# CSC3150 Assignment 1 Report

Chen Zhixin, 120090222

## How I design my program?

### Program 1

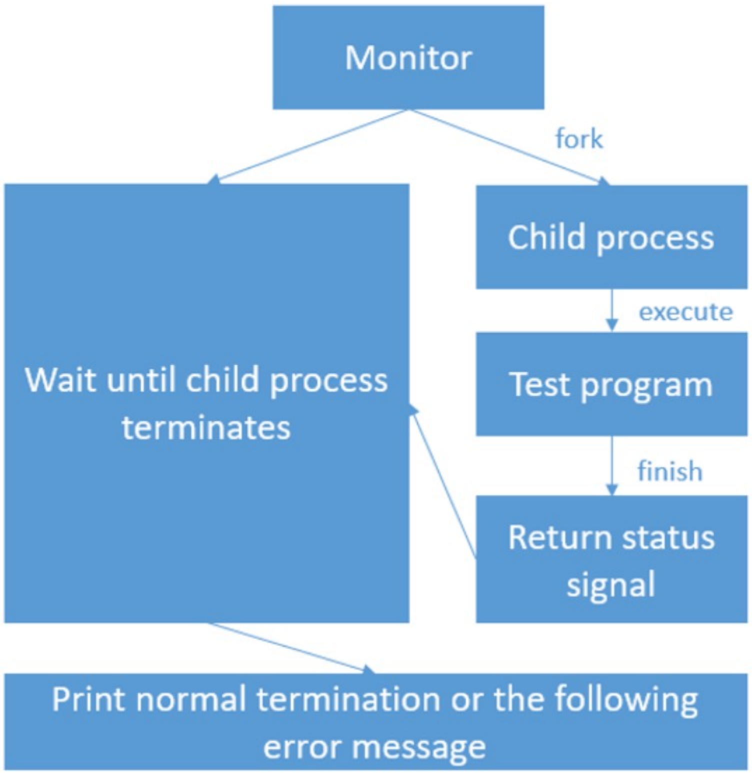
In program 1, I implemented a fork function in user mode. By referring to tutorial slides, I gained a basic understanding about how process works, and how process in user mode interact with the functions in kernel. The task is to clone a child process, let it run an executable and return a signal to the parent process. As for the parent process, all it need to do is to wait for the signal, then continue and display the signal received. There are three key functions that we need to utilize:

1. `fork()`: To folk a child process.
2. `execve()`: To run a executable with indicated path.
3. `waitpid()`: To wait for a specific process's termination, and receive signal *(raised by* `raise()` *in executable)* from it.

First, use `fork()` to folk a child process, then both the parent and child process will start executing at the next line, the only difference is the return value `pid` will be different, 0 for child process, non-zero integer for parent process. Then we use this to distinguish parent and child process, let them act differently.

Then, for the parent process, it uses `waitpid()` function to wait for the child's response. This function is blocking, i.e., It will be released only after it receives any signal from child process. Then, with the signal received, It can determine which type of signal is and display it to the user.

As for the child process, first it read the arguments passed in the command line, use it as the path of the executable, then run it with `execve()`. If the function run properly, it will never return and replace the original resource with the new executable. Then in the new one, signals are raised, received by the original parent process. Then it circle around, back to previous parent's waiting step and continue.


Monitor
fork
Child process
execute
Test program
finish
Return status signal
Wait until child process terminates
Print normal termination or the following error message

## Program 2

In program 2, I implemented a fork function in kernel mode. Similar to program 1, this time we need to fork a process, but not in user space, it's in kernel instead. Kernel environment is a little bit different than the user one. First, the way of running it is different. We need to utilize the "LKM" - or "loadable kernel modules" mechanism to load our own-written kernel codes. After compile, we need to insert the module by `insmod program2.ko`. Then the module get initialized, triggering the functions in this program.

After kernel module initializes, a kernel thread is created, in order to activate functions work in kernel space. `kthread_create()` is needed for creating a "task". And the `wake_up_process()` wake this "task" up, the function eventually runs in kernel space.

There are several key functions to be utilized in this program.

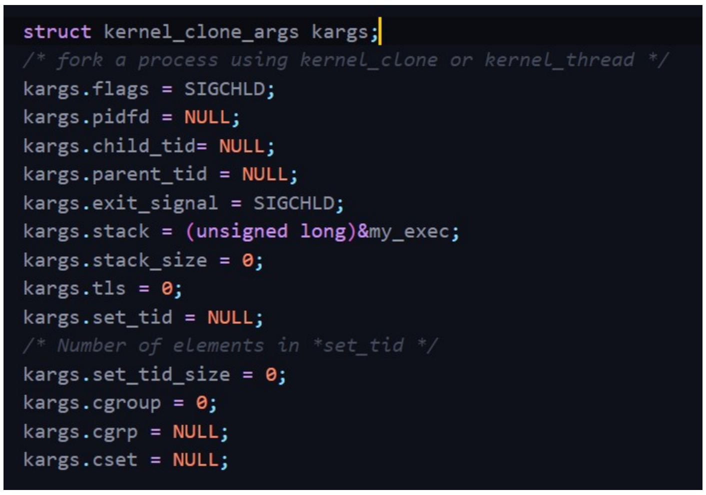
1. `do_execve()`: similar to `execve()`, but in kernel version.
2. `do_wait()`: similar to `wait()`, but in kernel version
3. `kernel_clone()`: a little bit similar to `fork()`, but in kernel version, also need to specify function to execute.
4. `getname_kernel()`: convert name in string to "filename" structure.

These functions are defined in the source file of the Linux kernel. Some of them are defined as "static", but some are not. For non-static part (`kernel_clone()` and `getname_kernel()`), we need to add `extern` on that function, and attach `EXPORT_SYMBOL(function())` below. Details will be illustrated in below parts.

For function design, I separate into three functions:

- `my_fork()`:
- `my_wait()`:
- `my_exec()`:

After the kernel thread is created, `my_fork()` function is under execution. By referring to the source code, to clone a process, we need to use `kernel_clone()`. It needs some arguments, whose structure `kernel_clone_args()` is defined in the source code. Then we create arguments to pass in:



```
struct kernel_clone_args kargs;
/* fork a process using kernel_clone or kernel_thread */
kargs.flags = SIGCHLD;
kargs.pidfd = NULL;
kargs.child_tid= NULL;
kargs.parent_tid = NULL;
kargs.exit_signal = SIGCHLD;
kargs.stack = (unsigned long)&my_exec;
kargs.stack_size = 0;
kargs.tls = 0;
kargs.set_tid = NULL;
/* Number of elements in *set_tid */
kargs.set_tid_size = 0;
kargs.cgroup = 0;
kargs.cgrp = NULL;
kargs.cset = NULL;
```

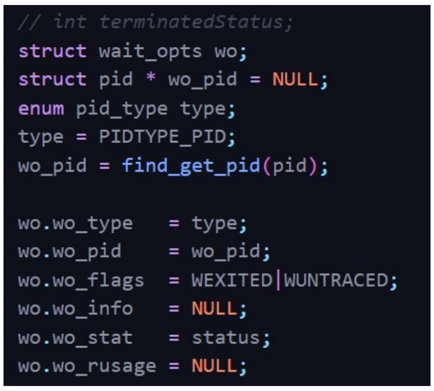
Some key points:

- .flags set to "SIGCHLD"
- .exit_signal set to "SIGCHLD"
- .stack set to "(unsigned long) & my_Exec"
- Others set to 0 or NULL, depending on it is integer or pointer.

Then we pass in the arguments. During the execution of `kernel_clone()`, actually a child process is created, running the `my_exec()` function, which will be introduced later. Then the parent process will print out the PID of both process, and wait for child process's termination by function `my_wait()`. the `my_wait()` function will return a value `status`, whose last 7 bits are exactly the standardized SIGNAL that can be analysis by us manually (We use and operation with 0x7f to get real status).

As for the `my_exec()` function run by child process, it is a bit similar to the function `execve()` in user mode. Here we indicate a string path `/tmp/test`, using function `getname_kernel()` convert it to structure "filename", latter can be passed into `do_execve()`, which runs a executable in user space and return value to be received in parent process.

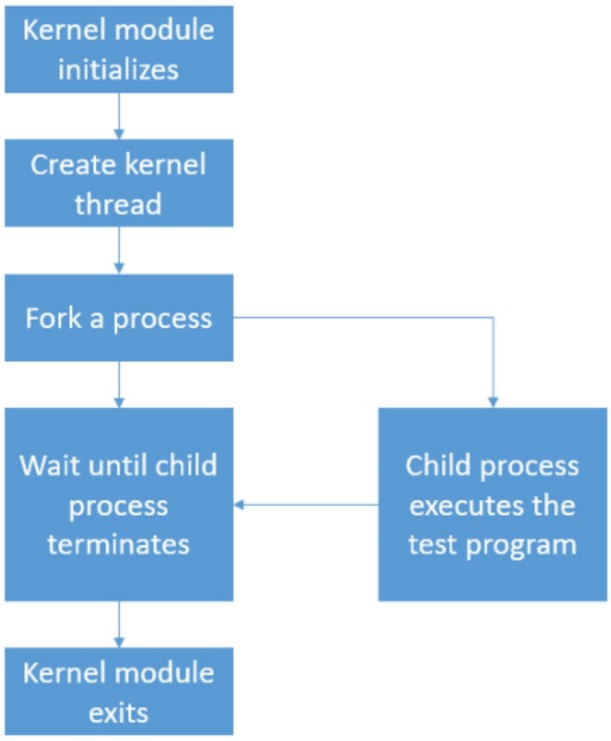
Meanwhile, parent process get into wait status with the function `my_wait()`. This function also needs a special structure called "wait_ops". We define it like this:



```
// int terminatedStatus;
struct wait_opts wo;
struct pid * wo_pid = NULL;
enum pid_type type;
type = PIDTYPE_PID;
wo_pid = find_get_pid(pid);

wo.wo_type   = type;
wo.wo_pid    = wo_pid;
wo.wo_flags  = WEXITED|WUNTRACED;
wo.wo_info   = NULL;
wo.wo_stat   = status;
wo.wo_rusage = NULL;
```

Notice that in kernel version 5.10.x, the type of `.wo_stat` has change from `int*` to `int`. So here we use an integer to initialize it. Then we pass it into the `do_execve()` function. After it receive signal from child process and release, we will get the signal back in `wo.wo_stat` and return it back to the main function. Then we utilize this to get signal, continue. The general routine is like the figure below:


Kernel module initializes
Create kernel thread
Fork a process
Wait until child process terminates
Child process executes the test program
Kernel module exits

### Bonus

In bonus, I implemented a pstree function.

## How to set up development environment?

### Prerequisites

**Make image in key points，get ready to rollback，reinstall the system if necessary.**

## Preparations

First, **enter super user(root) mode**. This applies for all the following steps.

```
sudo su
```

Install all dependencies

```
apt update && sudo apt install bc

apt-get install libncurses-dev gawk flex bison openssl libssl-dev dkms libelf-
dev libudev-dev libpci-dev libiberty-dev autoconf llvm dwarves
```

Use `cd` to find a place where you want to store source file. Make sure you have enough permission. (For example, enter super user(root) by `sudo su`, and `cd ~/` to download file on path `/root/`)
Download compressed package via wget

```
cd ~/
wget https://mirror.tuna.tsinghua.edu.cn/kernel/v5.x/linux-5.10.5.tar.xz
```

After download, the compressed package will be stored in the current folder.
Unzip it with:

```
tar xvf linux-5.10.5.tar.xz
```

Then cd into the unzipped folder and execute:

```
cd ./linux-5.10.5/
make mrproper
make clean
```

Then download the config file to the same folder via wget

```
wget https://ly-blog.oss-cn-shenzhen.aliyuncs.com/static/.config        # Expire
date: 2022/3/16
```

Then make sure you have a large-enough terminal window for the GUI of menuconfig. Enter:

```
make menuconfig
```

Then you get into a GUI. use four arrows to select, press enter to confirm.
First select "Load" -> "OK".
Back to homepage, select "Save" -> "OK".
Back to homepage, select "Exit"

Done. Now you are in command line terminal again.

## Allocate memory to 'Swap Space' (For Cloud VM only)

Some prior knowledge about Swap

Reference

All the commands should be executed in root mode as well.

```
cd /usr
mkdir swap      #create a new folder
dd if=/dev/zero of=/usr/swap/swapfile bs=1M count=4096 #Create a 4-GB memory space in SSD as virtual memory
du -sh /usr/swap/swapfile   #Check if this file occupy 4Gb
mkswap /usr/swap/swapfile
swapon /usr/swap/swapfile
```

Modify this file in vim.

```
vim /etc/fstab
```

In vim, add this line at the end of the file: (Press `o` into insertion mode starting from next new line, press `Esc` then `:wq` to save and quit vim)

```
/usr/swap/swapfile swap swap defaults 0 0
```

Then **reboot the machine**.
After reboot, you can check if the swap area is ready:

```
free -m
```

If now Swap has ~4096 free space, Done!

## Compile kernel (Choose either option)

### Option 1: In terminal

Make sure you are in root mode.

```
cd ~/linux-5.10.5/
make -j$(nproc)
```

It takes about 1~2 hrs to finish. Don't disconnect, don't close the terminal.

### Option 2: With `nohup` command (Recommended)

Reference

Make sure you are in root mode.

```
cd ~/linux-5.10.5/
nohup make -j$(nproc)       # Does not accept any intput, run in backend process, only can be killed by killing pid
```

It takes about 1~2 hrs to finish.

The command line output will be stored in ~/linux-5.10.5/nohup.out, use *vim* to inspect result. (Vim hints: `Shift + g` goes to the bottom of the file, press `:wq` to save and quit, `i` into insertion mode, `Esc` to quit, `/` to search, after search press `enter` to locate)

## Next steps

Make sure you are in root mode.

```
cd ~/linux-5.10.5/
make modules_install
make install
```

Then reboot the machine.
Enter this command to check kernel version.

```
uname -r
```

If you see 5.10.5, Now everything is done!

## Recompile & Reinstall

Modify `getname_kernel()` function:

```
vim ~/linux-5.10.5/fs/namei.c
```

then search for `struct filename * getname_kernel` (Hint: use search mode in vim as mentioned above, or line 212), add `extern` tag on it, then add `EXPORT_SYMBOL(getname_kernel);` behind (line 247), then save and quit.

Modify `kernel_clone()` function:

```
vim ~/linux-5.10.5/kernel/fork.c
```

then search for `pid_t kernel_clone(struct kernel_clone_args *args)` (Hint: use search mode in vim as mentioned above, or line 2416), add `extern` tag on it, then add `EXPORT_SYMBOL(kernel_clone);` behind (line 2495), then save and quit.

---

Then recompile & reinstall the kernel.

Please start from the step:

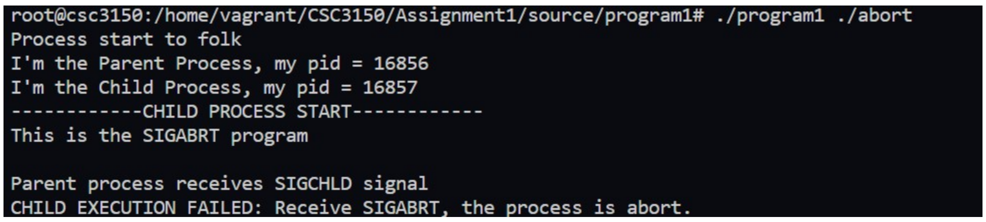
```
make -j$(nproc)
make modules_install
make install
```

Then reboot.
To save your time, don't start from `make mrproper`.

## Screenshot of the program output

### Program 1

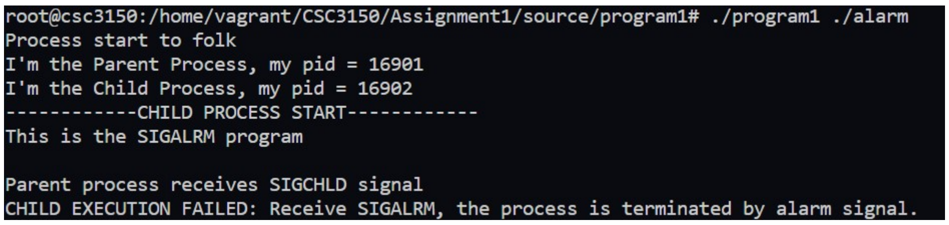
Received SIGABRT signal:



```
root@csc3150:/home/vagrant/CSC3150/Assignment1/source/program1# ./program1 ./abort
Process start to folk
I'm the Parent Process, my pid = 16856
I'm the Child Process, my pid = 16857
------------CHILD PROCESS START------------
This is the SIGABRT program

Parent process receives SIGCHLD signal
CHILD EXECUTION FAILED: Receive SIGABRT, the process is abort.
```

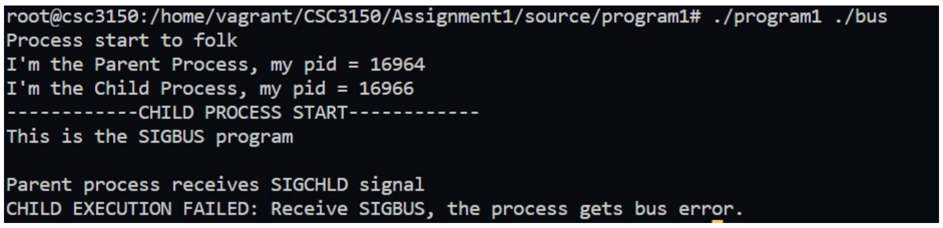
Received SIGALRM signal:



```
root@csc3150:/home/vagrant/CSC3150/Assignment1/source/program1# ./program1 ./alarm
Process start to folk
I'm the Parent Process, my pid = 16901
I'm the Child Process, my pid = 16902
------------CHILD PROCESS START------------
This is the SIGALRM program

Parent process receives SIGCHLD signal
CHILD EXECUTION FAILED: Receive SIGALRM, the process is terminated by alarm signal.
```

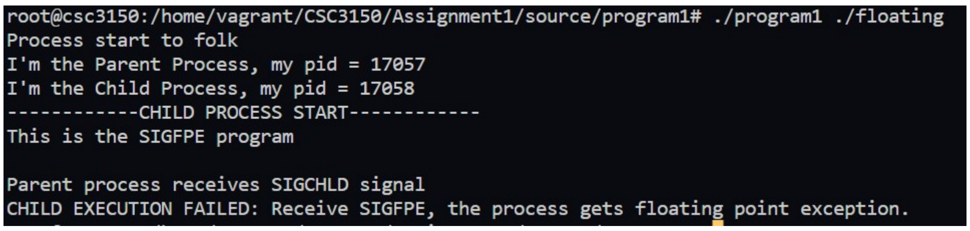
Received SIGBUS signal:



```
root@csc3150:/home/vagrant/CSC3150/Assignment1/source/program1# ./program1 ./bus
Process start to folk
I'm the Parent Process, my pid = 16964
I'm the Child Process, my pid = 16966
------------CHILD PROCESS START------------
This is the SIGBUS program

Parent process receives SIGCHLD signal
CHILD EXECUTION FAILED: Receive SIGBUS, the process gets bus error.
```

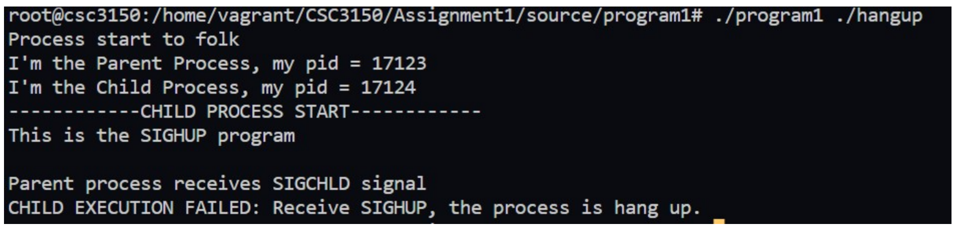
Received SIGFPE signal:



```
root@csc3150:/home/vagrant/CSC3150/Assignment1/source/program1# ./program1 ./floating
Process start to folk
I'm the Parent Process, my pid = 17057
I'm the Child Process, my pid = 17058
------------CHILD PROCESS START------------
This is the SIGFPE program

Parent process receives SIGCHLD signal
CHILD EXECUTION FAILED: Receive SIGFPE, the process gets floating point exception.
```

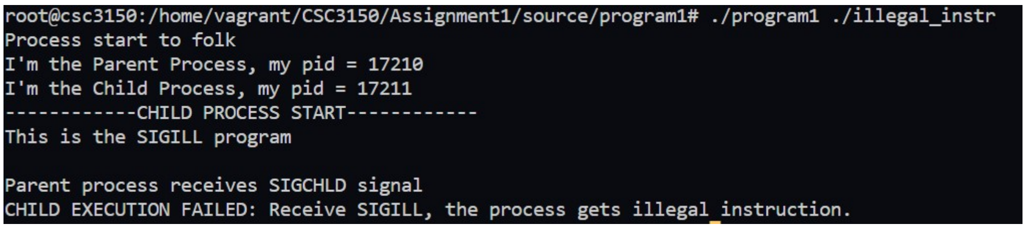
Received SIGHUP signal:



```
root@csc3150:/home/vagrant/CSC3150/Assignment1/source/program1# ./program1 ./hangup
Process start to folk
I'm the Parent Process, my pid = 17123
I'm the Child Process, my pid = 17124
------------CHILD PROCESS START------------
This is the SIGHUP program

Parent process receives SIGCHLD signal
CHILD EXECUTION FAILED: Receive SIGHUP, the process is hang up.
```

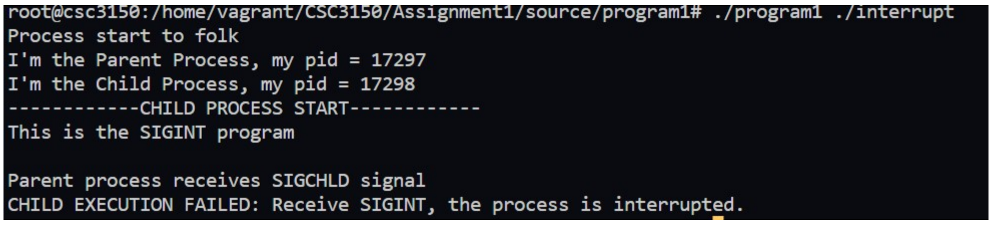
Received SIGILL signal:



```
root@csc3150:/home/vagrant/CSC3150/Assignment1/source/program1# ./program1 ./illegal_instr
Process start to folk
I'm the Parent Process, my pid = 17210
I'm the Child Process, my pid = 17211
------------CHILD PROCESS START------------
This is the SIGILL program

Parent process receives SIGCHLD signal
CHILD EXECUTION FAILED: Receive SIGILL, the process gets illegal instruction.
```

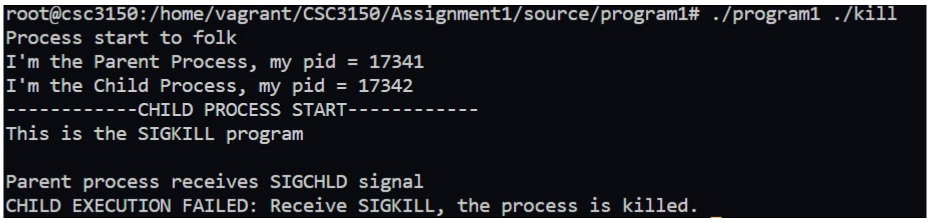
Received SIGINT signal:



```
root@csc3150:/home/vagrant/CSC3150/Assignment1/source/program1# ./program1 ./interrupt
Process start to folk
I'm the Parent Process, my pid = 17297
I'm the Child Process, my pid = 17298
------------CHILD PROCESS START------------
This is the SIGINT program

Parent process receives SIGCHLD signal
CHILD EXECUTION FAILED: Receive SIGINT, the process is interrupted.
```

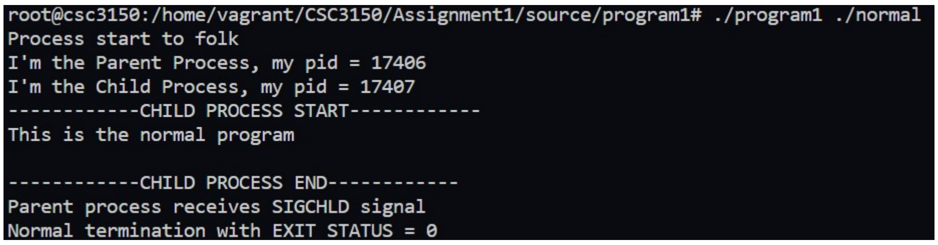
Received SIGKILL signal:



```
root@csc3150:/home/vagrant/CSC3150/Assignment1/source/program1# ./program1 ./kill
Process start to folk
I'm the Parent Process, my pid = 17341
I'm the Child Process, my pid = 17342
------------CHILD PROCESS START------------
This is the SIGKILL program

Parent process receives SIGCHLD signal
CHILD EXECUTION FAILED: Receive SIGKILL, the process is killed.
```

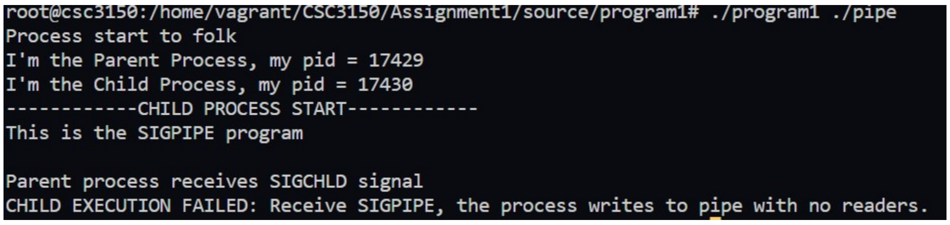
Received SIGCHLD signal (normal termination):



```
root@csc3150:/home/vagrant/CSC3150/Assignment1/source/program1# ./program1 ./normal
Process start to folk
I'm the Parent Process, my pid = 17406
I'm the Child Process, my pid = 17407
------------CHILD PROCESS START------------
This is the normal program

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

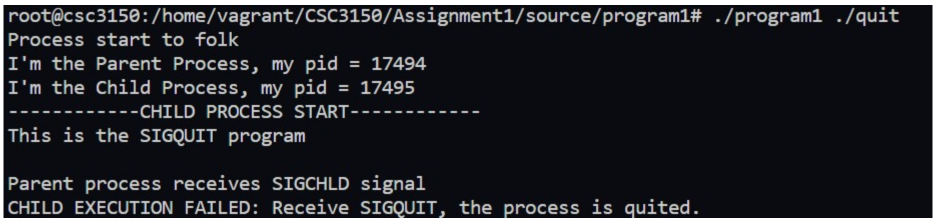
Received SIGPIPE signal:



```
root@csc3150:/home/vagrant/CSC3150/Assignment1/source/program1# ./program1 ./pipe
Process start to folk
I'm the Parent Process, my pid = 17429
I'm the Child Process, my pid = 17430
------------CHILD PROCESS START------------
This is the SIGPIPE program

Parent process receives SIGCHLD signal
CHILD EXECUTION FAILED: Receive SIGPIPE, the process writes to pipe with no readers.
```

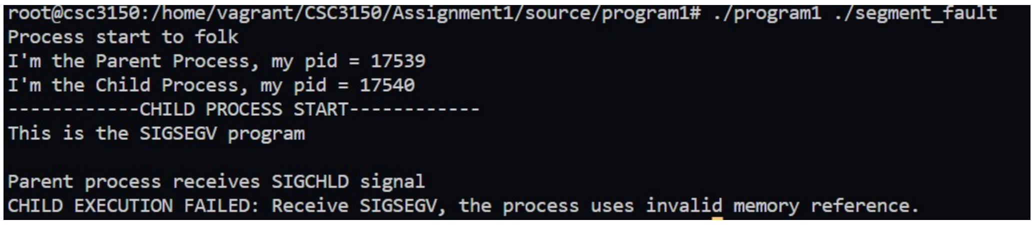
Received SIGQUIT signal:



```
root@csc3150:/home/vagrant/CSC3150/Assignment1/source/program1# ./program1 ./quit
Process start to folk
I'm the Parent Process, my pid = 17494
I'm the Child Process, my pid = 17495
------------CHILD PROCESS START------------
This is the SIGQUIT program

Parent process receives SIGCHLD signal
CHILD EXECUTION FAILED: Receive SIGQUIT, the process is quited.
```

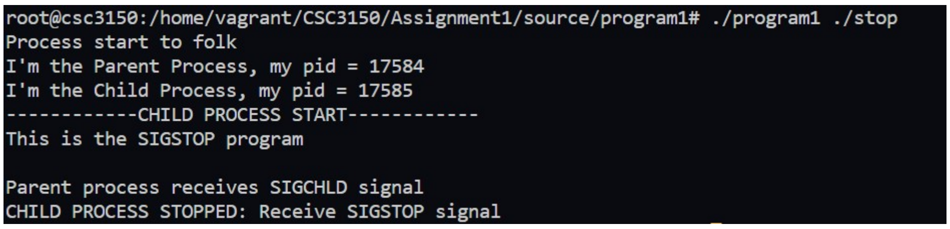
Received SIGSEGV signal:



```
root@csc3150:/home/vagrant/CSC3150/Assignment1/source/program1# ./program1 ./segment_fault
Process start to folk
I'm the Parent Process, my pid = 17539
I'm the Child Process, my pid = 17540
------------CHILD PROCESS START------------
This is the SIGSEGV program

Parent process receives SIGCHLD signal
CHILD EXECUTION FAILED: Receive SIGSEGV, the process uses invalid memory reference.
```

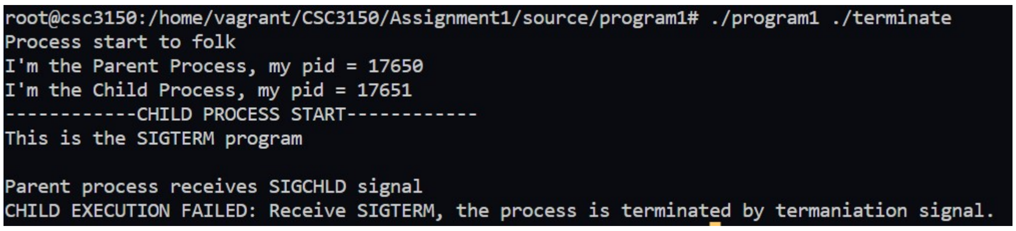
Received SIGSTOP signal:



```
root@csc3150:/home/vagrant/CSC3150/Assignment1/source/program1# ./program1 ./stop
Process start to folk
I'm the Parent Process, my pid = 17584
I'm the Child Process, my pid = 17585
------------CHILD PROCESS START------------
This is the SIGSTOP program

Parent process receives SIGCHLD signal
CHILD PROCESS STOPPED: Receive SIGSTOP signal
```

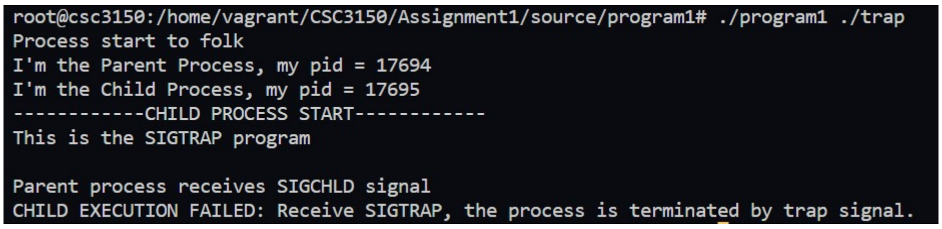
Received SIGCHLD signal:



```
root@csc3150:/home/vagrant/CSC3150/Assignment1/source/program1# ./program1 ./terminate
Process start to folk
I'm the Parent Process, my pid = 17650
I'm the Child Process, my pid = 17651
------------CHILD PROCESS START------------
This is the SIGTERM program

Parent process receives SIGCHLD signal
CHILD EXECUTION FAILED: Receive SIGTERM, the process is terminated by termaniation signal.
```

Received SIGTRAP signal:

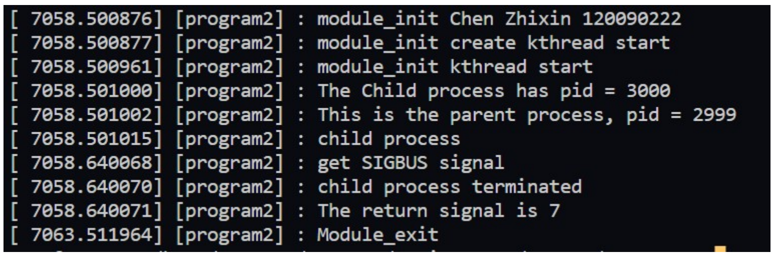


```
root@csc3150:/home/vagrant/CSC3150/Assignment1/source/program1# ./program1 ./trap
Process start to folk
I'm the Parent Process, my pid = 17694
I'm the Child Process, my pid = 17695
------------CHILD PROCESS START------------
This is the SIGTRAP program

Parent process receives SIGCHLD signal
CHILD EXECUTION FAILED: Receive SIGTRAP, the process is terminated by trap signal.
```

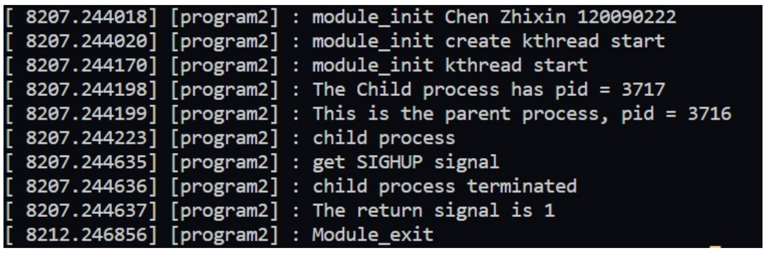
## Program 2

Received SIGBUS signal:



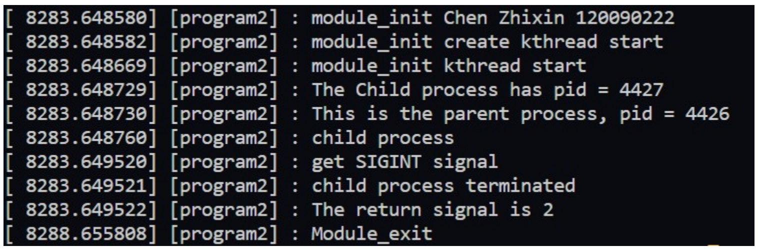
```
[ 7058.500876] [program2] : module_init Chen Zhixin 120090222
[ 7058.500877] [program2] : module_init create kthread start
[ 7058.500961] [program2] : module_init kthread start
[ 7058.501000] [program2] : The Child process has pid = 3000
[ 7058.501002] [program2] : This is the parent process, pid = 2999
[ 7058.501015] [program2] : child process
[ 7058.640068] [program2] : get SIGBUS signal
[ 7058.640070] [program2] : child process terminated
[ 7058.640071] [program2] : The return signal is 7
[ 7063.511964] [program2] : Module_exit
```

Received SIGHUP signal:


```
[ 8207.244018] [program2] : module_init Chen Zhixin 120090222
[ 8207.244020] [program2] : module_init create kthread start
[ 8207.244170] [program2] : module_init kthread start
[ 8207.244198] [program2] : The Child process has pid = 3717
[ 8207.244199] [program2] : This is the parent process, pid = 3716
[ 8207.244223] [program2] : child process
[ 8207.244635] [program2] : get SIGHUP signal
[ 8207.244636] [program2] : child process terminated
[ 8207.244637] [program2] : The return signal is 1
[ 8212.246856] [program2] : Module_exit
```

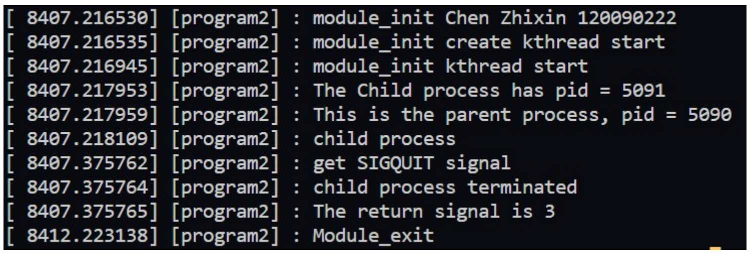
---

Received SIGINT signal:


```
[ 8283.648580] [program2] : module_init Chen Zhixin 120090222
[ 8283.648582] [program2] : module_init create kthread start
[ 8283.648669] [program2] : module_init kthread start
[ 8283.648729] [program2] : The Child process has pid = 4427
[ 8283.648730] [program2] : This is the parent process, pid = 4426
[ 8283.648760] [program2] : child process
[ 8283.649520] [program2] : get SIGINT signal
[ 8283.649521] [program2] : child process terminated
[ 8283.649522] [program2] : The return signal is 2
[ 8288.655808] [program2] : Module_exit
```

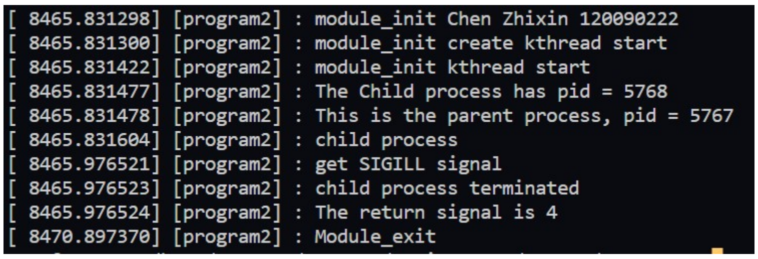
---

Received SIGQUIT signal:


```
[ 8407.216530] [program2] : module_init Chen Zhixin 120090222
[ 8407.216535] [program2] : module_init create kthread start
[ 8407.216945] [program2] : module_init kthread start
[ 8407.217953] [program2] : The Child process has pid = 5091
[ 8407.217959] [program2] : This is the parent process, pid = 5090
[ 8407.218109] [program2] : child process
[ 8407.375762] [program2] : get SIGQUIT signal
[ 8407.375764] [program2] : child process terminated
[ 8407.375765] [program2] : The return signal is 3
[ 8412.223138] [program2] : Module_exit
```

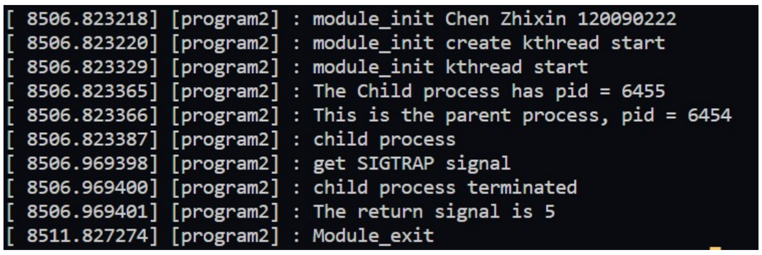
---

Received SIGILL signal:


```
[ 8465.831298] [program2] : module_init Chen Zhixin 120090222
[ 8465.831300] [program2] : module_init create kthread start
[ 8465.831422] [program2] : module_init kthread start
[ 8465.831477] [program2] : The Child process has pid = 5768
[ 8465.831478] [program2] : This is the parent process, pid = 5767
[ 8465.831604] [program2] : child process
[ 8465.976521] [program2] : get SIGILL signal
[ 8465.976523] [program2] : child process terminated
[ 8465.976524] [program2] : The return signal is 4
[ 8470.897370] [program2] : Module_exit
```

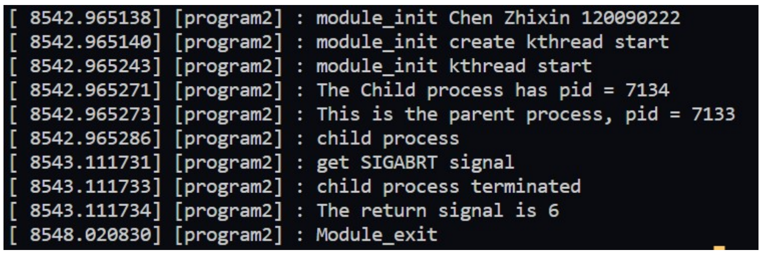
---

Received SIGTRAP signal:


```
[ 8506.823218] [program2] : module_init Chen Zhixin 120090222
[ 8506.823220] [program2] : module_init create kthread start
[ 8506.823329] [program2] : module_init kthread start
[ 8506.823365] [program2] : The Child process has pid = 6455
[ 8506.823366] [program2] : This is the parent process, pid = 6454
[ 8506.823387] [program2] : child process
[ 8506.969398] [program2] : get SIGTRAP signal
[ 8506.969400] [program2] : child process terminated
[ 8506.969401] [program2] : The return signal is 5
[ 8511.827274] [program2] : Module_exit
```

---

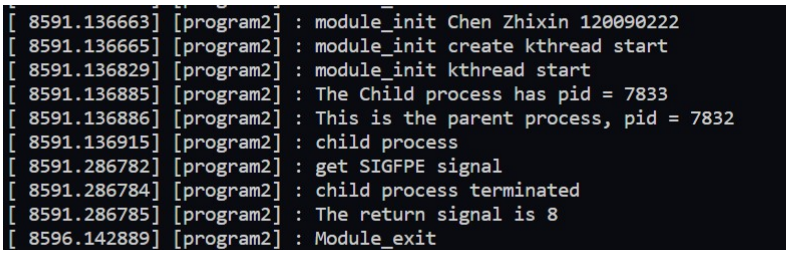
Received SIGABRT signal:



```
[ 8542.965138] [program2] : module_init Chen Zhixin 120090222
[ 8542.965140] [program2] : module_init create kthread start
[ 8542.965243] [program2] : module_init kthread start
[ 8542.965271] [program2] : The Child process has pid = 7134
[ 8542.965273] [program2] : This is the parent process, pid = 7133
[ 8542.965286] [program2] : child process
[ 8543.111731] [program2] : get SIGABRT signal
[ 8543.111733] [program2] : child process terminated
[ 8543.111734] [program2] : The return signal is 6
[ 8548.020830] [program2] : Module_exit
```

---

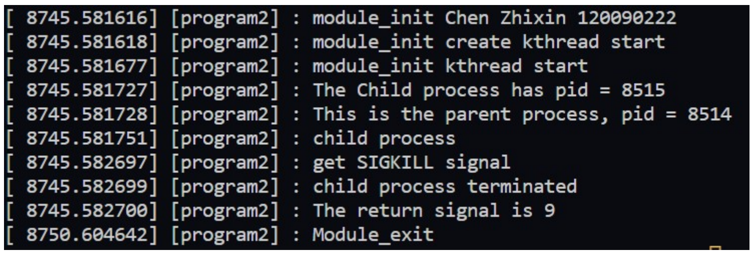
Received SIGFPE signal:



```
[ 8591.136663] [program2] : module_init Chen Zhixin 120090222
[ 8591.136665] [program2] : module_init create kthread start
[ 8591.136829] [program2] : module_init kthread start
[ 8591.136885] [program2] : The Child process has pid = 7833
[ 8591.136886] [program2] : This is the parent process, pid = 7832
[ 8591.136915] [program2] : child process
[ 8591.286782] [program2] : get SIGFPE signal
[ 8591.286784] [program2] : child process terminated
[ 8591.286785] [program2] : The return signal is 8
[ 8596.142889] [program2] : Module_exit
```

---

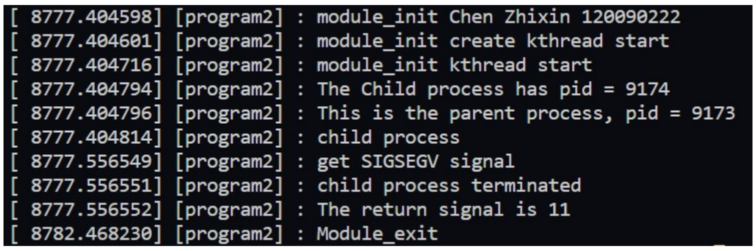
Received SIGKILL signal:



```
[ 8745.581616] [program2] : module_init Chen Zhixin 120090222
[ 8745.581618] [program2] : module_init create kthread start
[ 8745.581677] [program2] : module_init kthread start
[ 8745.581727] [program2] : The Child process has pid = 8515
[ 8745.581728] [program2] : This is the parent process, pid = 8514
[ 8745.581751] [program2] : child process
[ 8745.582697] [program2] : get SIGKILL signal
[ 8745.582699] [program2] : child process terminated
[ 8745.582700] [program2] : The return signal is 9
[ 8750.604642] [program2] : Module_exit
```

---

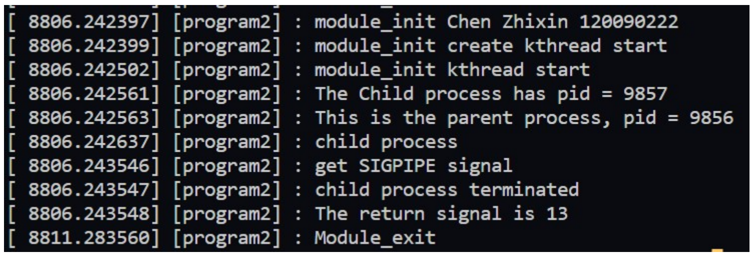
Received SIGSEGV signal:



```
[ 8777.404598] [program2] : module_init Chen Zhixin 120090222
[ 8777.404601] [program2] : module_init create kthread start
[ 8777.404716] [program2] : module_init kthread start
[ 8777.404794] [program2] : The Child process has pid = 9174
[ 8777.404796] [program2] : This is the parent process, pid = 9173
[ 8777.404814] [program2] : child process
[ 8777.556549] [program2] : get SIGSEGV signal
[ 8777.556551] [program2] : child process terminated
[ 8777.556552] [program2] : The return signal is 11
[ 8782.468230] [program2] : Module_exit
```

---

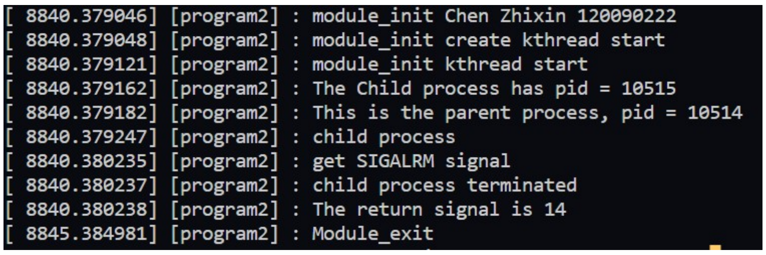
Received SIGPIPE signal:



```
[ 8806.242397] [program2] : module_init Chen Zhixin 120090222
[ 8806.242399] [program2] : module_init create kthread start
[ 8806.242502] [program2] : module_init kthread start
[ 8806.242561] [program2] : The Child process has pid = 9857
[ 8806.242563] [program2] : This is the parent process, pid = 9856
[ 8806.242637] [program2] : child process
[ 8806.243546] [program2] : get SIGPIPE signal
[ 8806.243547] [program2] : child process terminated
[ 8806.243548] [program2] : The return signal is 13
[ 8811.283560] [program2] : Module_exit
```

---

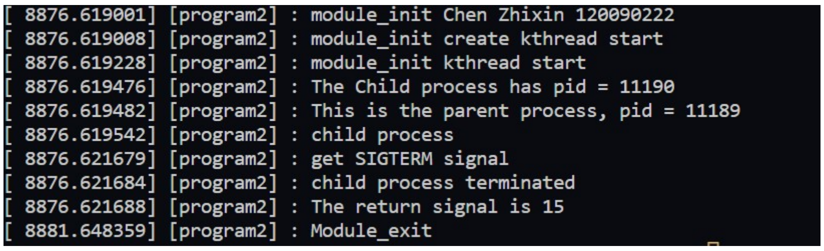
Received SIGALRM signal:



```
[ 8840.379046] [program2] : module_init Chen Zhixin 120090222
[ 8840.379048] [program2] : module_init create kthread start
[ 8840.379121] [program2] : module_init kthread start
[ 8840.379162] [program2] : The Child process has pid = 10515
[ 8840.379182] [program2] : This is the parent process, pid = 10514
[ 8840.379247] [program2] : child process
[ 8840.380235] [program2] : get SIGALRM signal
[ 8840.380237] [program2] : child process terminated
[ 8840.380238] [program2] : The return signal is 14
[ 8845.384981] [program2] : Module_exit
```

---

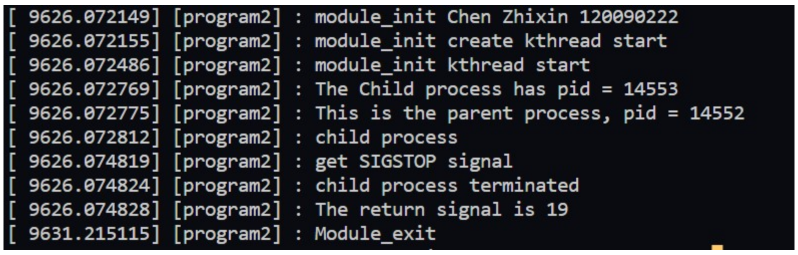
Received SIGTERM signal:



```
[ 8876.619001] [program2] : module_init Chen Zhixin 120090222
[ 8876.619008] [program2] : module_init create kthread start
[ 8876.619228] [program2] : module_init kthread start
[ 8876.619476] [program2] : The Child process has pid = 11190
[ 8876.619482] [program2] : This is the parent process, pid = 11189
[ 8876.619542] [program2] : child process
[ 8876.621679] [program2] : get SIGTERM signal
[ 8876.621684] [program2] : child process terminated
[ 8876.621688] [program2] : The return signal is 15
[ 8881.648359] [program2] : Module_exit
```

---

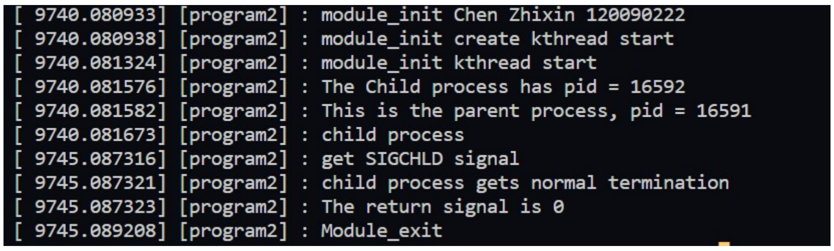
Received SIGSTOP signal:



```
[ 9626.072149] [program2] : module_init Chen Zhixin 120090222
[ 9626.072155] [program2] : module_init create kthread start
[ 9626.072486] [program2] : module_init kthread start
[ 9626.072769] [program2] : The Child process has pid = 14553
[ 9626.072775] [program2] : This is the parent process, pid = 14552
[ 9626.072812] [program2] : child process
[ 9626.074819] [program2] : get SIGSTOP signal
[ 9626.074824] [program2] : child process terminated
[ 9626.074828] [program2] : The return signal is 19
[ 9631.215115] [program2] : Module_exit
```

---

Received SIGCHLD signal:



```
[ 9740.080933] [program2] : module_init Chen Zhixin 120090222
[ 9740.080938] [program2] : module_init create kthread start
[ 9740.081324] [program2] : module_init kthread start
[ 9740.081576] [program2] : The Child process has pid = 16592
[ 9740.081582] [program2] : This is the parent process, pid = 16591
[ 9740.081673] [program2] : child process
[ 9745.087316] [program2] : get SIGCHLD signal
[ 9745.087321] [program2] : child process gets normal termination
[ 9745.087323] [program2] : The return signal is 0
[ 9745.089208] [program2] : Module_exit
```

### Program 3

## Things I learned from the tasks

From this task, I learn how the program interact with kernel. Also, I learned how kernel works, and what we can do with the kernel. Most important is, I learn the methodology of how to deal with a large project. Like Linux kernel, it is implemented by thousands of deliciated functions and definitions. When we need to modify certain functions or to add some new features, we can first take a look at its source code. Also, computer science is a subject which everything is updating in a rapid way. Many useful materials can be found on Google, GitHub, etc.

Also, I learned some useful technics of Linux command, which is very helpful for debugging. When something is stuck, we can use certain indicators (like `print` / `printf` / `printk` ) to locate the problem.