CSC3150 Assignment1 Report1

Nuoan Zhang 120090851

1. Design

Task1:

The program use fork() to create a child process and distinguish the process depending on the pid. Then use execve() to execute the test program. In the parent part, use waitpid() to wait it. After it terminates or stop,use WIFEXITED(status),WIFSIGNALED(status),WIFSTOPPED（status）to analyze the end.

Task2:

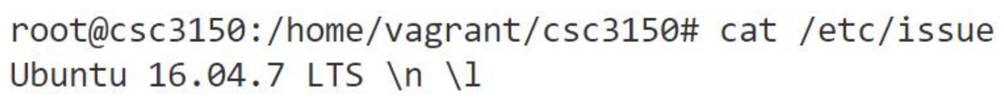
First, to get access to do wait(),kernel_clone(), do execve(), getname_kernel(), remove static before the declaration of do wait() and add “EXPORT SYMBOL GPL(function name);” in Linux kernel source code files and recompile kernel. And then add “extern function declaration;” in program2.c.

In my_fork(),fork a child process to do my_exec().And In my_exec(), it use do_execve() and the first parameter is the file name got by getname_kernel(). And in my_wait(),use do_wait to detect the termination and stop of the process.

After child process terminates or stops, analyze the end of child process by exit status.

2. Environment.

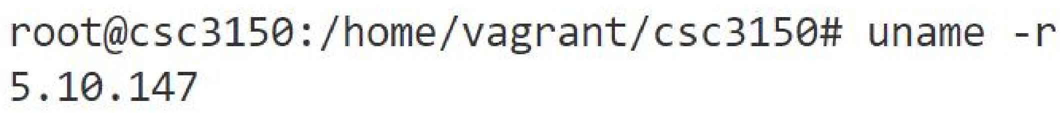
The Linux version:



```
root@csc3150:/home/vagrant/csc3150# cat /etc/issue
Ubuntu 16.04.7 LTS \n \l
```

Just install virualbox and vagrant to set up the VM.
Then set up the SSH plugin in VS code.

The kernel version:



```
root@csc3150:/home/vagrant/csc3150# uname -r
5.10.147
```

To compile the kernel, I first download the souce code and install dependency and development tools. Then extract the source file, copy config, login root account and go to kernel source directory, clean previous setting and start configuration, build kernel Image and modules, install kernel modules and kernel.

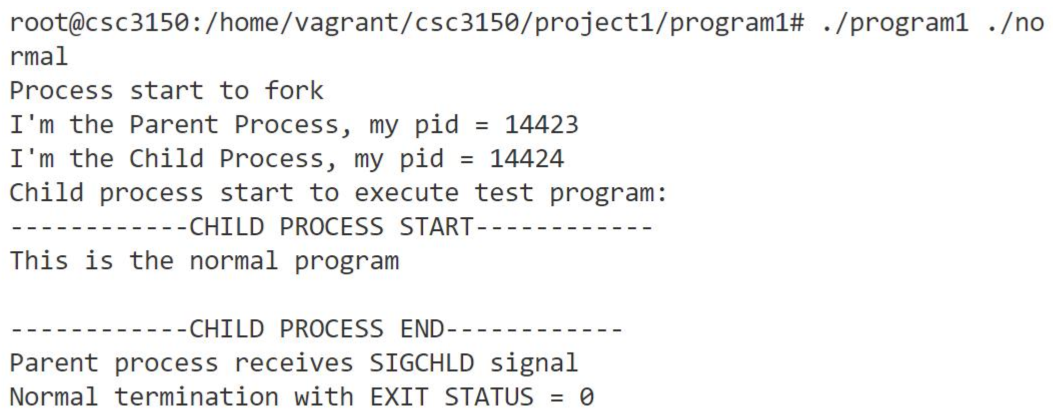
Finally, reboot to load the new kernel.

3. Output

Task1:

Output for normal termination

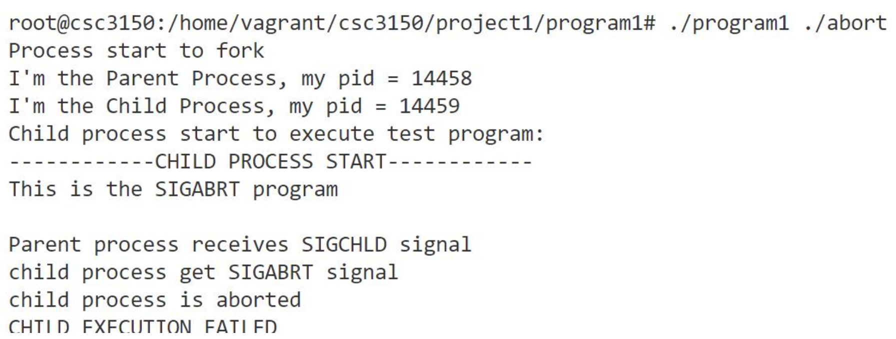
proc



```
root@csc3150:/home/vagrant/csc3150/project1/program1# ./program1 ./no
rmal
Process start to fork
I'm the Parent Process, my pid = 14423
I'm the Child Process, my pid = 14424
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
```

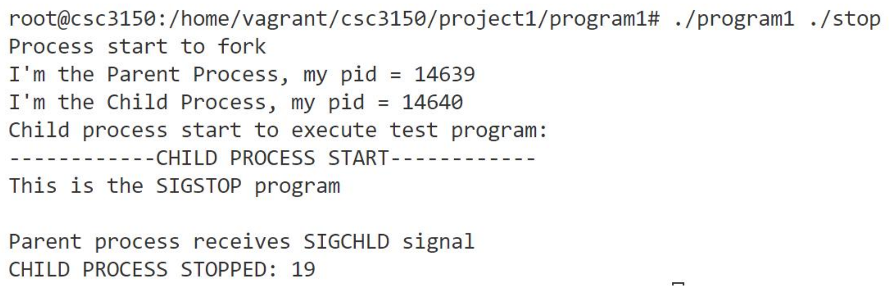
Output for signaled abort



```
root@csc3150:/home/vagrant/csc3150/project1/program1# ./program1 ./abort
Process start to fork
I'm the Parent Process, my pid = 14458
I'm the Child Process, my pid = 14459
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
child process is aborted
CHILD EXECUTION FAILED
```

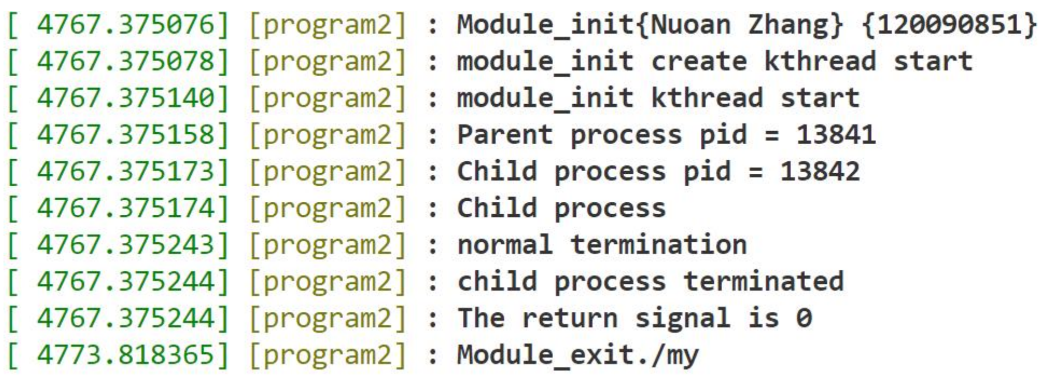
Output for stopped



```
root@csc3150:/home/vagrant/csc3150/project1/program1# ./program1 ./stop
Process start to fork
I'm the Parent Process, my pid = 14639
I'm the Child Process, my pid = 14640
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGSTOP program

Parent process receives SIGCHLD signal
CHILD PROCESS STOPPED: 19
```

Task2:



```
[ 4767.375076] [program2] : Module_init{Nuoan Zhang} {120090851}
[ 4767.375078] [program2] : module_init create kthread start
[ 4767.375140] [program2] : module_init kthread start
[ 4767.375158] [program2] : Parent process pid = 13841
[ 4767.375173] [program2] : Child process pid = 13842
[ 4767.375174] [program2] : Child process
[ 4767.375243] [program2] : normal termination
[ 4767.375244] [program2] : child process terminated
[ 4767.375244] [program2] : The return signal is 0
[ 4773.818365] [program2] : Module_exit./my
```

4. What did I learn.

Task1: process is identified by the pid, we can use fork() to create one. For its end, we can use functions WIFEXITED(), WIFSIGNALED() and WIFSTOPPED() to detected them(normal termination, abnormal termination and stop.

Task2:Learn many ways to modify some parts of Linux kernel source code and the 5 kinds of default disposition of signals.