Report

Name：Wu Jingqi
ID：120090357

1. How did I design my program?

(1) Program 1

(a) basic ideas of the program

The program is to run a process under user mode. The main process will fork a child process to execute a test program and wait for its returning signal. After receiving the terminated signal, the parent process will print out the related information of the signal.

(b) implementation

Firstly, the program will use fork( ) function to fork a child process at the beginning. Then, the program will check the pid of the process and do the corresponding operations. If pid equals to 0, which means the process is the child process, then the program will let it execute the test program. If pid not equals to 0 or -1, then it is the parent process and it will wait for the terminated signal of the child process (using waitpid( )) and print out the information of the signal.

(2) Program 2

(a) basic ideas of the program

This program will create a kernel thread. In the thread, the program will fork a child process and make it to execute another program. The parent process will wait for the child's terminated signal and print out related information.

(b) implementation

1. Firstly, since we need to use these 4 functions: "_do_fork" (/kernel/fork.c) , "do_execve" (/fs/exec.c), "getname" (/fs/namei.c) and "do_wait"(/kernel/exit.c), we need to find their corresponding files in kernel and export them using EXPORT_SYMBOL. After modifications, we need to recompile the kernel. Then we just need to use extern to import them in program2.c.

2. Then, when initiating the module, we need to create a kernel thread using kthread_create() to run my_fork() function, where the program will fork a child process to execute another program.

3.To implement my_fork( ), several functions need to be implemented in advance.

my_exec( ):This function will first get the information (using getname( )) of the test program using the address set inside it. After that, it will use do_execve( ) to execute the file.

my_wait( ):this function will get the terminated signal of the child process usin

g.

do_wait( ):With these 2 functions, my_fork( ) is implemented in the following: First, my_fork( ) will use _do_fork to generate a child process to run my_exec( ) function. Then it will use my_wait( ) to get the terminated signal of the child process and print out corresponding information.

4. After creating the kernel thread, the program will wake up the program if there kthread_create( ) create a thread successfully.

2. How to set up your development environment.
    1. Set visual machine: https://csc3150.cyzhu.dev/vm-configuration/windows-amd-intel-x86

       Install vitrualbox and vagrant.
       cd to new folder csc3150, and Execute vagrant init cyzhu/csc3150
       Then execute vagrant up.
       Then goto VScode to create terminal, and excute sudo apt update && sudo apt install -y build-essential
    2. Compile Kernel
       Download source code from
          http://www.kernel.org
          mirro: https://mirror.tuna.tsinghua.edu.cn/kernel/v5.x/
       Install Dependency and development tools
          sudo apt-get install libncurses-dev gawk flex bison openssl libssl-dev dkms libelf-devlibudev-dev libpci-dev libiberty-dev autoconf llvm dwarves
       Extract the source file to /home/seed/work
          cp KERNEL_FILE.tar.xz /home/seed/work
          cd /home/seed/work
          $sudo tar xvf KERNEL_FILE.tar.xz
       Copy config from /boot to /home/seed/work/KERNEL_FILE
       Login root account and go to kernel source directory
          $sudo su
          $cd /home/seed/work /KERNEL_FILE
       Clean previous setting and start configuration
          $make mrproper
          $make clean
          $make menuconfig
          save the config and exit

Build kernel Image and modules

    $make bzImage -j$(nproc)

    $make modules -j$(nproc)

    $make –j$(nproc)

Install kernel modules

    $make modules_install

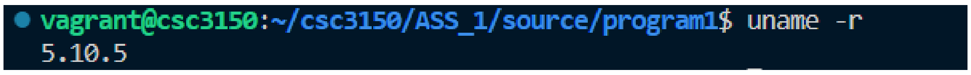
Install kernel

    $make install

Reboot to load new kernel

    $reboot

Then check the version:

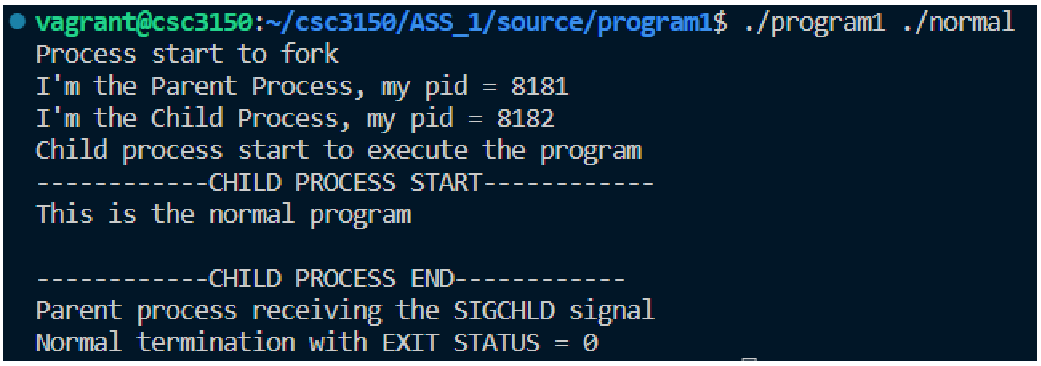


```
vagrant@csc3150:~/csc3150/ASS_1/source/program1$ uname -r
5.10.5
```

3. Screenshot of my program output.

(1) Program 1:

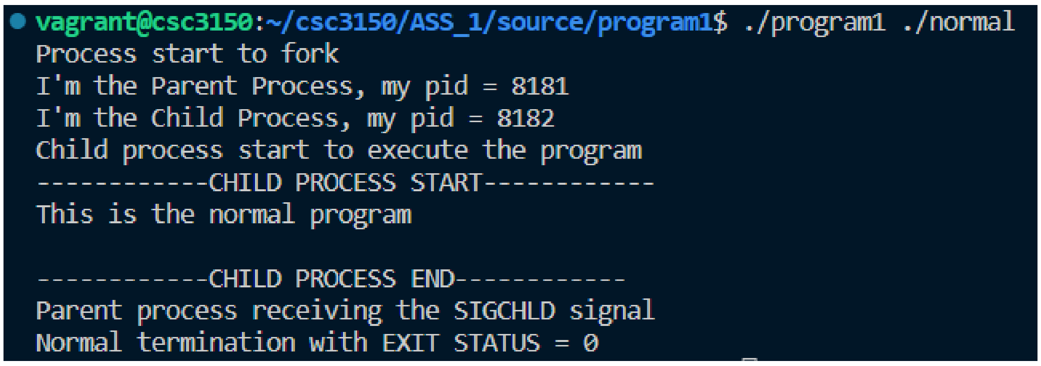
Normal:



```
vagrant@csc3150:~/csc3150/ASS_1/source/program1$ ./program1 ./normal
Process start to fork
I'm the Parent Process, my pid = 8181
I'm the Child Process, my pid = 8182
Child process start to execute the program
------------CHILD PROCESS START------------
This is the normal program

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

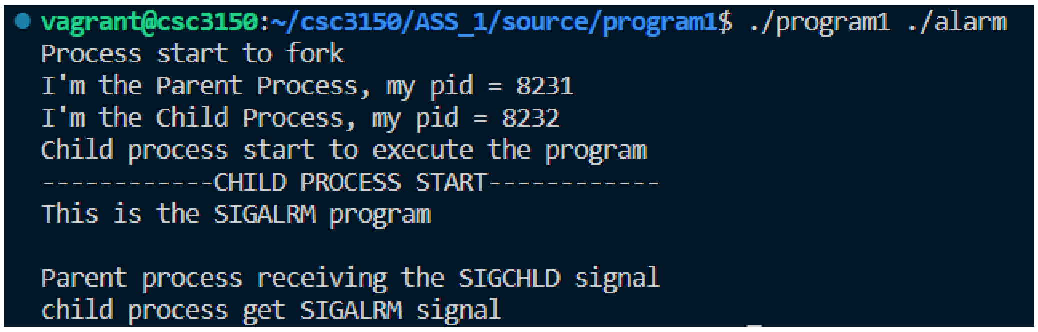
Abort:



```
vagrant@csc3150:~/csc3150/ASS_1/source/program1$ ./program1 ./normal
Process start to fork
I'm the Parent Process, my pid = 8181
I'm the Child Process, my pid = 8182
Child process start to execute the program
------------CHILD PROCESS START------------
This is the normal program

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

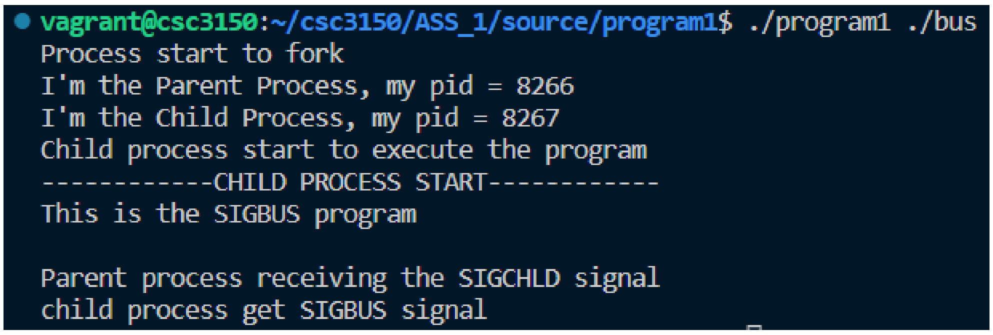
Alarm:



```
vagrant@csc3150:~/csc3150/ASS_1/source/program1$ ./program1 ./alarm
Process start to fork
I'm the Parent Process, my pid = 8231
I'm the Child Process, my pid = 8232
Child process start to execute the program
------------CHILD PROCESS START------------
This is the SIGALRM program

Parent process receiving the SIGCHLD signal
child process get SIGALRM signal
```

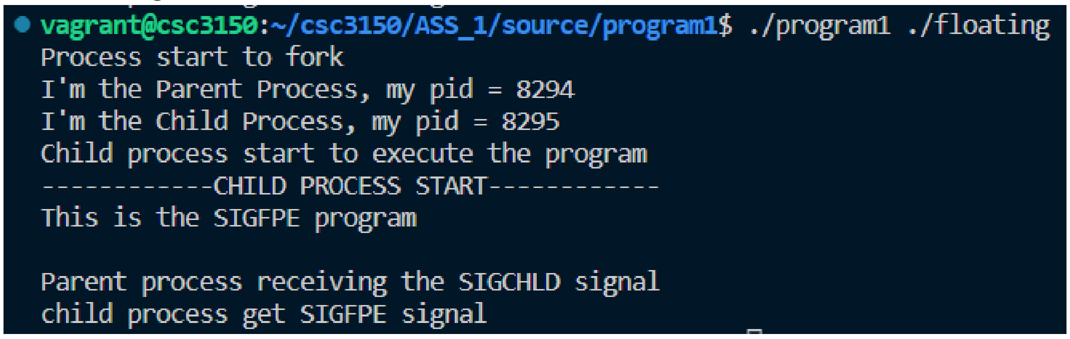
Bus:



```
vagrant@csc3150:~/csc3150/ASS_1/source/program1$ ./program1 ./bus
Process start to fork
I'm the Parent Process, my pid = 8266
I'm the Child Process, my pid = 8267
Child process start to execute the program
------------CHILD PROCESS START------------
This is the SIGBUS program

Parent process receiving the SIGCHLD signal
child process get SIGBUS signal
```

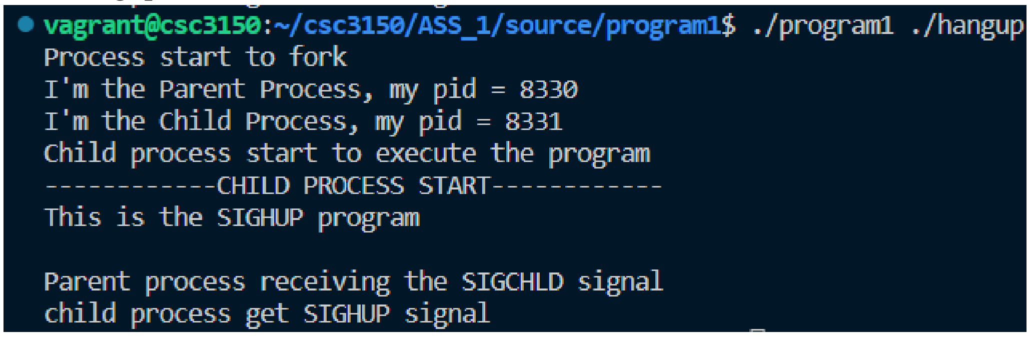
Floating:



```
vagrant@csc3150:~/csc3150/ASS_1/source/program1$ ./program1 ./floating
Process start to fork
I'm the Parent Process, my pid = 8294
I'm the Child Process, my pid = 8295
Child process start to execute the program
------------CHILD PROCESS START------------
This is the SIGFPE program

Parent process receiving the SIGCHLD signal
child process get SIGFPE signal
```

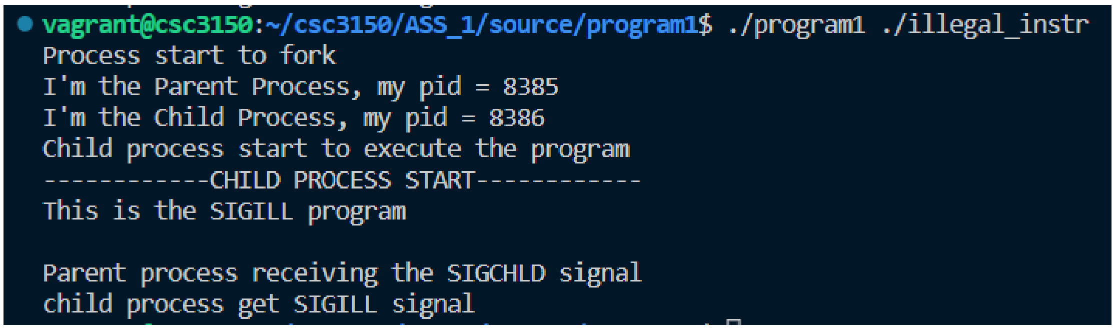
Hangup:



```
vagrant@csc3150:~/csc3150/ASS_1/source/program1$ ./program1 ./hangup
Process start to fork
I'm the Parent Process, my pid = 8330
I'm the Child Process, my pid = 8331
Child process start to execute the program
------------CHILD PROCESS START------------
This is the SIGHUP program

Parent process receiving the SIGCHLD signal
child process get SIGHUP signal
```

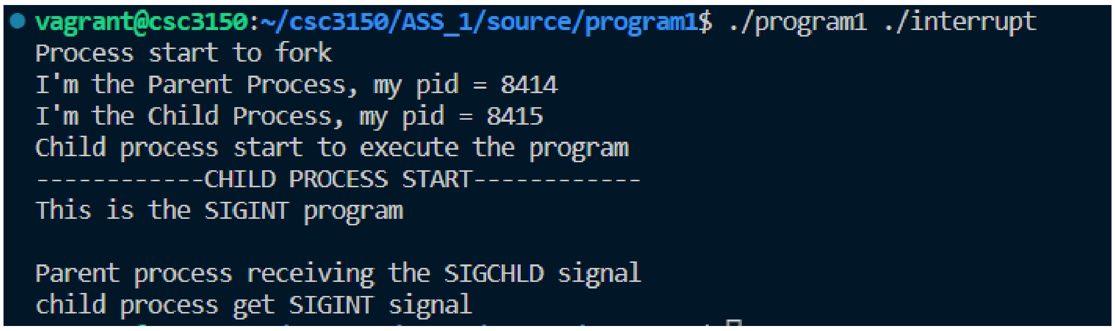
illegal_instr:



```
vagrant@csc3150:~/csc3150/ASS_1/source/program1$ ./program1 ./illegal_instr
Process start to fork
I'm the Parent Process, my pid = 8385
I'm the Child Process, my pid = 8386
Child process start to execute the program
------------CHILD PROCESS START------------
This is the SIGILL program

Parent process receiving the SIGCHLD signal
child process get SIGILL signal
```

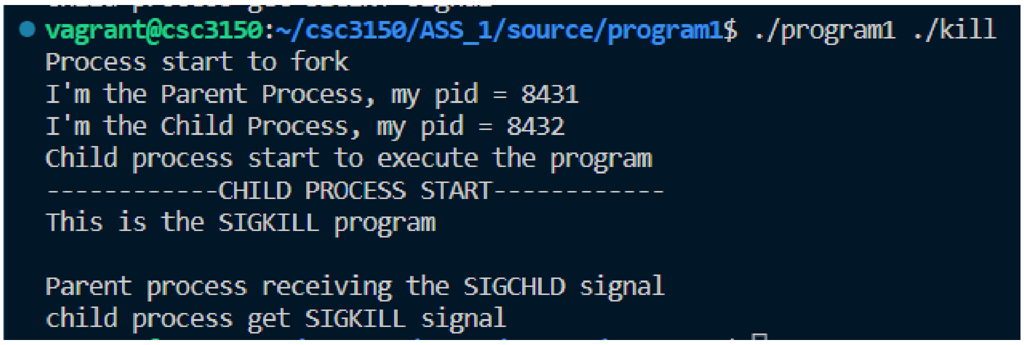
Interrupt:



```
vagrant@csc3150:~/csc3150/ASS_1/source/program1$ ./program1 ./interrupt
Process start to fork
I'm the Parent Process, my pid = 8414
I'm the Child Process, my pid = 8415
Child process start to execute the program
------------CHILD PROCESS START------------
This is the SIGINT program

Parent process receiving the SIGCHLD signal
child process get SIGINT signal
```

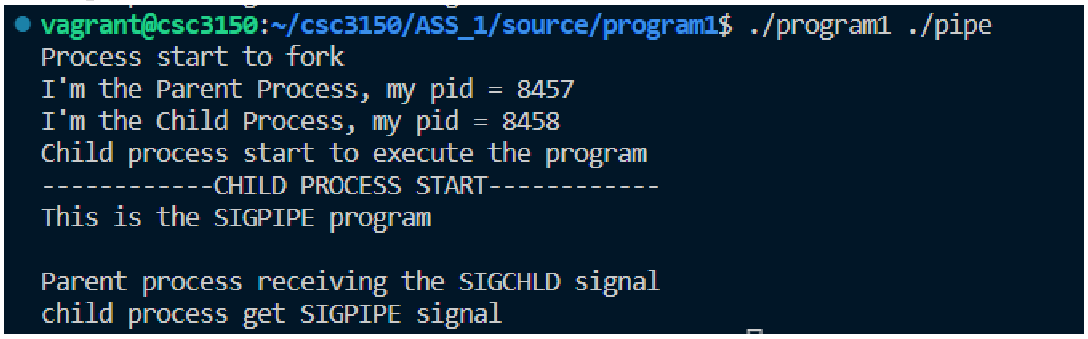
Kill:



```
vagrant@csc3150:~/csc3150/ASS_1/source/program1$ ./program1 ./kill
Process start to fork
I'm the Parent Process, my pid = 8431
I'm the Child Process, my pid = 8432
Child process start to execute the program
------------CHILD PROCESS START------------
This is the SIGKILL program

Parent process receiving the SIGCHLD signal
child process get SIGKILL signal
```

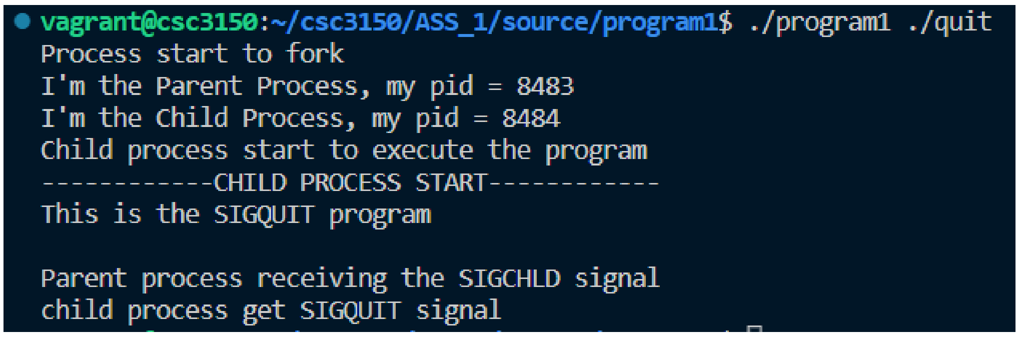
Pipe:



```
vagrant@csc3150:~/csc3150/ASS_1/source/program1$ ./program1 ./pipe
Process start to fork
I'm the Parent Process, my pid = 8457
I'm the Child Process, my pid = 8458
Child process start to execute the program
------------CHILD PROCESS START------------
This is the SIGPIPE program

Parent process receiving the SIGCHLD signal
child process get SIGPIPE signal
```

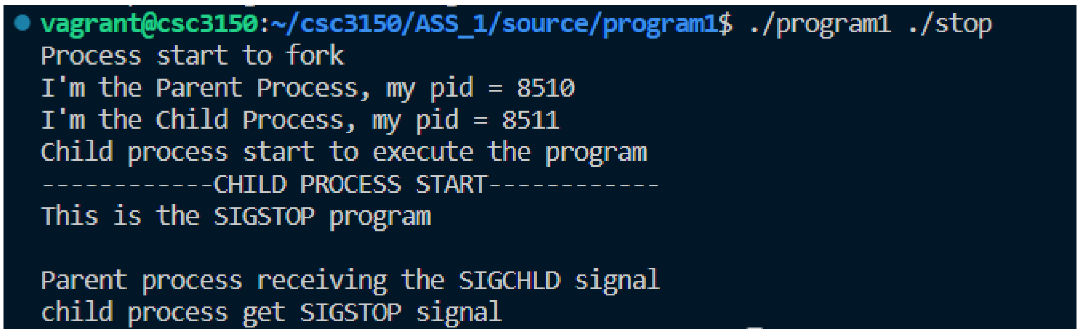
Quit:



```
● vagrant@csc3150:~/csc3150/ASS_1/source/program1$ ./program1 ./quit
  Process start to fork
  I'm the Parent Process, my pid = 8483
  I'm the Child Process, my pid = 8484
  Child process start to execute the program
  ------------CHILD PROCESS START------------
  This is the SIGQUIT program

  Parent process receiving the SIGCHLD signal
  child process get SIGQUIT signal
```

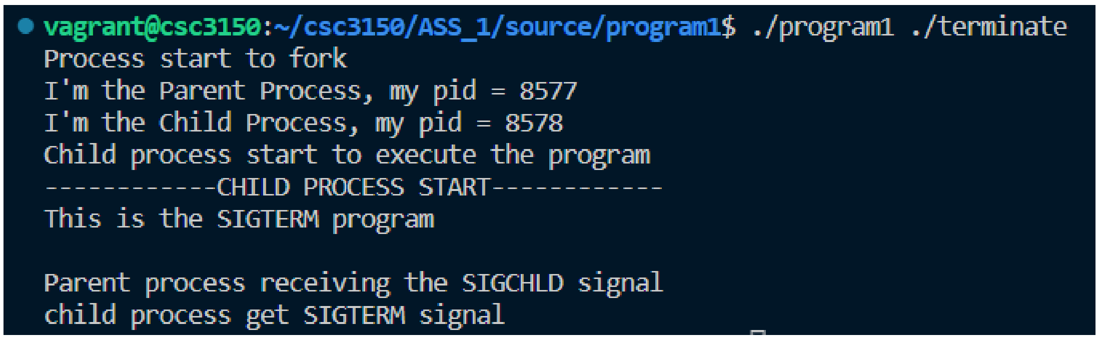
Stop:



```
● vagrant@csc3150:~/csc3150/ASS_1/source/program1$ ./program1 ./stop
  Process start to fork
  I'm the Parent Process, my pid = 8510
  I'm the Child Process, my pid = 8511
  Child process start to execute the program
  ------------CHILD PROCESS START------------
  This is the SIGSTOP program

  Parent process receiving the SIGCHLD signal
  child process get SIGSTOP signal
```

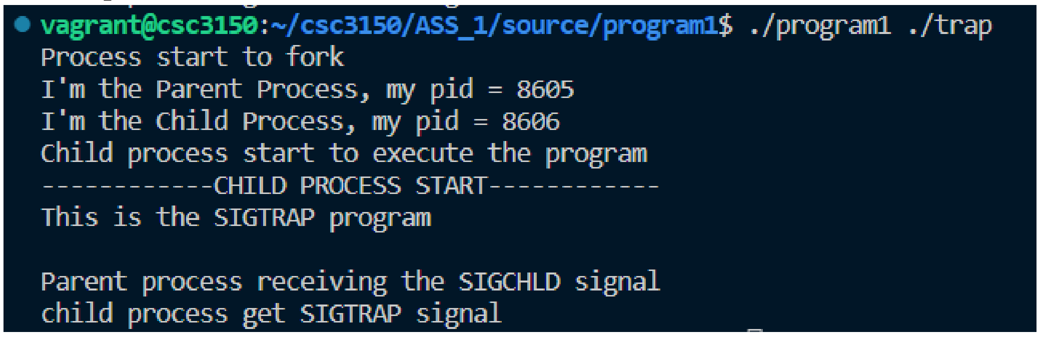
Terminate:



```
● vagrant@csc3150:~/csc3150/ASS_1/source/program1$ ./program1 ./terminate
  Process start to fork
  I'm the Parent Process, my pid = 8577
  I'm the Child Process, my pid = 8578
  Child process start to execute the program
  ------------CHILD PROCESS START------------
  This is the SIGTERM program

  Parent process receiving the SIGCHLD signal
  child process get SIGTERM signal
```

Trap:

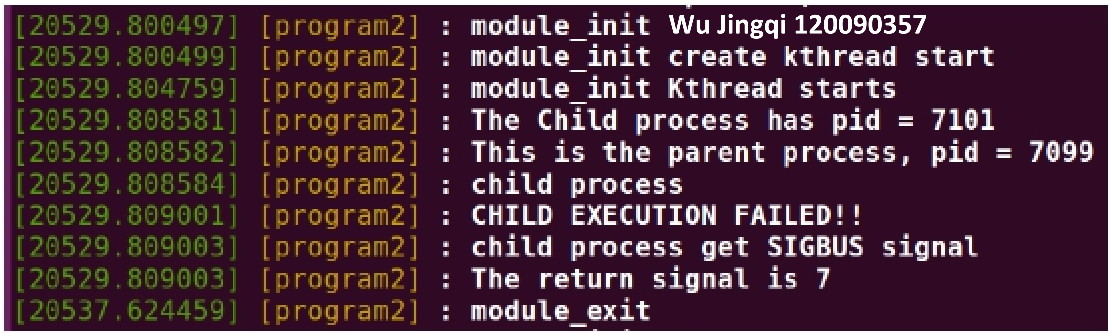


```
● vagrant@csc3150:~/csc3150/ASS_1/source/program1$ ./program1 ./trap
  Process start to fork
  I'm the Parent Process, my pid = 8605
  I'm the Child Process, my pid = 8606
  Child process start to execute the program
  ------------CHILD PROCESS START------------
  This is the SIGTRAP program

  Parent process receiving the SIGCHLD signal
  child process get SIGTRAP signal
```

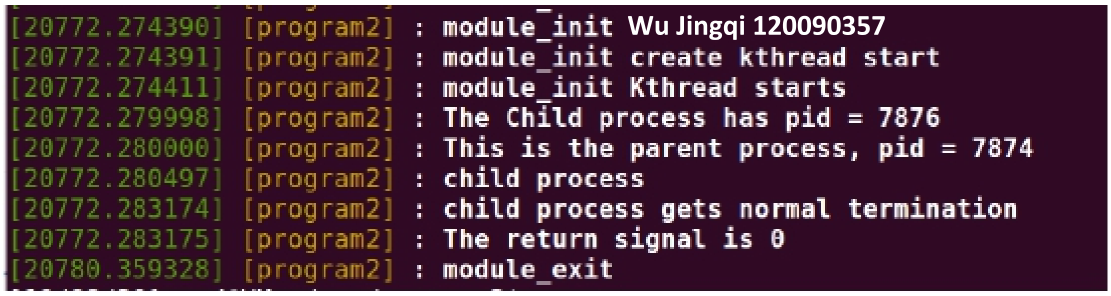
(2) Program 2:

1. test.c



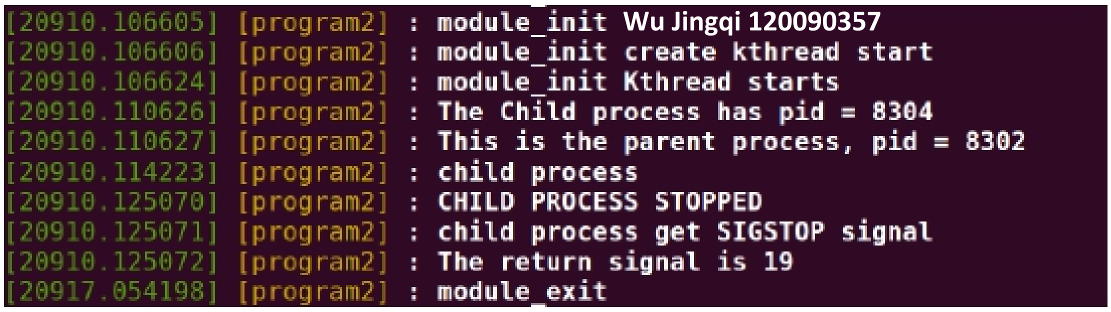
```
[20529.800497] [program2] : module_init Wu Jingqi 120090357
[20529.800499] [program2] : module_init create kthread start
[20529.804759] [program2] : module_init Kthread starts
[20529.808581] [program2] : The Child process has pid = 7101
[20529.808582] [program2] : This is the parent process, pid = 7099
[20529.808584] [program2] : child process
[20529.809001] [program2] : CHILD EXECUTION FAILED!!
[20529.809003] [program2] : child process get SIGBUS signal
[20529.809003] [program2] : The return signal is 7
[20537.624459] [program2] : module_exit
```

2. normal



```
[20772.274390] [program2] : module_init Wu Jingqi 120090357
[20772.274391] [program2] : module_init create kthread start
[20772.274411] [program2] : module_init Kthread starts
[20772.279998] [program2] : The Child process has pid = 7876
[20772.280000] [program2] : This is the parent process, pid = 7874
[20772.280497] [program2] : child process
[20772.283174] [program2] : child process gets normal termination
[20772.283175] [program2] : The return signal is 0
[20780.359328] [program2] : module_exit
```

3. stop



```
[20910.106605] [program2] : module_init Wu Jingqi 120090357
[20910.106606] [program2] : module_init create kthread start
[20910.106624] [program2] : module_init Kthread starts
[20910.110626] [program2] : The Child process has pid = 8304
[20910.110627] [program2] : This is the parent process, pid = 8302
[20910.114223] [program2] : child process
[20910.125070] [program2] : CHILD PROCESS STOPPED
[20910.125071] [program2] : child process get SIGSTOP signal
[20910.125072] [program2] : The return signal is 19
[20917.054198] [program2] : module_exit
```

4. What did I learn from the tasks?

(1) Program1

First, I learnt how to create child processes and how to use them to do some task. I also learnt how to execute other programs in the process using execve( ) function. Then, I also got some feelings about the terminated signals and know some basic knowledge of them.

(2) Program2

In this project, I learnt how to modify the kernel files and recompile the kernel to use it. I also learnt how to insert and remove modules to kernels.