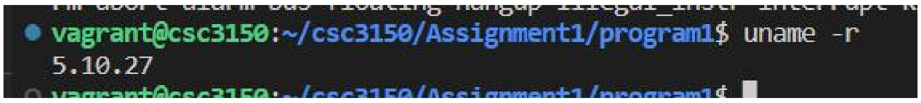
CSC3150 Assignment1 Report

Student Name: Shanjun Xie

Student ID: 120090745

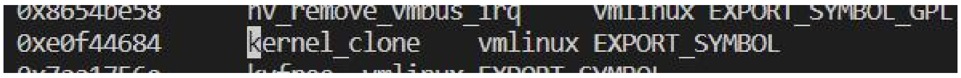
Setting up environment

This assignment must be execute on Linux 5.10.X and above. I downloaded the kernel source code and compile the kernel. After this step, my Linux version had been successfully updated to 5.10.27.

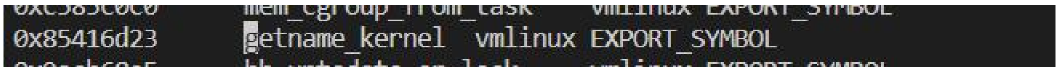


```
vagrant@csc3150:~/csc3150/Assignment1/program1$ uname -r
5.10.27
```

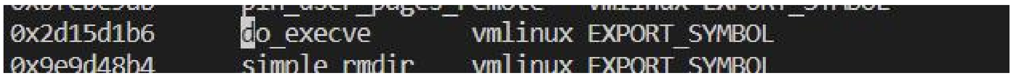
To use some functions in kernel mode, the functions have to be exported first. I used EXPORT_SYMBOL() function to export these functions in the kernel file, then recompile the kernel to make the changes available, as shown below:



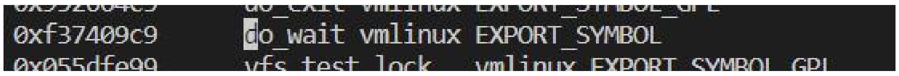
```
0xe0f44684      kernel_clone    vmlinux EXPORT_SYMBOL
```



```
0x85416d23      getname_kernel  vmlinux EXPORT_SYMBOL
```



```
0x2d15d1b6      do_execve       vmlinux EXPORT_SYMBOL
0x9e9d48b4      simple_rmdir    vmlinux EXPORT_SYMBOL
```



```
0xf37409c9      do_wait vmlinux EXPORT_SYMBOL
```

Program 1

1. The purpose of the program

This program is run in user mode. This program forks a child process to execute specific test programs, then gets the pid of both child process and parent process, and catches the return signal of the test programs, which will be used to determine what kind of signal(SIGABRT, SIGBUS, etc.) the child process sends. The information will be printed on the terminal.

2. Program design

Fork a child process: This program forks a child process using function fork().

Get pid of child and parent process: This program gets pid with function getpid(), and decides whether current process is child process or parent process using the return value of fork().

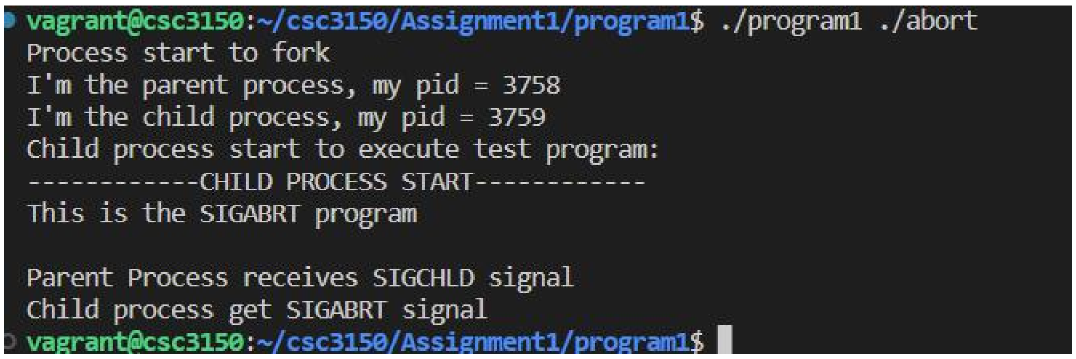
Execute the test program: This program uses execve() function to execute specific test program.

Waiting the child program to terminate: The parent process will be blocked by waitpid() function until the child process finishes running.

Determine the type of the signal: The waitpid() function will return a value "status", which is the return signal of the child process. This program then can determine specific signal type with this return value.

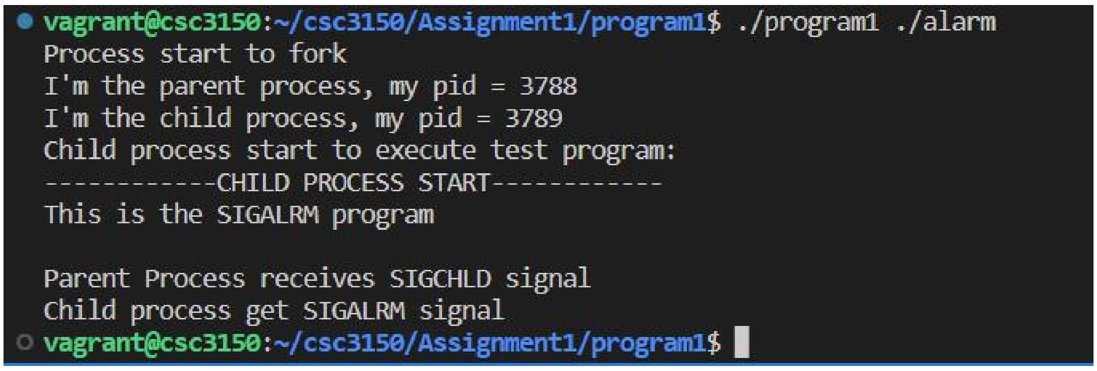
3. Sample output

Abort:



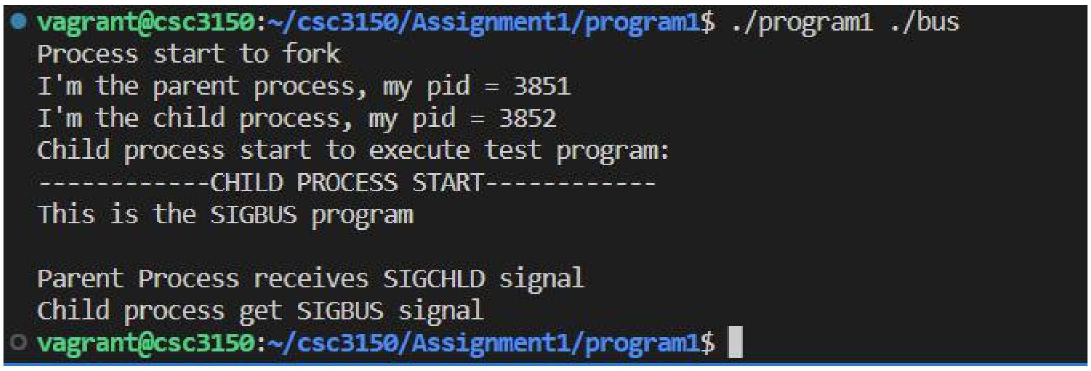
```
vagrant@csc3150:~/csc3150/Assignment1/program1$ ./program1 ./abort
Process start to fork
I'm the parent process, my pid = 3758
I'm the child process, my pid = 3759
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGABRT program

Parent Process receives SIGCHLD signal
Child process get SIGABRT signal
vagrant@csc3150:~/csc3150/Assignment1/program1$ 
```

Alarm:



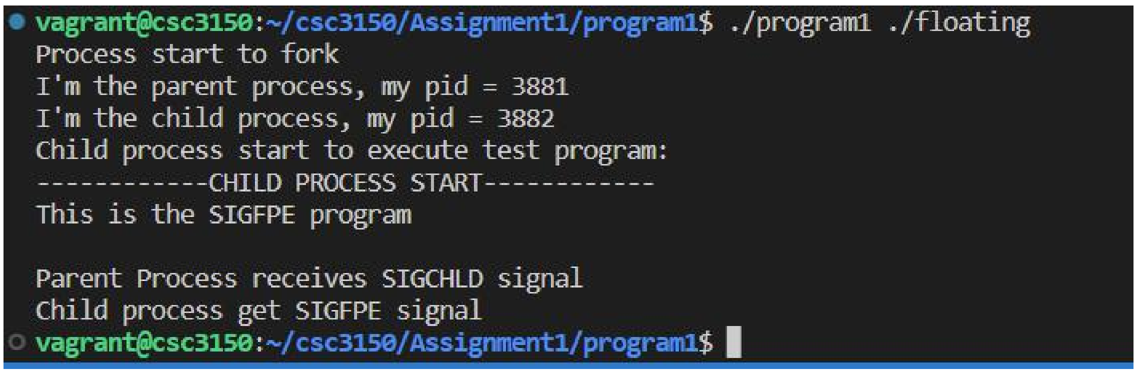
```
vagrant@csc3150:~/csc3150/Assignment1/program1$ ./program1 ./alarm
Process start to fork
I'm the parent process, my pid = 3788
I'm the child process, my pid = 3789
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGALRM program

Parent Process receives SIGCHLD signal
Child process get SIGALRM signal
vagrant@csc3150:~/csc3150/Assignment1/program1$ 
```

Bus:



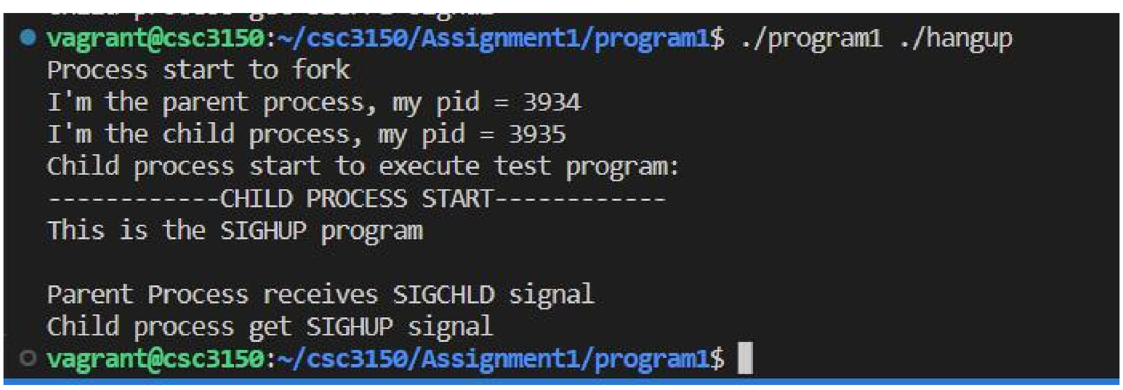
```
vagrant@csc3150:~/csc3150/Assignment1/program1$ ./program1 ./bus
Process start to fork
I'm the parent process, my pid = 3851
I'm the child process, my pid = 3852
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGBUS program

Parent Process receives SIGCHLD signal
Child process get SIGBUS signal
vagrant@csc3150:~/csc3150/Assignment1/program1$ 
```

Floating:



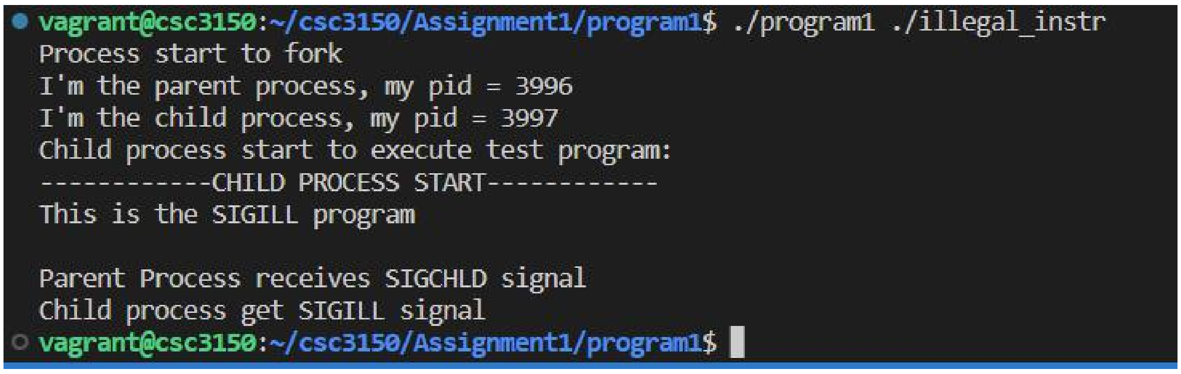
```
vagrant@csc3150:~/csc3150/Assignment1/program1$ ./program1 ./floating
Process start to fork
I'm the parent process, my pid = 3881
I'm the child process, my pid = 3882
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGFPE program

Parent Process receives SIGCHLD signal
Child process get SIGFPE signal
vagrant@csc3150:~/csc3150/Assignment1/program1$ 
```

Hang up:



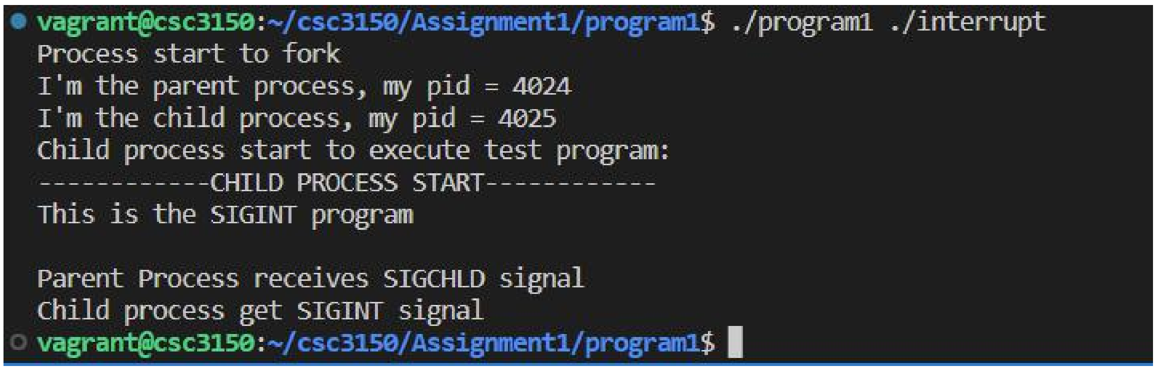
```
vagrant@csc3150:~/csc3150/Assignment1/program1$ ./program1 ./hangup
Process start to fork
I'm the parent process, my pid = 3934
I'm the child process, my pid = 3935
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGHUP program

Parent Process receives SIGCHLD signal
Child process get SIGHUP signal
vagrant@csc3150:~/csc3150/Assignment1/program1$ 
```

Illegal_instr:



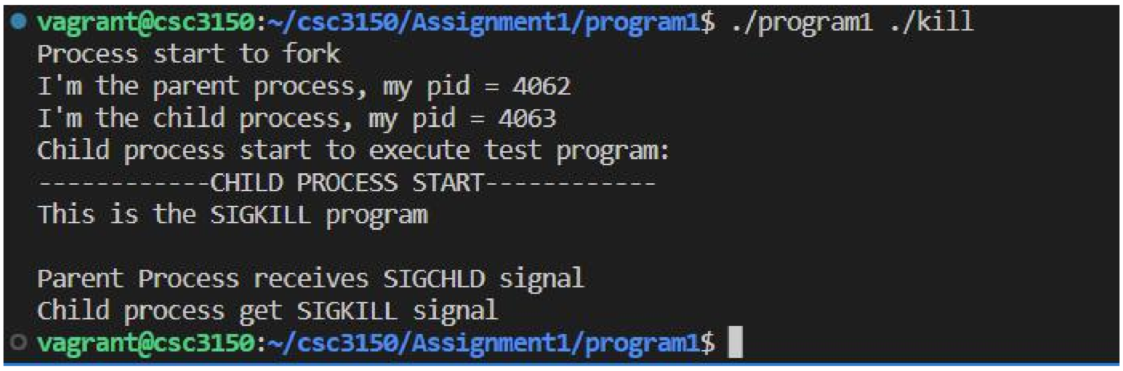
```
vagrant@csc3150:~/csc3150/Assignment1/program1$ ./program1 ./illegal_instr
Process start to fork
I'm the parent process, my pid = 3996
I'm the child process, my pid = 3997
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGILL program

Parent Process receives SIGCHLD signal
Child process get SIGILL signal
vagrant@csc3150:~/csc3150/Assignment1/program1$ 
```

Interrupt:



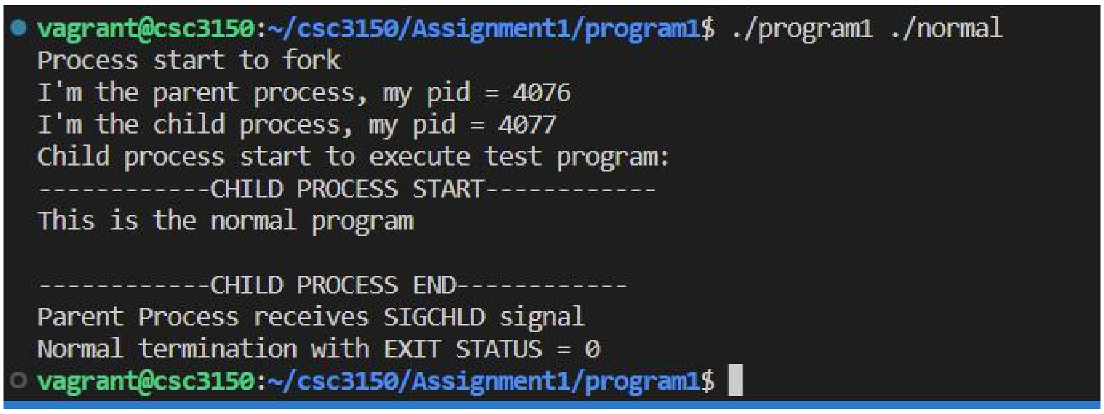
```
vagrant@csc3150:~/csc3150/Assignment1/program1$ ./program1 ./interrupt
Process start to fork
I'm the parent process, my pid = 4024
I'm the child process, my pid = 4025
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGINT program

Parent Process receives SIGCHLD signal
Child process get SIGINT signal
vagrant@csc3150:~/csc3150/Assignment1/program1$ 
```

Kill:



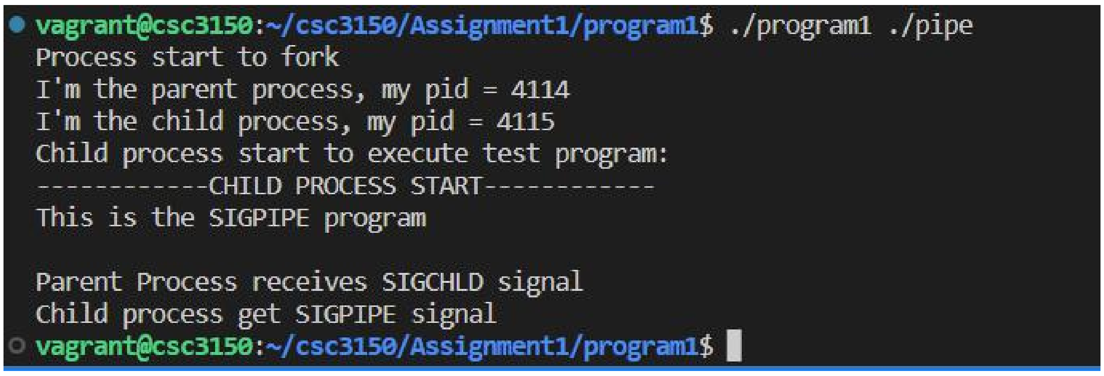
```
vagrant@csc3150:~/csc3150/Assignment1/program1$ ./program1 ./kill
Process start to fork
I'm the parent process, my pid = 4062
I'm the child process, my pid = 4063
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGKILL program

Parent Process receives SIGCHLD signal
Child process get SIGKILL signal
vagrant@csc3150:~/csc3150/Assignment1/program1$ 
```

Normal:

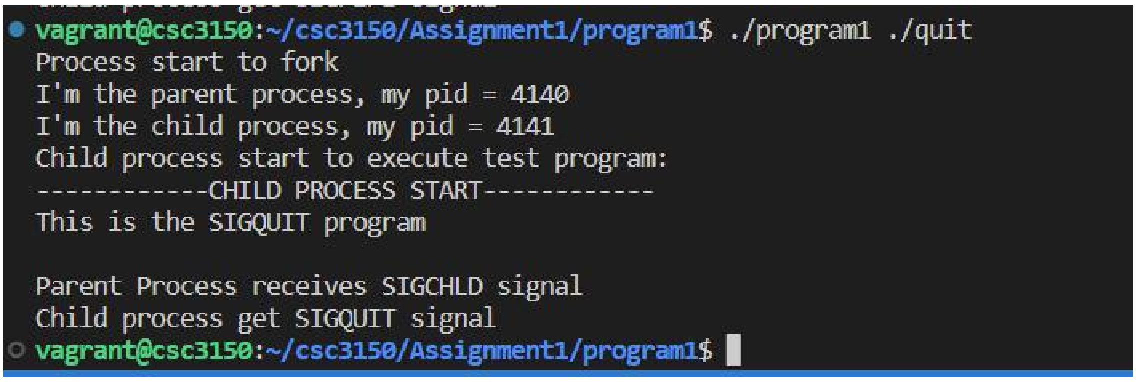


```
vagrant@csc3150:~/csc3150/Assignment1/program1$ ./program1 ./normal
Process start to fork
I'm the parent process, my pid = 4076
I'm the child process, my pid = 4077
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the normal program

------------CHILD PROCESS END------------
Parent Process receives SIGCHLD signal
Normal termination with EXIT STATUS = 0
vagrant@csc3150:~/csc3150/Assignment1/program1$ 
```

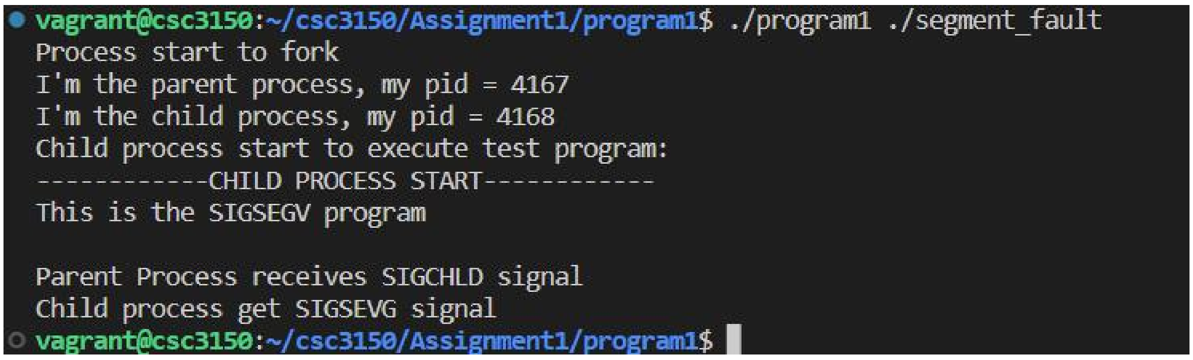
Pipe:


```
vagrant@csc3150:~/csc3150/Assignment1/program1$ ./program1 ./pipe
Process start to fork
I'm the parent process, my pid = 4114
I'm the child process, my pid = 4115
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
vagrant@csc3150:~/csc3150/Assignment1/program1$
```

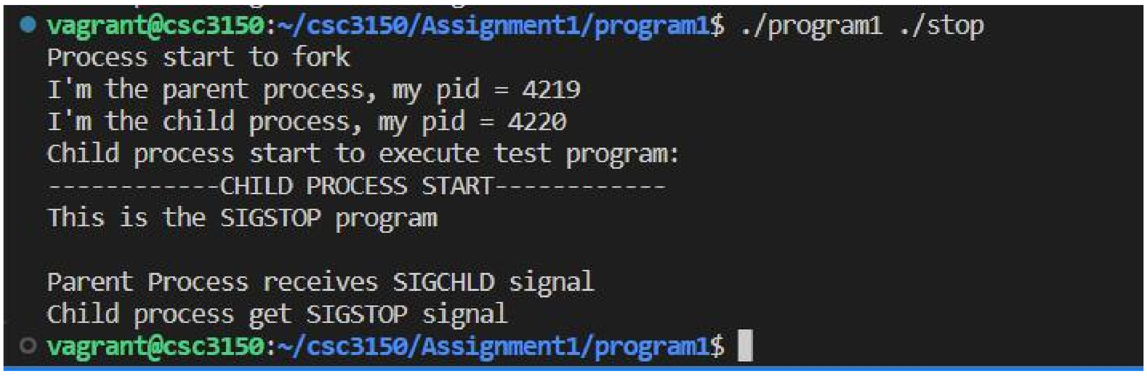
Quit:


```
vagrant@csc3150:~/csc3150/Assignment1/program1$ ./program1 ./quit
Process start to fork
I'm the parent process, my pid = 4140
I'm the child process, my pid = 4141
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
vagrant@csc3150:~/csc3150/Assignment1/program1$
```

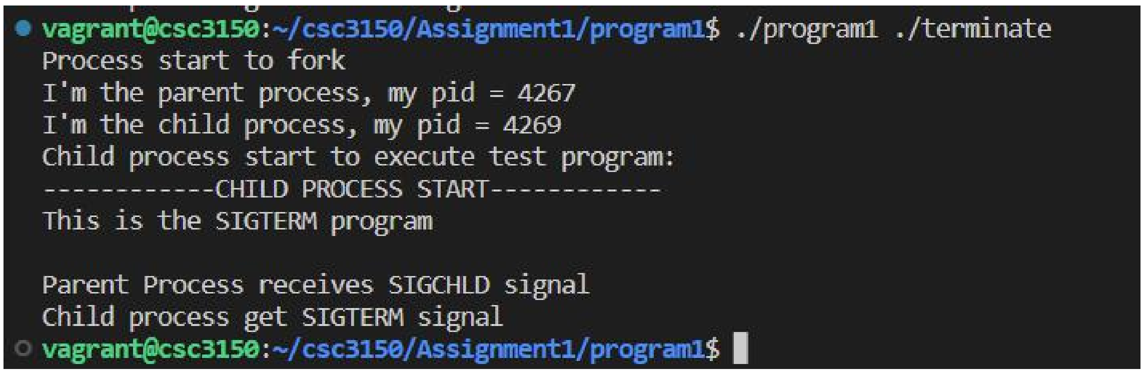
Segment_fault:


```
vagrant@csc3150:~/csc3150/Assignment1/program1$ ./program1 ./segment_fault
Process start to fork
I'm the parent process, my pid = 4167
I'm the child process, my pid = 4168
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGSEGV program

Parent Process receives SIGCHLD signal
Child process get SIGSEVG signal
vagrant@csc3150:~/csc3150/Assignment1/program1$
```

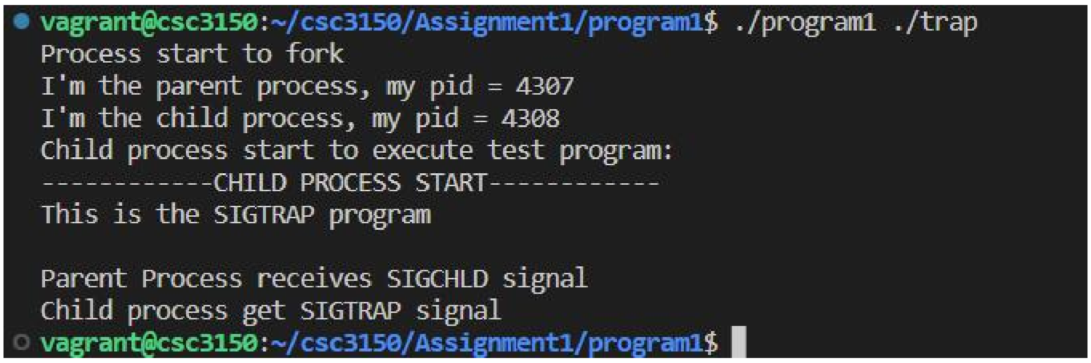
Stop:


```
vagrant@csc3150:~/csc3150/Assignment1/program1$ ./program1 ./stop
Process start to fork
I'm the parent process, my pid = 4219
I'm the child process, my pid = 4220
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
vagrant@csc3150:~/csc3150/Assignment1/program1$
```

Terminated:


```
vagrant@csc3150:~/csc3150/Assignment1/program1$ ./program1 ./terminate
Process start to fork
I'm the parent process, my pid = 4267
I'm the child process, my pid = 4269
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
vagrant@csc3150:~/csc3150/Assignment1/program1$
```

Trap:


```
vagrant@csc3150:~/csc3150/Assignment1/program1$ ./program1 ./trap
Process start to fork
I'm the parent process, my pid = 4307
I'm the child process, my pid = 4308
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
vagrant@csc3150:~/csc3150/Assignment1/program1$
```

4. What I learned from the program

This program made me understand how to fork a child process and using wait function to control the running order of child process and parent process in user mode. I also learned what pid is and how to catch the return signal of child process.

Program 2

1. The purpose of the program

This program is a LKM program. This program creates a kernel thread and forks a child process to run specific test programs directly in kernel mode. Similar to program 1, this program also gets the pid of both child process and parent process, and then catches the return status of the child process and determine what type of return signal is with the return value after the child process terminates.

2. Program design

Create a new kernel thread: To directly create a new kernel thread, this program uses function kthread_create().

Fork a child process: This program uses kernel_clone() to fork a child process.

Get pid of child process and parent process: the return value of the kernel_clone() is the pid of the child process. To ge the pid of the parent process, this function evokes function find_get_pid() to get the structure “pid” of the parent process, then get parent pid with this structure.

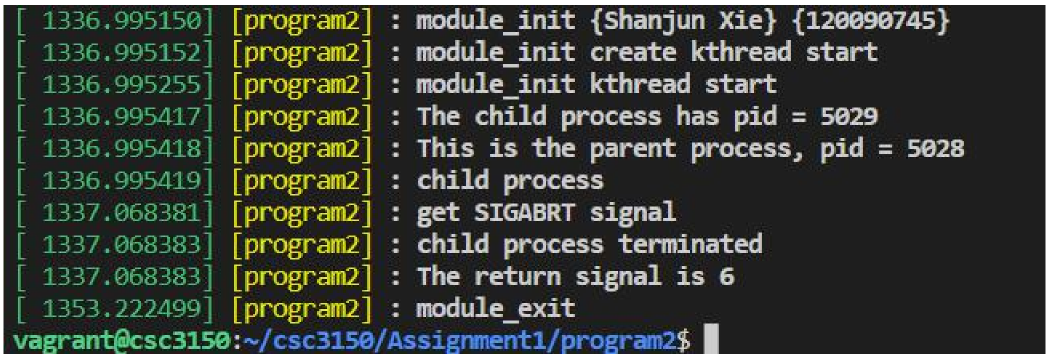
Execute the test programs: This function first gets the file name with function getname_kernel(), then evokes do_execve() to run the test program.

Wait until the child process terminates: This function uses function do_wait() to block the parent process until the child process finishes running.

Return signal and determine the type of the child signal: This function gets return value of the do_wait function, and then uses the structure wait_opts and functions such as __WEXITED() to get the actual return signal, with which can this program determine what type of child signal is.

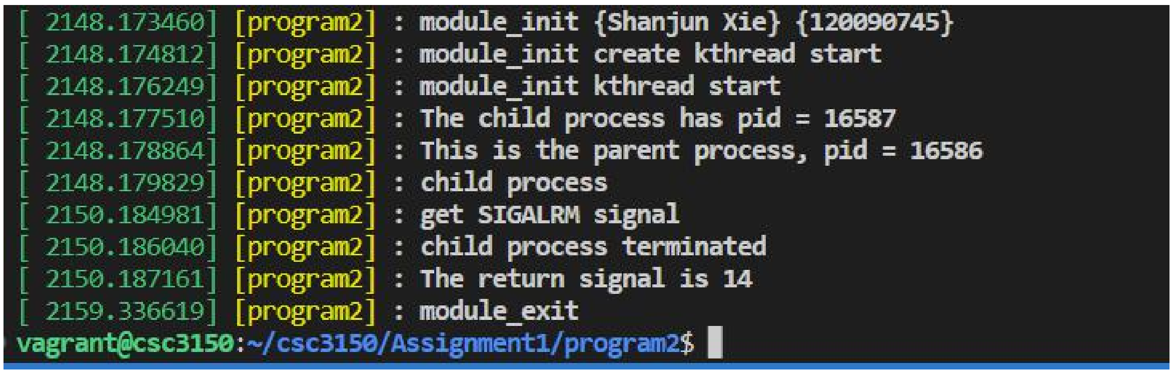
3. Sample output

Abort:

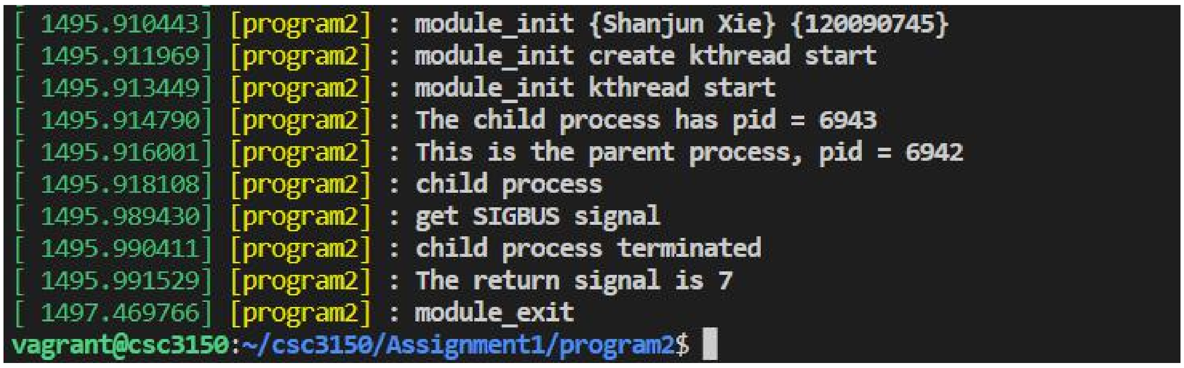


```
[ 1336.995150] [program2] : module_init {Shanjun Xie} {120090745}
[ 1336.995152] [program2] : module_init create kthread start
[ 1336.995255] [program2] : module_init kthread start
[ 1336.995417] [program2] : The child process has pid = 5029
[ 1336.995418] [program2] : This is the parent process, pid = 5028
[ 1336.995419] [program2] : child process
[ 1337.068381] [program2] : get SIGABRT signal
[ 1337.068383] [program2] : child process terminated
[ 1337.068383] [program2] : The return signal is 6
[ 1353.222499] [program2] : module_exit
vagrant@csc3150:~/csc3150/Assignment1/program2$ 
```

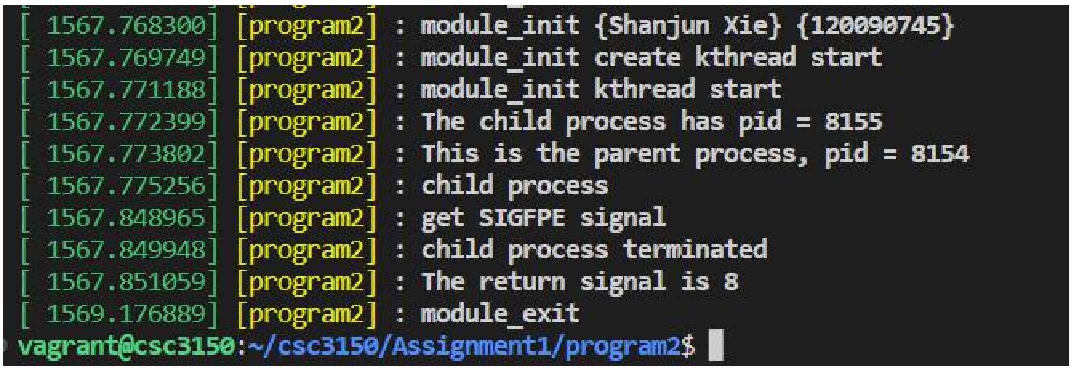
Alarm:


```
[ 2148.173460] [program2] : module_init {Shanjun Xie} {120090745}
[ 2148.174812] [program2] : module_init create kthread start
[ 2148.176249] [program2] : module_init kthread start
[ 2148.177510] [program2] : The child process has pid = 16587
[ 2148.178864] [program2] : This is the parent process, pid = 16586
[ 2148.179829] [program2] : child process
[ 2150.184981] [program2] : get SIGALRM signal
[ 2150.186040] [program2] : child process terminated
[ 2150.187161] [program2] : The return signal is 14
[ 2159.336619] [program2] : module_exit
vagrant@csc3150:~/csc3150/Assignment1/program2$
```

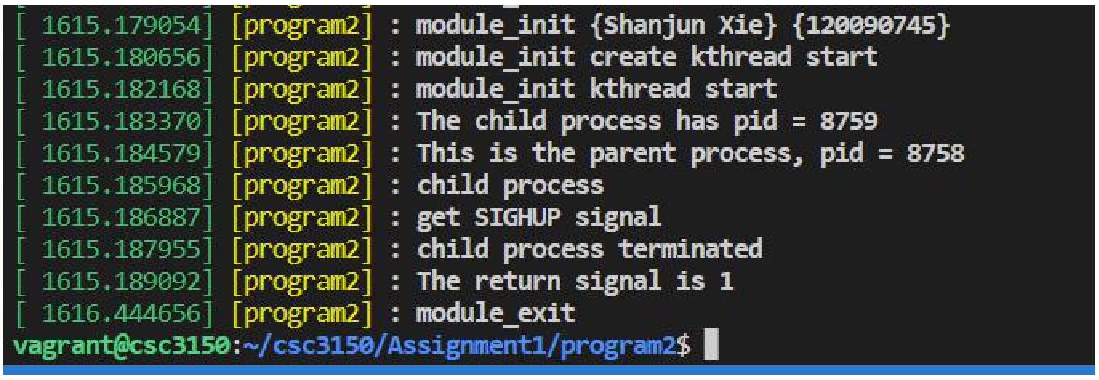
Bus:


```
[ 1495.910443] [program2] : module_init {Shanjun Xie} {120090745}
[ 1495.911969] [program2] : module_init create kthread start
[ 1495.913449] [program2] : module_init kthread start
[ 1495.914790] [program2] : The child process has pid = 6943
[ 1495.916001] [program2] : This is the parent process, pid = 6942
[ 1495.918108] [program2] : child process
[ 1495.989430] [program2] : get SIGBUS signal
[ 1495.990411] [program2] : child process terminated
[ 1495.991529] [program2] : The return signal is 7
[ 1497.469766] [program2] : module_exit
vagrant@csc3150:~/csc3150/Assignment1/program2$
```

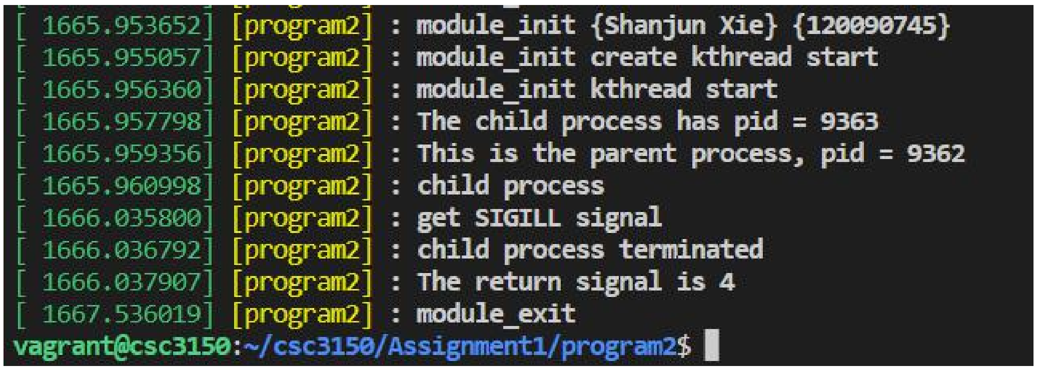
Floating:


```
[ 1567.768300] [program2] : module_init {Shanjun Xie} {120090745}
[ 1567.769749] [program2] : module_init create kthread start
[ 1567.771188] [program2] : module_init kthread start
[ 1567.772399] [program2] : The child process has pid = 8155
[ 1567.773802] [program2] : This is the parent process, pid = 8154
[ 1567.775256] [program2] : child process
[ 1567.848965] [program2] : get SIGFPE signal
[ 1567.849948] [program2] : child process terminated
[ 1567.851059] [program2] : The return signal is 8
[ 1569.176889] [program2] : module_exit
vagrant@csc3150:~/csc3150/Assignment1/program2$
```

Hangup:

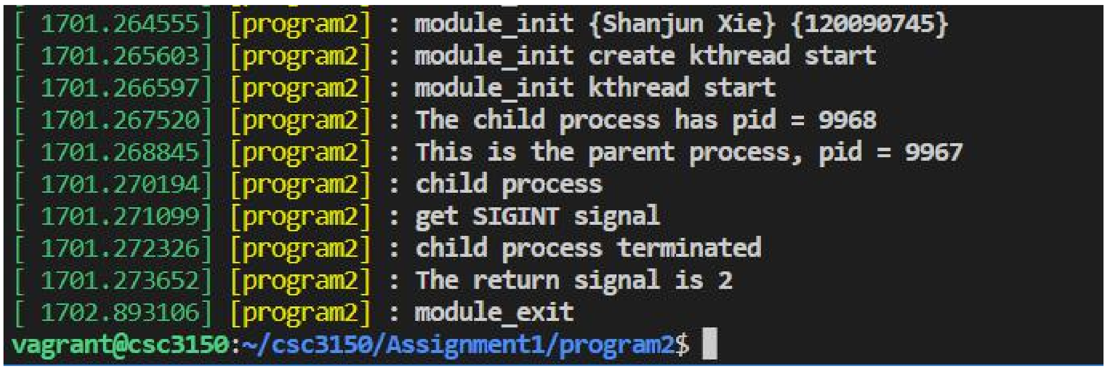

```
[ 1615.179054] [program2] : module_init {Shanjun Xie} {120090745}
[ 1615.180656] [program2] : module_init create kthread start
[ 1615.182168] [program2] : module_init kthread start
[ 1615.183370] [program2] : The child process has pid = 8759
[ 1615.184579] [program2] : This is the parent process, pid = 8758
[ 1615.185968] [program2] : child process
[ 1615.186887] [program2] : get SIGHUP signal
[ 1615.187955] [program2] : child process terminated
[ 1615.189092] [program2] : The return signal is 1
[ 1616.444656] [program2] : module_exit
vagrant@csc3150:~/csc3150/Assignment1/program2$
```

Illegal_instr:



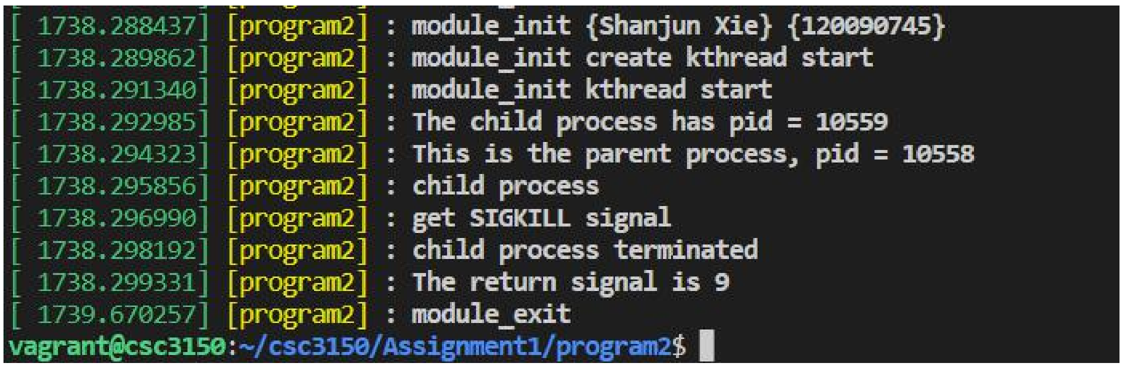
```
[ 1665.953652] [program2] : module_init {Shanjun Xie} {120090745}
[ 1665.955057] [program2] : module_init create kthread start
[ 1665.956360] [program2] : module_init kthread start
[ 1665.957798] [program2] : The child process has pid = 9363
[ 1665.959356] [program2] : This is the parent process, pid = 9362
[ 1665.960998] [program2] : child process
[ 1666.035800] [program2] : get SIGILL signal
[ 1666.036792] [program2] : child process terminated
[ 1666.037907] [program2] : The return signal is 4
[ 1667.536019] [program2] : module_exit
vagrant@csc3150:~/csc3150/Assignment1/program2$
```

Interrupt:



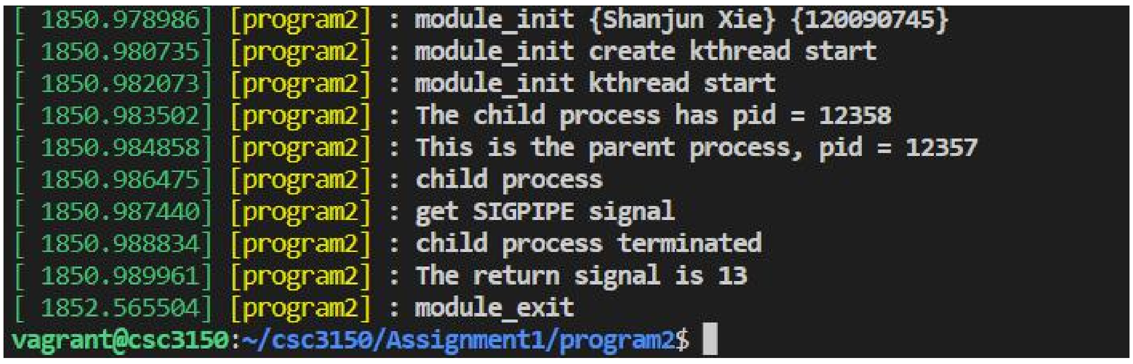
```
[ 1701.264555] [program2] : module_init {Shanjun Xie} {120090745}
[ 1701.265603] [program2] : module_init create kthread start
[ 1701.266597] [program2] : module_init kthread start
[ 1701.267520] [program2] : The child process has pid = 9968
[ 1701.268845] [program2] : This is the parent process, pid = 9967
[ 1701.270194] [program2] : child process
[ 1701.271099] [program2] : get SIGINT signal
[ 1701.272326] [program2] : child process terminated
[ 1701.273652] [program2] : The return signal is 2
[ 1702.893106] [program2] : module_exit
vagrant@csc3150:~/csc3150/Assignment1/program2$
```

Kill:



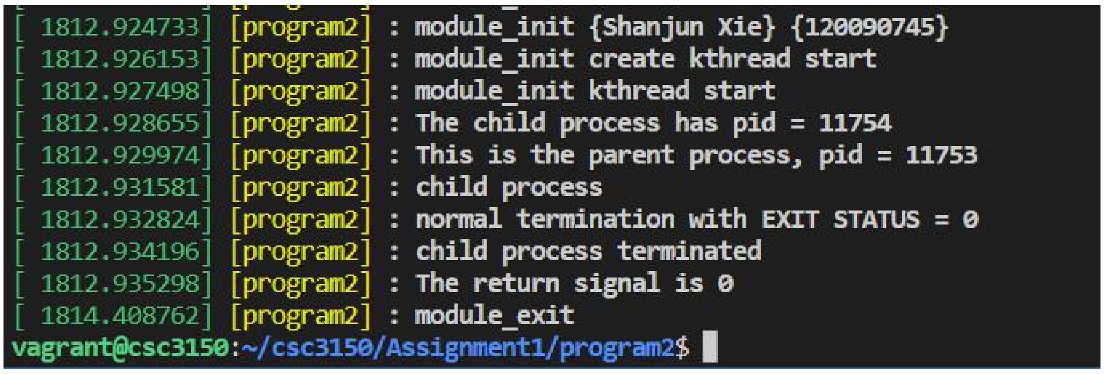
```
[ 1738.288437] [program2] : module_init {Shanjun Xie} {120090745}
[ 1738.289862] [program2] : module_init create kthread start
[ 1738.291340] [program2] : module_init kthread start
[ 1738.292985] [program2] : The child process has pid = 10559
[ 1738.294323] [program2] : This is the parent process, pid = 10558
[ 1738.295856] [program2] : child process
[ 1738.296990] [program2] : get SIGKILL signal
[ 1738.298192] [program2] : child process terminated
[ 1738.299331] [program2] : The return signal is 9
[ 1739.670257] [program2] : module_exit
vagrant@csc3150:~/csc3150/Assignment1/program2$
```

Pipe:

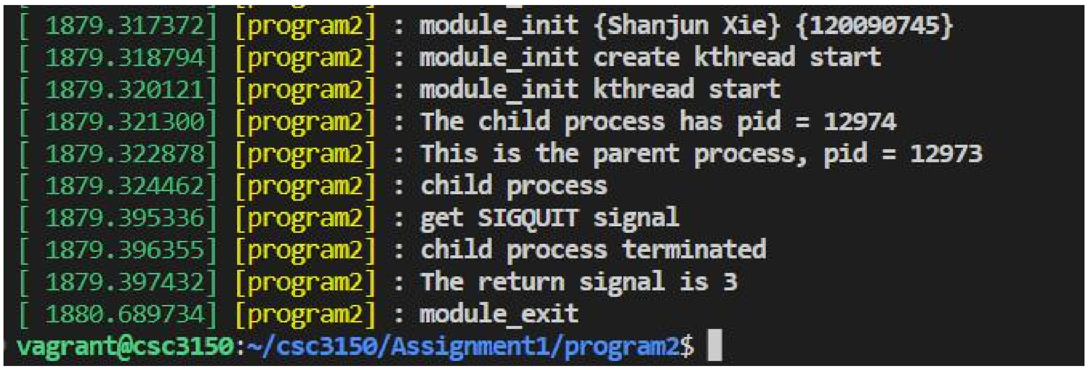


```
[ 1850.978986] [program2] : module_init {Shanjun Xie} {120090745}
[ 1850.980735] [program2] : module_init create kthread start
[ 1850.982073] [program2] : module_init kthread start
[ 1850.983502] [program2] : The child process has pid = 12358
[ 1850.984858] [program2] : This is the parent process, pid = 12357
[ 1850.986475] [program2] : child process
[ 1850.987440] [program2] : get SIGPIPE signal
[ 1850.988834] [program2] : child process terminated
[ 1850.989961] [program2] : The return signal is 13
[ 1852.565504] [program2] : module_exit
vagrant@csc3150:~/csc3150/Assignment1/program2$
```

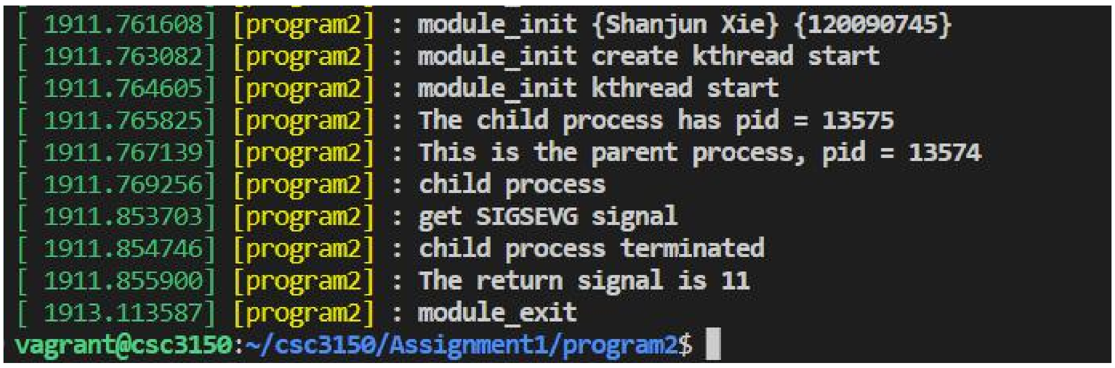
Normal:


```
[ 1812.924733] [program2] : module_init {Shanjun Xie} {120090745}
[ 1812.926153] [program2] : module_init create kthread start
[ 1812.927498] [program2] : module_init kthread start
[ 1812.928655] [program2] : The child process has pid = 11754
[ 1812.929974] [program2] : This is the parent process, pid = 11753
[ 1812.931581] [program2] : child process
[ 1812.932824] [program2] : normal termination with EXIT STATUS = 0
[ 1812.934196] [program2] : child process terminated
[ 1812.935298] [program2] : The return signal is 0
[ 1814.408762] [program2] : module_exit
vagrant@csc3150:~/csc3150/Assignment1/program2$
```

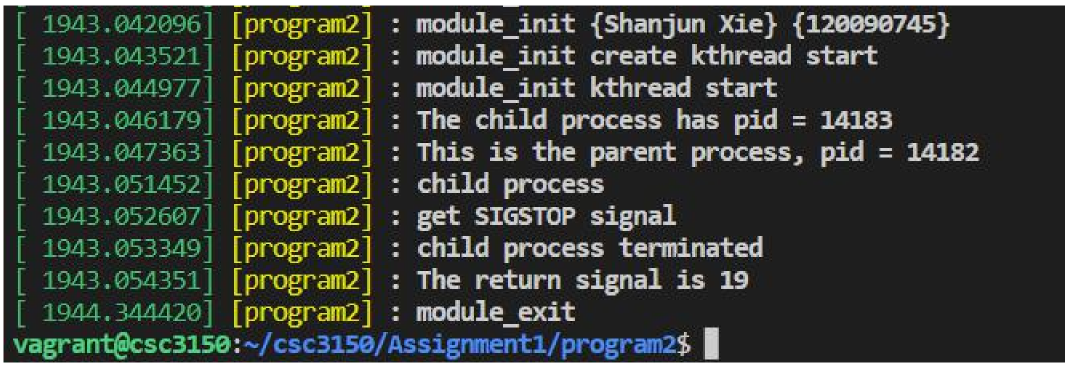
Quit:


```
[ 1879.317372] [program2] : module_init {Shanjun Xie} {120090745}
[ 1879.318794] [program2] : module_init create kthread start
[ 1879.320121] [program2] : module_init kthread start
[ 1879.321300] [program2] : The child process has pid = 12974
[ 1879.322878] [program2] : This is the parent process, pid = 12973
[ 1879.324462] [program2] : child process
[ 1879.395336] [program2] : get SIGQUIT signal
[ 1879.396355] [program2] : child process terminated
[ 1879.397432] [program2] : The return signal is 3
[ 1880.689734] [program2] : module_exit
vagrant@csc3150:~/csc3150/Assignment1/program2$
```

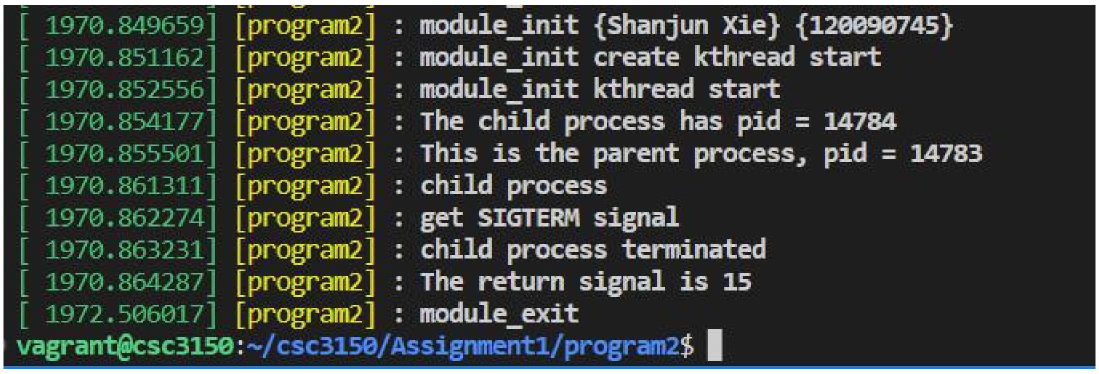
Segment_fault:


```
[ 1911.761608] [program2] : module_init {Shanjun Xie} {120090745}
[ 1911.763082] [program2] : module_init create kthread start
[ 1911.764605] [program2] : module_init kthread start
[ 1911.765825] [program2] : The child process has pid = 13575
[ 1911.767139] [program2] : This is the parent process, pid = 13574
[ 1911.769256] [program2] : child process
[ 1911.853703] [program2] : get SIGSEVG signal
[ 1911.854746] [program2] : child process terminated
[ 1911.855900] [program2] : The return signal is 11
[ 1913.113587] [program2] : module_exit
vagrant@csc3150:~/csc3150/Assignment1/program2$
```

Stop:

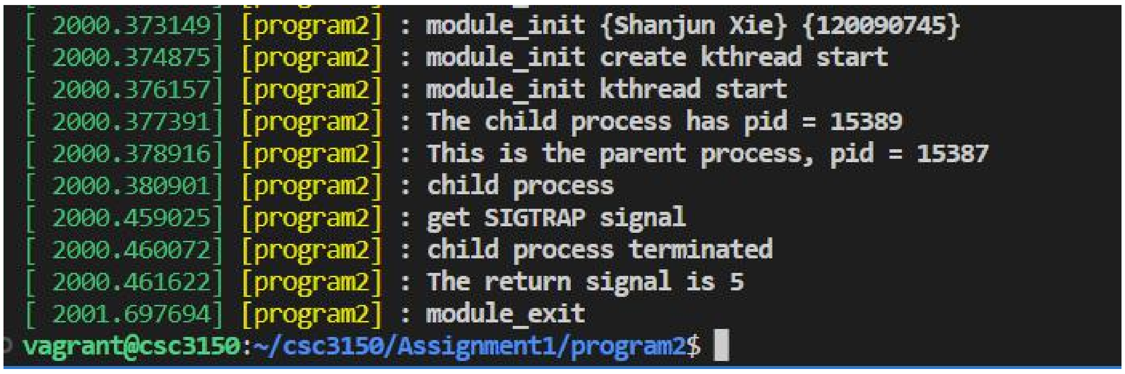

```
[ 1943.042096] [program2] : module_init {Shanjun Xie} {120090745}
[ 1943.043521] [program2] : module_init create kthread start
[ 1943.044977] [program2] : module_init kthread start
[ 1943.046179] [program2] : The child process has pid = 14183
[ 1943.047363] [program2] : This is the parent process, pid = 14182
[ 1943.051452] [program2] : child process
[ 1943.052607] [program2] : get SIGSTOP signal
[ 1943.053349] [program2] : child process terminated
[ 1943.054351] [program2] : The return signal is 19
[ 1944.344420] [program2] : module_exit
vagrant@csc3150:~/csc3150/Assignment1/program2$
```

Terminate:



```
[ 1970.849659] [program2] : module_init {Shanjun Xie} {120090745}
[ 1970.851162] [program2] : module_init create kthread start
[ 1970.852556] [program2] : module_init kthread start
[ 1970.854177] [program2] : The child process has pid = 14784
[ 1970.855501] [program2] : This is the parent process, pid = 14783
[ 1970.861311] [program2] : child process
[ 1970.862274] [program2] : get SIGTERM signal
[ 1970.863231] [program2] : child process terminated
[ 1970.864287] [program2] : The return signal is 15
[ 1972.506017] [program2] : module_exit
vagrant@csc3150:~/csc3150/Assignment1/program2$ 
```

Trap:



```
[ 2000.373149] [program2] : module_init {Shanjun Xie} {120090745}
[ 2000.374875] [program2] : module_init create kthread start
[ 2000.376157] [program2] : module_init kthread start
[ 2000.377391] [program2] : The child process has pid = 15389
[ 2000.378916] [program2] : This is the parent process, pid = 15387
[ 2000.380901] [program2] : child process
[ 2000.459025] [program2] : get SIGTRAP signal
[ 2000.460072] [program2] : child process terminated
[ 2000.461622] [program2] : The return signal is 5
[ 2001.697694] [program2] : module_exit
vagrant@csc3150:~/csc3150/Assignment1/program2$ 
```

4. What I learned from the program

I learned that we can directly operate on the kernel with LKM. To finish the task, I also needed to change some code in the kernel code and recompile the kernel, which gave me a sense about how to directly modify the Linux kernel. This program also showed how to directly create a kernel thread and fork a child process.