120090850 Long Jiyuan

**Report**

1. **The design of the program**

1.1

For program1, The structure is that:

Firstly use fork() function to fork a child process, split to child process part and parent process part.

In child process part, use the given template to execute the file. And print out message related to child process. In parent process part, after waitpid(), status contains all return information of child process. If so, print out “Normal termination ……”.

Using WIFSIGNALED to check if child process is terminated by signal. And get the signal value by using WTERMSIG. With a signal table found online, complete the corresponding output. And using WIFSTOPPED to check if child process is stopped.

1.2

For program2, The structure is that:

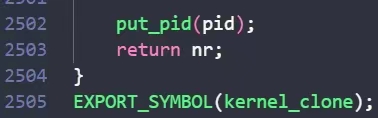
Firstly, program2\_init is invoked by module\_init(). Then the initialization function call kthread\_create() and pass my\_fork as a parameter. After wake\_up\_process(), it goes to my\_fork(). In my\_fork(), fork a process to execute the test program, using kernel\_clone(). Then according to mywait() function which calls the dowait() function, the parent process will wait until child process terminates.

Also the kernel\_clone() function will return a pid for child process and we can also know its parent pid. And print them out.

The do\_execve() function will invoke the test case, which will use getname\_kernel() to return the path of the test case.

After that within this test program, it will raise signal. The signal could be caught and related message should be printed out in kernel log.

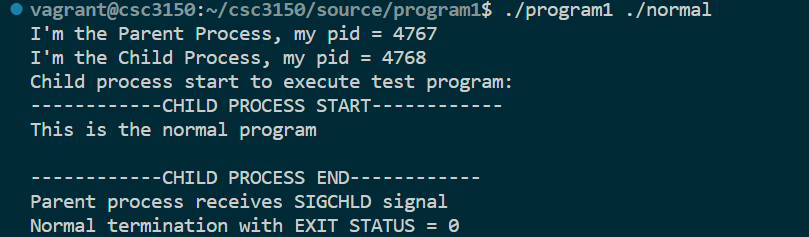
1. **To set up the development environment, including how to compile kernel**
2. Download and install VMBox and set up the connection within SSH
3. Compile the kernel according to tut’s ppt
4. Find the kernel\_clone(), do\_execve(), getname\_kernel() and do\_wait() functions and use EXPORT\_SYMBOL() to export them
5. The screenshot of (3)



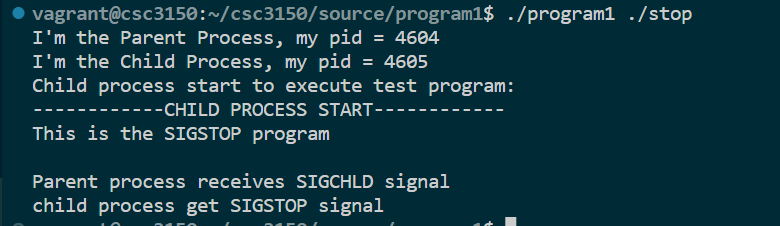
1. Compile the kernel 1. make bzImage, 2. make modules 3. Install kernel modules 4. make install
2. Reboot
3. **Screenshot of your program output**

Program1

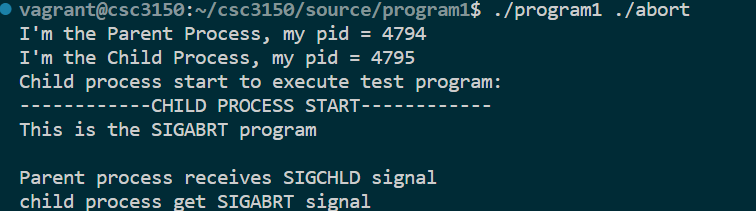
1. In the 'program1' directory, type 'make' command and enter.
2. In the 'program1' directory, type './program1 $TEST\_CASE $ARG1 $ARG2 ...'
3. Normal



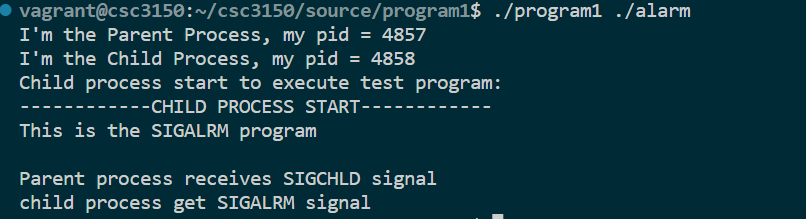
1. Stop



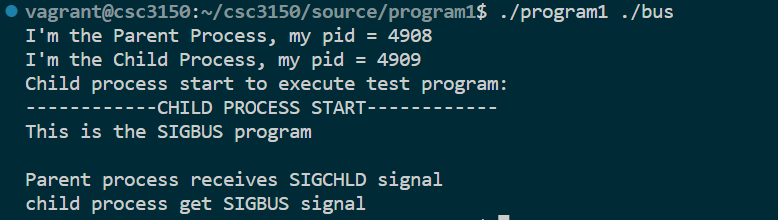
1. Abort



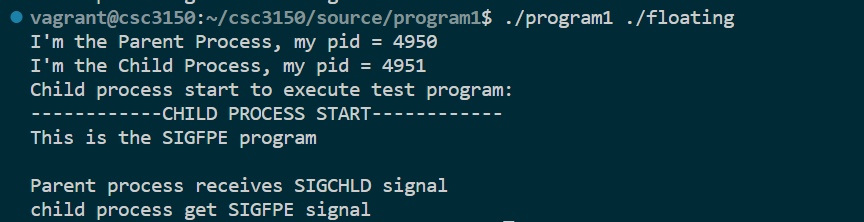
1. Alarm



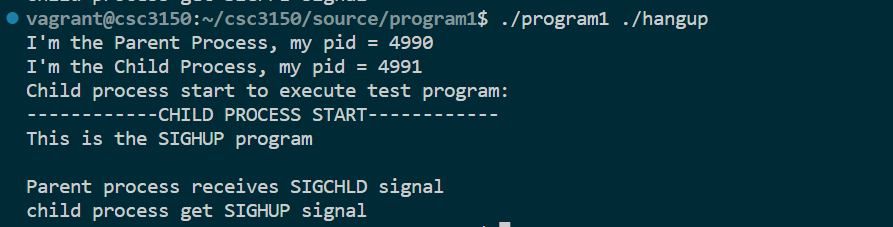
1. Bus



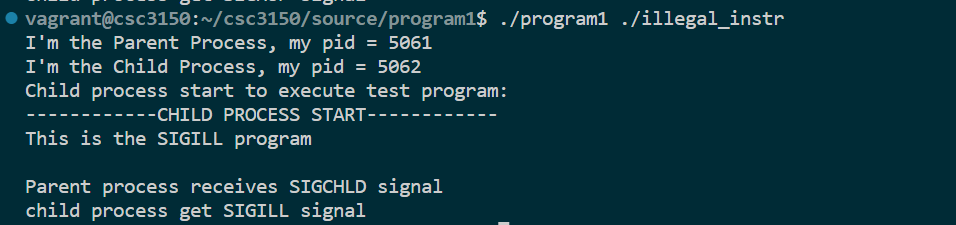
1. Floating



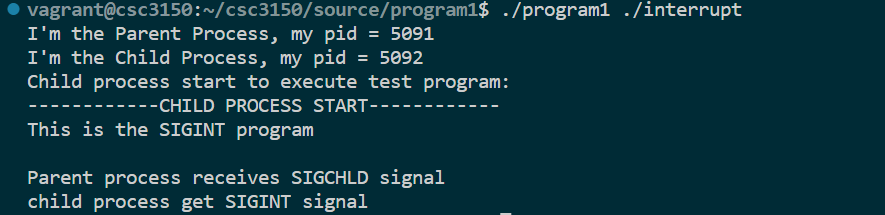
1. Hangup



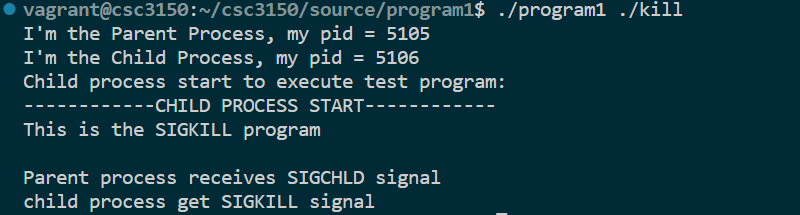
1. Illegal\_instr



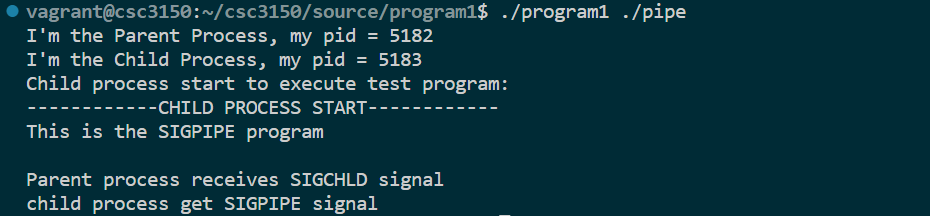
1. Interrupt



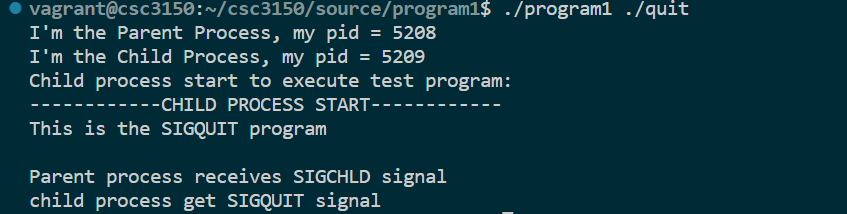
1. Kill



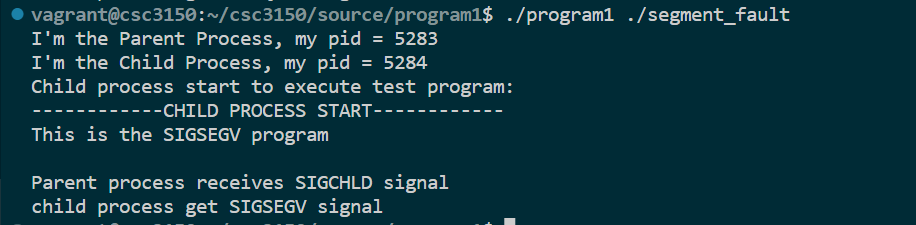
1. Pipe



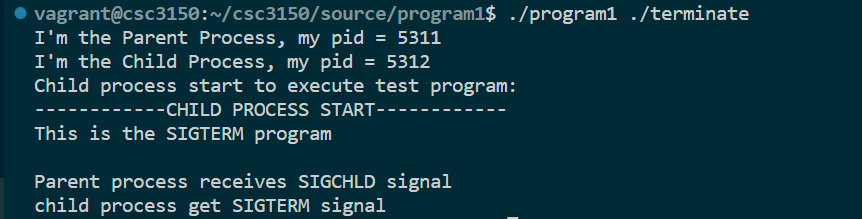
1. Quit



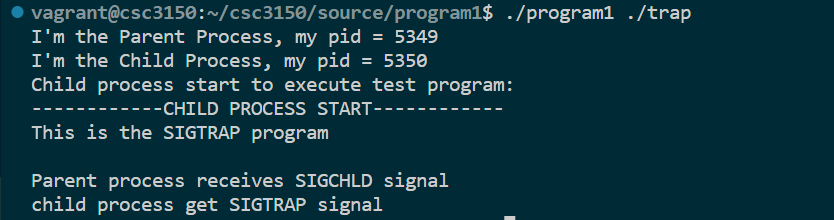
1. Segment\_fault



1. Terminate

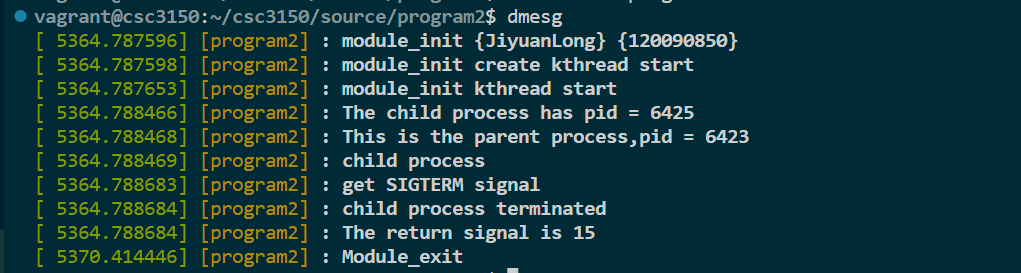


1. Trap

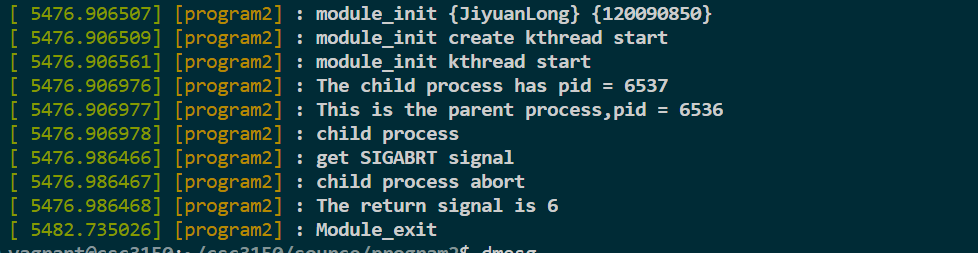


Program2

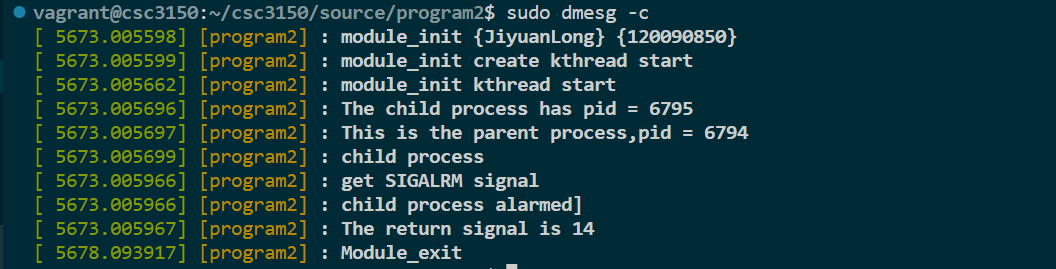
1. Make
2. Sudo insmod program2.ko
3. Sudo rmmod program2
4. Dmesg
5. Term



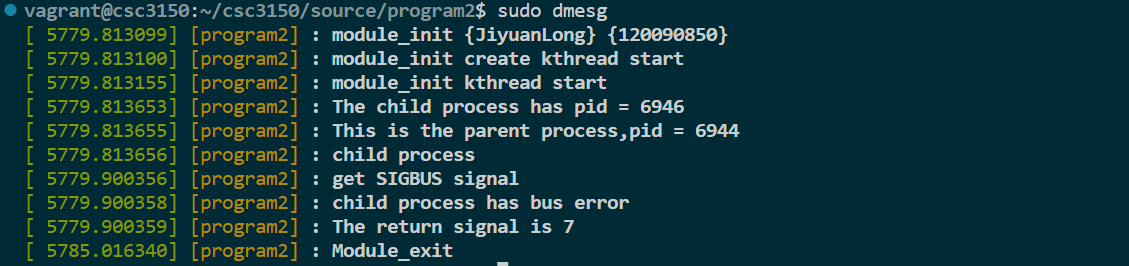
1. Abort



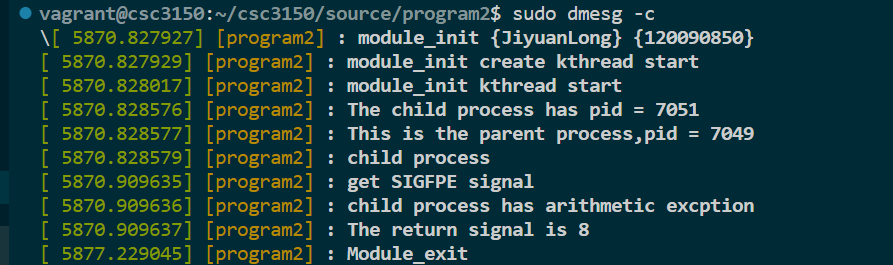
1. Alarm



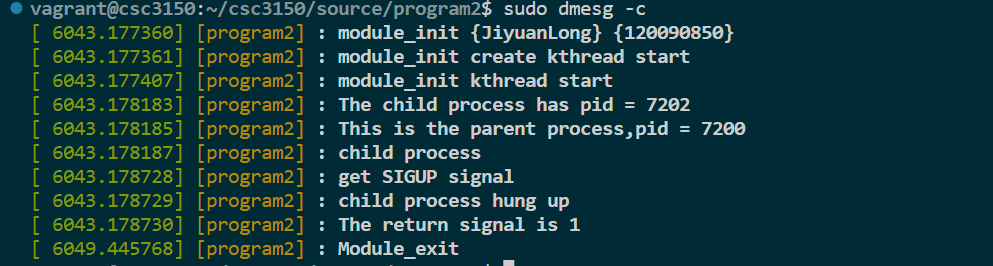
1. Bus



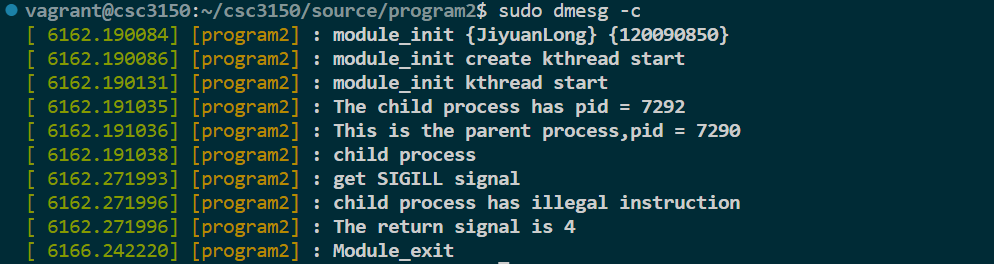
1. Float



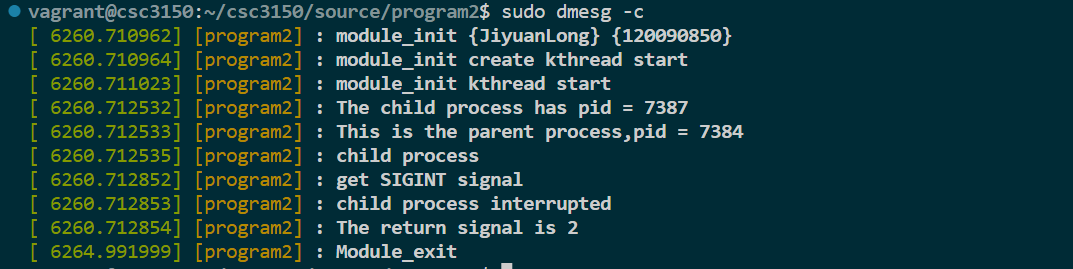
1. Hangup



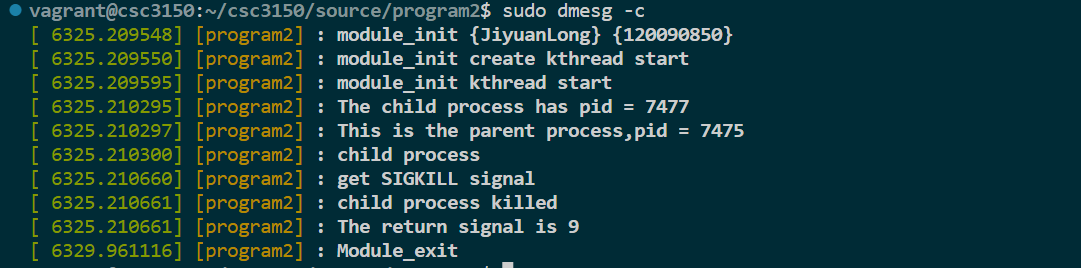
1. Illegal



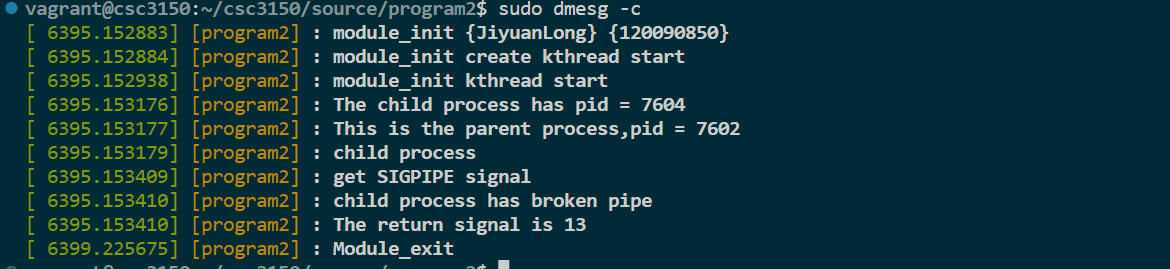
1. Interrupt



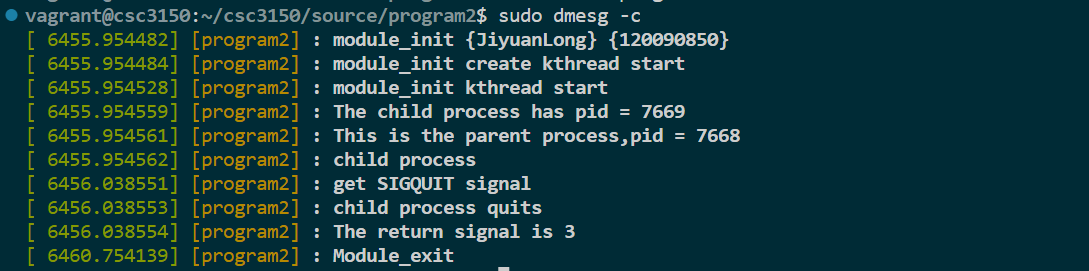
1. Kill



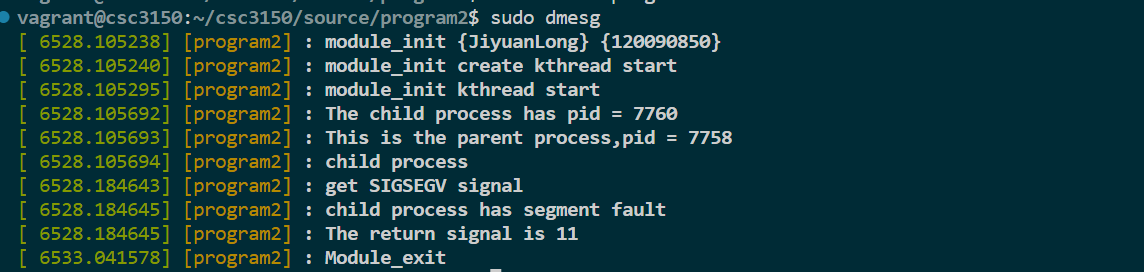
1. Pipe



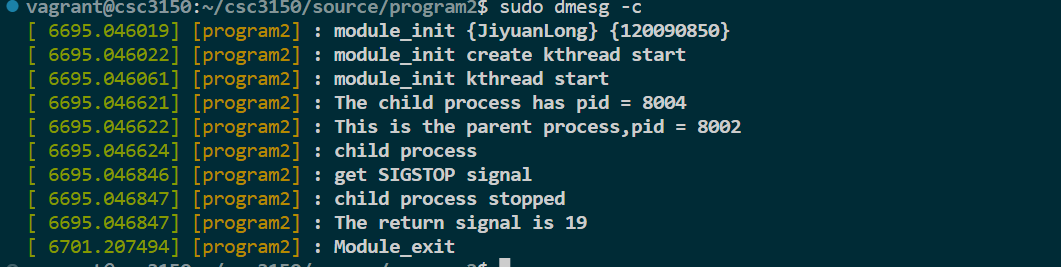
1. Quit



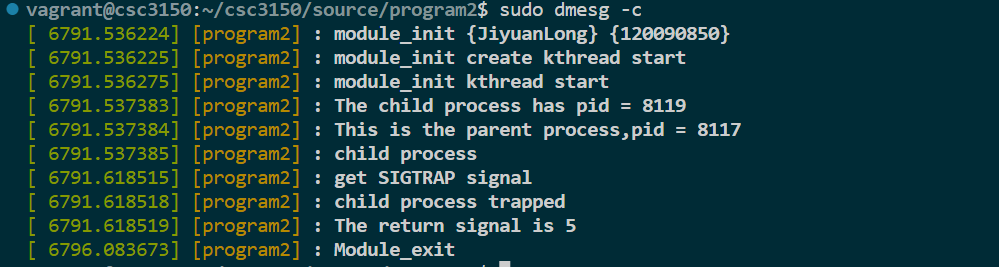
1. Segment fault



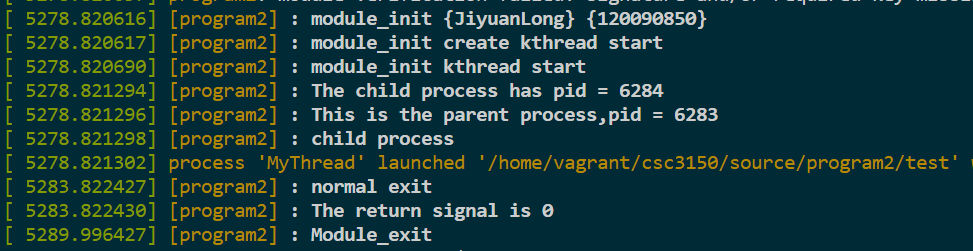
1. Stop



1. Trap



1. Normal



1. **Things I have learned**

I have learned some kernel functions. And how a process was create and how it can be terminated, in other words, the whole life of a process. What’s more I have learned to check the kernel functions.

Using clang-format:

