CSC3150 Operating System

Assignment1 Report

Yuhang Wang 王禹杭

120090246

1. Design
   1. Task 1
2. Using fork() to fork a child process. If it fails to fork, it will exit and signal an error. And then child process and parent process will proceed concurrently.
3. In child process, use execve() to execute the test program. The parameter of this function is that you inputted in the command line.
4. In parent process, using waitpid() to get the signal.

Here is the usage of waitpid():

**pid\_t wait\_child** = **waitpid(pid, &status, WUNTRACED);**

the optionWUNTRACEDtraces the status of stopped children.

1. Using WIFEXITED, WEXITSTATUS, WIFSIGNALED, WTERMSIG, and WIFSTOPPED to receive and parse the signal and check the child process’ termination status.
2. Identify the signal: using switch case expression to match the exit code of the signal to their corresponding name. Then print it.
   1. Task 2
3. In linux system file, I have add lines of code In the corresponding file:

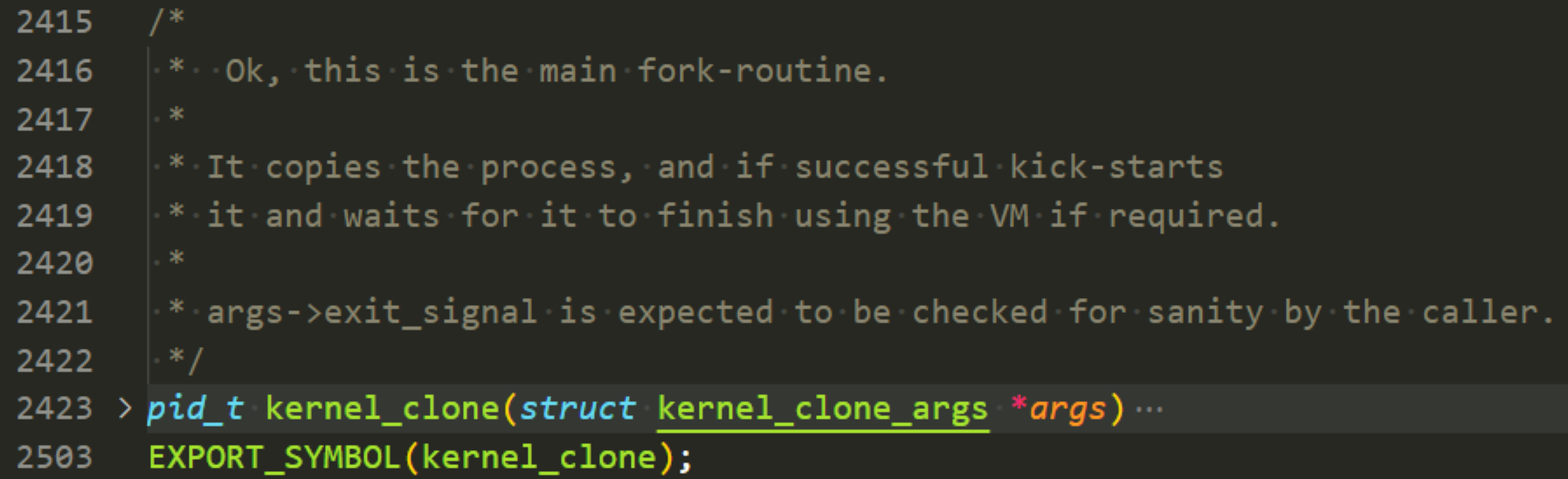
EXPORT\_SYMBOL(kernel\_clone)

EXPORT\_SYMBOL(do\_wait)

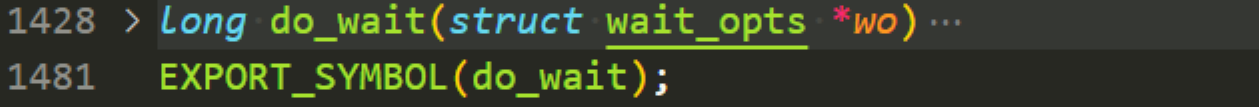
EXPORT\_SYMBOL(do\_execve)

EXPORT\_SYMBOL(getname\_kernel)

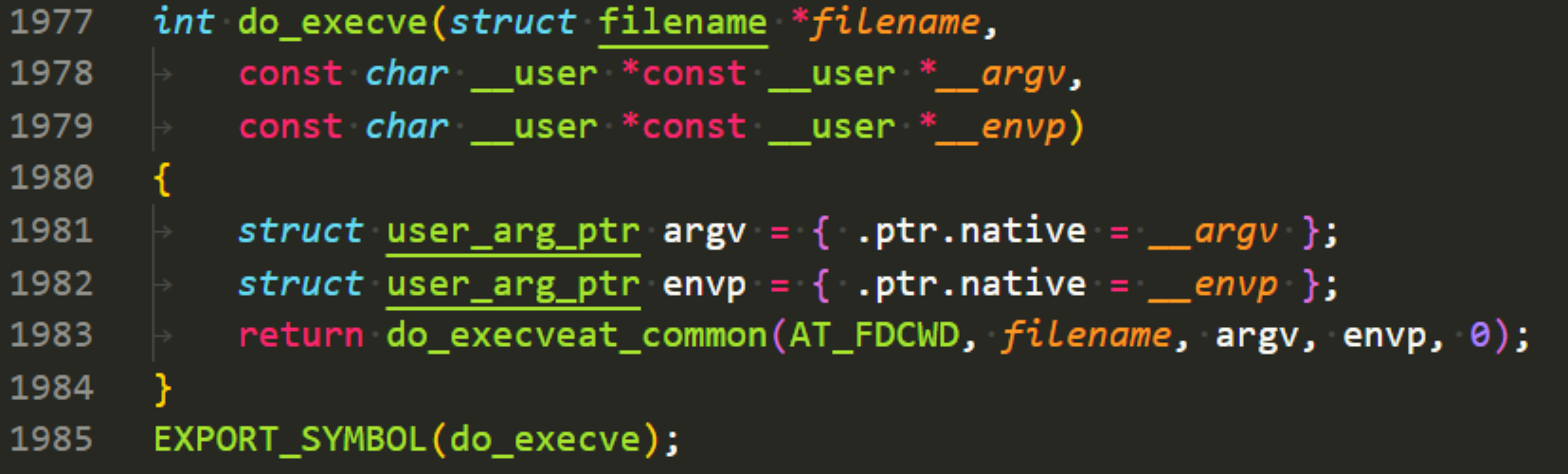
Linux-5.10.99/kernel/fork.c:



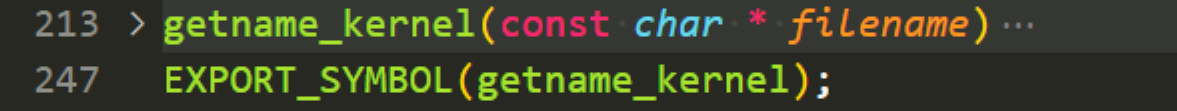
Linux-5.10.99/kernel/exit.c:



Linux-5.10.99/fs/exit.c:



Linux-5.10.99/fs/namei.c:



1. I also extern the symbol in program2.c:

**extern int do\_execve(struct filename \*filename,**

**const char \_\_user \*const \_\_user \*\_\_argv,**

**const char \_\_user \*const \_\_user \*\_\_envp);**

**extern struct filename \*getname\_kernel(const char \*filename);**

**extern long do\_wait(struct wait\_opts \*wo);**

**extern pid\_t kernel\_clone(struct kernel\_clone\_args \*kargs);**

1. I also constructed a type to better use the **do\_wait()** function:

**struct wait\_opts {**

**enum pid\_type wo\_type;**

**int wo\_flags;**

**struct pid \*wo\_pid;**

**struct waitid\_info \*wo\_info;**

**int wo\_stat;**

**struct rusage \*wo\_rusage;**

**wait\_queue\_entry\_t child\_wait;**

**int notask\_error;**

**};**

1. Create kernel thread: In kernel mode, using kthread\_create() function to

create a new thread to do my\_fork() function.

1. In my\_fork() function, using kernel\_clone() function to fork a process linked to my\_exec() function.
2. In my\_exec(), using do\_execve() to execute the test program. The path should be **"/tmp/test"**, where the test is an executable file and act as a child thread.

And I set the argv[] and envp[] argument in do\_execve() function, and it is as shown below:

**result = do\_execve(getname\_kernel(path), NULL, NULL);**

1. In my\_wait() : using do\_wait() to wait the child process and get the exit signal code in wo.wo\_stat. Then using switch case expression to identify the signal received and print the corresponding signal raised.

And I set:

**wo.wo\_flags = WEXITED | WUNTRACED**

Besides, I add a command:

**return\_signal &= 0x7f;**

to the returned signal, which converts the signal greater than 128 to the signal from range 0-19.

1. Development Environment
   1. Version of Linux Distribution:

Distributor ID: Ubuntu

Description: Ubuntu 16.04.7 LTS

Release: 16.04

Codename: xenial

* 1. Version of kernel: 5.10.99
  2. Version of gcc: 5.4.0 20160609

1. The steps to compile the kernel and execute my program
   1. Compile the kernel:
2. Download source code from

<http://www.kernel.org>

mirro: <https://mirror.tuna.tsinghua.edu.cn/kernel/v5.x/>

1. Install Dependency and development tools:

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

1. 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**

1. Copy config from /boot to /home/seed/work/KERNEL\_FILE
2. Login root account and go to kernel source directory

**sudo su**

**cd /home/seed/work /KERNEL\_FILE**

1. Clean previous setting and start configuration

**make mrproper**

**make clean**

**make menuconfig**

**save the config and exit**

1. kernel Image and modules(start from here when recompile the kernel)

**make bzImage -j$(nproc)**

**make modules -j$(nproc)**

1. Install kernel modules

**make modules\_install**

1. Install kernel

**make install**

1. Reboot to load new kernel

**reboot**

(When rebooting, you should select the updated kernel)

* 1. Program1 Compile:

How to compile:

In the 'program1' directory, type 'make' command and enter.

How to clear:

In the 'program1' directory, type 'make clean' command and enter.

* 1. Program1 execution:

After changing into **Assignment\_1\_120090246/source/program1** directory, then type:

**./program1 ./$TEST\_CASE**

where $TEST\_CASE is the name of test program, for example:

**./program1 ./abort**

* 1. Program2 Complie

How to compile:

In the 'program1' directory, type 'make' command and enter.

How to clear:

In the 'program1' directory, type 'make clean' command and enter.

* 1. Program2 execution

In the 'program1' directory, first type “make” to run Makefile, then type:

**sudo insmod program2.ko**

**sudo rmmod program2**

**dmesg**

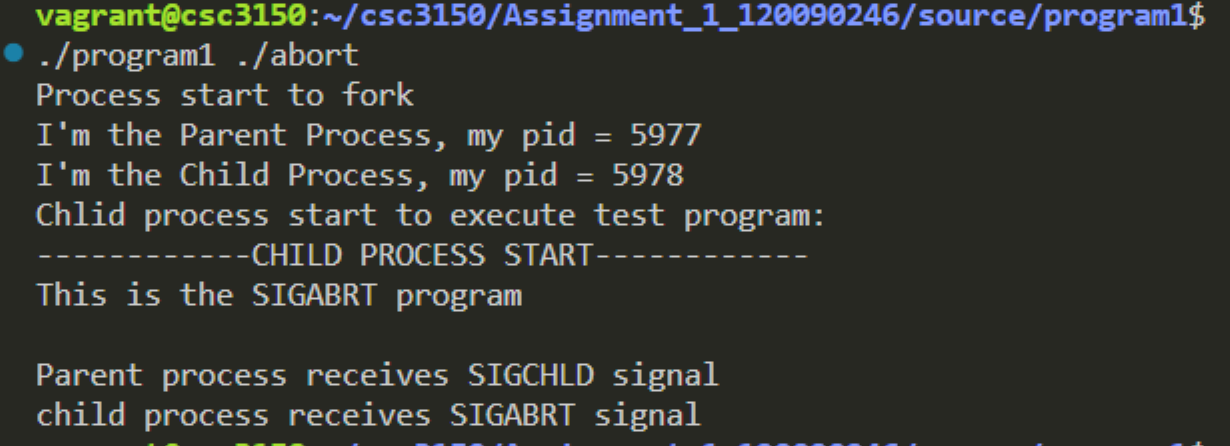
Then , you could see messages appearing. The messages are between the messages 'Module init' and 'module exit'. You could replace “dmesg” command by

**dmesg | tail -n 20**

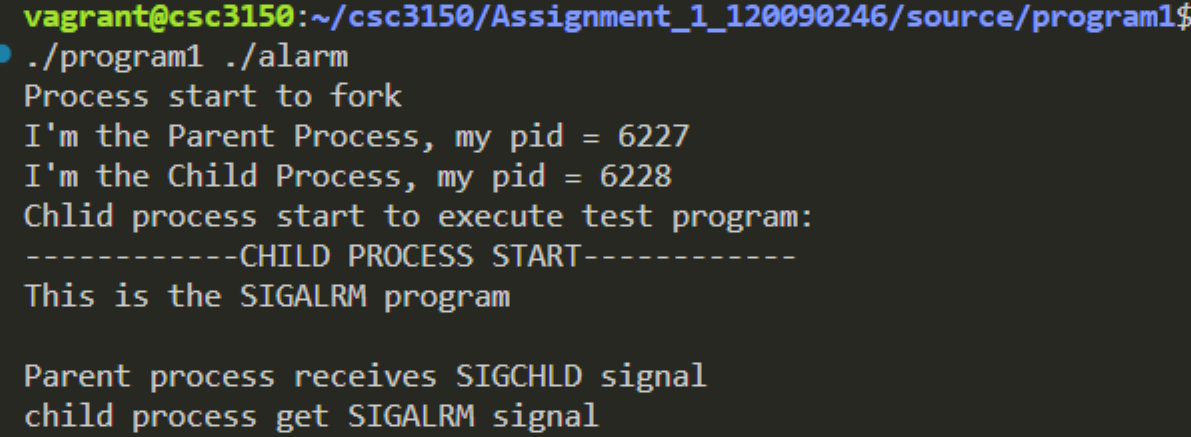
if you want to show last 20 lines of messages.

1. Output Demonstration
   1. Task1

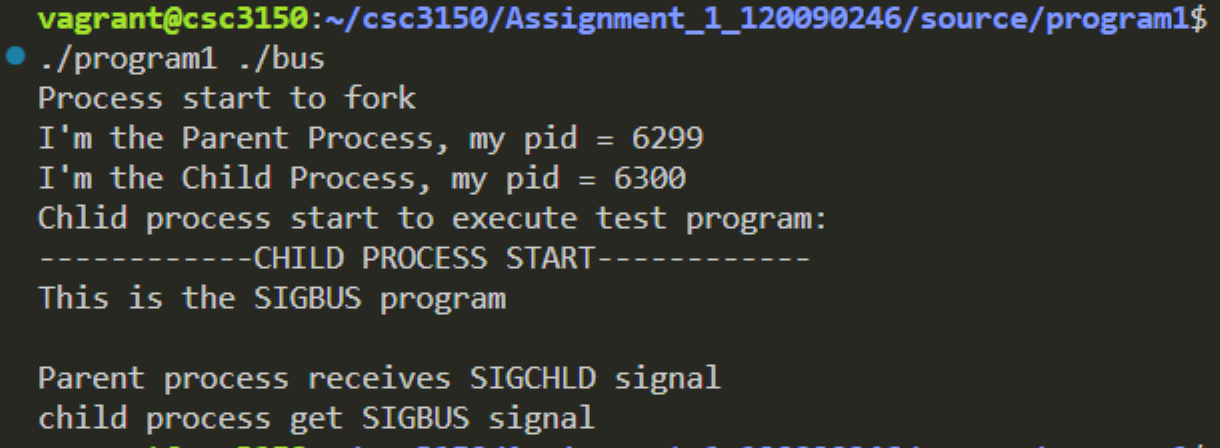
Abort



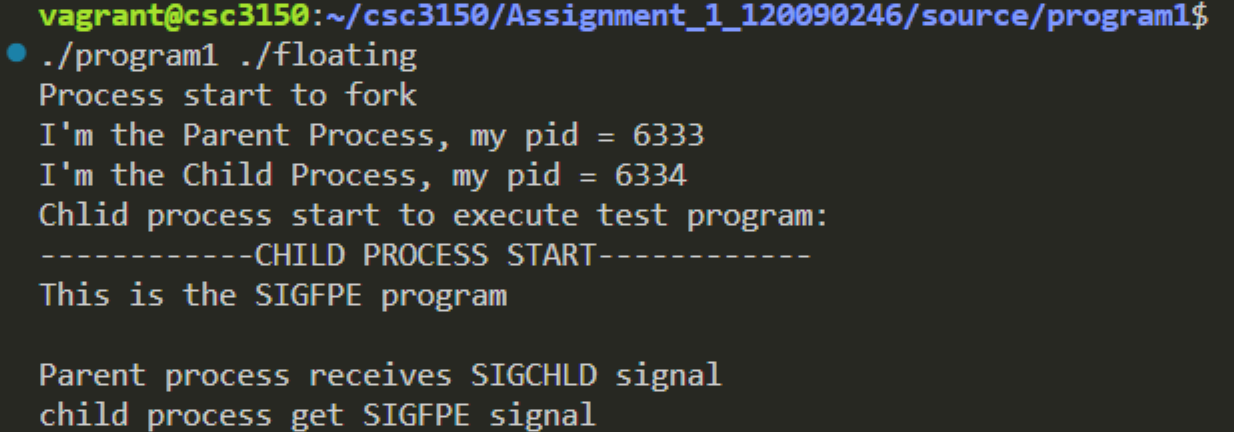
Alarm



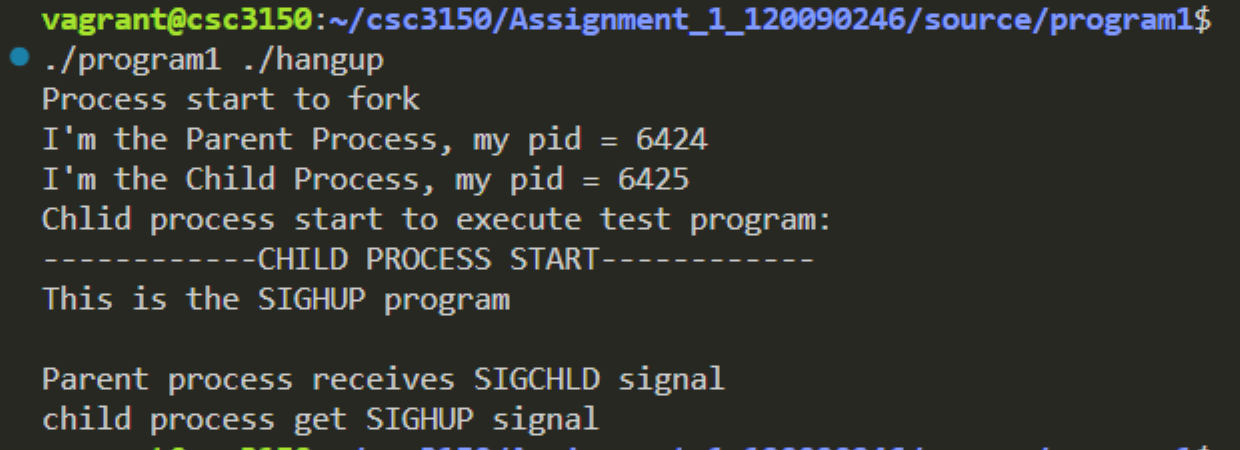
Bus



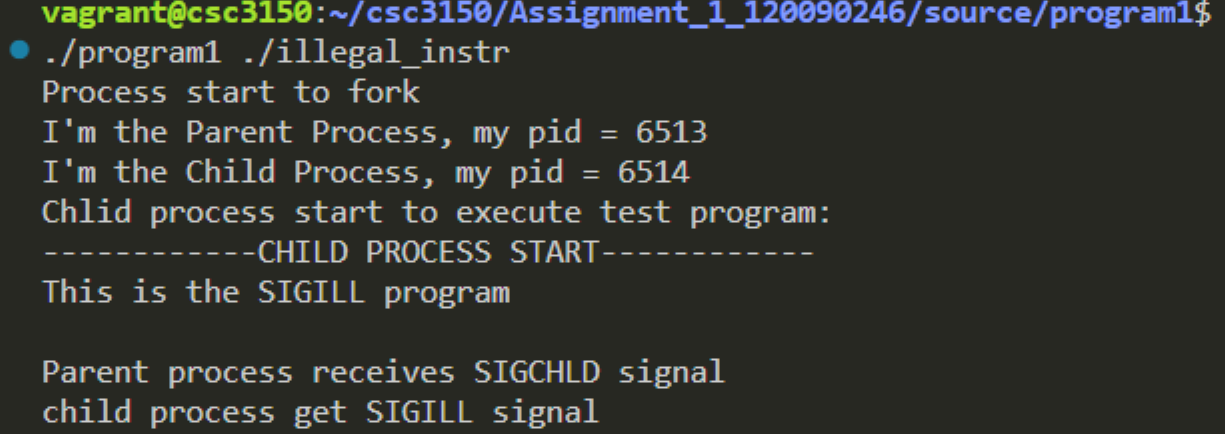
Floating



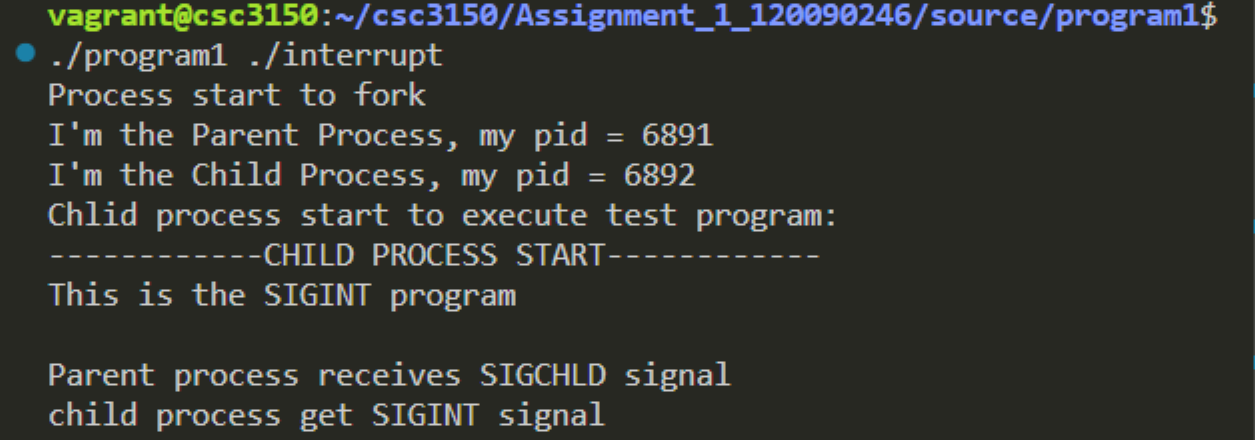
Hangup



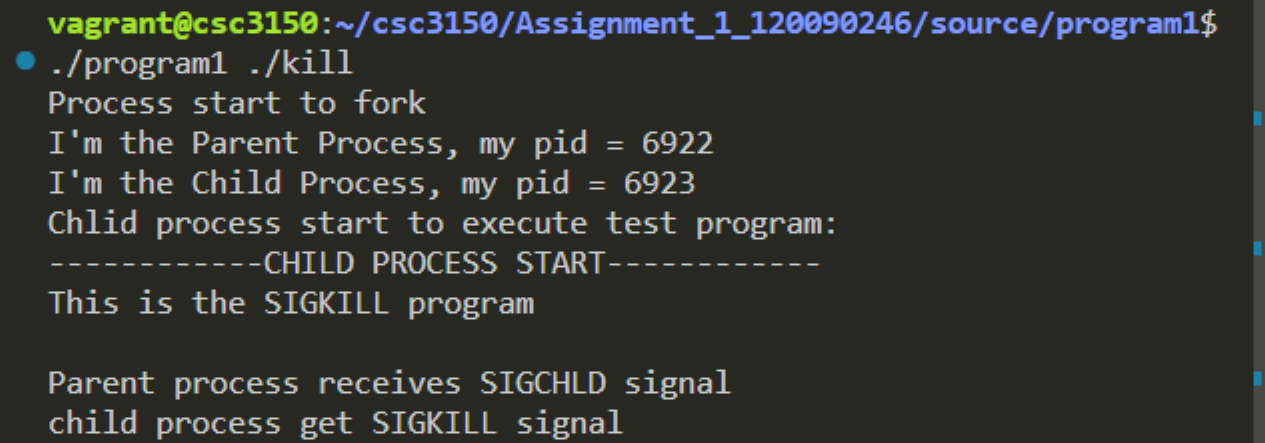
Illegal\_instr



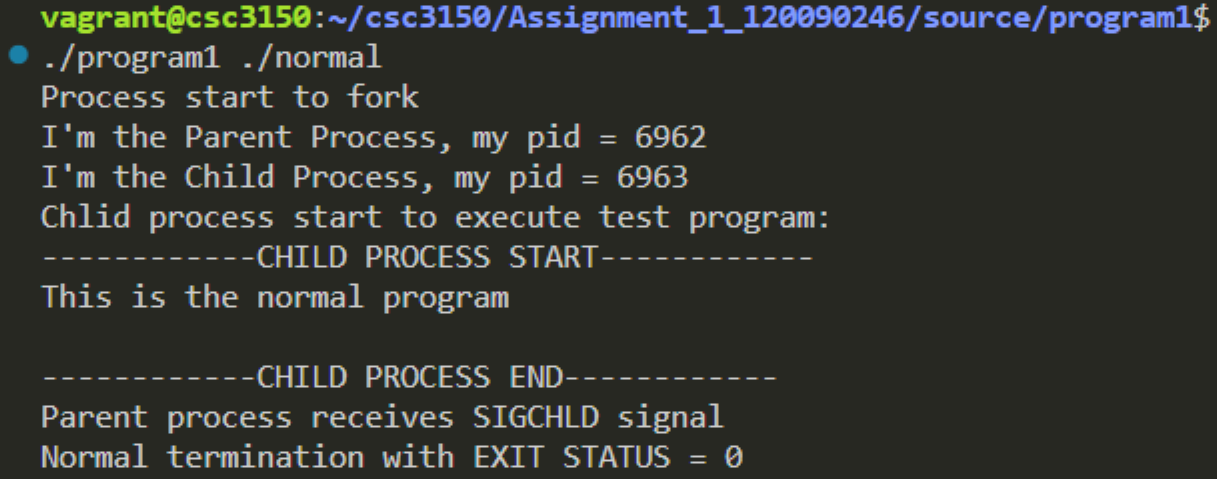
Interrupt



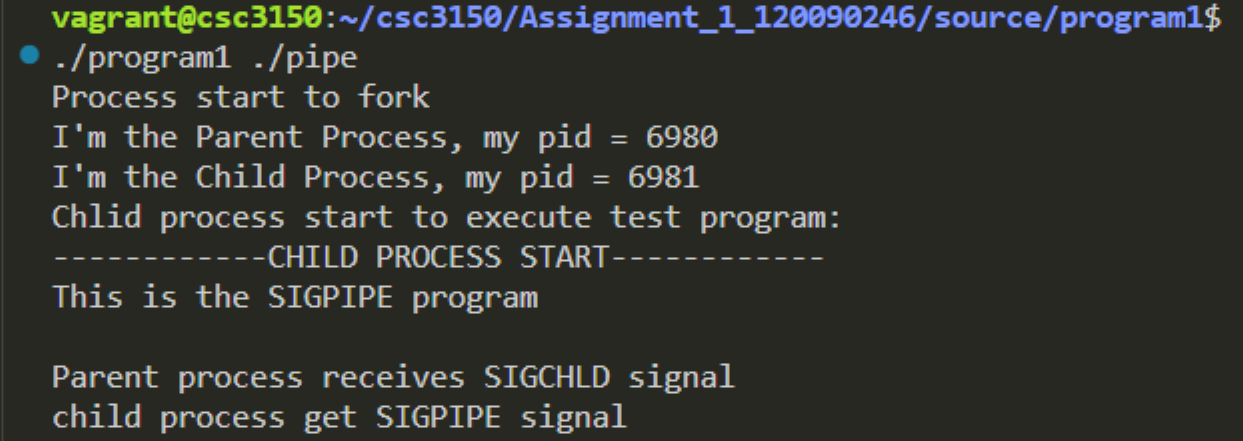
Kill



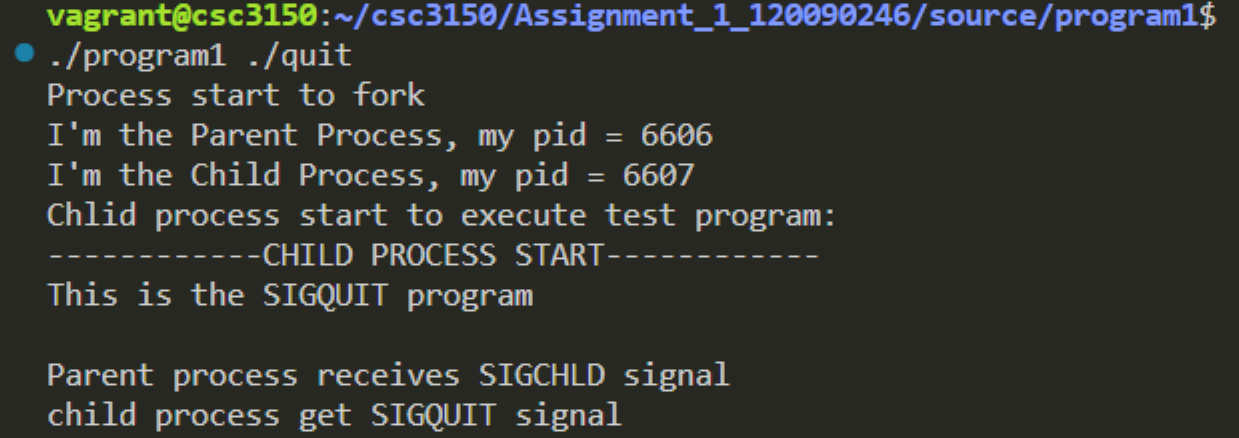
Normal



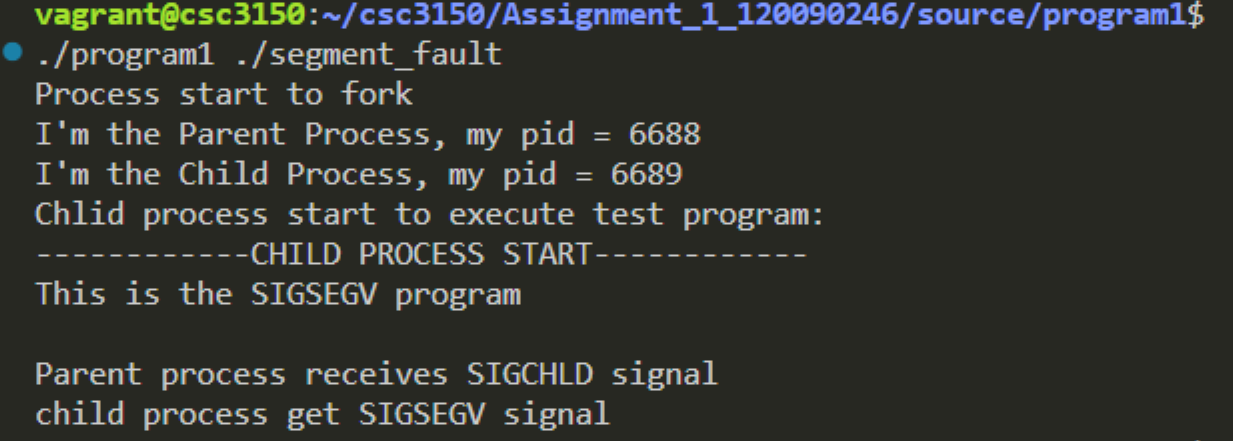
Pipe



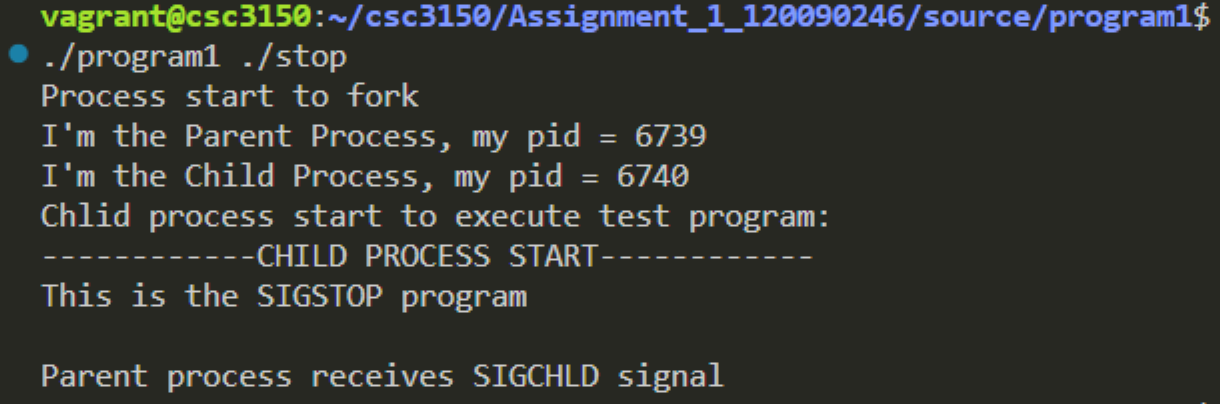
Quit



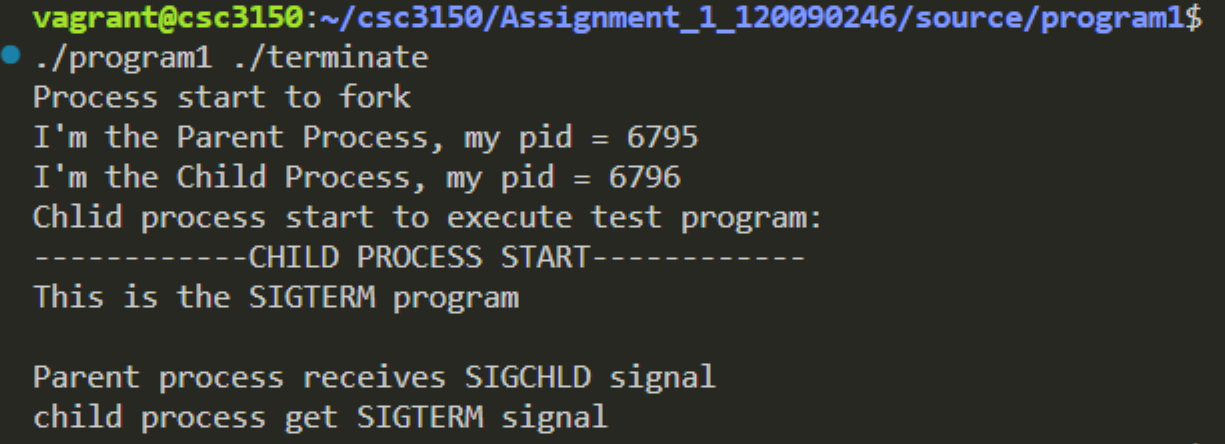
Segment\_fault



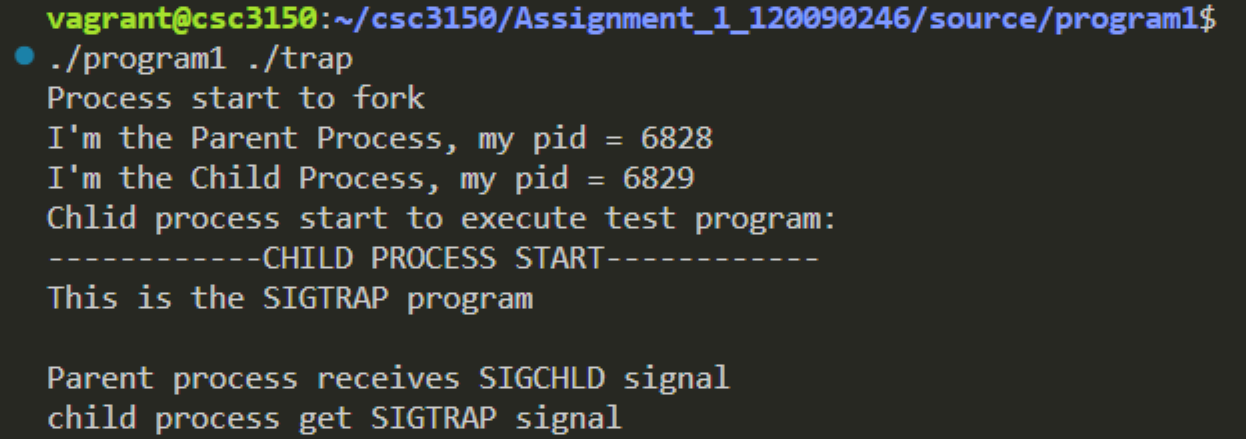
Stop



Terminate

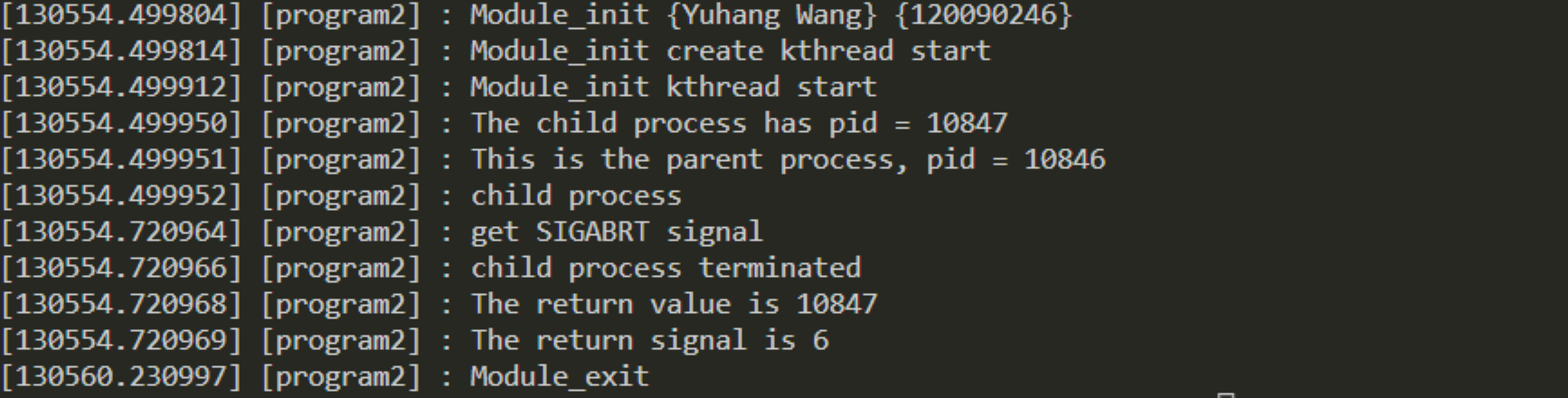


Trap

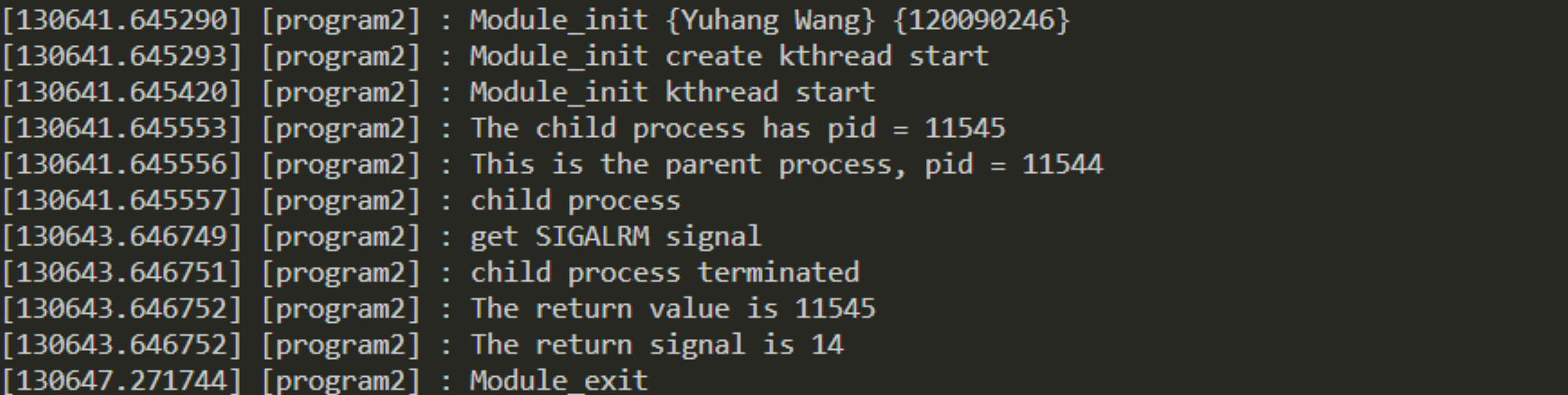


* 1. task2

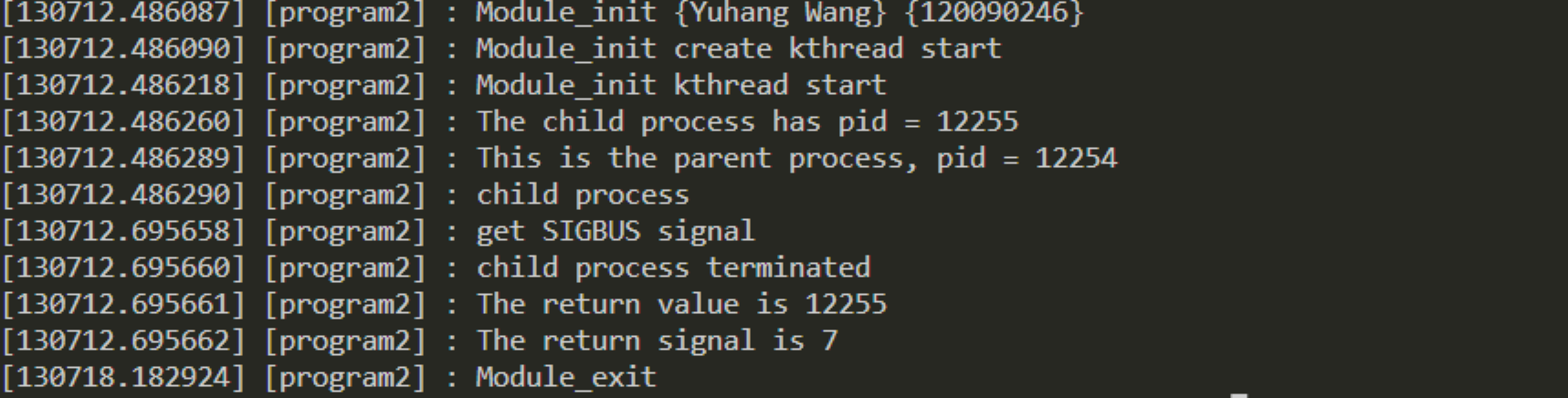
Abort



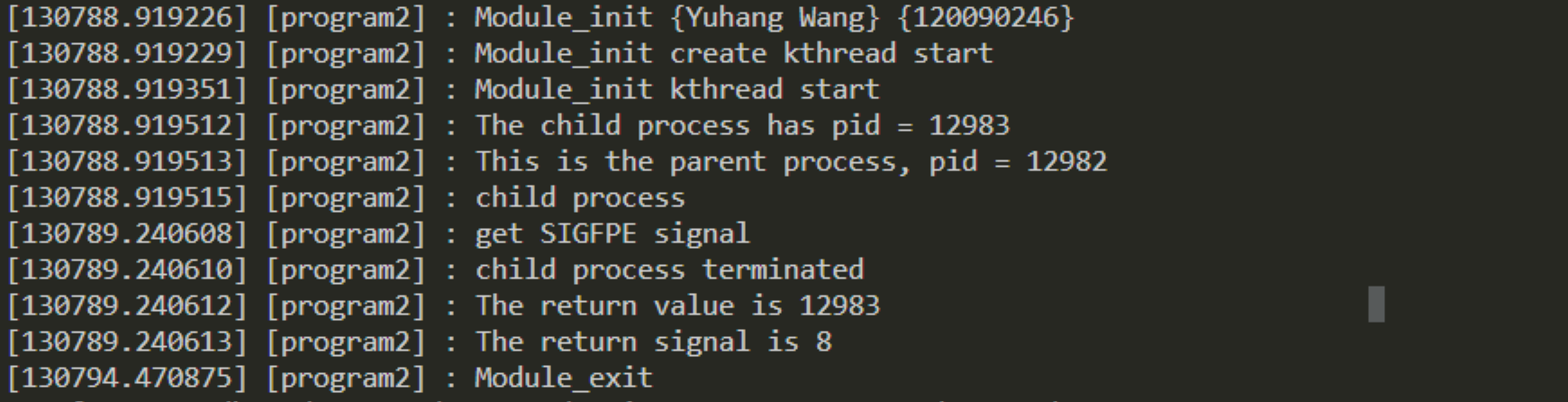
Alarm



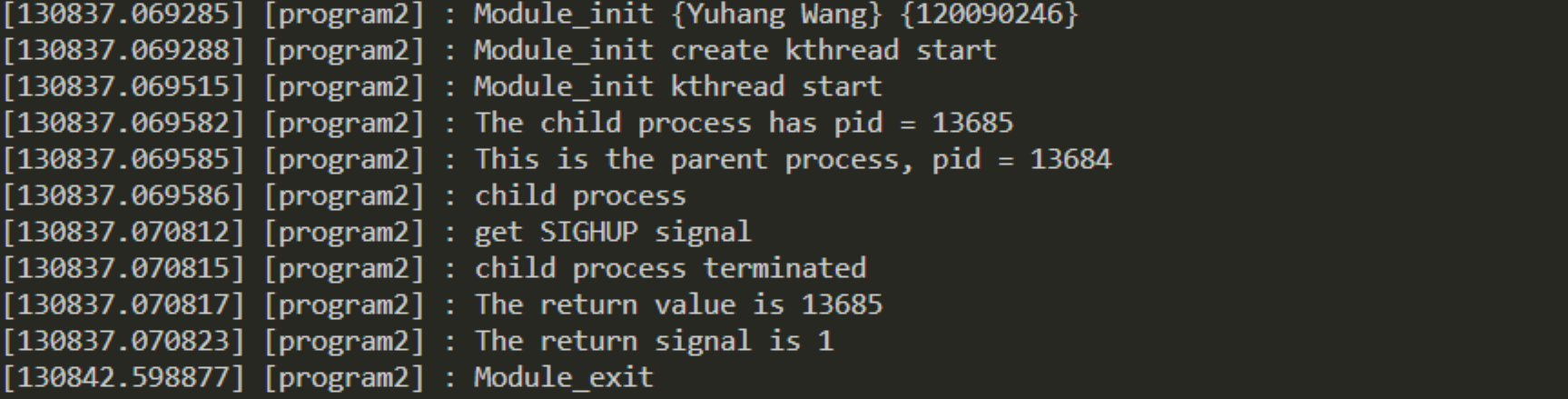
Bus



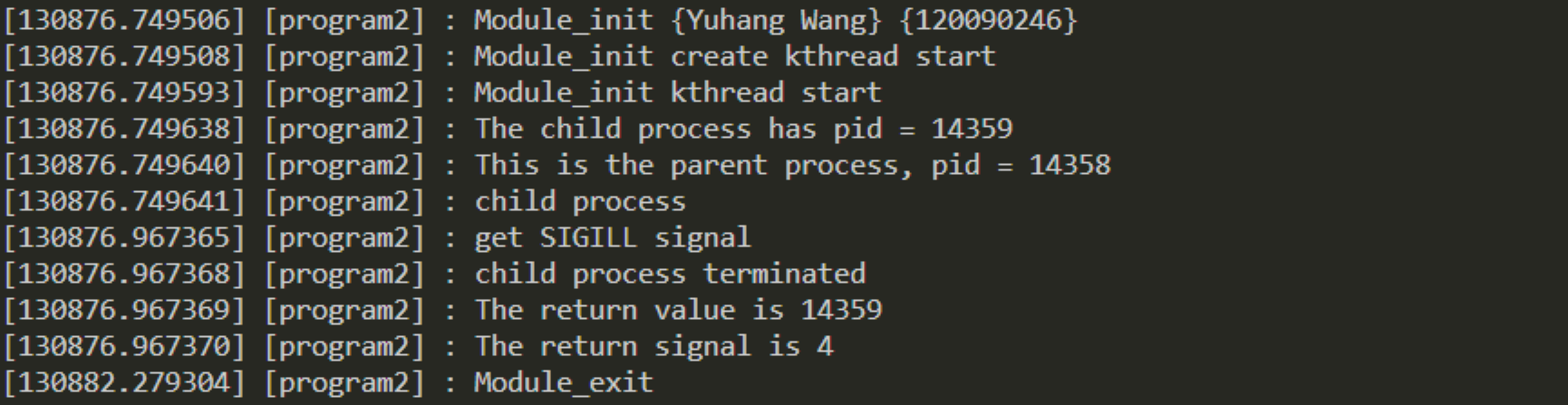
Floating



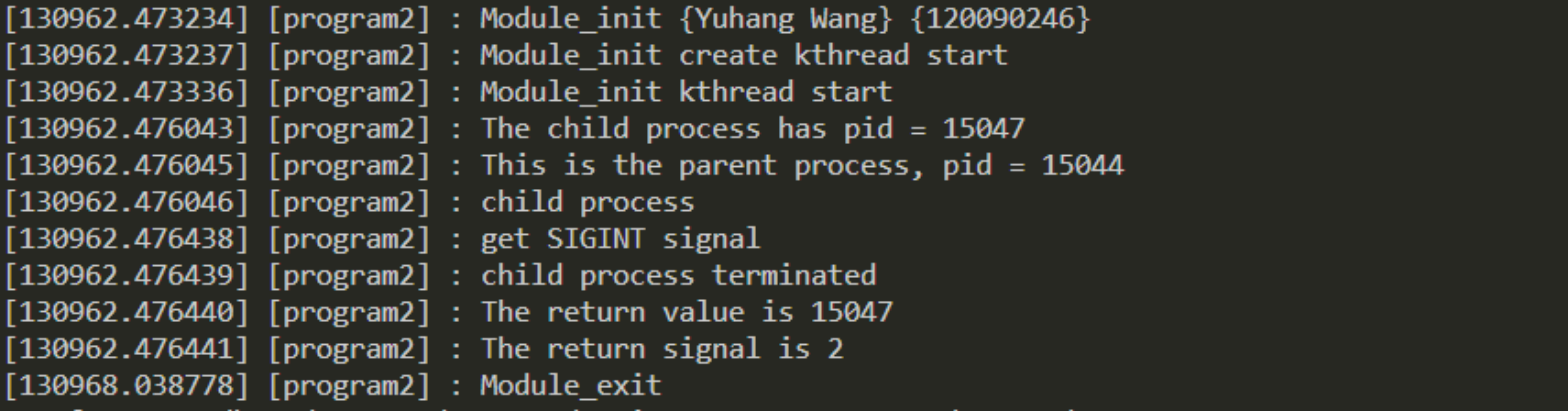
Hangup



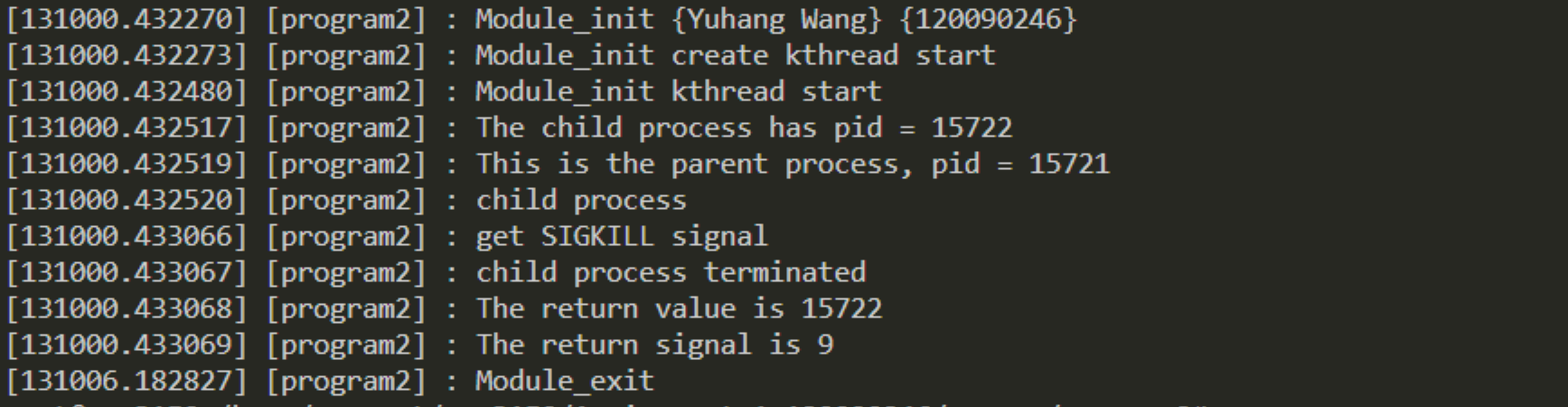
Illegal\_instr



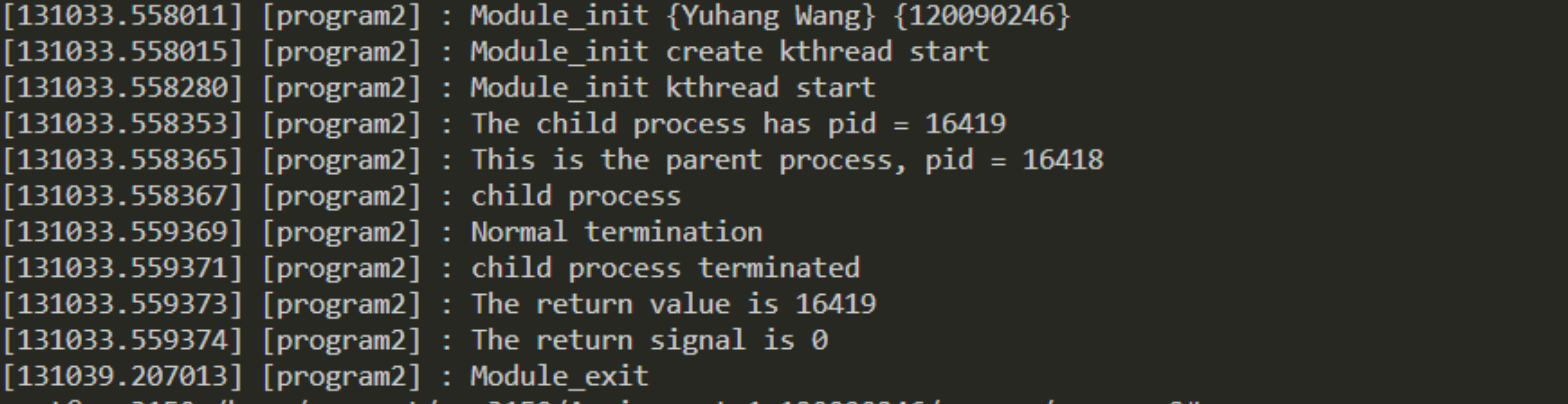
Interrupt



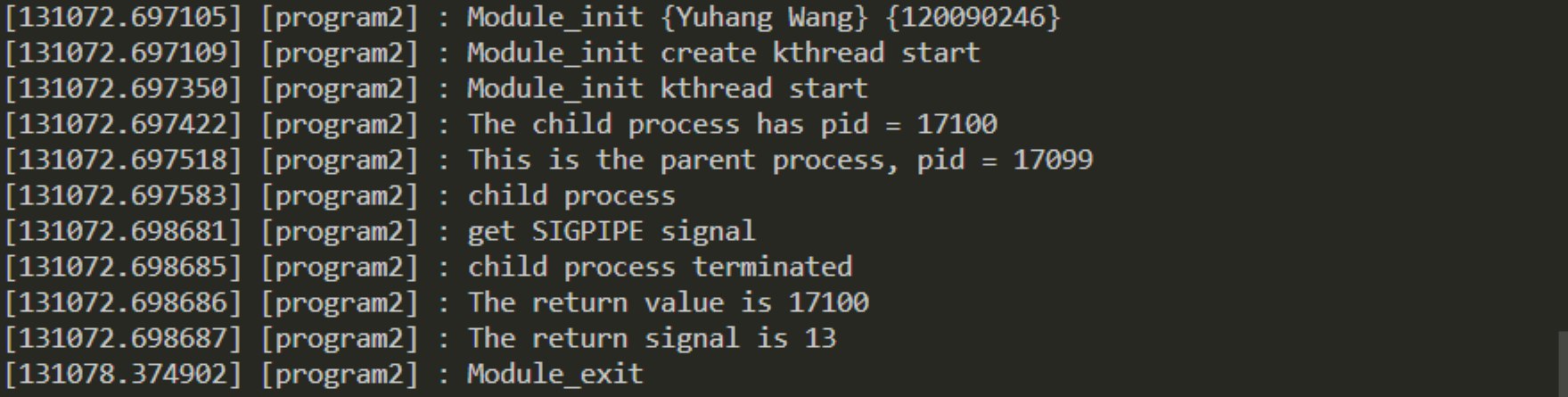
Kill



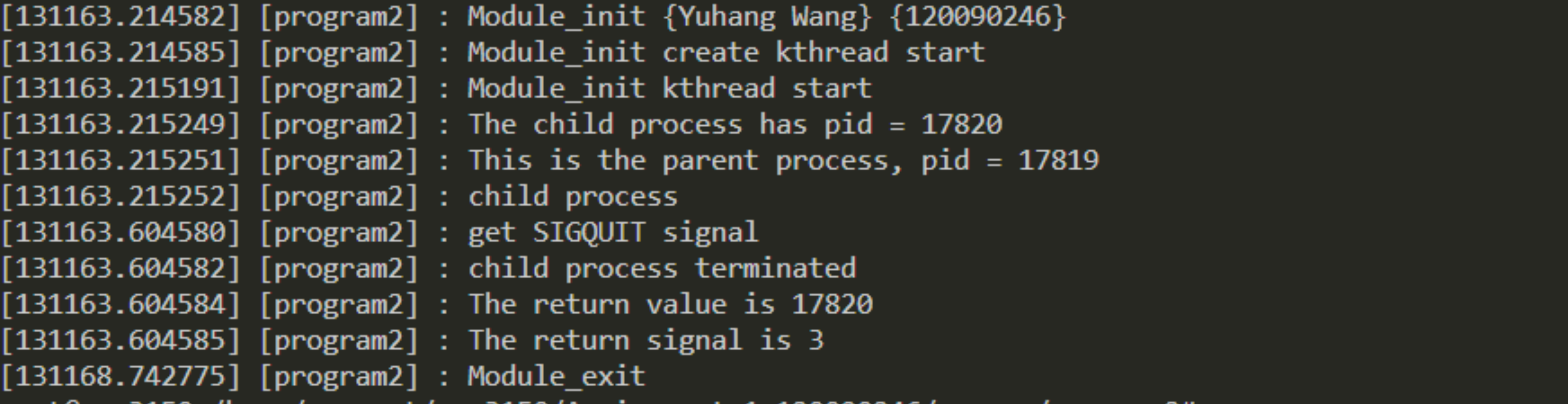
Normal



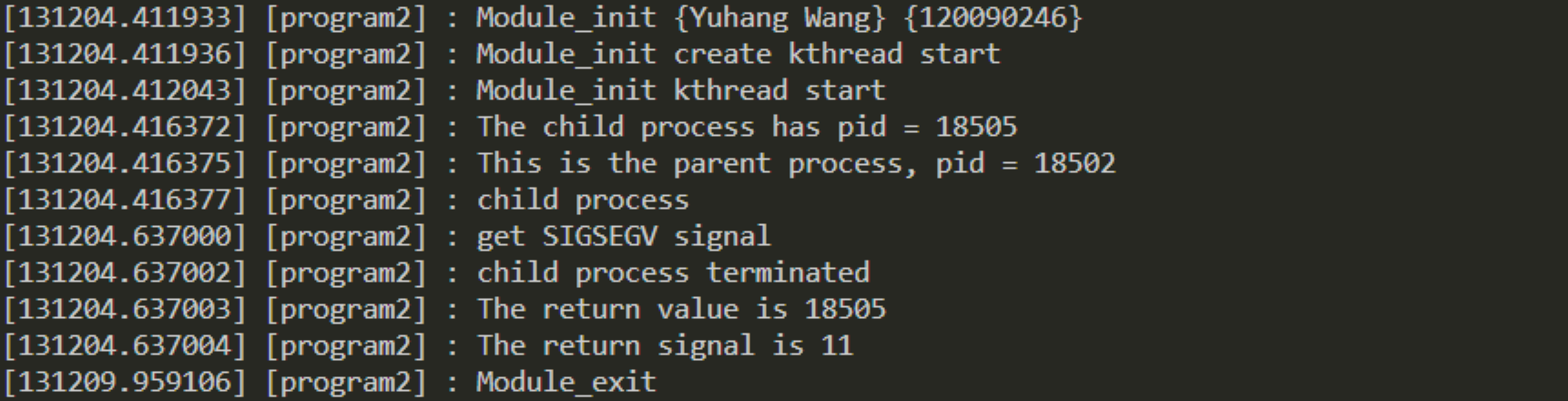
Pipe



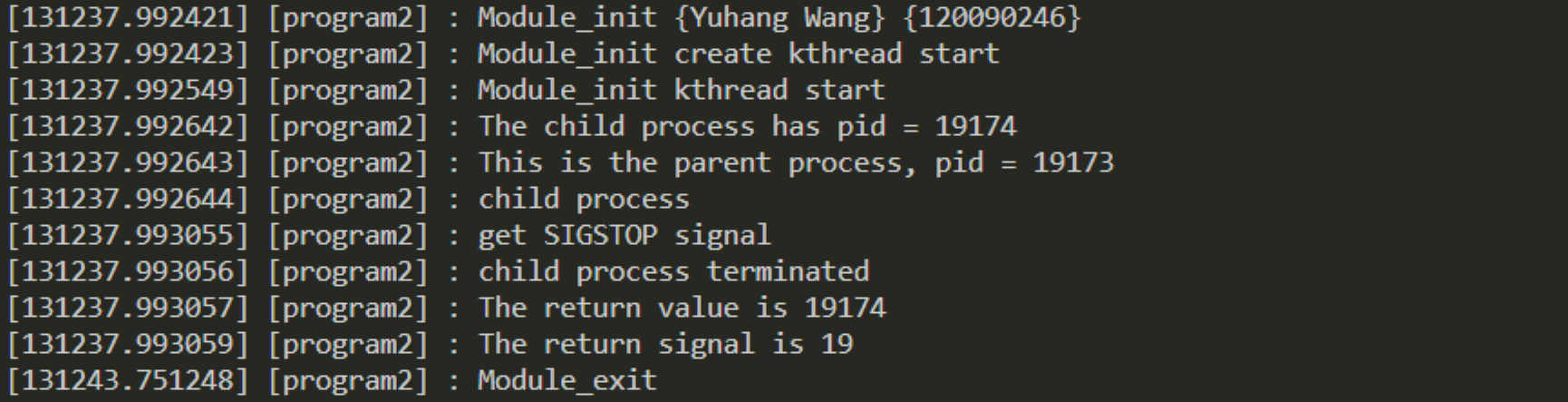
Quit



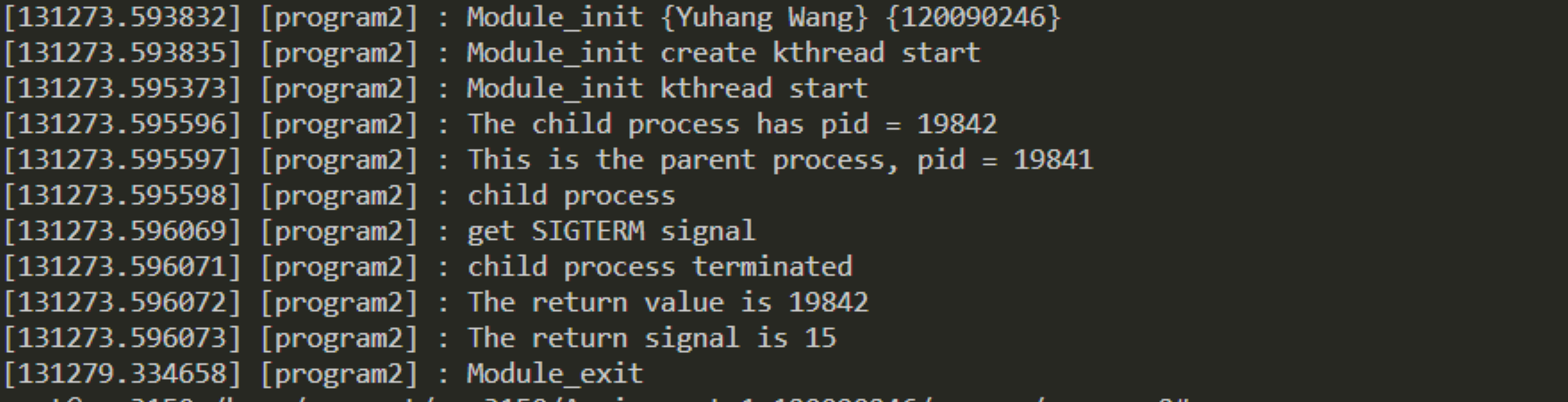
Segment\_fault



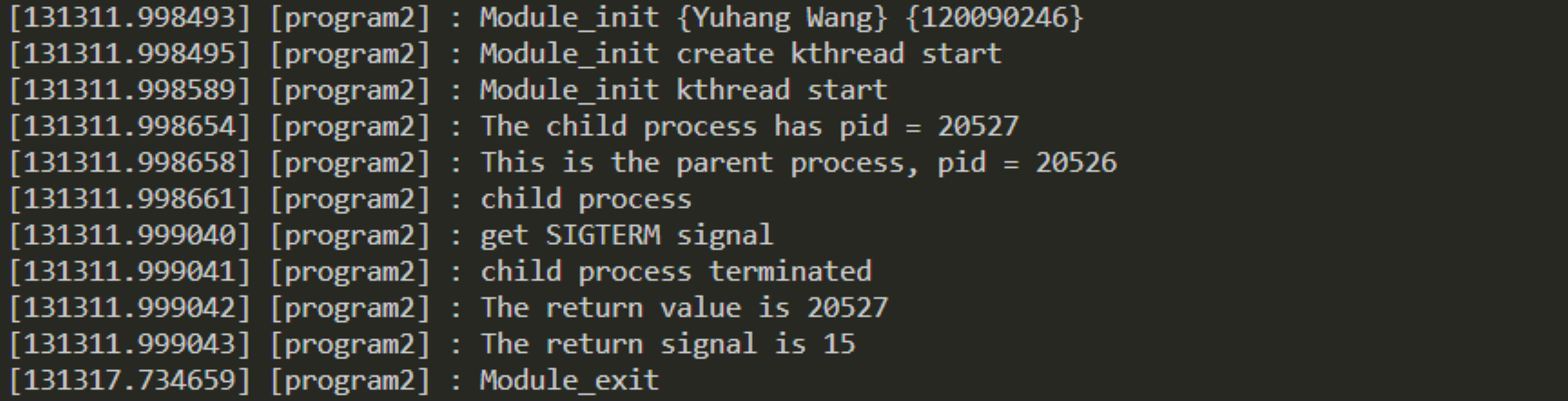
Stop



Terminate



Trap



1. What I Learned
   1. In user mode, I learned to

fork a children process,

use children process to execute certain file or program,

wait until child process terminates,

receive signal raised from the child process,

process the signal, and finally print the signal.

* 1. In kernel mode, I learned to

initialize the kernel module and create kernel thread,

fork a process, execute certain file or program in child module,

wait the children process to terminate, how to execute file in kernel mode, generate kernel object, insert kernel object to kernel module,

remove kernel object, and open kernel log to look into the process.

* 1. I also learned how to compile the kernel, how to modify the source code in the kernel source file, how to recompile the kernel, and how to write Makefile code, and the usage of clang-format.