CSC 3150 Assignment 1 Report

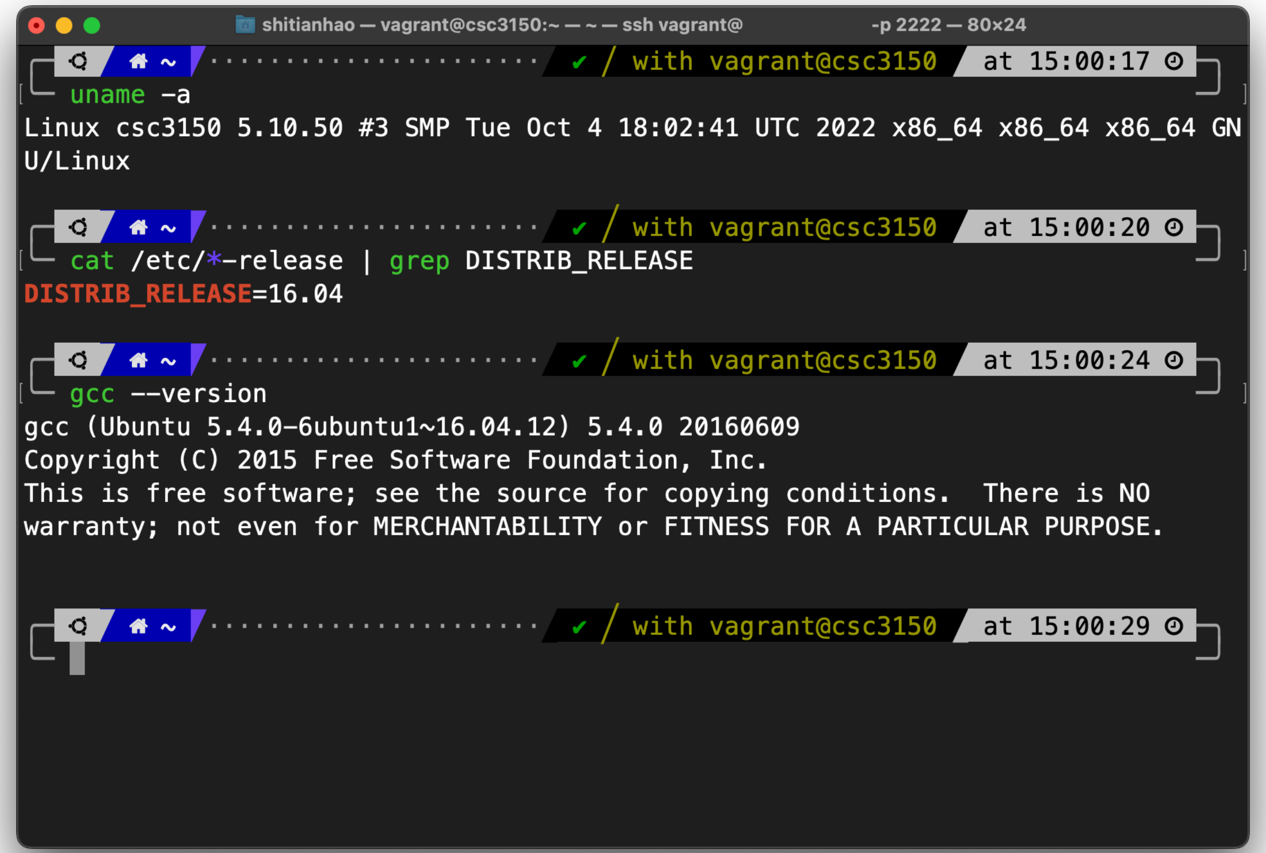
Tianhao SHI, 120090472

Oct 2022

Environment

Environment Info

- **Linux Kernel Version**: 5.10
- **GCC Version**: 5.4.0
- **Ubuntu Distribution Version**: 16.04


shitianhao — vagrant@csc3150:~ — ~ — ssh vagrant@ -p 2222 — 80×24

```
~ ······················ ✔ / with vagrant@csc3150  at 15:00:17
uname -a
Linux csc3150 5.10.50 #3 SMP Tue Oct 4 18:02:41 UTC 2022 x86_64 x86_64 x86_64 GN
U/Linux

~ ······················ ✔ / with vagrant@csc3150  at 15:00:20
cat /etc/*-release | grep DISTRIB_RELEASE
DISTRIB_RELEASE=16.04

~ ······················ ✔ / with vagrant@csc3150  at 15:00:24
gcc --version
gcc (Ubuntu 5.4.0-6ubuntu1~16.04.12) 5.4.0 20160609
Copyright (C) 2015 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.


~ ······················ ✔ / with vagrant@csc3150  at 15:00:29
```

Environment Setup

The following steps are executed during my VM setup:

1. Expand the disk space
    1. Make a copy of the default `vmdk` file in the format of `vdl`
    2. Modify the size of `vdl` file
    3. Convert the format back to `vmdk` and update VM setting to boot from new disk
2. Updating/ Upgrading packages (`sudo apt-get update && sudo apt`)
    - Apt source are modified for faster download
3. Install essential packages
4. Download kernel source code
5. Modify the source code and export some symbols using `EXPORT_SYMBOL()`
    - `do_wait()` from kernel/exit.c
    - `do_execve()` from fs/exec.c
    - `getname_kernel()` from fs/namei.c
    - `kernel_clone()` from kernel/fork.c
6. Compile kernel source code and reboot
    1. save a snapshot of VM
    2. switch into root user `sudo su`
    3. copy the original `.config` file to the downloaded source code folder
    4. `make mrproper`
    5. `make clean`
    6. `make menuconfig` load and save the config file
    7. `make bzImage -j$(nproc)`
    8. `make modules -j$(nproc)`
    9. `make modules_install`
    10. `make install`
    11. `reboot`

Code design

The output of the tasks can be found in the appendix

Task 1

Fork a child process

In task 1, a child is created by calling the `fork()` function. If the child process is created successfully, the program will enter a conditional statement. The child will execute codes in the `pid==0` branch.

Execute the program

In the previous step, the child process will execute the code enclosed in `pid==0` branch. In this branch, the `execve()` fuction is called. This function takes in 3 arguments: the *filename*, which is the path to the executable file (machine code); the *arguments* that the exeutable file takes (there should be none in this assignment, however); and the *environment variable* array used during the execution. It should be noted that the path to the program to be executed is passed as an argument, and is stored at *argv[1]*. The *arguments* corresponds to `argv[2:]` in the arguments passed to `program1.c`, and should end with a `NULL`. The child will then load and execute the specified file.

Parent receive SIGCHLD

In order to receive SIGCHLD sent from the child process, a signal handler isneeded. I defined a function `sigchld_handler` function in `program1.c` and configed the `signal()` function so that whenever SIGCHLD is raised, a message will be printed.

Print out the termination status of child process

In the first step, while child process executes code in `pid==0` branch, parent process will go into the `else` branch. It will call the `wait_pid(pid_t pid, int* status, int options)` function. From the parent process's point of view, the `pid` will be the child process's pid. The `status`, passed by refrence, will record the termination status. In task 1, the `options` is set as `WUNTRACED` because it is possible that the child will be killed/stopped.

Macros defined in nolib.c can be used to analyze the exit status

- Whether the child process is exited can be checked by `WIFEXITED`

    - The return status can be required by `WEXITSTATUS`
- Whether the child process is terminated by a signal can be checked by `WIFSIGNALED`
    - The signal can be further achieved analyzed by `WTERMSIG`
- Whether the child process is stopped by a signal can be checked by `WIFSTOPPED`
    - The signal can be further acquired by `WSTOPSIG`
- Although not covered in the testcase, whether a process is resumed can be checked by `WIFCONTINUED`

To make the output more semantic, I wrote a function called `getsig(int sig)` that returns the string of the signal name. The underlying value of each process can be found here. It should be noted that the signals are platform-dependent. In this assignment I selected x86 architecture. I assume the testing environment should be x86 as well.

```
#define SIGHUP           1
#define SIGINT           2
#define SIGQUIT          3
#define SIGILL           4
#define SIGTRAP          5
#define SIGABRT          6
#define SIGIOT           6
#define SIGBUS           7
#define SIGFPE           8
#define SIGKILL          9
#define SIGUSR1         10
#define SIGSEGV         11
#define SIGUSR2         12
#define SIGPIPE         13
#define SIGALRM         14
#define SIGTERM         15
#define SIGSTKFLT       16
#define SIGCHLD         17
#define SIGCONT         18
#define SIGSTOP         19
#define SIGTSTP         20
#define SIGTTIN         21
#define SIGTTOU         22
#define SIGURG          23
#define SIGXCPU         24
#define SIGXFSZ         25
#define SIGVTALRM       26
#define SIGPROF         27
#define SIGWINCH        28
#define SIGIO           29
#define SIGPOLL         SIGIO
/*
#define SIGLOST         29
*/
#define SIGPWR          30
#define SIGSYS          31
#define SIGUNUSED       31
```

Task 2

Create a kernel thread to run my_fork

A new kernel thread will be created using the `kthread_create(threadfn, data, namefmt, arg...)`. The first argument is the self-implemented function `my_fork`. The `data` argument is set to NULL, and `namefmt` is just a name for the thread.

The `kthread_create` only create a new thread, and a `task_struct` struct is returned., to start the thread, I used `wake_up_process(struct taskstruct *p)`, and passed a pointer to the newly created task_struct.

Fork a process to execute the test program

As of version 5.10.50, the system call `fork` is done through calling the `kernel_thread()` function, which is just a wrapper function of `kernel_clone()`. Therefore, I only exported the latter. This function takes in a `kernel_clone_args` struct, which describes how the child process is forked. To guarantee same behaviour, my args are modified from the settings in the `kernel_thread` function:

```
struct kernel_clone_args args = {
    .flags      = ((lower_32_bits(flags) | CLONE_VM |
            CLONE_UNTRACED) & ~CSIGNAL),
    .exit_signal   = (lower_32_bits(flags) & CSIGNAL),
    .stack      = (unsigned long)fn,
    .stack_size = (unsigned long)arg,
  };
```

In the context of task 2, the flags are set to **SIGCHLD**, the fn is a self-written **`my_exec`**, which is responsible for executing the test program. The arg is simply set as NULL.

Print the process id of both the parent and child process

The parent process's pid is readily achieved though the `current` pointer (current->pid). The child process's pid, assuming successful creation, is simply the return value of `kernel_clone`.

Exectue the test program

The execution is completed by calling the `do_execve` function, which is the kernel-space version of the `execve` function. It also takes three arguments. The first argument is a `struct filename` pointer. The actual path to the file needs to be transferred to this type by calling the `getname_kernel()` function (as discussed in the piazza forum, the getname function will not work in 5.10.50). The other two arguments, `argv` and `envp` are simply set to NULL.

Parent wait for child process and capture signal

The waiting process is achieved by calling the `my_wait` function, which takes 2 arguments: the target process's pid (in this case, the pid of child process), and an integer status (passed by refrence). This function is a wrapper function for the acutal `do_wait` function. To configure the waiting process, a `wait_opts` struct is defined:

```
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;
};
```

It should be noted that wo_stat is the `status` argument. After the `do_wait` completion, the status should be updated. The wo_flags should be WEXITED | WUNTRACED because we want to trace not only normal exit of child process, but also whether it is beed stopped or termnated by signals. The wo_pid is achieved through looking up a hash table using the `find_get_pid` function.

Catching signal and Parsing exit status

The `status` variable is passed by refrence to the `my_wait` function. Its value at the end of function execution contains information about the child process. However, the macros used in task 1 cannot be used in kernel space. Therefore, in program2.c, I copied these macros so that they can be readily applied to the analyze the status. The macros used are:

- __WEXITSTATUS
- __WTERMSIG
- __WSTOPSIG
- __WIFEXITED
- __WIFSIGNALED
- __WIFSTOPPED

For semantic output, as I did in task 1, I used the `getsig` to output the signal name as string

Bonus

The bonus program asked us to implement the `pstree` command in linux system.

Key data structure

N-ary tree

An n-ary tree is used as the data structure. Each node of the tree is defined as a struct:

```
struct proc_node{
  proc_info info;

  proc_node *parent;
  proc_node *first_child;
  proc_node *next_sibling;
} ;
```

The `proc_info` field contains information abut the process. The `parent` is a pointer pointing to the process's parent process. The `first_child` is a pointer to the first child process node, and any sibling process can be get by traversing the `next_sibling` pointer.

The proc_info is also a struct:

```
struct proc_info{
  pid_t pid;
  pid_t ppid;
  std::string name;
  std::string cmdline;
};
```

Map

To guaranteen quick access, a <int, proc_node*> map is created. The address of a given node can be accessed in constant time complexity.

Basic Information

In linux system, the `/proc` folder in Linux contains information about the current state of kernel. The folder contains nunmbered folders, e.g: */proc/2*, which contains information about a process (in this example, process with PID=2). The 'status' file in this folder contains the PID, PPID, name of this process.

Workflow

1. Parse the arguments and config the output stype
2. Scan the **/proc** directory, use regex to match all process folder
3. For each process foder, if it's not in the map, put it in the map
    1. Access the /proc/PID/status file and the /proc/PID/cmdline
    2. Acquire the PID, PPID, Name from the file
    3. Create a node using the above-mentioned information
    4. Traverse using PPID until a parent node is already present in the map
4. After step 2, the tree is created.
5. Print the tree out using Depth-first-search

Arguments I implemented

- -V: print out the verbose information about pstree
- -p: print out the PID of each process
- -c: print out the commandline of each process
- -s: show the parent process

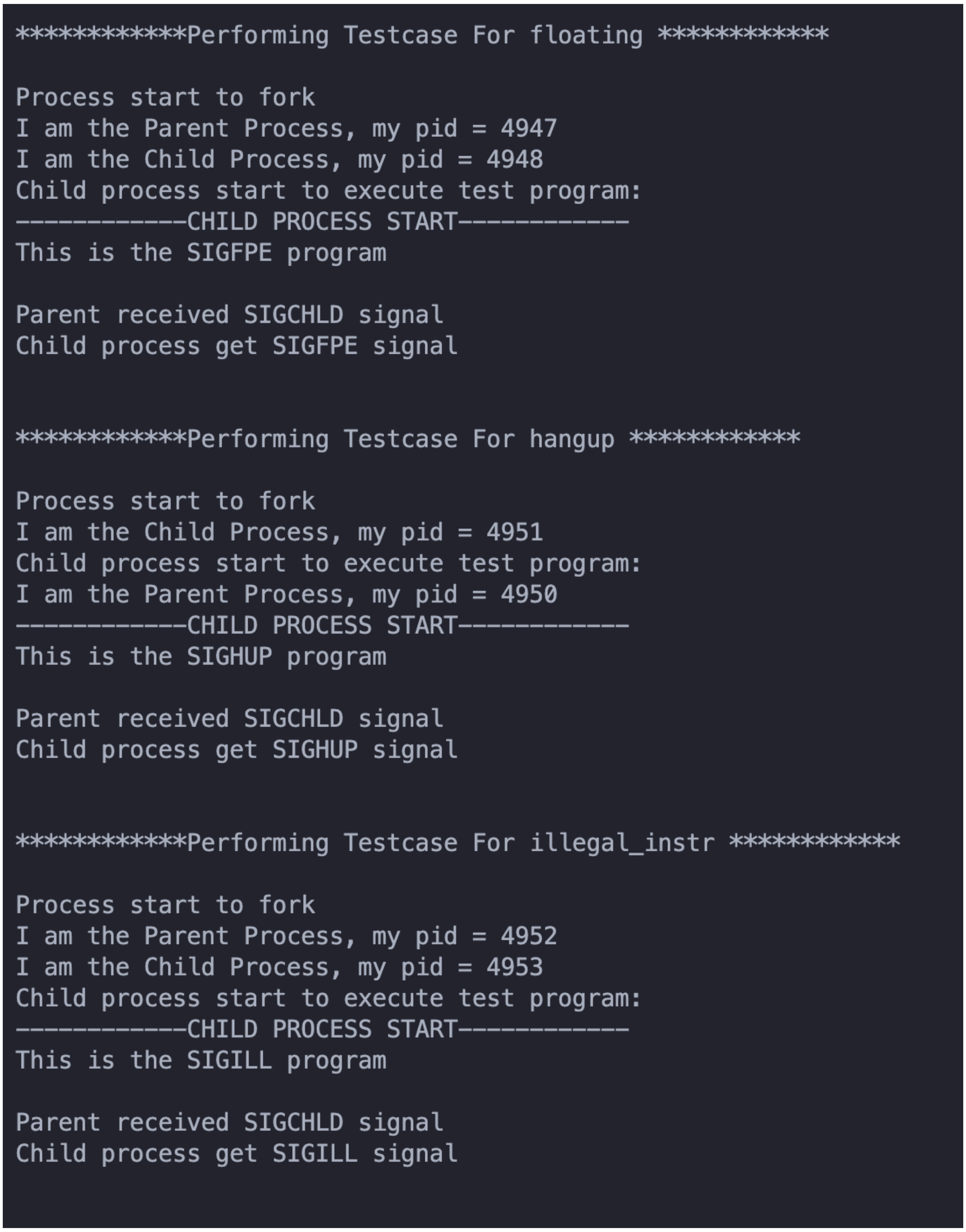
Learning Outcome

Both task 1 and task 2 allows me to gain hand-on experience in C programming. They allow me to understand how Linux processes are created both from the user space and the kernel space. Moreover, I learned what it mean to 'execute' a binary file.

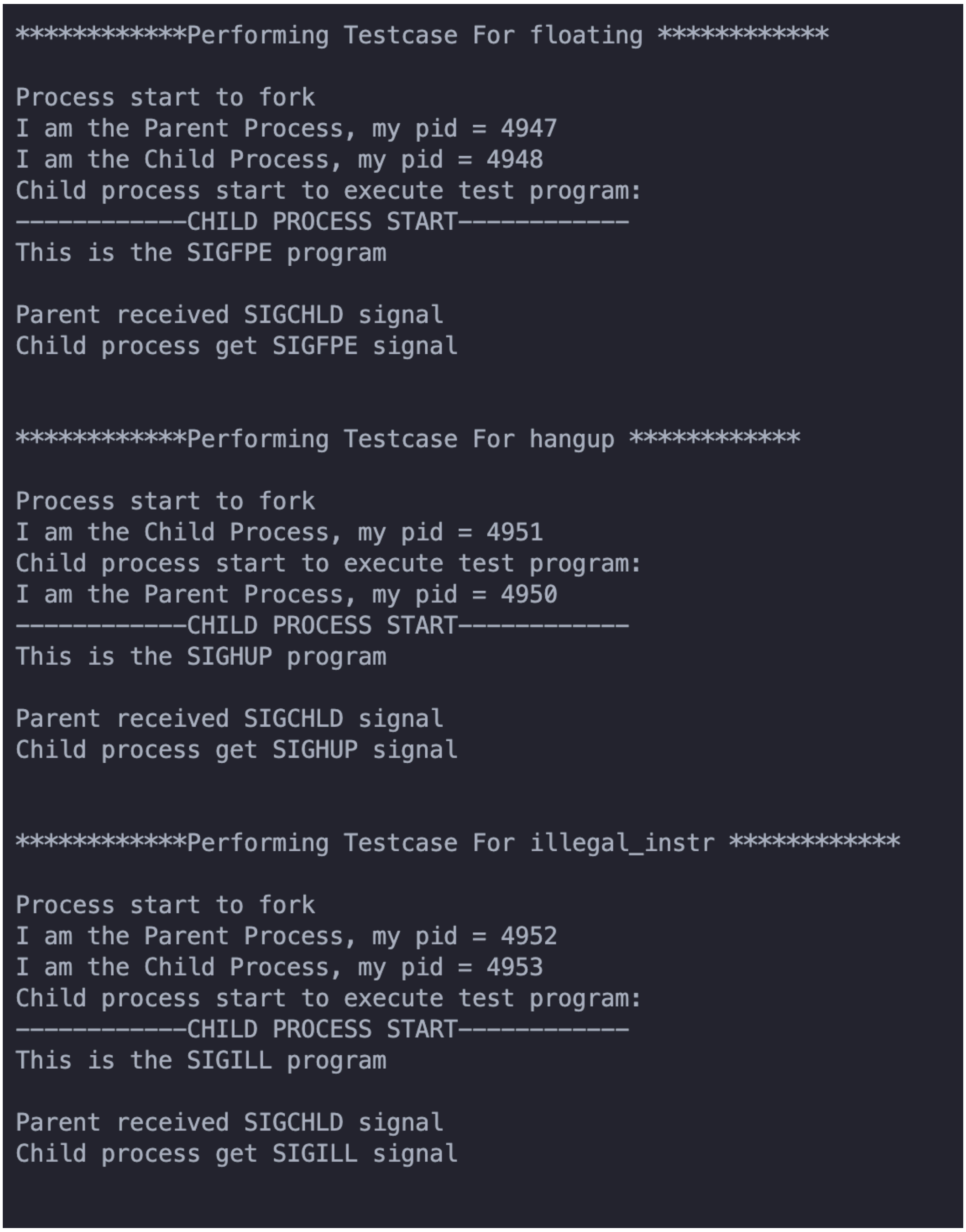
After this assgnment, I learned how to write a basic linux kernel module and insert it into the running kernel using LKM. I also learned to patiently read the source code of Linux Kernel and understand the APIs(where a symbol is defined and refrenced etc.).

Appendix

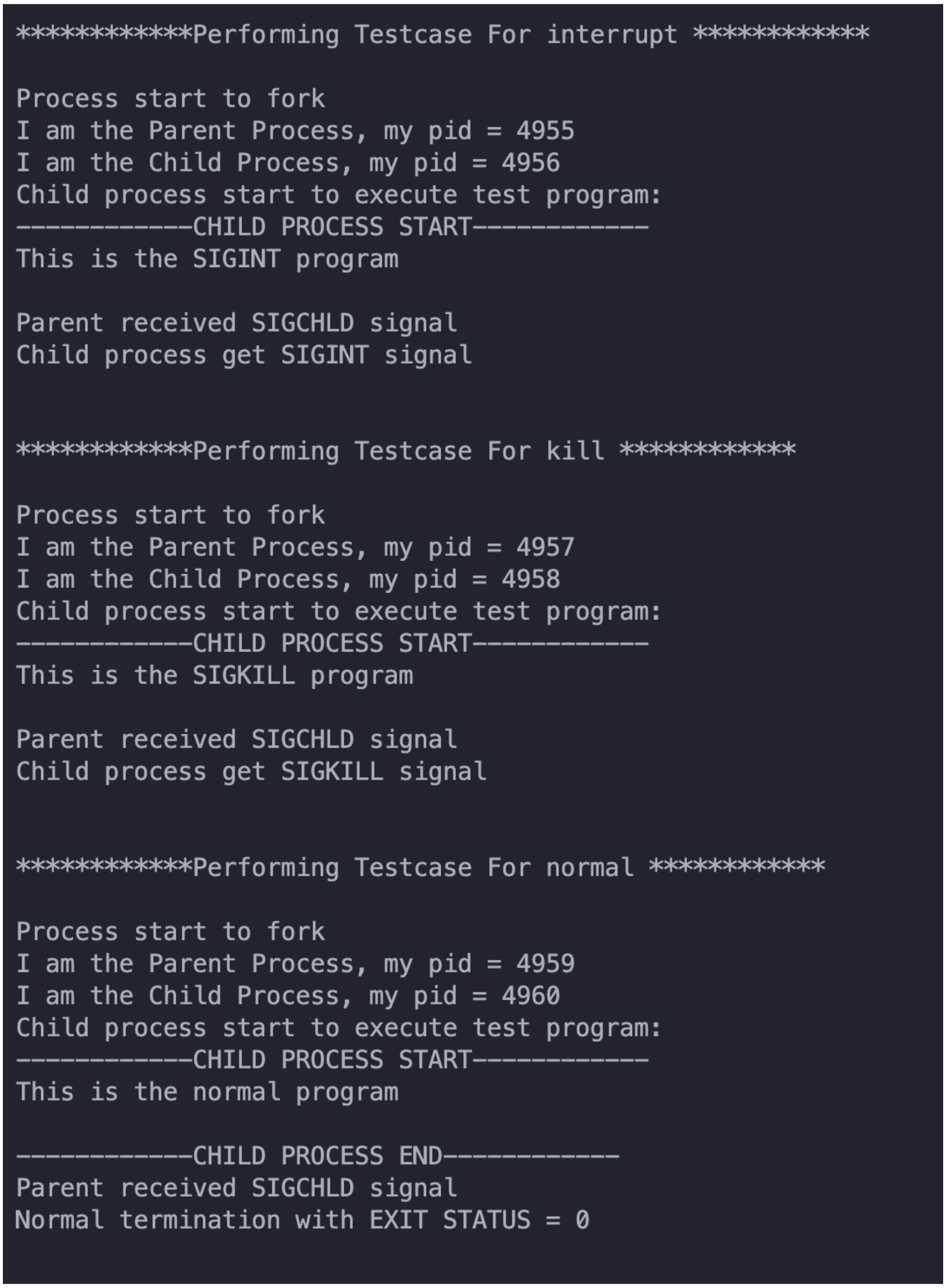
Program output for Task 1



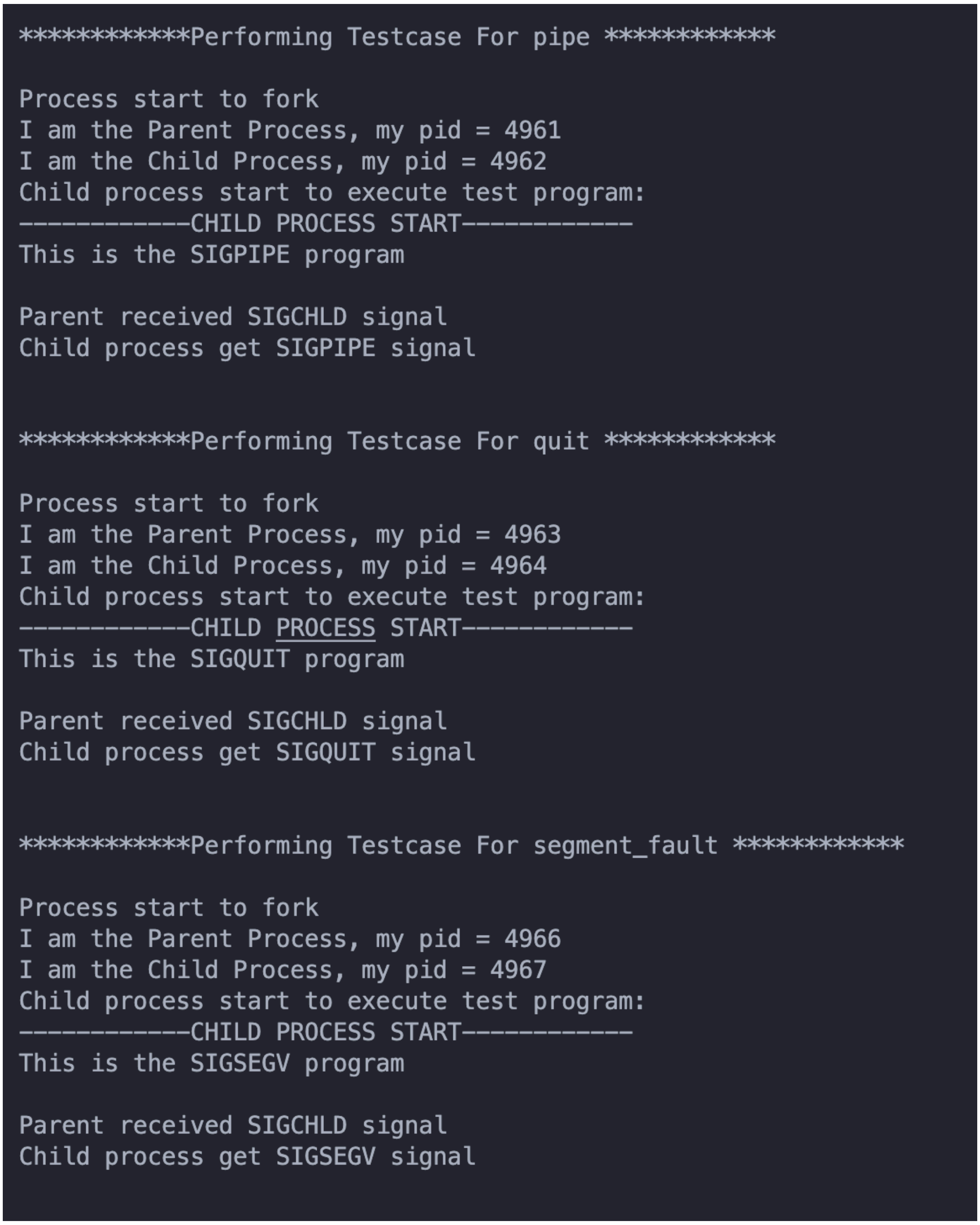
```
************Performing Testcase For floating ************

Process start to fork
I am the Parent Process, my pid = 4947
I am the Child Process, my pid = 4948
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGFPE program

Parent received SIGCHLD signal
Child process get SIGFPE signal


************Performing Testcase For hangup ************

Process start to fork
I am the Child Process, my pid = 4951
Child process start to execute test program:
I am the Parent Process, my pid = 4950
------------CHILD PROCESS START------------
This is the SIGHUP program

Parent received SIGCHLD signal
Child process get SIGHUP signal


************Performing Testcase For illegal_instr ************

Process start to fork
I am the Parent Process, my pid = 4952
I am the Child Process, my pid = 4953
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGILL program

Parent received SIGCHLD signal
Child process get SIGILL signal
```



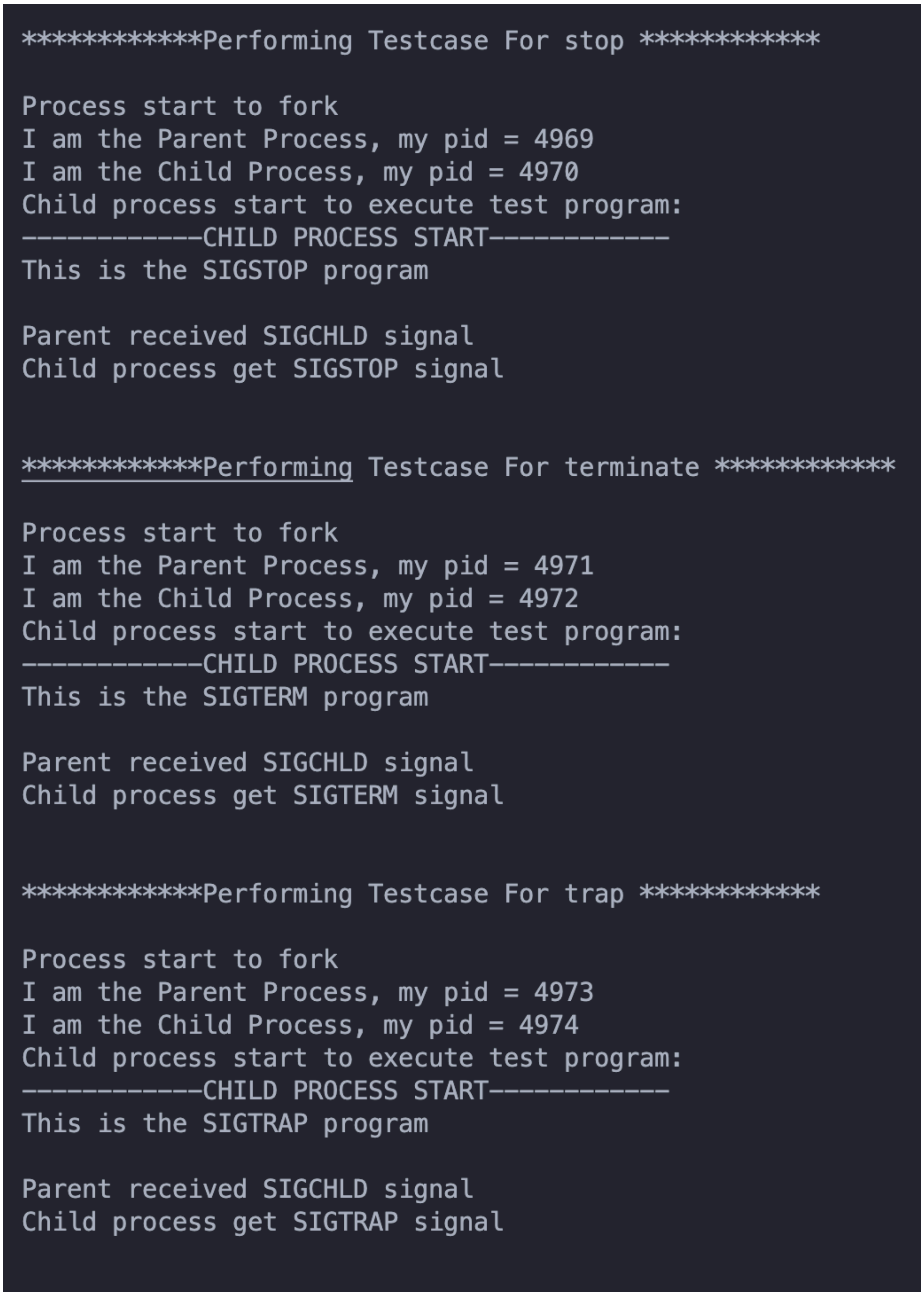
```
************Performing Testcase For floating ************

Process start to fork
I am the Parent Process, my pid = 4947
I am the Child Process, my pid = 4948
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGFPE program

Parent received SIGCHLD signal
Child process get SIGFPE signal


************Performing Testcase For hangup ************

Process start to fork
I am the Child Process, my pid = 4951
Child process start to execute test program:
I am the Parent Process, my pid = 4950
------------CHILD PROCESS START------------
This is the SIGHUP program

Parent received SIGCHLD signal
Child process get SIGHUP signal


************Performing Testcase For illegal_instr ************

Process start to fork
I am the Parent Process, my pid = 4952
I am the Child Process, my pid = 4953
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGILL program

Parent received SIGCHLD signal
Child process get SIGILL signal
```



```
************Performing Testcase For interrupt ************

Process start to fork
I am the Parent Process, my pid = 4955
I am the Child Process, my pid = 4956
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGINT program

Parent received SIGCHLD signal
Child process get SIGINT signal


************Performing Testcase For kill ************

Process start to fork
I am the Parent Process, my pid = 4957
I am the Child Process, my pid = 4958
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGKILL program

Parent received SIGCHLD signal
Child process get SIGKILL signal


************Performing Testcase For normal ************

Process start to fork
I am the Parent Process, my pid = 4959
I am the Child Process, my pid = 4960
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the normal program

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

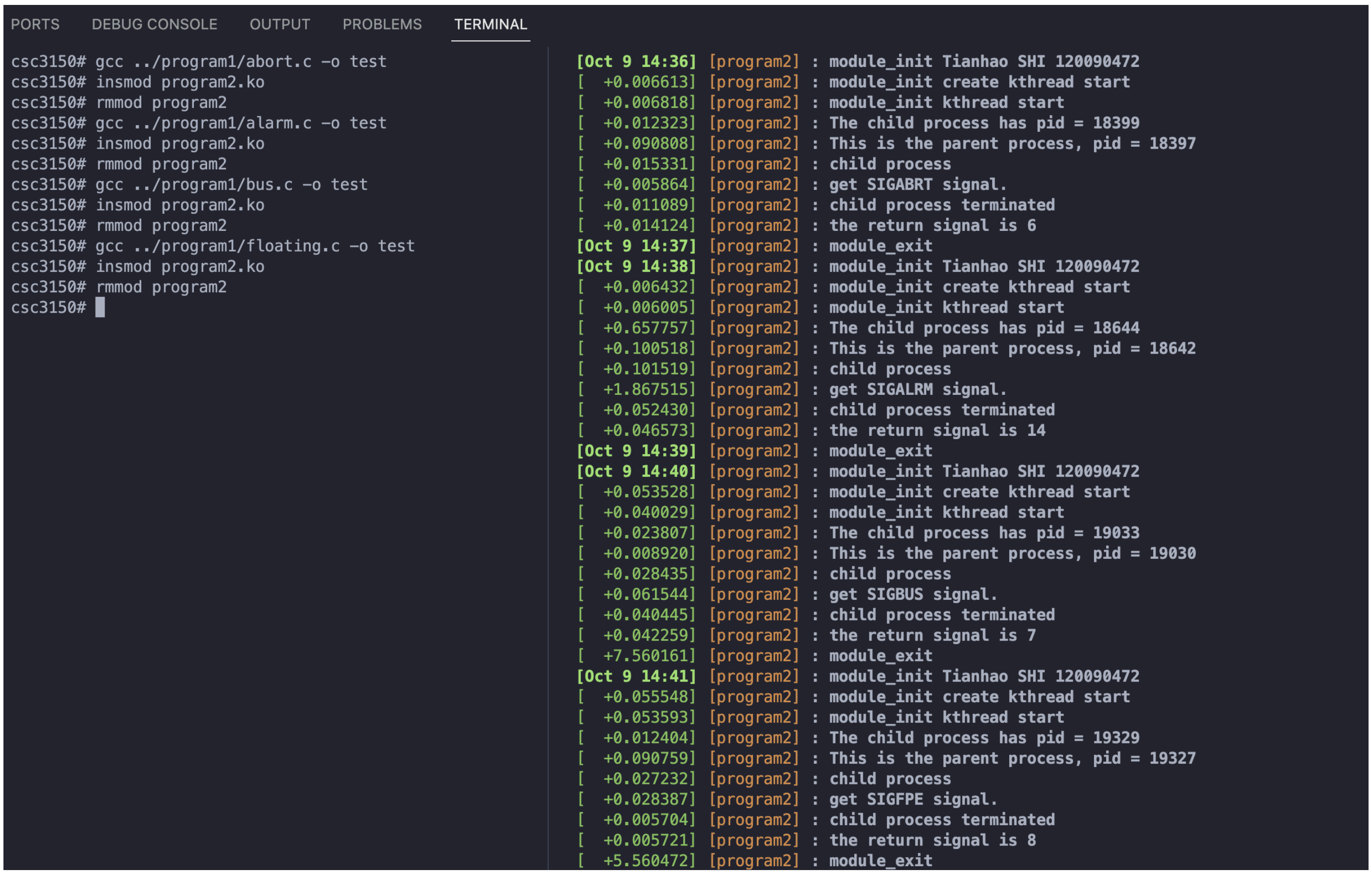


```
************Performing Testcase For pipe ************

Process start to fork
I am the Parent Process, my pid = 4961
I am the Child Process, my pid = 4962
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGPIPE program

Parent received SIGCHLD signal
Child process get SIGPIPE signal


************Performing Testcase For quit ************

Process start to fork
I am the Parent Process, my pid = 4963
I am the Child Process, my pid = 4964
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGQUIT program

Parent received SIGCHLD signal
Child process get SIGQUIT signal


************Performing Testcase For segment_fault ************

Process start to fork
I am the Parent Process, my pid = 4966
I am the Child Process, my pid = 4967
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGSEGV program

Parent received SIGCHLD signal
Child process get SIGSEGV signal
```



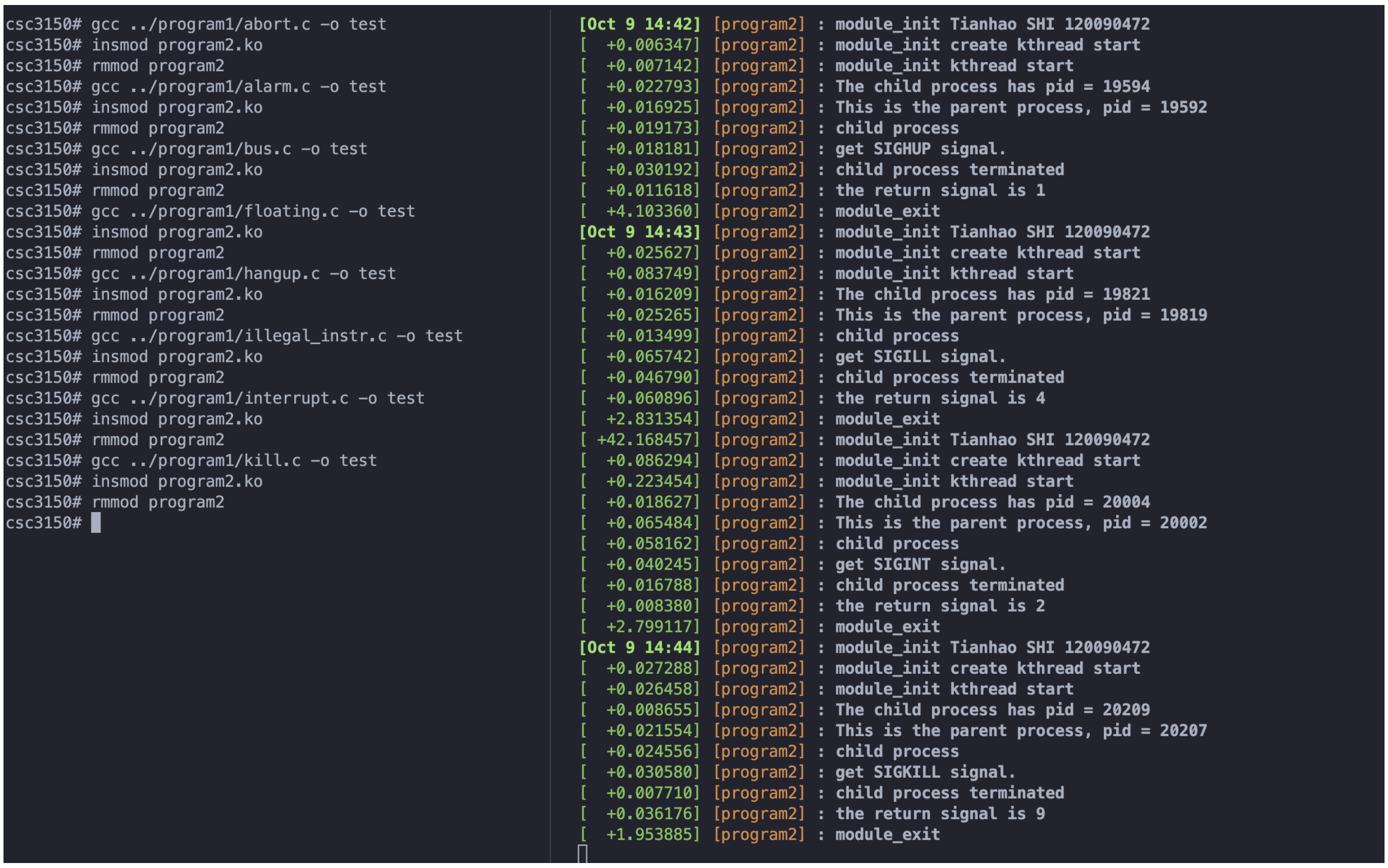
```
************Performing Testcase For stop ************

Process start to fork
I am the Parent Process, my pid = 4969
I am the Child Process, my pid = 4970
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGSTOP program

Parent received SIGCHLD signal
Child process get SIGSTOP signal


************Performing Testcase For terminate ************

Process start to fork
I am the Parent Process, my pid = 4971
I am the Child Process, my pid = 4972
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGTERM program

Parent received SIGCHLD signal
Child process get SIGTERM signal


************Performing Testcase For trap ************

Process start to fork
I am the Parent Process, my pid = 4973
I am the Child Process, my pid = 4974
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGTRAP program

Parent received SIGCHLD signal
Child process get SIGTRAP signal
```

Program output for Task 2



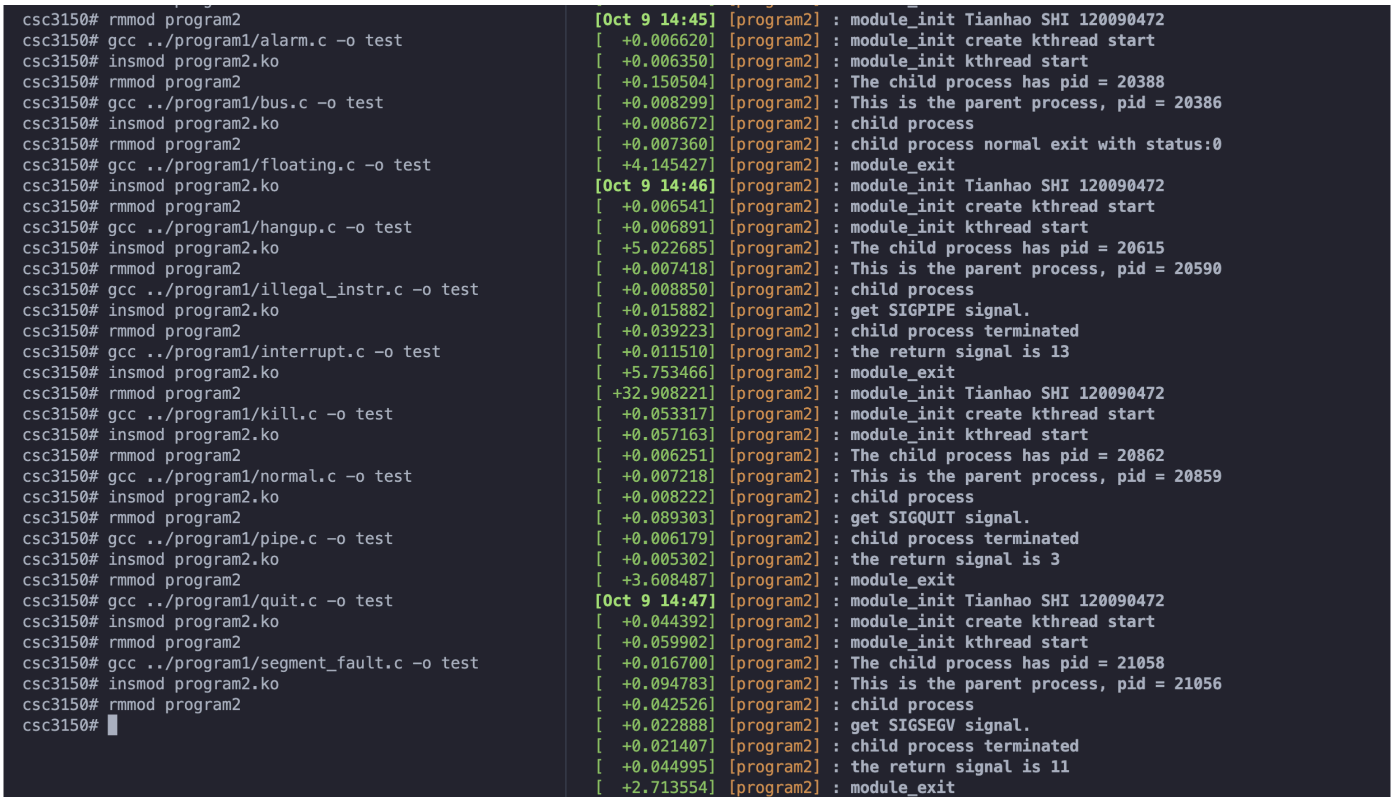
```
csc3150# gcc ../program1/abort.c -o test
csc3150# insmod program2.ko
csc3150# rmmod program2
csc3150# gcc ../program1/alarm.c -o test
csc3150# insmod program2.ko
csc3150# rmmod program2
csc3150# gcc ../program1/bus.c -o test
csc3150# insmod program2.ko
csc3150# rmmod program2
csc3150# gcc ../program1/floating.c -o test
csc3150# insmod program2.ko
csc3150# rmmod program2
csc3150#
```

```
[Oct 9 14:36] [program2] : module_init Tianhao SHI 120090472
[  +0.006613] [program2] : module_init create kthread start
[  +0.006818] [program2] : module_init kthread start
[  +0.012323] [program2] : The child process has pid = 18399
[  +0.090808] [program2] : This is the parent process, pid = 18397
[  +0.015331] [program2] : child process
[  +0.005864] [program2] : get SIGABRT signal.
[  +0.011089] [program2] : child process terminated
[  +0.014124] [program2] : the return signal is 6
[Oct 9 14:37] [program2] : module_exit
[Oct 9 14:38] [program2] : module_init Tianhao SHI 120090472
[  +0.006432] [program2] : module_init create kthread start
[  +0.006005] [program2] : module_init kthread start
[  +0.657757] [program2] : The child process has pid = 18644
[  +0.100518] [program2] : This is the parent process, pid = 18642
[  +0.101519] [program2] : child process
[  +1.867515] [program2] : get SIGALRM signal.
[  +0.052430] [program2] : child process terminated
[  +0.046573] [program2] : the return signal is 14
[Oct 9 14:39] [program2] : module_exit
[Oct 9 14:40] [program2] : module_init Tianhao SHI 120090472
[  +0.053528] [program2] : module_init create kthread start
[  +0.040029] [program2] : module_init kthread start
[  +0.023807] [program2] : The child process has pid = 19033
[  +0.008920] [program2] : This is the parent process, pid = 19030
[  +0.028435] [program2] : child process
[  +0.061544] [program2] : get SIGBUS signal.
[  +0.040445] [program2] : child process terminated
[  +0.042259] [program2] : the return signal is 7
[  +7.560161] [program2] : module_exit
[Oct 9 14:41] [program2] : module_init Tianhao SHI 120090472
[  +0.055548] [program2] : module_init create kthread start
[  +0.053593] [program2] : module_init kthread start
[  +0.012404] [program2] : The child process has pid = 19329
[  +0.090759] [program2] : This is the parent process, pid = 19327
[  +0.027232] [program2] : child process
[  +0.028387] [program2] : get SIGFPE signal.
[  +0.005704] [program2] : child process terminated
[  +0.005721] [program2] : the return signal is 8
[  +5.560472] [program2] : module_exit
```



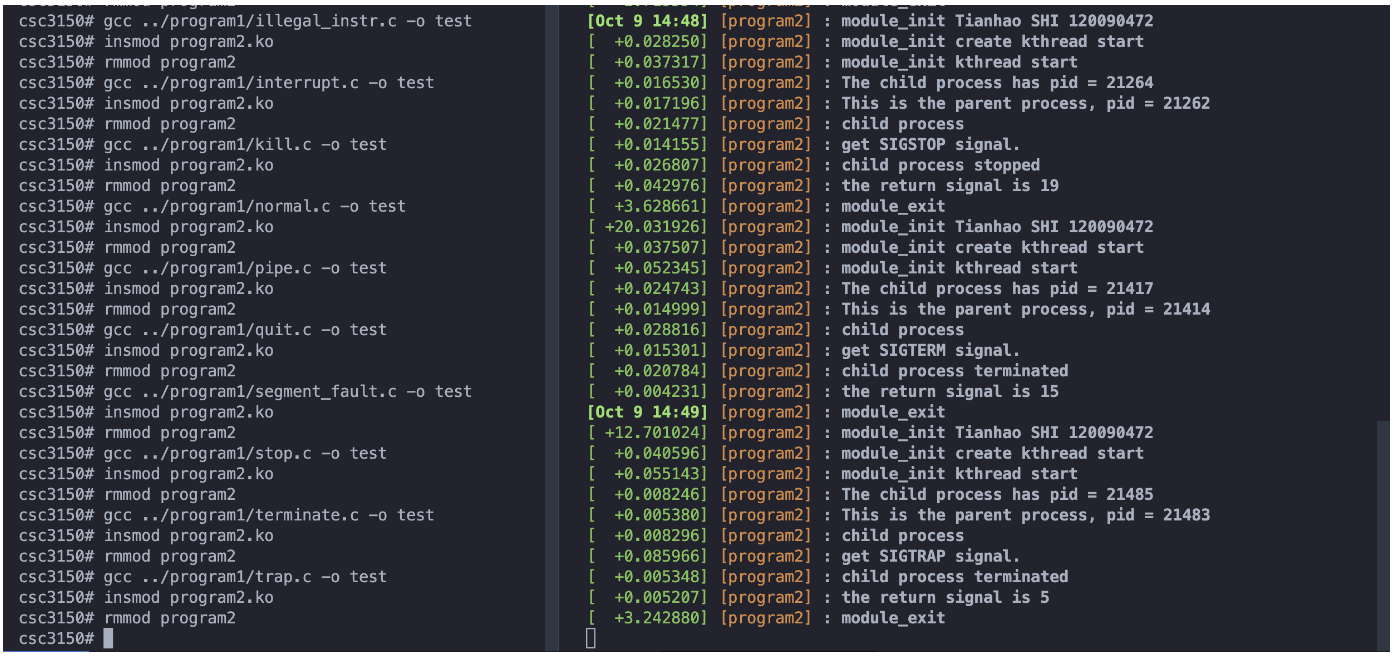
```
csc3150# gcc ../program1/abort.c -o test
csc3150# insmod program2.ko
csc3150# rmmod program2
csc3150# gcc ../program1/alarm.c -o test
csc3150# insmod program2.ko
csc3150# rmmod program2
csc3150# gcc ../program1/bus.c -o test
csc3150# insmod program2.ko
csc3150# rmmod program2
csc3150# gcc ../program1/floating.c -o test
csc3150# insmod program2.ko
csc3150# rmmod program2
csc3150# gcc ../program1/hangup.c -o test
csc3150# insmod program2.ko
csc3150# rmmod program2
csc3150# gcc ../program1/illegal_instr.c -o test
csc3150# insmod program2.ko
csc3150# rmmod program2
csc3150# gcc ../program1/interrupt.c -o test
csc3150# insmod program2.ko
csc3150# rmmod program2
csc3150# gcc ../program1/kill.c -o test
csc3150# insmod program2.ko
csc3150# rmmod program2
csc3150#
```

```
[Oct 9 14:42] [program2] : module_init Tianhao SHI 120090472
[  +0.006347] [program2] : module_init create kthread start
[  +0.007142] [program2] : module_init kthread start
[  +0.022793] [program2] : The child process has pid = 19594
[  +0.016925] [program2] : This is the parent process, pid = 19592
[  +0.019173] [program2] : child process
[  +0.018181] [program2] : get SIGHUP signal.
[  +0.030192] [program2] : child process terminated
[  +0.011618] [program2] : the return signal is 1
[  +4.103360] [program2] : module_exit
[Oct 9 14:43] [program2] : module_init Tianhao SHI 120090472
[  +0.025627] [program2] : module_init create kthread start
[  +0.083749] [program2] : module_init kthread start
[  +0.016209] [program2] : The child process has pid = 19821
[  +0.025265] [program2] : This is the parent process, pid = 19819
[  +0.013499] [program2] : child process
[  +0.065742] [program2] : get SIGILL signal.
[  +0.046790] [program2] : child process terminated
[  +0.060896] [program2] : the return signal is 4
[  +2.831354] [program2] : module_exit
[ +42.168457] [program2] : module_init Tianhao SHI 120090472
[  +0.086294] [program2] : module_init create kthread start
[  +0.223454] [program2] : module_init kthread start
[  +0.018627] [program2] : The child process has pid = 20004
[  +0.065484] [program2] : This is the parent process, pid = 20002
[  +0.058162] [program2] : child process
[  +0.040245] [program2] : get SIGINT signal.
[  +0.016788] [program2] : child process terminated
[  +0.008380] [program2] : the return signal is 2
[  +2.799117] [program2] : module_exit
[Oct 9 14:44] [program2] : module_init Tianhao SHI 120090472
[  +0.027288] [program2] : module_init create kthread start
[  +0.026458] [program2] : module_init kthread start
[  +0.008655] [program2] : The child process has pid = 20209
[  +0.021554] [program2] : This is the parent process, pid = 20207
[  +0.024556] [program2] : child process
[  +0.030580] [program2] : get SIGKILL signal.
[  +0.007710] [program2] : child process terminated
[  +0.036176] [program2] : the return signal is 9
[  +1.953885] [program2] : module_exit
```



```
csc3150# rmmod program2
csc3150# gcc ../program1/alarm.c -o test
csc3150# insmod program2.ko
csc3150# rmmod program2
csc3150# gcc ../program1/bus.c -o test
csc3150# insmod program2.ko
csc3150# rmmod program2
csc3150# gcc ../program1/floating.c -o test
csc3150# insmod program2.ko
csc3150# rmmod program2
csc3150# gcc ../program1/hangup.c -o test
csc3150# insmod program2.ko
csc3150# rmmod program2
csc3150# gcc ../program1/illegal_instr.c -o test
csc3150# insmod program2.ko
csc3150# rmmod program2
csc3150# gcc ../program1/interrupt.c -o test
csc3150# insmod program2.ko
csc3150# rmmod program2
csc3150# gcc ../program1/kill.c -o test
csc3150# insmod program2.ko
csc3150# rmmod program2
csc3150# gcc ../program1/normal.c -o test
csc3150# insmod program2.ko
csc3150# rmmod program2
csc3150# gcc ../program1/pipe.c -o test
csc3150# insmod program2.ko
csc3150# rmmod program2
csc3150# gcc ../program1/quit.c -o test
csc3150# insmod program2.ko
csc3150# rmmod program2
csc3150# gcc ../program1/segment_fault.c -o test
csc3150# insmod program2.ko
csc3150# rmmod program2
csc3150# 
```

```
[Oct 9 14:45] [program2] : module_init Tianhao SHI 120090472
[  +0.006620] [program2] : module_init create kthread start
[  +0.006350] [program2] : module_init kthread start
[  +0.150504] [program2] : The child process has pid = 20388
[  +0.008299] [program2] : This is the parent process, pid = 20386
[  +0.008672] [program2] : child process
[  +0.007360] [program2] : child process normal exit with status:0
[  +4.145427] [program2] : module_exit
[Oct 9 14:46] [program2] : module_init Tianhao SHI 120090472
[  +0.006541] [program2] : module_init create kthread start
[  +0.006891] [program2] : module_init kthread start
[  +5.022685] [program2] : The child process has pid = 20615
[  +0.007418] [program2] : This is the parent process, pid = 20590
[  +0.008850] [program2] : child process
[  +0.015882] [program2] : get SIGPIPE signal.
[  +0.039223] [program2] : child process terminated
[  +0.011510] [program2] : the return signal is 13
[  +5.753466] [program2] : module_exit
[ +32.908221] [program2] : module_init Tianhao SHI 120090472
[  +0.053317] [program2] : module_init create kthread start
[  +0.057163] [program2] : module_init kthread start
[  +0.006251] [program2] : The child process has pid = 20862
[  +0.007218] [program2] : This is the parent process, pid = 20859
[  +0.008222] [program2] : child process
[  +0.089303] [program2] : get SIGQUIT signal.
[  +0.006179] [program2] : child process terminated
[  +0.005302] [program2] : the return signal is 3
[  +3.608487] [program2] : module_exit
[Oct 9 14:47] [program2] : module_init Tianhao SHI 120090472
[  +0.044392] [program2] : module_init create kthread start
[  +0.059902] [program2] : module_init kthread start
[  +0.016700] [program2] : The child process has pid = 21058
[  +0.094783] [program2] : This is the parent process, pid = 21056
[  +0.042526] [program2] : child process
[  +0.022888] [program2] : get SIGSEGV signal.
[  +0.021407] [program2] : child process terminated
[  +0.044995] [program2] : the return signal is 11
[  +2.713554] [program2] : module_exit
```



```
csc3150# gcc ../program1/illegal_instr.c -o test
csc3150# insmod program2.ko
csc3150# rmmod program2
csc3150# gcc ../program1/interrupt.c -o test
csc3150# insmod program2.ko
csc3150# rmmod program2
csc3150# gcc ../program1/kill.c -o test
csc3150# insmod program2.ko
csc3150# rmmod program2
csc3150# gcc ../program1/normal.c -o test
csc3150# insmod program2.ko
csc3150# rmmod program2
csc3150# gcc ../program1/pipe.c -o test
csc3150# insmod program2.ko
csc3150# rmmod program2
csc3150# gcc ../program1/quit.c -o test
csc3150# insmod program2.ko
csc3150# rmmod program2
csc3150# gcc ../program1/segment_fault.c -o test
csc3150# insmod program2.ko
csc3150# rmmod program2
csc3150# gcc ../program1/stop.c -o test
csc3150# insmod program2.ko
csc3150# rmmod program2
csc3150# gcc ../program1/terminate.c -o test
csc3150# insmod program2.ko
csc3150# rmmod program2
csc3150# gcc ../program1/trap.c -o test
csc3150# insmod program2.ko
csc3150# rmmod program2
csc3150# 
```

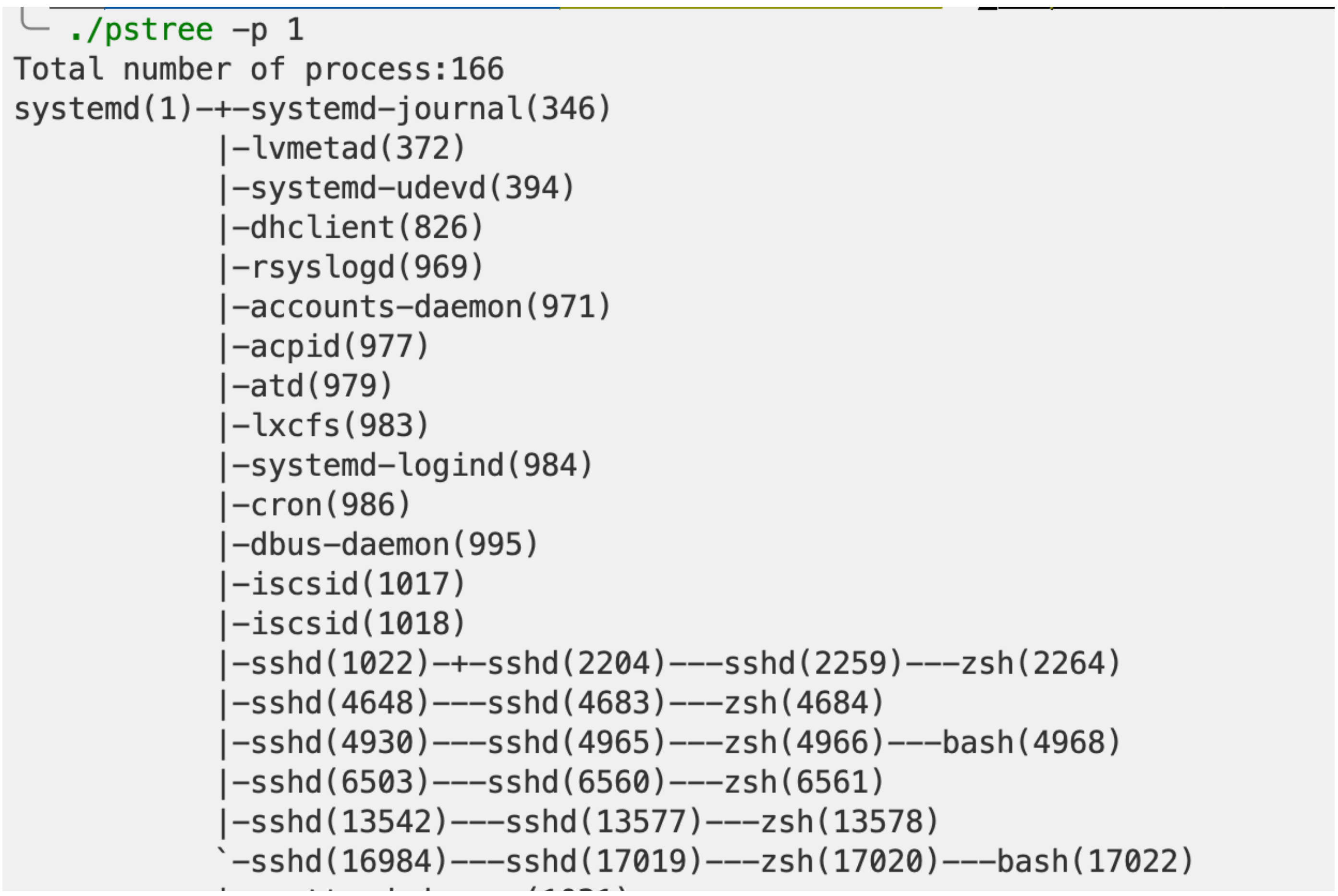
```
[Oct 9 14:48] [program2] : module_init Tianhao SHI 120090472
[  +0.028250] [program2] : module_init create kthread start
[  +0.037317] [program2] : module_init kthread start
[  +0.016530] [program2] : The child process has pid = 21264
[  +0.017196] [program2] : This is the parent process, pid = 21262
[  +0.021477] [program2] : child process
[  +0.014155] [program2] : get SIGSTOP signal.
[  +0.026807] [program2] : child process stopped
[  +0.042976] [program2] : the return signal is 19
[  +3.628661] [program2] : module_exit
[ +20.031926] [program2] : module_init Tianhao SHI 120090472
[  +0.037507] [program2] : module_init create kthread start
[  +0.052345] [program2] : module_init kthread start
[  +0.024743] [program2] : The child process has pid = 21417
[  +0.014999] [program2] : This is the parent process, pid = 21414
[  +0.028816] [program2] : child process
[  +0.015301] [program2] : get SIGTERM signal.
[  +0.020784] [program2] : child process terminated
[  +0.004231] [program2] : the return signal is 15
[Oct 9 14:49] [program2] : module_exit
[ +12.701024] [program2] : module_init Tianhao SHI 120090472
[  +0.040596] [program2] : module_init create kthread start
[  +0.055143] [program2] : module_init kthread start
[  +0.008246] [program2] : The child process has pid = 21485
[  +0.005380] [program2] : This is the parent process, pid = 21483
[  +0.008296] [program2] : child process
[  +0.085966] [program2] : get SIGTRAP signal.
[  +0.005348] [program2] : child process terminated
[  +0.005207] [program2] : the return signal is 5
[  +3.242880] [program2] : module_exit
```

Modified Makefile for batch running Task 1

```
CFILES:= $(shell ls|grep .c)
PROGS:=$(patsubst %.c,%,$(CFILES))
all: $(PROGS)

%:%.c
```

```
	$(CC) -o $@ $<

clean:$(PROGS)
	rm $(PROGS)

run: $(PROGS)
	@for testcase in $(PROGS) ; do \
	      if [ $$testcase != "program1" ]; then \
	    echo "************Performing Testcase For $$testcase
************\n"; \
	    ./program1 $$testcase; \
	    echo "\n"; \
	  fi \
	done
```

Bonus program output



```
└ ./pstree -p 1
Total number of process:166
systemd(1)-+-systemd-journal(346)
           |-lvmetad(372)
           |-systemd-udevd(394)
           |-dhclient(826)
           |-rsyslogd(969)
           |-accounts-daemon(971)
           |-acpid(977)
           |-atd(979)
           |-lxcfs(983)
           |-systemd-logind(984)
           |-cron(986)
           |-dbus-daemon(995)
           |-iscsid(1017)
           |-iscsid(1018)
           |-sshd(1022)-+-sshd(2204)---sshd(2259)---zsh(2264)
           |-sshd(4648)---sshd(4683)---zsh(4684)
           |-sshd(4930)---sshd(4965)---zsh(4966)---bash(4968)
           |-sshd(6503)---sshd(6560)---zsh(6561)
           |-sshd(13542)---sshd(13577)---zsh(13578)
           `-sshd(16984)---sshd(17019)---zsh(17020)---bash(17022)
```