



香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Report for project1

Zhao Yang

120090865@link.cuhk.edu.cn

Contents

1 Overview	3
2 Environment	3
3 Program	3
3.1 Task 1	3
3.1.1 Design	3
3.1.2 Execution	4
3.1.3 Screenshots	5
3.2 Task 2	10
3.2.1 Design	10
3.2.2 Execution	11
3.2.3 SCREENSHOTS	11
3.3 Bonus	16
3.3.1 Execution	16
3.3.2 Test example	17
4 What I Learn	20

1 OVERVIEW

This report is about CSC3150 project. The assignment mainly focuses on revealing the details of processes and multi-processes programming. Task one is about user mode in multi process programming, while the other task is about kernel mode in multi-process programming. The bonus is like a algorithm programming problem to implement a pstree.

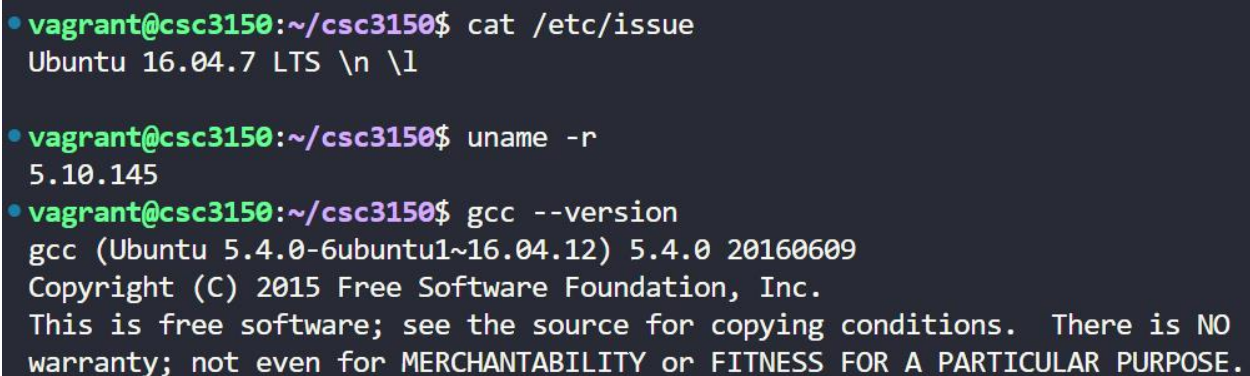
2 ENVIRONMENT

The environment of running my programs is the following:

OS version: Ubuntu 16.04.7 LTS

kernel version: Linux-5.10.145

gcc version: gcc (Ubuntu 5.4.0-6ubuntu1 16.04.12) 5.4.0 20160609

A terminal window with a dark background and light-colored text. It shows three commands and their outputs: 1. 'cat /etc/issue' outputs 'Ubuntu 16.04.7 LTS \n \1'. 2. 'uname -r' outputs '5.10.145'. 3. 'gcc --version' outputs 'gcc (Ubuntu 5.4.0-6ubuntu1~16.04.12) 5.4.0 20160609 Copyright (C) 2015 Free Software Foundation, Inc. This is free software; see the source for copying conditions. There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.'

```
• vagrant@csc3150:~/csc3150$ cat /etc/issue
Ubuntu 16.04.7 LTS \n \1

• vagrant@csc3150:~/csc3150$ uname -r
5.10.145

• vagrant@csc3150:~/csc3150$ gcc --version
gcc (Ubuntu 5.4.0-6ubuntu1~16.04.12) 5.4.0 20160609
Copyright (C) 2015 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

FIGURE 1: ENVIRONMENT

3 PROGRAM

3.1 TASK 1

3.1.1 DESIGN

The chart flow of program 1 is:

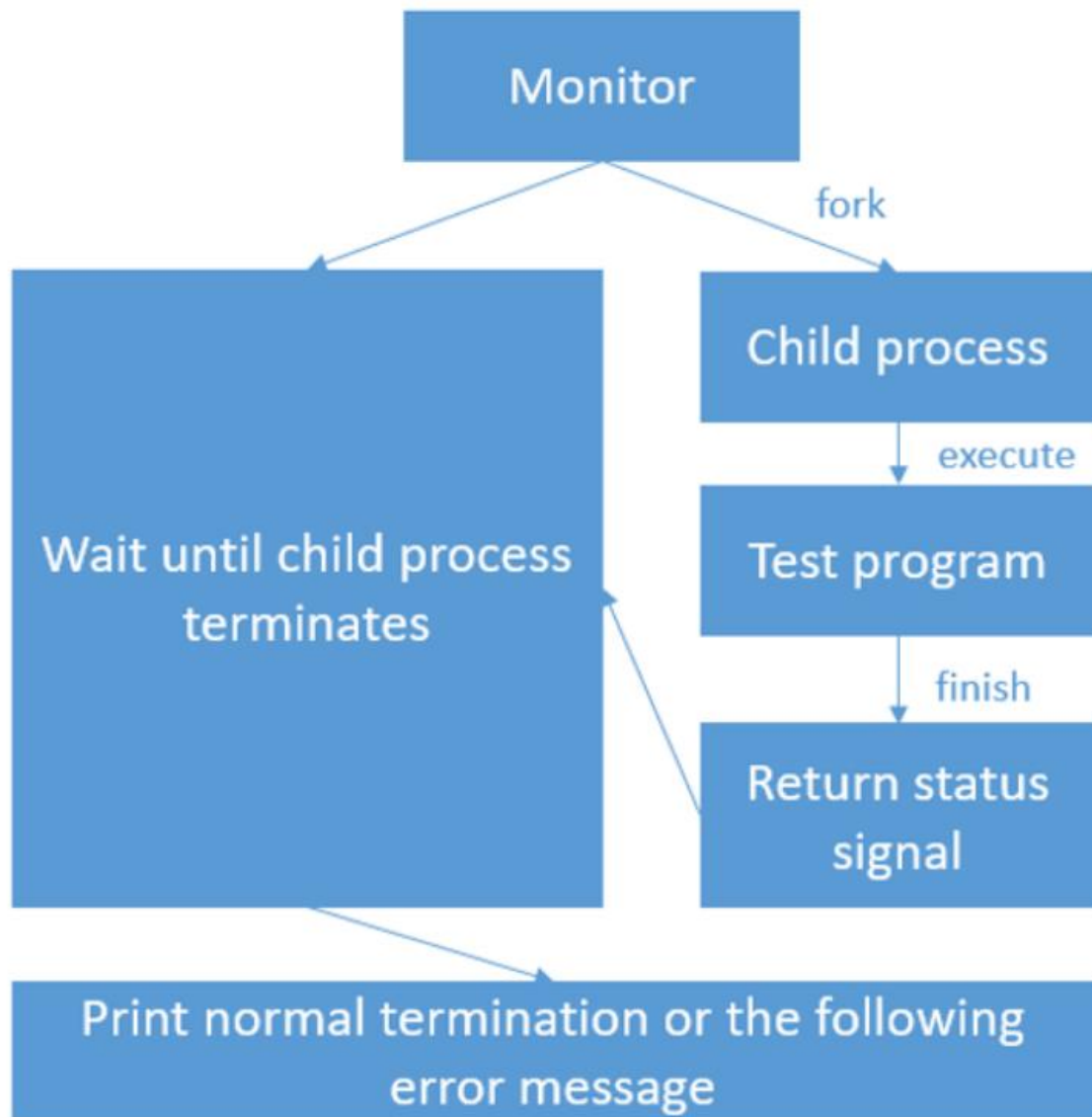


FIGURE 2: CHART FLOW

For This problem, we first use `fork()` function to fork a children process with same state machine of parent process. After that, we should use `execve()` function to execute clear and reset the state machine of child process in order to build a new program. At the same time, parent process should wait the child process until it terminate and send some signals back, we use `wait_pid()` here to handle some different cases for different send back signal.

3.1.2 EXECUTION

In order to execute the task 1, you should first enter into the program1 directory and than follow these steps:

Listing 1: step

```

1 $ cd /* directory where the program.c located in */
2 $ make # compile the program
3 $ ./program1 ./test_name
4 $ make clean # remember to clear

```

3.1.3 SCREENSHOTS

```

vagrant@csc3150:~/csc3150/Homework/project1/program1$ ./program1 ./normal
Process start to fork
I'm the Parent Process, my pid = 15311
I'm the Child Process, my pid = 15312
Child process start to execute test program:
-----CHILD PROCESS START-----
This is the normal program

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

```

FIGURE 3: NORMAL

```

vagrant@csc3150:~/csc3150/Homework/project1/program1$ ./program1 ./abort
Process start to fork
I'm the Parent Process, my pid = 15348
I'm the Child Process, my pid = 15349
Child process start to execute test program:
-----CHILD PROCESS START-----
This is the SIGABRT program

Parent process receives SIGCHILD signal
child process get SIGABRT signal

```

FIGURE 4: ABORT

```
vagrant@csc3150:~/csc3150/Homework/project1/program1$ ./program1 ./stop
Process start to fork
I'm the Parent Process, my pid = 16035
I'm the Child Process, my pid = 16036
Child process start to execute test program:
-----CHILD PROCESS START-----
This is the SIGSTOP program

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

FIGURE 5: STOP

```
vagrant@csc3150:~/csc3150/Homework/project1/program1$ ./program1 ./alarm
Process start to fork
I'm the Parent Process, my pid = 16571
I'm the Child Process, my pid = 16572
Child process start to execute test program:
-----CHILD PROCESS START-----
This is the SIGALRM program

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

FIGURE 6: ALARM

```
vagrant@csc3150:~/csc3150/Homework/project1/program1$ ./program1 ./bus
Process start to fork
I'm the Parent Process, my pid = 17196
I'm the Child Process, my pid = 17197
Child process start to execute test program:
-----CHILD PROCESS START-----
This is the SIGBUS program

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

FIGURE 7: BUS


```
vagrant@csc3150:~/csc3150/Homework/project1/program1$ ./program1 ./floating
Process start to fork
I'm the Parent Process, my pid = 19055
I'm the Child Process, my pid = 19056
Child process start to execute test program:
-----CHILD PROCESS START-----
This is the SIGFPE program

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

FIGURE 8: FLOAT

```
vagrant@csc3150:~/csc3150/Homework/project1/program1$ ./program1 ./hangup
Process start to fork
I'm the Parent Process, my pid = 1809
I'm the Child Process, my pid = 1810
Child process start to execute test program:
-----CHILD PROCESS START-----
This is the SIGHUP program

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

FIGURE 9: HANGUP

```
vagrant@csc3150:~/csc3150/Homework/project1/program1$ ./program1 ./illegal_instr
Process start to fork
I'm the Parent Process, my pid = 2009
I'm the Child Process, my pid = 2010
Child process start to execute test program:
-----CHILD PROCESS START-----
This is the SIGILL program

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

FIGURE 10: ILLEGAL_INSTR

```
vagrant@csc3150:~/csc3150/Homework/project1/program1$ ./program1 ./interrupt
Process start to fork
I'm the Parent Process, my pid = 2928
I'm the Child Process, my pid = 2929
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
```

FIGURE 11: INTERRUPT

```
vagrant@csc3150:~/csc3150/Homework/project1/program1$ ./program1 ./kill
Process start to fork
I'm the Parent Process, my pid = 2807
I'm the Child Process, my pid = 2808
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
```

FIGURE 12: KILL

```
vagrant@csc3150:~/csc3150/Homework/project1/program1$ ./program1 ./pipe
Process start to fork
I'm the Parent Process, my pid = 2983
I'm the Child Process, my pid = 2984
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
```

FIGURE 13: PIPE


```
vagrant@csc3150:~/csc3150/Homework/project1/program1$ ./program1 ./quit
Process start to fork
I'm the Parent Process, my pid = 3028
I'm the Child Process, my pid = 3029
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
```

FIGURE 14: QUIT

```
vagrant@csc3150:~/csc3150/Homework/project1/program1$ ./program1 ./segment_fault
Process start to fork
I'm the Parent Process, my pid = 3098
I'm the Child Process, my pid = 3099
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
```

FIGURE 15: SEGMENT FAULT

```
vagrant@csc3150:~/csc3150/Homework/project1/program1$ ./program1 ./stop
Process start to fork
I'm the Parent Process, my pid = 3128
I'm the Child Process, my pid = 3129
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
```

FIGURE 16: STOP

```
vagrant@csc3150:~/csc3150/Homework/project1/program1$ ./program1 ./terminate
Process start to fork
I'm the Parent Process, my pid = 3144
I'm the Child Process, my pid = 3145
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
```

FIGURE 17: TEMINATE

3.2 TASK 2

3.2.1 DESIGN

The chart flow of program 2 is:

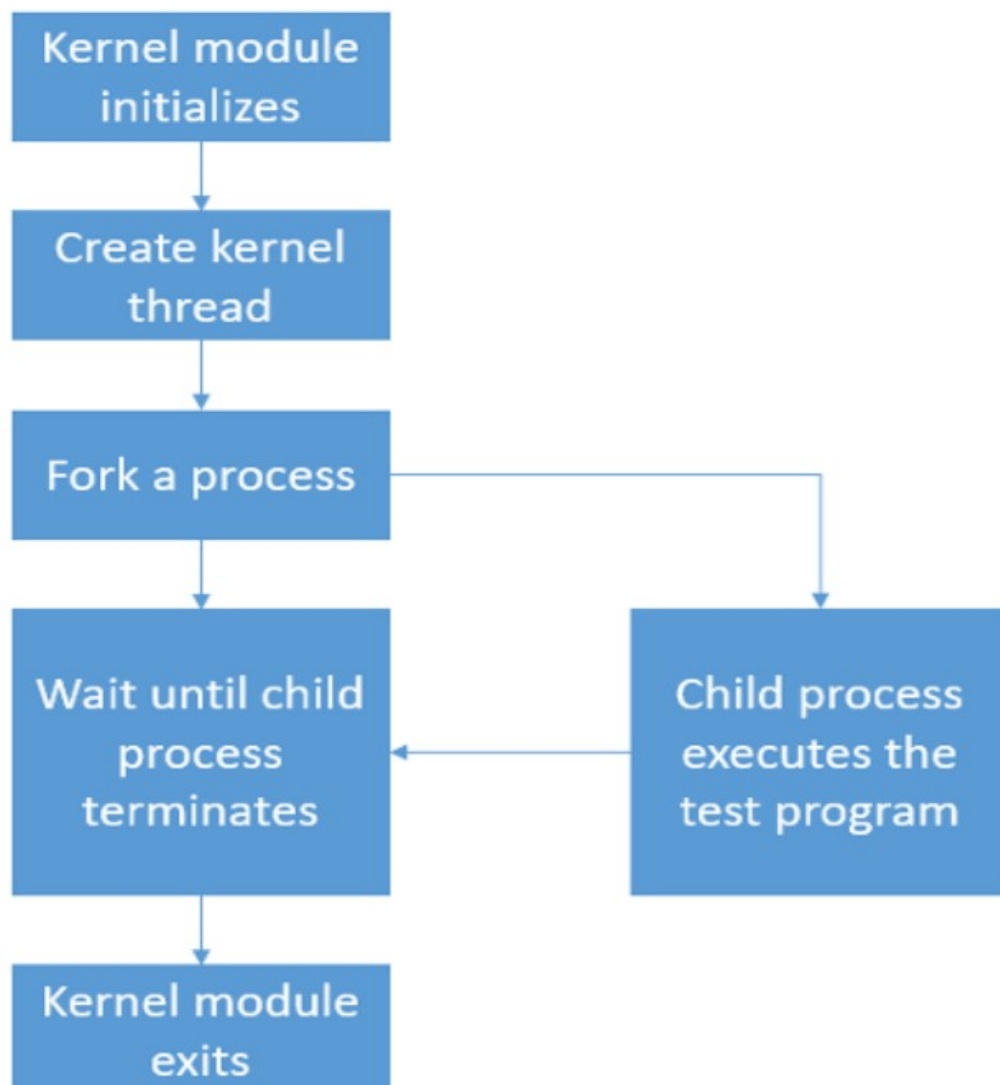


FIGURE 18: FLOW CHART

For this problem, first we should modify the kernel into proper version(5.10.145) and export some kernel function in the kernel source code. In my program, the `program_init()` function is used to initialize the kernel and create the kernel thread. The `my_exec()` function is used to execute the test program. The `my_fork()` function is used to fork the process and get the pid for parent and child process. The `my_wait()` function is used to wait for the child process terminate and get the signal.

3.2.2 EXECUTION

In order to execute the task 2, first you should make sure you already install Ubuntu 5.10.145 and install the modules, export and extern the function including `kernel_thread`, `do_wait`, `do_exec`, `getname_kernel`. Then you can do the following steps:

Listing 2: step

```

1 $ su root
2 $ cd /* directory where your program2.c located in */
3 $ gcc -o test test.c
4 $ make
5 $ insmod program2.ko
6 $ rmmmod program2.ko
7 $ dmesg | tail -n 10
8 $ make clean
9 $ rm -f test

```

3.2.3 SCREENSHOTS

```

[ 6925.890085] [program2] : module_init {Yang zhao} {120090865}
[ 6925.890086] [program2] : module_init create kthread start
[ 6925.890202] [program2] : module_init Kthread starts
[ 6925.890311] [program2] : The Child process has pid = 10678
[ 6925.890312] [program2] : This is the parent process, pid = 10677
[ 6925.890452] [program2] : child process
[ 6925.890647] [program2] : get SIGTERM signal
[ 6925.890648] [program2] : child process terminated
[ 6925.890649] [program2] : The return signal is 15
[ 6928.896855] [program2] : module_exit

```

FIGURE 19: TERMINATE


```

[ 6987.772612] [program2] : module_init {Yang zhao} {120090865}
[ 6987.772613] [program2] : module_init create kthread start
[ 6987.772701] [program2] : module_init Kthread starts
[ 6987.772793] [program2] : The Child process has pid = 11331
[ 6987.772794] [program2] : This is the parent process, pid = 11330
[ 6987.772864] [program2] : child process
[ 6987.850297] [program2] : get SIGABRT signal
[ 6987.850298] [program2] : child process aborted
[ 6987.850299] [program2] : The return signal is 6
[ 6990.777037] [program2] : module_exit

```

FIGURE 20: ABORT

```

[ 7140.340951] [program2] : module_init {Yang zhao} {120090865}
[ 7140.340952] [program2] : module_init create kthread start
[ 7140.341077] [program2] : module_init Kthread starts
[ 7140.341169] [program2] : The Child process has pid = 12017
[ 7140.341170] [program2] : This is the parent process, pid = 12016
[ 7140.341257] [program2] : child process
[ 7142.341562] [program2] : get SIGALRM signal
[ 7142.341563] [program2] : child process get a SIGALRM signal
[ 7142.341564] [program2] : The return signal is 14
[ 7143.361611] [program2] : module_exit

```

FIGURE 21: ALARM

```

[ 7290.733051] [program2] : module_init {Yang zhao} {120090865}
[ 7290.733053] [program2] : module_init create kthread start
[ 7290.733122] [program2] : module_init Kthread starts
[ 7290.733156] [program2] : The Child process has pid = 12673
[ 7290.733161] [program2] : This is the parent process, pid = 12672
[ 7290.733289] [program2] : child process
[ 7290.806750] [program2] : get SIGBUS signal
[ 7290.806751] [program2] : child process got BUS error
[ 7290.806751] [program2] : The return signal is 7
[ 7293.785440] [program2] : module_exit

```

FIGURE 22: BUS


```
[ 7334.562821] [program2] : module_init {Yang zhao} {120090865}
[ 7334.562822] [program2] : module_init create kthread start
[ 7334.562954] [program2] : module_init Kthread starts
[ 7334.562985] [program2] : The Child process has pid = 13332
[ 7334.562986] [program2] : This is the parent process, pid = 13331
[ 7334.563064] [program2] : child process
[ 7334.641387] [program2] : get SIGFPE signal
[ 7334.641388] [program2] : child process got Floating-point exception
[ 7334.641389] [program2] : The return signal is 8
[ 7337.567452] [program2] : module_exit
```

FIGURE 23: FLOAT

```
[ 7443.955751] [program2] : module_init {Yang zhao} {120090865}
[ 7443.955752] [program2] : module_init create kthread start
[ 7443.955844] [program2] : module_init Kthread starts
[ 7443.955871] [program2] : The Child process has pid = 13954
[ 7443.955872] [program2] : This is the parent process, pid = 13953
[ 7443.955927] [program2] : child process
[ 7443.956186] [program2] : get SIGHUP signal
[ 7443.956187] [program2] : child process hung up
[ 7443.956187] [program2] : The return signal is 1
[ 7446.960116] [program2] : module_exit
```

FIGURE 24: HANGUP

```
[ 7480.566238] [program2] : module_init {Yang zhao} {120090865}
[ 7480.566240] [program2] : module_init create kthread start
[ 7480.566328] [program2] : module_init Kthread starts
[ 7480.566387] [program2] : The Child process has pid = 14566
[ 7480.566388] [program2] : This is the parent process, pid = 14565
[ 7480.566475] [program2] : child process
[ 7480.664012] [program2] : get SIGILL signal
[ 7480.664014] [program2] : child process got Illegal instruction
[ 7480.664014] [program2] : The return signal is 4
[ 7483.570808] [program2] : module_exit
```

FIGURE 25: ILLEGAL_INSTR


```

[11221.026920] [program2] : module_init {Yang zhao} {120090865}
[11221.026922] [program2] : module_init create kthread start
[11221.027043] [program2] : module_init Kthread starts
[11221.027086] [program2] : The Child process has pid = 15714
[11221.027086] [program2] : This is the parent process, pid = 15713
[11221.027223] [program2] : child process
[11221.027427] [program2] : get SIGINT signal
[11221.027428] [program2] : child process interrupted
[11221.027428] [program2] : The return signal is 2
[11224.038460] [program2] : module_exit

```

FIGURE 26: INTERRUPT

```

[11417.361994] [program2] : module_init {Yang zhao} {120090865}
[11417.361996] [program2] : module_init create kthread start
[11417.362113] [program2] : module_init Kthread starts
[11417.362137] [program2] : The Child process has pid = 16373
[11417.362138] [program2] : This is the parent process, pid = 16372
[11417.362270] [program2] : child process
[11417.362602] [program2] : get SIGKILL signal
[11417.362603] [program2] : child process are killed
[11417.362604] [program2] : The return signal is 9
[11420.367677] [program2] : module_exit

```

FIGURE 27: KILL

```

[11554.302137] [program2] : module_init {Yang zhao} {120090865}
[11554.302138] [program2] : module_init create kthread start
[11554.302272] [program2] : module_init Kthread starts
[11554.302303] [program2] : The Child process has pid = 16992
[11554.302307] [program2] : This is the parent process, pid = 16991
[11554.302455] [program2] : child process
[11554.302999] [program2] : child process gets normal termination
[11554.303000] [program2] : The return signal is 0
[11557.315050] [program2] : module_exit

```

FIGURE 28: NORMAL


```

[11787.161936] [program2] : module_init {Yang zhao} {120090865}
[11787.161937] [program2] : module_init create kthread start
[11787.162025] [program2] : module_init Kthread starts
[11787.162048] [program2] : The Child process has pid = 17633
[11787.162049] [program2] : This is the parent process, pid = 17632
[11787.162142] [program2] : child process
[11787.162378] [program2] : get SIGPIPE signal
[11787.162379] [program2] : child process got Broken pipe
[11787.162379] [program2] : The return signal is 13
[11790.220219] [program2] : module_exit

```

FIGURE 29: PIPE

```

[11903.746951] [program2] : module_init {Yang zhao} {120090865}
[11903.746952] [program2] : module_init create kthread start
[11903.747183] [program2] : module_init Kthread starts
[11903.747242] [program2] : The Child process has pid = 18260
[11903.747243] [program2] : This is the parent process, pid = 18259
[11903.747397] [program2] : child process
[11903.828758] [program2] : get SIGQUIT signal
[11903.828759] [program2] : child process quited
[11903.828760] [program2] : The return signal is 3
[11906.751781] [program2] : module_exit

```

FIGURE 30: QUIT

```

[11973.080913] [program2] : module_init {Yang zhao} {120090865}
[11973.080914] [program2] : module_init create kthread start
[11973.081054] [program2] : module_init Kthread starts
[11973.081089] [program2] : The Child process has pid = 18888
[11973.081089] [program2] : This is the parent process, pid = 18887
[11973.081292] [program2] : child process
[11973.160759] [program2] : get SIGSEGV signal
[11973.160760] [program2] : child process got Segmentation violation
[11973.160761] [program2] : The return signal is 11
[11976.086359] [program2] : module_exit

```

FIGURE 31: SEGMENTATION FAULT

```

[12044.206990] [program2] : module_init {Yang zhao} {120090865}
[12044.206991] [program2] : module_init create kthread start
[12044.207123] [program2] : module_init Kthread starts
[12044.207225] [program2] : The Child process has pid = 19501
[12044.207225] [program2] : This is the parent process, pid = 19500
[12044.207302] [program2] : child process
[12044.207440] [program2] : CHILD PROCESS STOPPED
[12044.207440] [program2] : child process get SIGSTOP signal
[12044.207441] [program2] : The return signal is 19
[12047.211689] [program2] : module_exit

```

FIGURE 32: STOP

```

[12121.352815] [program2] : module_init {Yang zhao} {120090865}
[12121.352816] [program2] : module_init create kthread start
[12121.352912] [program2] : module_init Kthread starts
[12121.352946] [program2] : The Child process has pid = 20128
[12121.352946] [program2] : This is the parent process, pid = 20127
[12121.353080] [program2] : child process
[12121.435248] [program2] : get SIGTRAP signal
[12121.435250] [program2] : child process trapped
[12121.435250] [program2] : The return signal is 5
[12124.356924] [program2] : module_exit

```

FIGURE 33: TRAP

3.3 BONUS

In this task, I create a file to implement the pstree and the file name is pstree.c . My program support [-c] [-h] [-n] [-p] [-V] [-A] [-U] [PID] eight commands. There are a makefile to build the output program.

3.3.1 EXECUTION

First build the executable file named pstree.




```
vagrant@csc3150:~/csc3150/Homework/project1/bonus$ make
gcc pstree.c -c -g -o pstree.o
gcc basic.c -c -g -o basic.o
gcc get.c -c -g -o get.o
gcc sort.c -c -g -o sort.o
gcc build.c -c -g -o build.o
gcc pstree.o basic.o get.o sort.o build.o -o pstree
```

FIGURE 34: MAKE

3.3.2 TEST EXAMPLE

```
vagrant@csc3150:~/csc3150/Homework/project1/bonus$ ./pstree
systemd-+-accounts-daemon-+-2*[{accounts-daemon}]
        |-acpid
        |-agetty
        |-agetty
        |-atd
        |-cpptools-srv-+-15*[{cpptools-srv}]
        |-cpptools-srv-+-15*[{cpptools-srv}]
        |-cron
        |-dbus-daemon
        |-dhclient
        |-irqbalance
        |-iscsid
```

FIGURE 35: PSTREE

```
vagrant@csc3150:~/csc3150/Homework/project1/bonus$ ./pstree -c
systemd-+-accounts-daemon-+-{gmain}
          |                  +-{gdbus}
          |
          |-acpid
          |-agetty
          |-agetty
          |-atd
          |-cpptools-srv-+-{cpptools-srv}
                        |+-{cpptools-srv}
                        |+-{cpptools-srv}
                        |+-{cpptools-srv}
                        |+-{cpptools-srv}
                        |+-{cpptools-srv}
                        |+-{cpptools-srv}
                        |+-{cpptools-srv}
                        |+-{cpptools-srv}
                        |+-{cpptools-srv}
                        |+-{cpptools-srv}
```

FIGURE 36: PSTREE -C

```
vagrant@csc3150:~/csc3150/Homework/project1/bonus$ ./pstree -h
systemd-+-accounts-daemon-+-{gmain) S 1 1050
          |                  +-{gdbus}
          |
          |-acpid
          |-agetty
          |-agetty
          |-atd
          |-cpptools-srv-+-{cpptools-srv}
```

FIGURE 37: PSTREE -H


```

vagrant@csc3150:~/csc3150/Homework/project1/bonus$ ./pstree -n
systemd+-systemd-journal
| -lvmetad
| -systemd-udevd
| -dhclient
| -dbus-daemon
| -accounts-daemon+-2*[{accounts-daemon}]
| -systemd-logind
| -lxcfs+-2*[{lxcfs}]
| -atd
| -rsyslogd+-3*[{rsyslogd}]
| -acpid
| -cron
| -iscsid

```

FIGURE 38: PSTREE -N

```

vagrant@csc3150:~/csc3150/Homework/project1/bonus$ ./pstree -p
systemd(1) +-accounts-daemon(1050) +-{gmain}(1070)
|                                     `-{gdbus}(1075)
| -acpid(1078)
| -agetty(1165)
| -agetty(1171)
| -atd(1061)
| -cpptools-srv(23214) +-{cpptools-srv}(23215)
|                       | -{cpptools-srv}(23216)
|                       | -{cpptools-srv}(23217)
|                       | -{cpptools-srv}(23218)
|                       | -{cpptools-srv}(23219)
|                       | -{cpptools-srv}(23220)
|                       | -{cpptools-srv}(23221)
|                       | -{cpptools-srv}(23222)
|                       | -{cpptools-srv}(23223)
|                       | -{cpptools-srv}(23224)
|                       | -{cpptools-srv}(23225)

```

FIGURE 39: PSTREE -P

```

vagrant@csc3150:~/csc3150/Homework/project1/bonus$ ./pstree -V
pstree 1.0
Copyright (C) 2022 Yang Zhao

```

FIGURE 40: PSTREE -V

```
vagrant@csc3150:~/csc3150/Homework/project1/bonus$ ./pstree -A
systemd--accounts-daemon--2*[{accounts-daemon}]
    |-acpid
    |-agetty
    |-agetty
    |-atd
    |-cpptools-srv--15*[{cpptools-srv}]
    |-cpptools-srv--15*[{cpptools-srv}]
    |-cron
    |-dbus-daemon
    |-dhclient
```

FIGURE 41: PSTREE -A

```
vagrant@csc3150:~/csc3150/Homework/project1/bonus$ ./pstree -U
systemd--accounts-daemon--2*[{accounts-daemon}]
    |-acpid
    |-agetty
    |-agetty
    |-atd
    |-cpptools-srv--15*[{cpptools-srv}]
    |-cpptools-srv--15*[{cpptools-srv}]
    |-cron
```

FIGURE 42: PSTREE -U

```
vagrant@csc3150:~/csc3150/Homework/project1/bonus$ ./pstree 1050
accounts-daemon(1050)--{gmain}(1070)
    `--{gdbus}(1075)
```

FIGURE 43: PSTREE PID

```
vagrant@csc3150:~/csc3150/Homework/project1/bonus$ ./pstree wkbdbqj
pstree: invalid option -- wkbdbqj
Usage: pstree [ -c ] [ -h ] [ -n ] [ -p ] [ -V ] [ -A ] [ -U ] [ PID ]
```

FIGURE 44: INVALID

4 WHAT I LEARN

In the project 1, I work on parent processes and child processes and get a better understanding of their relationship. And I also learn system call such as fork, wait and so on. Besides, I modify the kernel module and then have a better insight of kernel and thread. Finally, by solving bonus task, I get knowledge of pstree, having a deep understanding of linux file system.