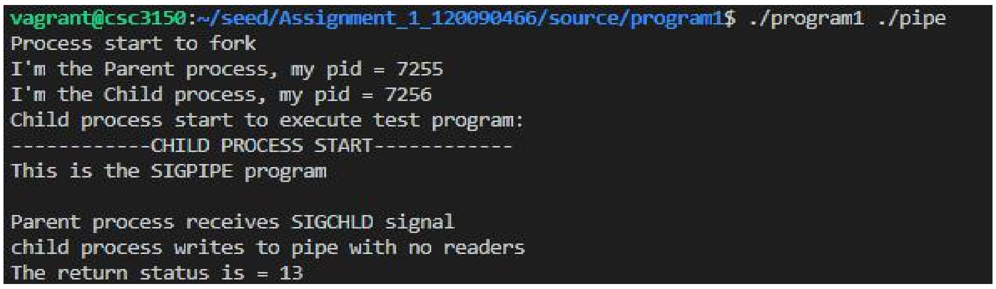


```
vagrant@csc3150:~/seed/Assignment_1_120090466/source/program1$ ./program1 ./normal
Process start to fork
I'm the Parent process, my pid = 7226
I'm the Child process, my pid = 7227
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the normal program

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

Figure12 The output of testing normal.c

10. Pipe signal

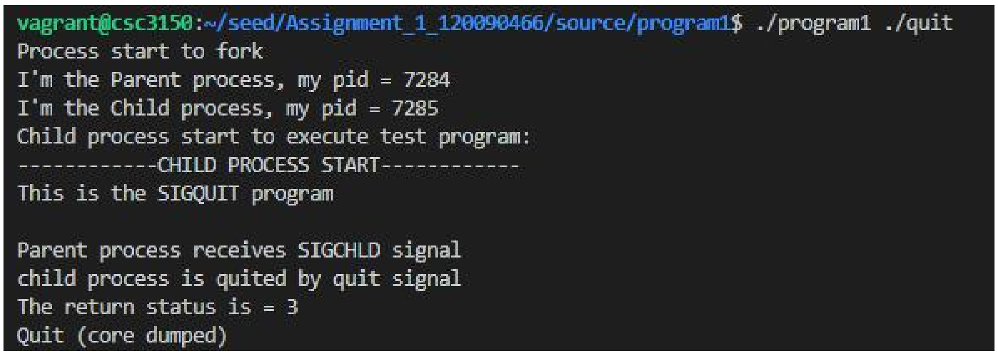


```
vagrant@csc3150:~/seed/Assignment_1_120090466/source/program1$ ./program1 ./pipe
Process start to fork
I'm the Parent process, my pid = 7255
I'm the Child process, my pid = 7256
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGPIPE program

Parent process receives SIGCHLD signal
child process writes to pipe with no readers
The return status is = 13
```

Figure13 The output of testing pipe.c

11. Quit signal

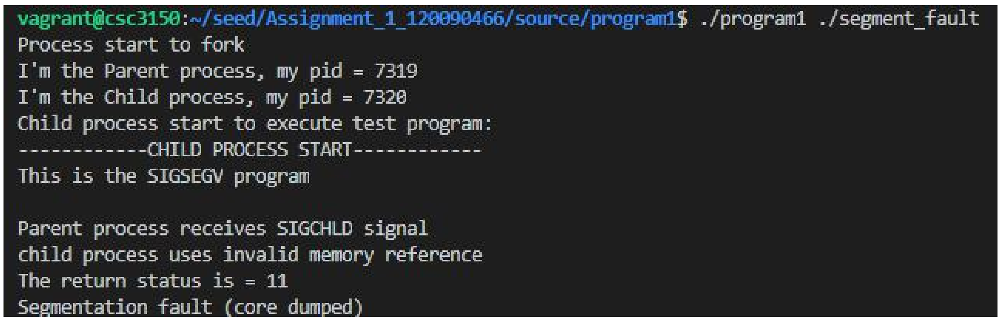


```
vagrant@csc3150:~/seed/Assignment_1_120090466/source/program1$ ./program1 ./quit
Process start to fork
I'm the Parent process, my pid = 7284
I'm the Child process, my pid = 7285
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGQUIT program

Parent process receives SIGCHLD signal
child process is quited by quit signal
The return status is = 3
Quit (core dumped)
```

Figure14 The output of testing quit.c

12. Segment_fault signal

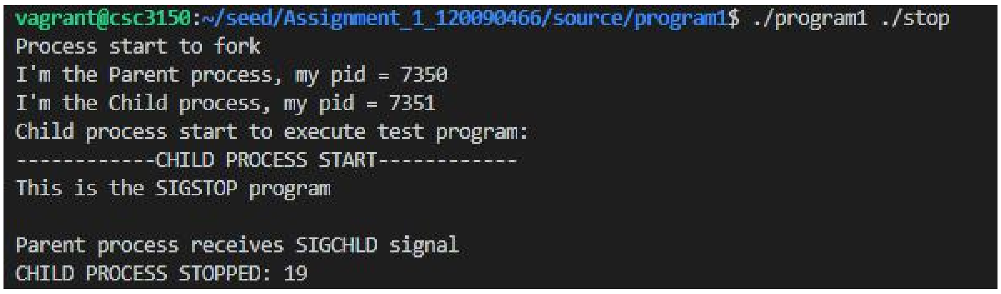


```
vagrant@csc3150:~/seed/Assignment_1_120090466/source/program1$ ./program1 ./segment_fault
Process start to fork
I'm the Parent process, my pid = 7319
I'm the Child process, my pid = 7320
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGSEGV program

Parent process receives SIGCHLD signal
child process uses invalid memory reference
The return status is = 11
Segmentation fault (core dumped)
```

Figure15 The output of testing segment_fault.c

13. Stop signal

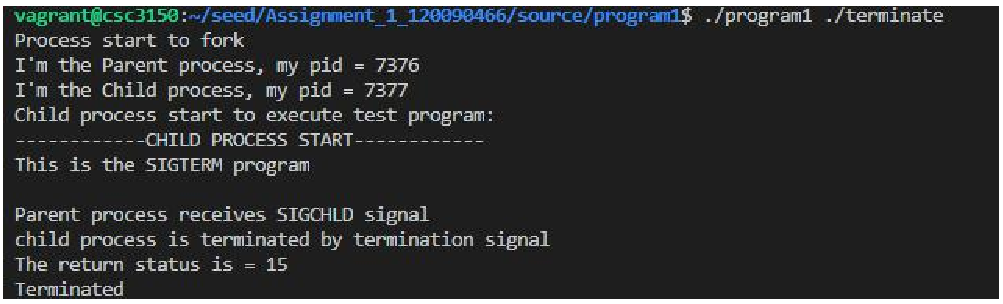


```
vagrant@csc3150:~/seed/Assignment_1_120090466/source/program1$ ./program1 ./stop
Process start to fork
I'm the Parent process, my pid = 7350
I'm the Child process, my pid = 7351
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGSTOP program

Parent process receives SIGCHLD signal
CHILD PROCESS STOPPED: 19
```

Figure16 The output of testing stop.c

14. Terminate signal

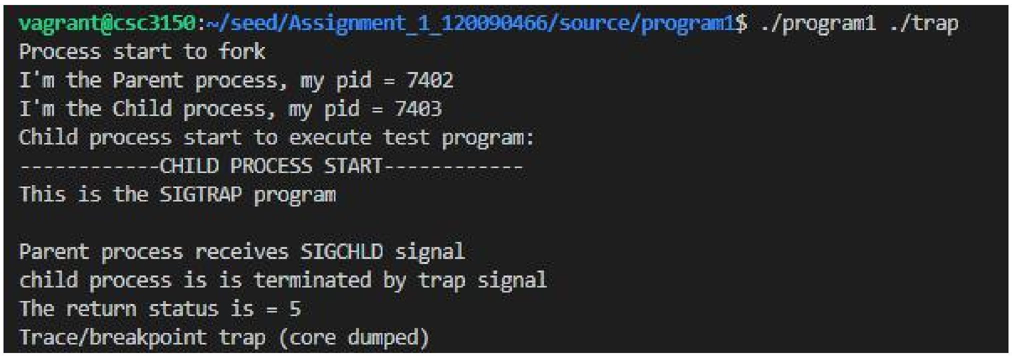


```
vagrant@csc3150:~/seed/Assignment_1_120090466/source/program1$ ./program1 ./terminate
Process start to fork
I'm the Parent process, my pid = 7376
I'm the Child process, my pid = 7377
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGTERM program

Parent process receives SIGCHLD signal
child process is terminated by termination signal
The return status is = 15
Terminated
```

Figure17 The output of testing terminate.c

15. Trap signal



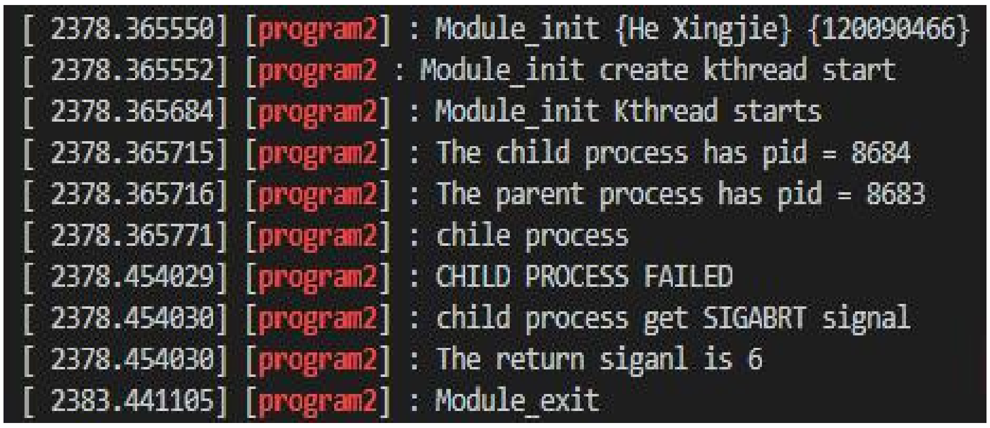
```
vagrant@csc3150:~/seed/Assignment_1_120090466/source/program1$ ./program1 ./trap
Process start to fork
I'm the Parent process, my pid = 7402
I'm the Child process, my pid = 7403
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGTRAP program

Parent process receives SIGCHLD signal
child process is is terminated by trap signal
The return status is = 5
Trace/breakpoint trap (core dumped)
```

Figure 18 The output of testing trap.c

4.2 Task 2

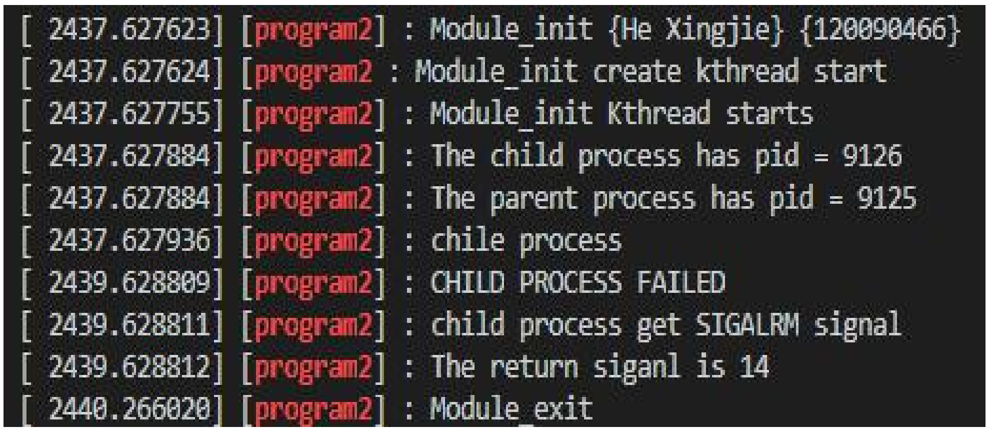
1. Execute abort signal.



```
[ 2378.365550] [program2] : Module_init {He Xingjie} {120090466}
[ 2378.365552] [program2 : Module_init create kthread start
[ 2378.365684] [program2] : Module_init Kthread starts
[ 2378.365715] [program2] : The child process has pid = 8684
[ 2378.365716] [program2] : The parent process has pid = 8683
[ 2378.365771] [program2] : chile process
[ 2378.454029] [program2] : CHILD PROCESS FAILED
[ 2378.454030] [program2] : child process get SIGABRT signal
[ 2378.454030] [program2] : The return siganl is 6
[ 2383.441105] [program2] : Module_exit
```

Figure 19 The output of testing abort signal

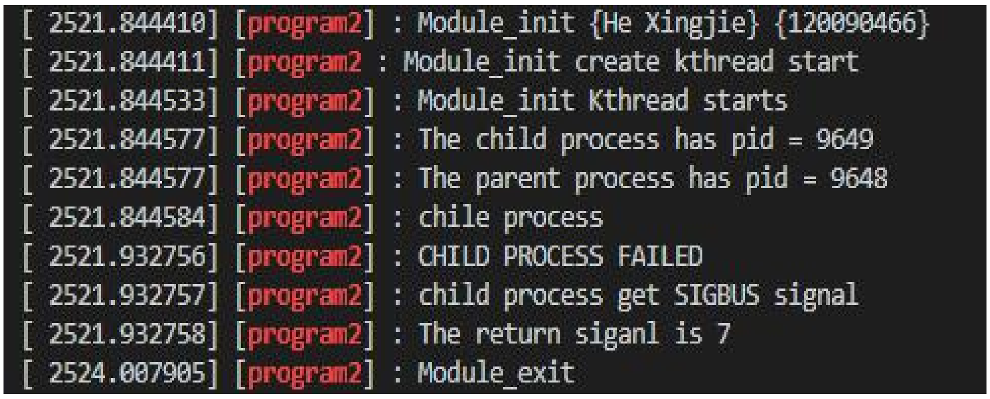
2. Execute alarm signal



```
[ 2437.627623] [program2] : Module_init {He Xingjie} {120090466}
[ 2437.627624] [program2 : Module_init create kthread start
[ 2437.627755] [program2] : Module_init Kthread starts
[ 2437.627884] [program2] : The child process has pid = 9126
[ 2437.627884] [program2] : The parent process has pid = 9125
[ 2437.627936] [program2] : chile process
[ 2439.628809] [program2] : CHILD PROCESS FAILED
[ 2439.628811] [program2] : child process get SIGALRM signal
[ 2439.628812] [program2] : The return siganl is 14
[ 2440.266020] [program2] : Module_exit
```

Figure 20 The output of testing alarm signal

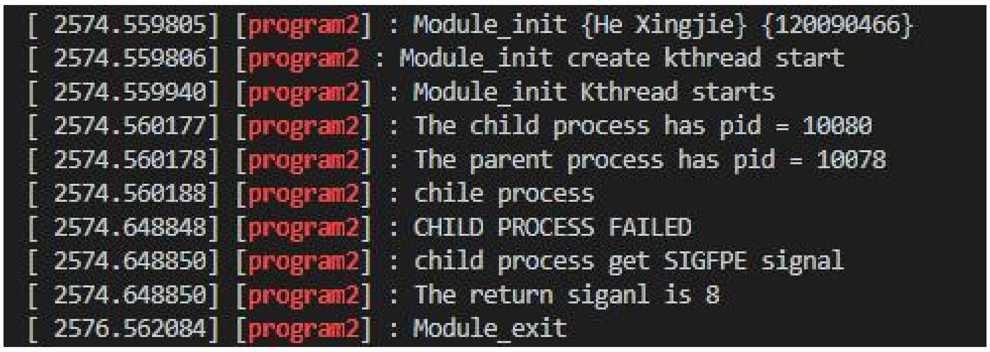
3. Execute bus signal



```
[ 2521.844410] [program2] : Module_init {He Xingjie} {120090466}
[ 2521.844411] [program2 : Module_init create kthread start
[ 2521.844533] [program2] : Module_init Kthread starts
[ 2521.844577] [program2] : The child process has pid = 9649
[ 2521.844577] [program2] : The parent process has pid = 9648
[ 2521.844584] [program2] : chile process
[ 2521.932756] [program2] : CHILD PROCESS FAILED
[ 2521.932757] [program2] : child process get SIGBUS signal
[ 2521.932758] [program2] : The return siganl is 7
[ 2524.007905] [program2] : Module_exit
```

Figure 21 The output of testing bus signal

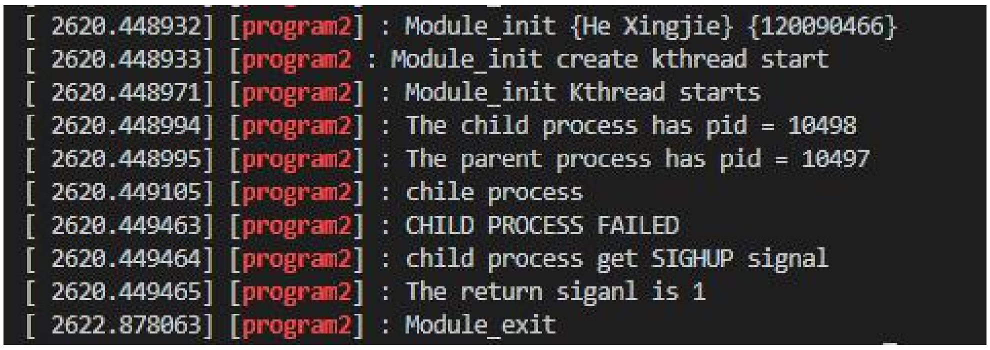
4. Execute floating signal



```
[ 2574.559805] [program2] : Module_init {He Xingjie} {120090466}
[ 2574.559806] [program2 : Module_init create kthread start
[ 2574.559940] [program2] : Module_init Kthread starts
[ 2574.560177] [program2] : The child process has pid = 10080
[ 2574.560178] [program2] : The parent process has pid = 10078
[ 2574.560188] [program2] : chile process
[ 2574.648848] [program2] : CHILD PROCESS FAILED
[ 2574.648850] [program2] : child process get SIGFPE signal
[ 2574.648850] [program2] : The return siganl is 8
[ 2576.562084] [program2] : Module_exit
```

Figure 22 The output of testing floating signal

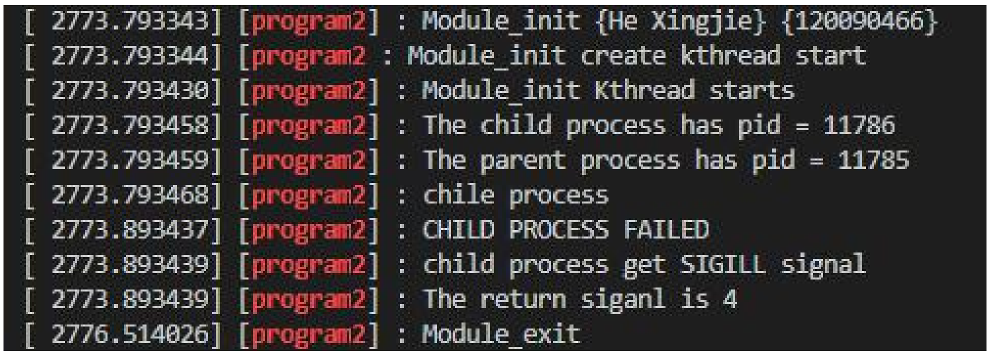
5. Execute hangup signal



```
[ 2620.448932] [program2] : Module_init {He Xingjie} {120090466}
[ 2620.448933] [program2 : Module_init create kthread start
[ 2620.448971] [program2] : Module_init Kthread starts
[ 2620.448994] [program2] : The child process has pid = 10498
[ 2620.448995] [program2] : The parent process has pid = 10497
[ 2620.449105] [program2] : chile process
[ 2620.449463] [program2] : CHILD PROCESS FAILED
[ 2620.449464] [program2] : child process get SIGHUP signal
[ 2620.449465] [program2] : The return siganl is 1
[ 2622.878063] [program2] : Module_exit
```

Figure 23 The output of testing hangup signal

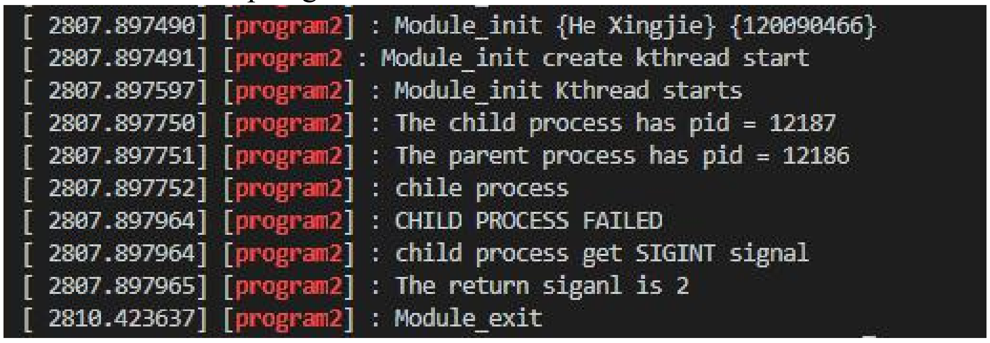
6. Execute illegal_instr signal



```
[ 2773.793343] [program2] : Module_init {He Xingjie} {120090466}
[ 2773.793344] [program2 : Module_init create kthread start
[ 2773.793430] [program2] : Module_init Kthread starts
[ 2773.793458] [program2] : The child process has pid = 11786
[ 2773.793459] [program2] : The parent process has pid = 11785
[ 2773.793468] [program2] : chile process
[ 2773.893437] [program2] : CHILD PROCESS FAILED
[ 2773.893439] [program2] : child process get SIGILL signal
[ 2773.893439] [program2] : The return siganl is 4
[ 2776.514026] [program2] : Module_exit
```

Figure 24 The output of testing illegal_instr signal

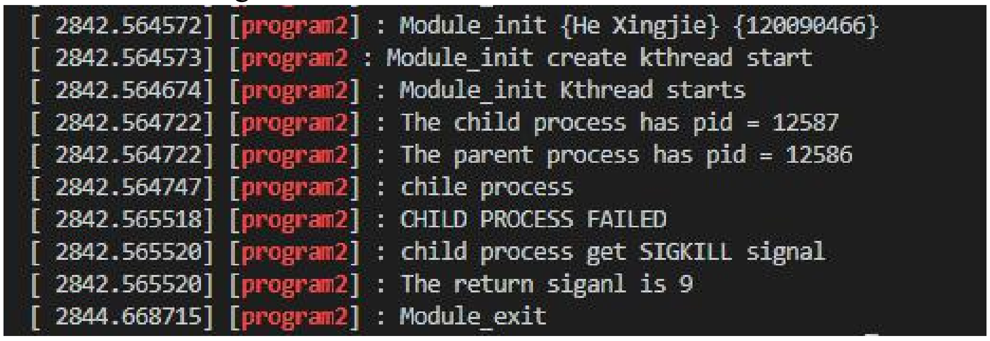
7. Execute interrupt signal



```
[ 2807.897490] [program2] : Module_init {He Xingjie} {120090466}
[ 2807.897491] [program2 : Module_init create kthread start
[ 2807.897597] [program2] : Module_init Kthread starts
[ 2807.897750] [program2] : The child process has pid = 12187
[ 2807.897751] [program2] : The parent process has pid = 12186
[ 2807.897752] [program2] : chile process
[ 2807.897964] [program2] : CHILD PROCESS FAILED
[ 2807.897964] [program2] : child process get SIGINT signal
[ 2807.897965] [program2] : The return siganl is 2
[ 2810.423637] [program2] : Module_exit
```

Figure 25 The output of testing interrupt signal

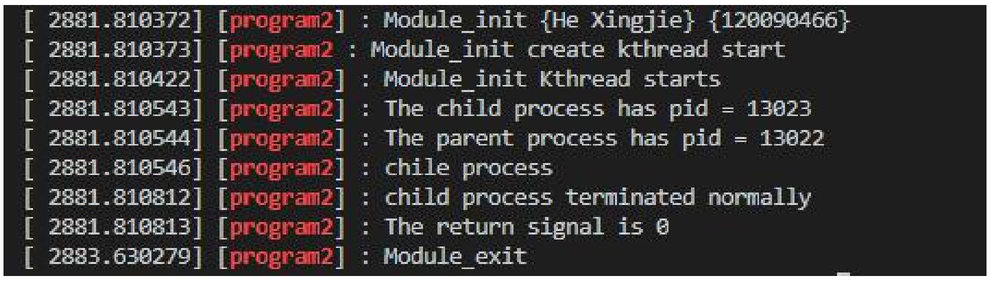
8. Execute kill signal



```
[ 2842.564572] [program2] : Module_init {He Xingjie} {120090466}
[ 2842.564573] [program2 : Module_init create kthread start
[ 2842.564674] [program2] : Module_init Kthread starts
[ 2842.564722] [program2] : The child process has pid = 12587
[ 2842.564722] [program2] : The parent process has pid = 12586
[ 2842.564747] [program2] : chile process
[ 2842.565518] [program2] : CHILD PROCESS FAILED
[ 2842.565520] [program2] : child process get SIGKILL signal
[ 2842.565520] [program2] : The return siganl is 9
[ 2844.668715] [program2] : Module_exit
```

Figure 26 The output of testing kill signal

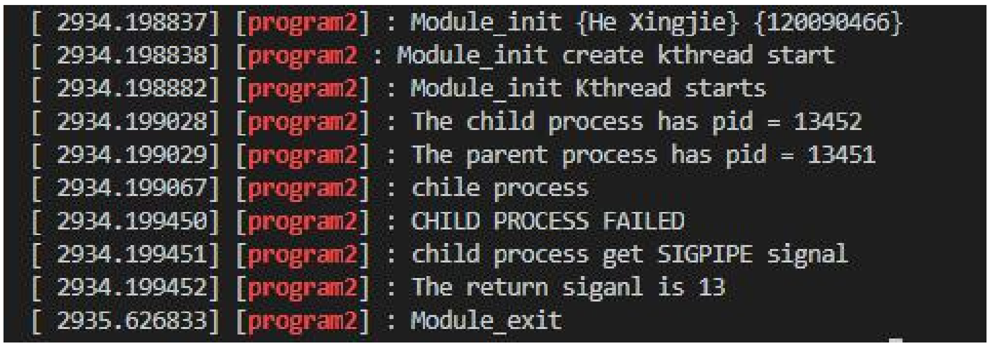
9. Execute norm termination signal



```
[ 2881.810372] [program2] : Module_init {He Xingjie} {120090466}
[ 2881.810373] [program2 : Module_init create kthread start
[ 2881.810422] [program2] : Module_init Kthread starts
[ 2881.810543] [program2] : The child process has pid = 13023
[ 2881.810544] [program2] : The parent process has pid = 13022
[ 2881.810546] [program2] : chile process
[ 2881.810812] [program2] : child process terminated normally
[ 2881.810813] [program2] : The return signal is 0
[ 2883.630279] [program2] : Module_exit
```

Figure 27 The output of testing norm termination signal

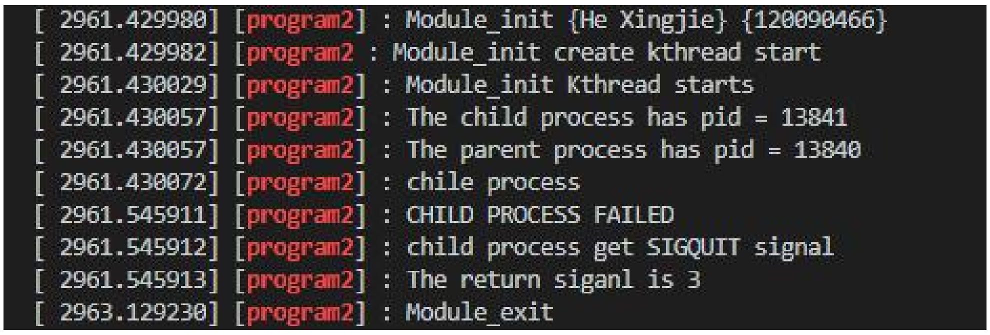
10. Execute pipe signal



```
[ 2934.198837] [program2] : Module_init {He Xingjie} {120090466}
[ 2934.198838] [program2 : Module_init create kthread start
[ 2934.198882] [program2] : Module_init Kthread starts
[ 2934.199028] [program2] : The child process has pid = 13452
[ 2934.199029] [program2] : The parent process has pid = 13451
[ 2934.199067] [program2] : chile process
[ 2934.199450] [program2] : CHILD PROCESS FAILED
[ 2934.199451] [program2] : child process get SIGPIPE signal
[ 2934.199452] [program2] : The return siganl is 13
[ 2935.626833] [program2] : Module_exit
```

Figure 28 The output of testing kill signal

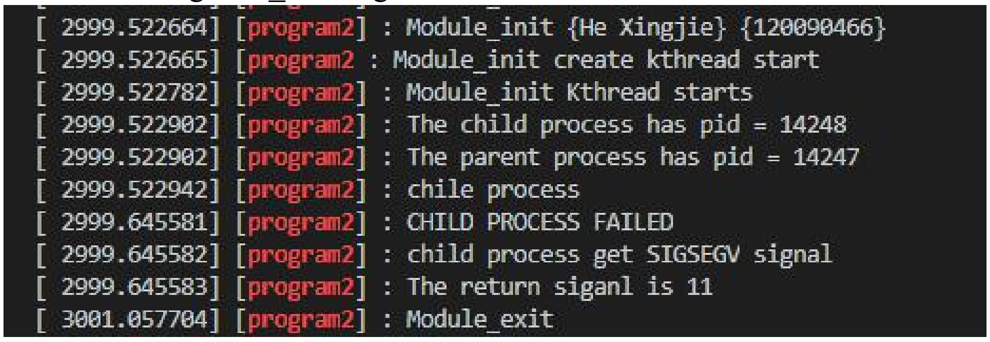
11. Execute quit signal



```
[ 2961.429980] [program2] : Module_init {He Xingjie} {120090466}
[ 2961.429982] [program2 : Module_init create kthread start
[ 2961.430029] [program2] : Module_init Kthread starts
[ 2961.430057] [program2] : The child process has pid = 13841
[ 2961.430057] [program2] : The parent process has pid = 13840
[ 2961.430072] [program2] : chile process
[ 2961.545911] [program2] : CHILD PROCESS FAILED
[ 2961.545912] [program2] : child process get SIGQUIT signal
[ 2961.545913] [program2] : The return siganl is 3
[ 2963.129230] [program2] : Module_exit
```

Figure 29 The output of testing quit signal

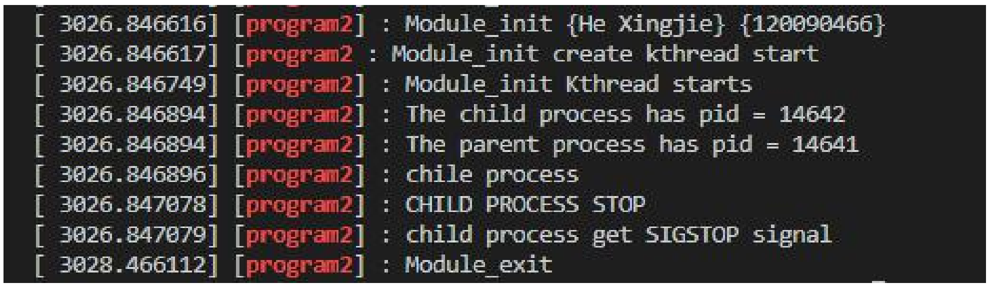
12. Execute segment_fault sigal



```
[ 2999.522664] [program2] : Module_init {He Xingjie} {120090466}
[ 2999.522665] [program2 : Module_init create kthread start
[ 2999.522782] [program2] : Module_init Kthread starts
[ 2999.522902] [program2] : The child process has pid = 14248
[ 2999.522902] [program2] : The parent process has pid = 14247
[ 2999.522942] [program2] : chile process
[ 2999.645581] [program2] : CHILD PROCESS FAILED
[ 2999.645582] [program2] : child process get SIGSEGV signal
[ 2999.645583] [program2] : The return siganl is 11
[ 3001.057704] [program2] : Module_exit
```

Figure 30 The output of testing segment_fault signal

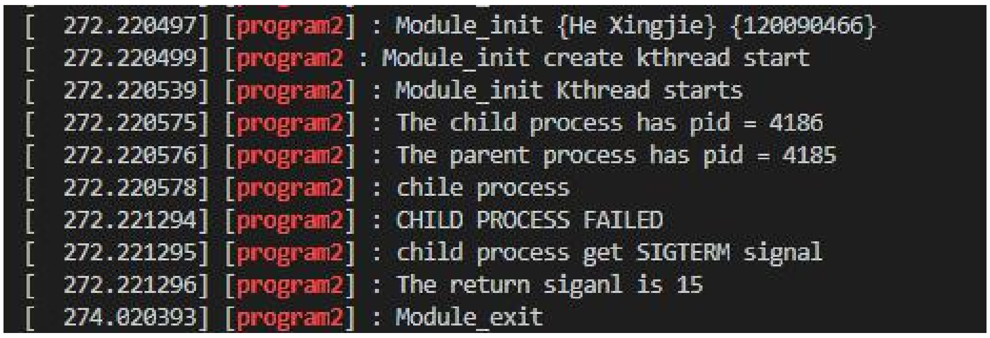
13. Execute stop signal



```
[ 3026.846616] [program2] : Module_init {He Xingjie} {120090466}
[ 3026.846617] [program2 : Module_init create kthread start
[ 3026.846749] [program2] : Module_init Kthread starts
[ 3026.846894] [program2] : The child process has pid = 14642
[ 3026.846894] [program2] : The parent process has pid = 14641
[ 3026.846896] [program2] : chile process
[ 3026.847078] [program2] : CHILD PROCESS STOP
[ 3026.847079] [program2] : child process get SIGSTOP signal
[ 3028.466112] [program2] : Module_exit
```

Figure 31 The output of testing stop signal

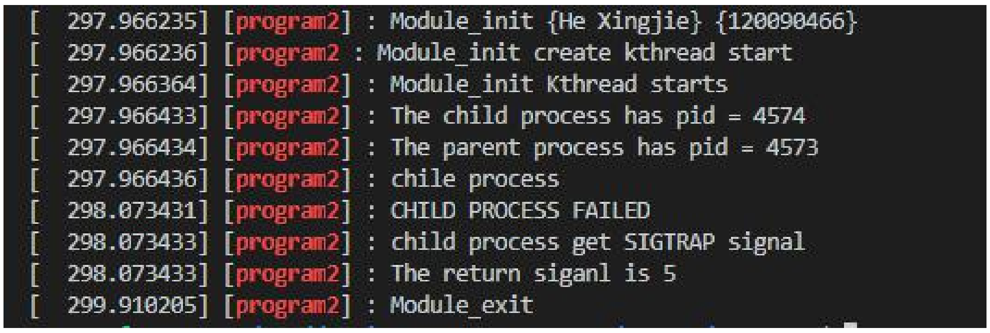
14. Execute termination signal



```
[  272.220497] [program2] : Module_init {He Xingjie} {120090466}
[  272.220499] [program2 : Module_init create kthread start
[  272.220539] [program2] : Module_init Kthread starts
[  272.220575] [program2] : The child process has pid = 4186
[  272.220576] [program2] : The parent process has pid = 4185
[  272.220578] [program2] : chile process
[  272.221294] [program2] : CHILD PROCESS FAILED
[  272.221295] [program2] : child process get SIGTERM signal
[  272.221296] [program2] : The return siganl is 15
[  274.020393] [program2] : Module_exit
```

Figure 32 The output of testing termination signal

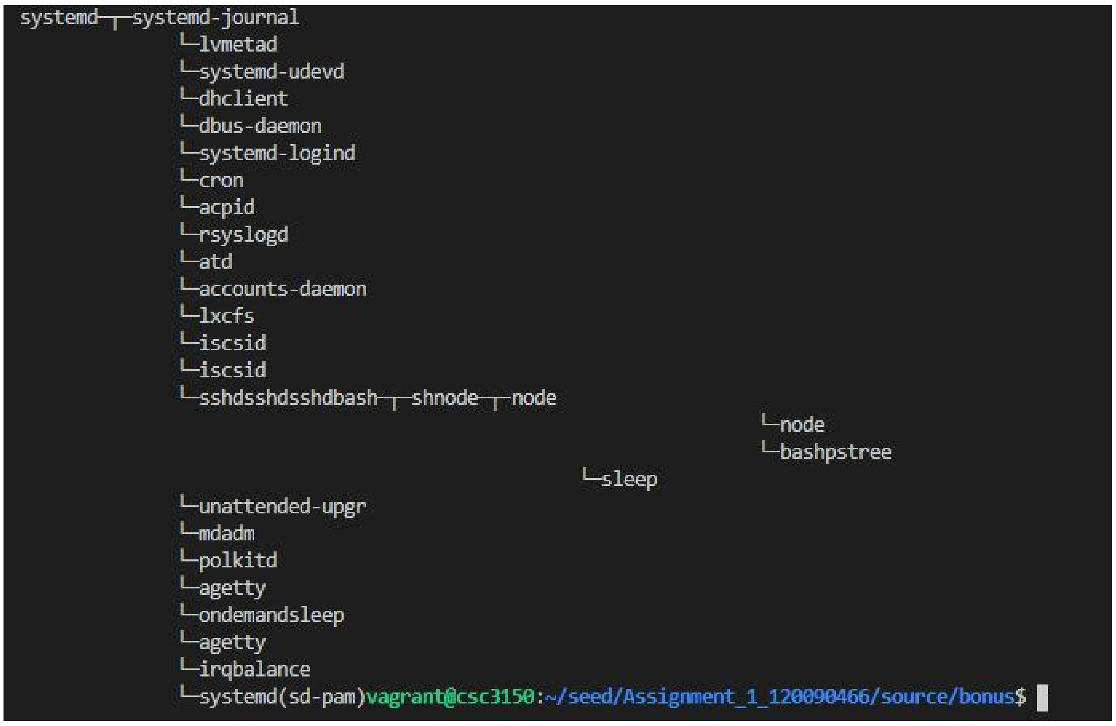
15. Execute trap signal



```
[  297.966235] [program2] : Module_init {He Xingjie} {120090466}
[  297.966236] [program2 : Module_init create kthread start
[  297.966364] [program2] : Module_init Kthread starts
[  297.966433] [program2] : The child process has pid = 4574
[  297.966434] [program2] : The parent process has pid = 4573
[  297.966436] [program2] : chile process
[  298.073431] [program2] : CHILD PROCESS FAILED
[  298.073433] [program2] : child process get SIGTRAP signal
[  298.073433] [program2] : The return siganl is 5
[  299.910205] [program2] : Module_exit
```

Figure 33 The output of testing trap signal

4.3 Bonus



```
systemd─┬─systemd-journal
              └─lvmetad
              └─systemd-udevd
              └─dhclient
              └─dbus-daemon
              └─systemd-logind
              └─cron
              └─acpid
              └─rsyslogd
              └─atd
              └─accounts-daemon
              └─lxcfs
              └─iscsid
              └─iscsid
              └─sshdsshdsshdbash─┬─shnode─┬─node
                                                └─node
                                                └─bashpstree
                                 └─sleep
              └─unattended-upgr
              └─mdadm
              └─polkitd
              └─agetty
              └─ondemandsleep
              └─agetty
              └─irqbalance
              └─systemd(sd-pam)vagrant@csc3150:~/seed/Assignment_1_120090466/source/bonus$
```

Figure 34 Bonus tree.

5 What I have learned from this assignment

CSC3150 is another hard course taught by professor Zhong. I think we must prepare well for the challenges.

Assignment 1 itself is not too difficult but there are a lot of problems about how to
set the environment and other issues. For the knowledge and coding part, we just need to understand some definitions about the process and kernel and imitate the codes

from tutorial and the source code. However, we need to set up the environment by ourselves during which there will be mountains of errors and bugs from the system. For me, I am not so familiar with the Linux system and need to search for the solutions of bugs from piazza and google. However, after doing this project I become more skillful in Linux system and virtual machine. I have learned some operations about process, how to compile and modify kernel, how to execute a program in the codes, and so on. It does improve my computer ability.

Discussion is quite important in handling the large project like this. We encounter different kinds of problems when finishing the tasks, and others may also meet the same trouble. As for the discussion platform, piazza is a very good platform for us to discuss our problems we have met during our assignment.

I can feel that TAs are hard-working and laborious for this course. They have large
workloads and will be disturbed by mountains of questions raised by students.
GOOD LUCK TO YOU!