Luo Zhexu's CSC3150 Assignment1 Report

Name: LuoZhexu **Student ID:** 120090427

1.Overview

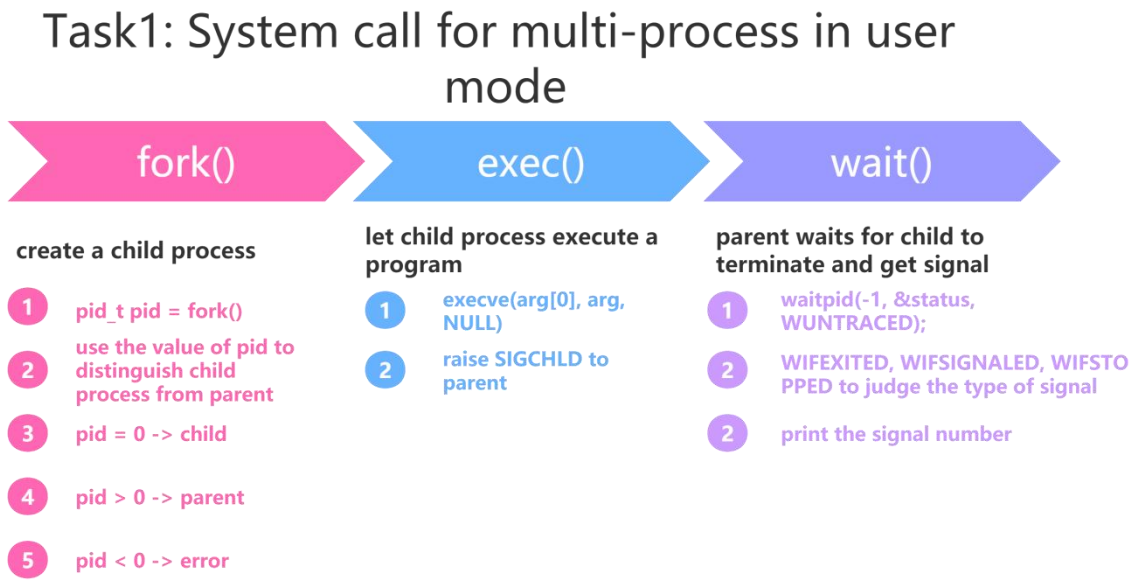
In this project, we mainly implemented kernel mode multi process programming to understand how users call functions in the kernel to operate, and how the kernel modules are designed and operated.

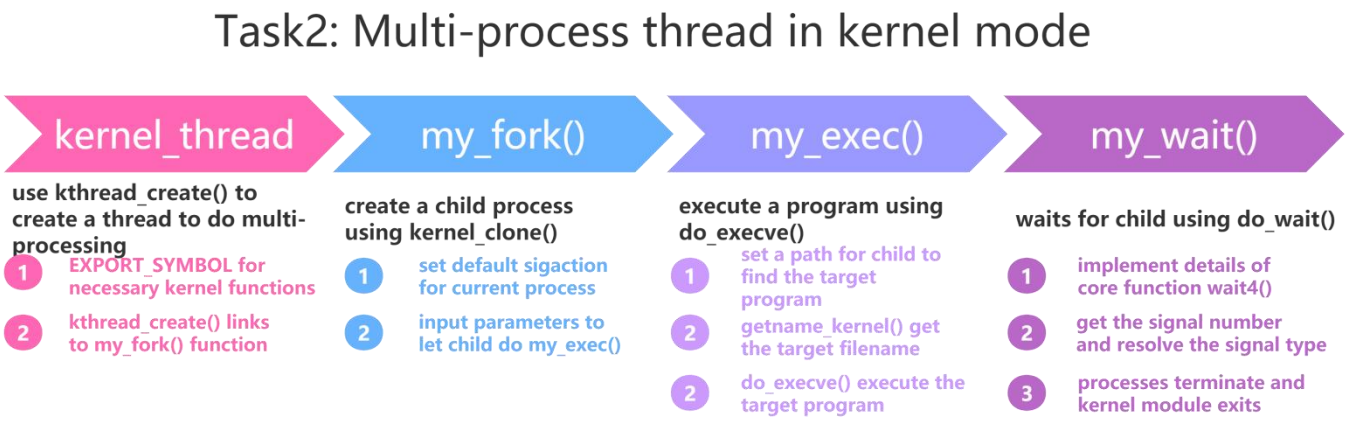
In Task 1, we mainly use the system-call function fork() in user mode to create a child process, and use the exec() and wait() functions to simulate the execution file of the child process and the parent process's waiting for the termination of the child process. Finally, the parent process receives the termination signal from the child process to complete the entire process.

In Task 2, we go deep into the kernel mode, directly called the corresponding functions used to create child processes, execute processes, and wait for processes in the kernel mode, write a kernel module, learn to load and exit the kernel, and achieve creating threads in the kernel mode to realize multiple processes.

2.Program Structure

The general structure of the two main tasks is as follows.





3.Details

(1)Preliminary work

Before proceeding with the two main tasks, we must first create a suitable running environment for them. In order not to affect the use of the host, we use a virtual machine for experiments. The virtual machine system is Linux Ubuntu.

After selecting the virtual machine, we should select the appropriate Linux kernel version. In this experiment, we downloaded Linux 5.10.38 from the store, placed it in the appropriate directory, and copied the config file to the same directory. Next, we use $make mrproper, $make clean, $make menuconfig to clean previous setting and start configuration. After that, we use $make bzImage - j $(nproc), $make modules - j $(nproc) to build kernel images and modules. Finally, we downloaded the kernel module and kernel, and the new kernel was successfully installed.

The following are the virtual machine attributes of this project:

Linux ubuntu-xenial

Kernel version 5.10.38

gcc version 5.4.0 20160609

(2)Task1

This task is relatively simple. When we want to create a new sub process in user mode, we can use the system call function fork() to create it. The fork() function returns the pid twice, representing the child process and the parent process respectively. The child process has a pid of 0, and the parent process has a pid greater than 0. This part can be distinguished by a simple “if” conditional statement. After calling the fork() function, the following contents will be executed by the child

process and the parent process at the same time.

In this task, we want to let the subprocess execute some test files to output various signals. Here we will call execve() in the exec() function. When we input the target file on the command line, the execve() function will get the corresponding parameters and execute the target file, and then output the signal.

In general, the child process and the parent process work at the same time, but here we hope that the parent process can wait until the child process is finished before continuing. The wait() or waitpid() function is used here. The waitpid() function will track the state of the child process. When the state of the child process changes, the parent process will receive necessary information such as signals, and understand the reason for the termination of the child process (such as exceptions). This part is determined by WIFEXITED (to determine whether to exit normally), WIFSIGNALED (to determine whether to exit abnormally), WIFSTOPPED (to determine whether to exit abnormally). If the WIFSIGNALED gets an abnormal signal, we can also use the WTERMSIG function to get the type and number of the abnormal signal. The following are the outputs under normal and abnormal conditions:

```
vagrant@ubuntu-xenial:~/csc3150/source/program1$ ./program1 abort
Process start to fork
I'm the Parent Process, my pid = 10845
I'm the Child Process, my pid = 10846
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 EXECUTION FAILED: 6
vagrant@ubuntu-xenial:~/csc3150/source/program1$
```

```
vagrant@ubuntu-xenial:~/csc3150/source/program1$ ./program1 alarm
Process start to fork
I'm the Parent Process, my pid = 10893
I'm the Child Process, my pid = 10894
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
CHILD EXECUTION FAILED: 14
vagrant@ubuntu-xenial:~/csc3150/source/program1$
```

```
vagrant@ubuntu-xenial:~/csc3150/source/program1$ ./program1 bus
Process start to fork
I'm the Parent Process, my pid = 10923
I'm the Child Process, my pid = 10924
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
CHILD EXECUTION FAILED: 7
vagrant@ubuntu-xenial:~/csc3150/source/program1$
```

```
vagrant@ubuntu-xenial:~/csc3150/source/program1$ ./program1 floating
Process start to fork
I'm the Parent Process, my pid = 10964
I'm the Child Process, my pid = 10965
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
CHILD EXECUTION FAILED: 8
vagrant@ubuntu-xenial:~/csc3150/source/program1$
```

```
vagrant@ubuntu-xenial:~/csc3150/source/program1$ ./program1 hangup
Process start to fork
I'm the Parent Process, my pid = 11005
I'm the Child Process, my pid = 11006
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
CHILD EXECUTION FAILED: 1
vagrant@ubuntu-xenial:~/csc3150/source/program1$
```

```
vagrant@ubuntu-xenial:~/csc3150/source/program1$ ./program1 illegal_instr
Process start to fork
I'm the Parent Process, my pid = 11054
I'm the Child Process, my pid = 11055
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
CHILD EXECUTION FAILED: 4
vagrant@ubuntu-xenial:~/csc3150/source/program1$
```

```
vagrant@ubuntu-xenial:~/csc3150/source/program1$ ./program1 interrupt
Process start to fork
I'm the Parent Process, my pid = 11098
I'm the Child Process, my pid = 11099
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
CHILD EXECUTION FAILED: 2
vagrant@ubuntu-xenial:~/csc3150/source/program1$
```

```
vagrant@ubuntu-xenial:~/csc3150/source/program1$ ./program1 kill
Process start to fork
I'm the Parent Process, my pid = 11139
I'm the Child Process, my pid = 11140
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
CHILD EXECUTION FAILED: 9
vagrant@ubuntu-xenial:~/csc3150/source/program1$
```

```
vagrant@ubuntu-xenial:~/csc3150/source/program1$ ./program1 normal
Process start to fork
I'm the Parent Process, my pid = 11179
I'm the Child Process, my pid = 11180
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@ubuntu-xenial:~/csc3150/source/program1$
```

```
vagrant@ubuntu-xenial:~/csc3150/source/program1$ ./program1 pipe
Process start to fork
I'm the Parent Process, my pid = 11219
I'm the Child Process, my pid = 11220
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
CHILD EXECUTION FAILED: 13
vagrant@ubuntu-xenial:~/csc3150/source/program1$
```

```
vagrant@ubuntu-xenial:~/csc3150/source/program1$ ./program1 quit
Process start to fork
I'm the Parent Process, my pid = 11259
I'm the Child Process, my pid = 11260
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
CHILD EXECUTION FAILED: 3
vagrant@ubuntu-xenial:~/csc3150/source/program1$
```

```
vagrant@ubuntu-xenial:~/csc3150/source/program1$ ./program1 segment_fault
Process start to fork
I'm the Parent Process, my pid = 11300
I'm the Child Process, my pid = 11301
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGSEGV program

Parent process receives SIGCHLD signal
child process get SIGSEGV signal
CHILD EXECUTION FAILED: 11
vagrant@ubuntu-xenial:~/csc3150/source/program1$
```

```
vagrant@ubuntu-xenial:~/csc3150/source/program1$ ./program1 stop
Process start to fork
I'm the Parent Process, my pid = 11330
I'm the Child Process, my pid = 11331
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
CHILD PROCESS STOPPED: 19
vagrant@ubuntu-xenial:~/csc3150/source/program1$
```

```
vagrant@ubuntu-xenial:~/csc3150/source/program1$ ./program1 terminate
Process start to fork
I'm the Parent Process, my pid = 11370
I'm the Child Process, my pid = 11371
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
CHILD EXECUTION FAILED: 15
vagrant@ubuntu-xenial:~/csc3150/source/program1$
```

```
vagrant@ubuntu-xenial:~/csc3150/source/program1$ ./program1 trap
Process start to fork
I'm the Parent Process, my pid = 11410
I'm the Child Process, my pid = 11411
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
CHILD EXECUTION FAILED: 5
vagrant@ubuntu-xenial:~/csc3150/source/program1$
```

(3)Task2

In this task, we no longer only operate in user mode, but directly experience in the kernel module. We will create a kernel module that can execute multiple processes. After the kernel module is created, a thread will be created automatically, and the thread will call the function we have set to create and conduct multiple processes.

In Task 1, we can easily call system functions to implement multiple processes. However, these functions have more detailed implementation methods in the kernel. Among them, the fork() function will mainly pass the kernel_ clone(), and exec() function is mainly implemented through do_ execve() implementation. The essence of the wait() function is the wait4() function, which mainly uses do_wait() to wait for the child process.

Since it is in the kernel mode, we need to directly use these functions to perform multiple processes. But these functions were not originally open to the public. We need to use EXPORT_SYMBOL to export the functions we need, so that our own kernel modules can directly use it. Therefore, we also need to add this statement to the files corresponding to these kernel functions, which means that we need to modify the kernel. So we need to configure and download the module as we did at the beginning after the modification.

When these things are done, we can start to implement multi-process work. Use "extern" to call these exported functions, as shown in the following figure.

```
/* export functions from kernel */
extern pid_t kernel_clone(struct kernel_clone_args *args);
extern long do_wait(struct wait_opts *wo);
extern int do_execve(struct filename *filename,
                     const char __user *const __user *__argv,
                     const char __user *const __user *__envp);
extern struct filename *getname(const char __user *filename);
extern struct filename *getname_kernel(const char *filename);
```

The main life cycle of our kernel module is as follows: during initialization, the module will create a thread task to execute the "my_fork()" function, which will create a child process to execute the target file through the "my_exec()" function, and then the "my_wait()" function will make the parent process wait for the end of the child process and recycle the signal. When this is done, we exit the module from the kernel.

```
task = kthread_create(&my_fork, NULL, "MyThread");
```

In "my_fork()", we need to call "kernel_clone()". When creating a sub process, the exit signal is SIGCHLD, and the stack address is the function pointer of "my_exec()".

```
/* fork a process using kernel_clone */
struct kernel_clone_args args = {
    .flags = SIGCHLD,
    .child_tid = NULL,
    .parent_tid = NULL,
    .stack = &my_exec,
    .stack_size = 0,
    .exit_signal = SIGCHLD,
};
pid = kernel_clone(
    &args); // the test program will be executed in my_exec() function
```

In "my_exec()", we need to set the correct target file name for "do_execve()", which is implemented through "getname_kernel()". We need to set the correct file path for it.

```c
/* implement exec function */
int my_exec(void)
{
    /* execute a test program in child process */
    const char path[] =
        "/home/vagrant/csc3150/source/program2/test"; // describe the path
    // const char path[] = "/tmp/test";
    struct filename *testfile = getname_kernel(path);
    int exec = do_execve(testfile, NULL, NULL);
    return 0;
}
```

After that, we input the pid of the child process into the "my_wait()" function. This function is very similar to the source code of wait4(). We input appropriate parameters to the structure "wait_opts" to enable it to execute the "do_wait()" key function, receive normal or abnormal signals through pointer analysis, use "&0x7f" to process the signal value, and then output the signal value.

```c
/* implement wait function */
int my_wait(pid_t pid)
{
    int status;
    struct wait_opts wo;
    struct pid *wo_pid = find_get_pid(pid);
    enum pid_type type;
    type = PIDTYPE_PID;

    wo.wo_type = type;
    wo.wo_pid = wo_pid;
    wo.wo_flags = WEXITED | WUNTRACED;
    wo.wo_info = NULL;
    wo.wo_stat = (int __user)status;
    wo.wo_rusage = NULL;

    int a;
    a = do_wait(&wo);
```

The test output under 15 different signals is shown in the figure below:

```
[16480.827107] [program2] : module_init {Luozhexu} {120090427}
[16480.827110] [program2] : module_init create kthread start
[16480.827111] [program2] : module_init kthread start
[16480.828235] [program2] : The child process has pid = 12997
[16480.828237] [program2] : This is the parent process, pid = 12996
[16480.828237] [program2] : child process
[16480.995516] [program2] : get SIGABRT signal
[16480.995518] [program2] : child process terminated
[16480.995519] [program2] : The return signal is 6
[16483.938364] [program2] : Module_exit
root@ubuntu-xenial:/home/vagrant/csc3150/source/program2#
```

```
[16556.254078] [program2] : module_init {Luozhexu} {120090427}
[16556.283203] [program2] : module_init create kthread start
[16556.305835] [program2] : module_init kthread start
[16556.327670] [program2] : The child process has pid = 13150
[16556.346410] [program2] : This is the parent process, pid = 13149
[16556.367307] [program2] : child process
[16558.370331] [program2] : get SIGALRM signal
[16558.384642] [program2] : child process terminated
[16558.401174] [program2] : The return signal is 14
[16559.544496] [program2] : Module_exit
root@ubuntu-xenial:/home/vagrant/csc3150/source/program2#
```

```
[16620.767486] [program2] : module_init {Luozhexu} {120090427}
[16620.793428] [program2] : module_init create kthread start
[16620.817453] [program2] : module_init kthread start
[16620.839303] [program2] : The child process has pid = 13333
[16620.863111] [program2] : This is the parent process, pid = 13331
[16620.891451] [program2] : child process
[16621.015723] [program2] : get SIGBUS signal
[16621.032989] [program2] : child process terminated
[16621.051804] [program2] : The return signal is 7
[16622.542733] [program2] : Module_exit
root@ubuntu-xenial:/home/vagrant/csc3150/source/program2#
```

```
[16664.686091] [program2] : module_init {Luozhexu} {120090427}
[16664.711946] [program2] : module_init create kthread start
[16664.734409] [program2] : module_init kthread start
[16664.755083] [program2] : The child process has pid = 13436
[16664.774872] [program2] : This is the parent process, pid = 13434
[16664.795501] [program2] : child process
[16664.927174] [program2] : get SIGFPE signal
[16664.941618] [program2] : child process terminated
[16664.958352] [program2] : The return signal is 8
[16666.902132] [program2] : Module_exit
root@ubuntu-xenial:/home/vagrant/csc3150/source/program2#
```

```
[16713.785574] [program2] : module_init {Luozhexu} {120090427}
[16713.810475] [program2] : module_init create kthread start
[16713.831829] [program2] : module_init kthread start
[16713.855311] [program2] : The child process has pid = 13538
[16713.874397] [program2] : This is the parent process, pid = 13537
[16713.895735] [program2] : child process
[16713.912035] [program2] : get SIGHUP signal
[16713.928387] [program2] : child process terminated
[16713.946416] [program2] : The return signal is 1
[16715.485701] [program2] : Module_exit
root@ubuntu-xenial:/home/vagrant/csc3150/source/program2#
```

```
[16752.216120] [program2] : module_init {Luozhexu} {120090427}
[16752.244347] [program2] : module_init create kthread start
[16752.270453] [program2] : module_init kthread start
[16752.296023] [program2] : The child process has pid = 13629
[16752.314960] [program2] : This is the parent process, pid = 13628
[16752.335735] [program2] : child process
[16752.452297] [program2] : get SIGILL signal
[16752.466287] [program2] : child process terminated
[16752.482429] [program2] : The return signal is 4
[16753.563456] [program2] : Module_exit
root@ubuntu-xenial:/home/vagrant/csc3150/source/program2#
```

```
[16867.800524] [program2] : module_init {Luozhexu} {120090427}
[16867.825401] [program2] : module_init create kthread start
[16867.849990] [program2] : module_init kthread start
[16867.872776] [program2] : The child process has pid = 13861
[16867.896134] [program2] : This is the parent process, pid = 13859
[16867.920691] [program2] : child process
[16867.936129] [program2] : get SIGINT signal
[16867.950988] [program2] : child process terminated
[16867.971139] [program2] : The return signal is 2
[16869.997847] [program2] : Module_exit
root@ubuntu-xenial:/home/vagrant/csc3150/source/program2#
```

```
[16964.333214] [program2] : module_init {Luozhexu} {120090427}
[16964.356740] [program2] : module_init create kthread start
[16964.379658] [program2] : module_init kthread start
[16964.403264] [program2] : The child process has pid = 13953
[16964.425400] [program2] : This is the parent process, pid = 13951
[16964.449259] [program2] : child process
[16964.465312] [program2] : get SIGKILL signal
[16964.480938] [program2] : child process terminated
[16964.501115] [program2] : The return signal is 9
[16965.865306] [program2] : Module_exit
root@ubuntu-xenial:/home/vagrant/csc3150/source/program2#
```

```
[17488.062453] [program2] : module_init {Luozhexu} {120090427}
[17488.088537] [program2] : module_init create kthread start
[17488.110636] [program2] : module_init kthread start
[17488.142099] [program2] : The child process has pid = 15441
[17488.165333] [program2] : This is the parent process, pid = 15440
[17488.191598] [program2] : child process
[17493.169750] [program2] : get NORMAL signal
[17493.186788] [program2] : child process terminated
[17493.205738] [program2] : The return signal is 0
[17494.174924] [program2] : Module_exit
root@ubuntu-xenial:/home/vagrant/csc3150/source/program2#
```

```
[16789.372959] [program2] : module_init {Luozhexu} {120090427}
[16789.398938] [program2] : module_init create kthread start
[16789.420848] [program2] : module_init kthread start
[16789.443032] [program2] : The child process has pid = 13722
[16789.462640] [program2] : This is the parent process, pid = 13720
[16789.483082] [program2] : child process
[16789.496298] [program2] : get SIGPIPE signal
[16789.512507] [program2] : child process terminated
[16789.528529] [program2] : The return signal is 13
[16790.546239] [program2] : Module_exit
root@ubuntu-xenial:/home/vagrant/csc3150/source/program2#
```

```
[17009.610900] [program2] : module_init {Luozhexu} {120090427}
[17009.629861] [program2] : module_init create kthread start
[17009.649368] [program2] : module_init kthread start
[17009.669746] [program2] : The child process has pid = 14042
[17009.692845] [program2] : This is the parent process, pid = 14041
[17009.716887] [program2] : child process
[17009.820172] [program2] : get SIGQUIT signal
[17009.837775] [program2] : child process terminated
[17009.857105] [program2] : The return signal is 3
[17011.191453] [program2] : Module_exit
root@ubuntu-xenial:/home/vagrant/csc3150/source/program2#
```

```
[17051.137976] [program2] : module_init {Luozhexu} {120090427}
[17051.168774] [program2] : module_init create kthread start
[17051.192604] [program2] : module_init kthread start
[17051.215488] [program2] : The child process has pid = 14131
[17051.237027] [program2] : This is the parent process, pid = 14130
[17051.261424] [program2] : child process
[17051.410106] [program2] : get SIGSEGV signal
[17051.433609] [program2] : child process terminated
[17051.453817] [program2] : The return signal is 11
[17054.538279] [program2] : Module_exit
root@ubuntu-xenial:/home/vagrant/csc3150/source/program2#
```

```
[17098.872430] [program2] : module_init {Luozhexu} {120090427}
[17098.906030] [program2] : module_init create kthread start
[17098.926232] [program2] : module_init kthread start
[17098.946906] [program2] : The child process has pid = 14203
[17098.968016] [program2] : This is the parent process, pid = 14202
[17098.991790] [program2] : child process
[17099.006953] [program2] : get SIGSTOP signal
[17099.023233] [program2] : child process terminated
[17099.040431] [program2] : The return signal is 19
[17100.262268] [program2] : Module_exit
root@ubuntu-xenial:/home/vagrant/csc3150/source/program2#
```

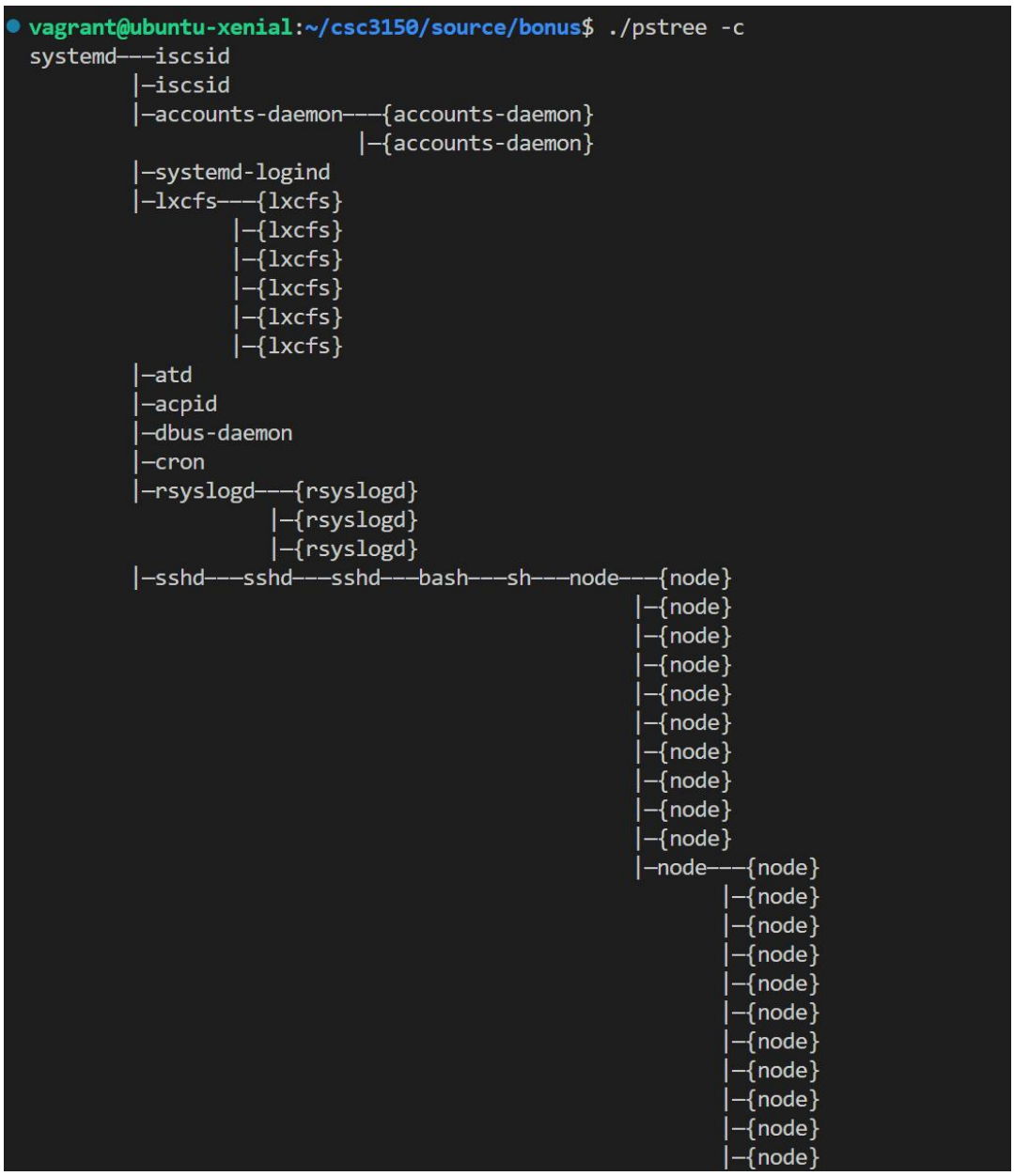
```
[17129.372595] [program2] : module_init {Luozhexu} {120090427}
[17129.396976] [program2] : module_init create kthread start
[17129.420155] [program2] : module_init kthread start
[17129.442249] [program2] : The child process has pid = 14269
[17129.466003] [program2] : This is the parent process, pid = 14268
[17129.492512] [program2] : child process
[17129.511229] [program2] : get SIGTERM signal
[17129.527580] [program2] : child process terminated
[17129.550871] [program2] : The return signal is 15
[17130.865152] [program2] : Module_exit
root@ubuntu-xenial:/home/vagrant/csc3150/source/program2#
```

```
[17206.564819] [program2] : module_init {Luozhexu} {120090427}
[17206.602216] [program2] : module_init create kthread start
[17206.625908] [program2] : module_init kthread start
[17206.648455] [program2] : The child process has pid = 14376
[17206.667187] [program2] : This is the parent process, pid = 14374
[17206.687177] [program2] : child process
[17206.809737] [program2] : get SIGTRAP signal
[17206.823656] [program2] : child process terminated
[17206.840278] [program2] : The return signal is 5
[17207.812510] [program2] : Module_exit
root@ubuntu-xenial:/home/vagrant/csc3150/source/program2#
```

4.Bonus

In the bonus section, I mainly implemented the rough prototype of the pstree. Find the process files in the "/proc" directory, record their names and pid, and form an array. Then build a tree connection between them. The output of the two options "pstree - A" and "pstree - c" is shown in the figure.

```
vagrant@ubuntu-xenial:~/csc3150/source/bonus$ ./pstree -A
systemd-+-iscsid
        |-iscsid
        |-accounts-daemon-+-2*[{accounts-daemon}]
        |-systemd-logind
        |-lxcfs-+-6*[{lxcfs}]
        |-atd
        |-acpid
        |-dbus-daemon
        |-cron
        |-rsyslogd-+-3*[{rsyslogd}]
        |-sshd-+-sshd-+-sshd-+-bash-+-sh-+-node-+-10*[{node}]
                                                |-node-+-11*[{node}]
                                                       |-bash-+-pstree
                                                       `-sh-+-cpuUsage.sh-+-sleep
                                                |-node-+-11*[{node}]
                                                       `-cpptools-+-23*[{cpptools}]
                                                `-node-+-12*[{node}]
                                  `-sleep
                        `-sshd-+-sshd-+-bash-+-sleep
        |-unattended-upgr-+-{unattended-upgr}
        |-mdadm
        |-polkitd-+-2*[{polkitd}]
        |-irqbalance
        |-login-+-bash
        |-agetty
        |-systemd-+-sd-pam)
        |-test
        |-cpptools-srv-+-11*[{cpptools-srv}]
        |-systemd-journal
        |-lvmetad
        |-systemd-udevd
        |-cpptools-srv-+-11*[{cpptools-srv}]
        |-systemd-timesyn-+-{systemd-timesyn}
        `-dhclientvagrant@ubuntu-xenial:~/csc3150/source/bonus$
```



5.Reflections

In this project, the most challenging thing for me is to understand the source code of Linux. Because there are many strange variables and functions in the source code of Linux, it requires patience to understand their principles and how to operate. However, when figuring out how the kernel performs multi-process operations, I can feel the fantasy of the Linux system. In addition, continuous command line operations in Linux also cultivate my knowledge about terminals, which is very important for subsequent programming. In addition, in the process of installing and compiling the kernel, I also learned the structure of the kernel, which deepened my impression of the Linux system.