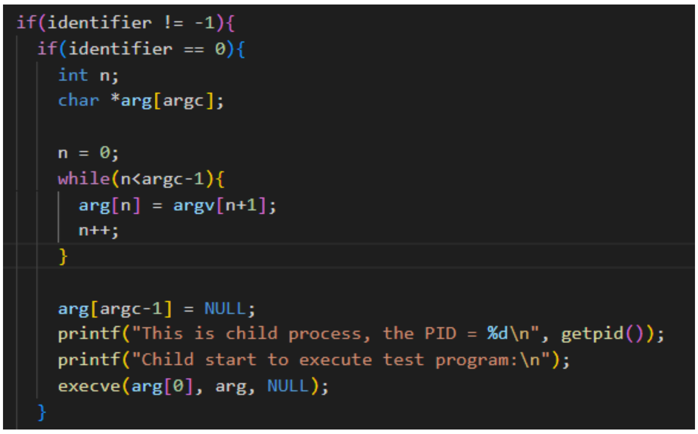
Assignment 1 Project Report

Background

The purpose of this programming exercise is to familiarize us with calling and running new processes and threads. In order to complete the assignment, we must fork a child process and run test programs inside of it. The assignment's goal is to expose us with operating system and kernel interaction and editing, as well as moving between user mode and kernel mode. The Ubuntu Version I use is version **16.04.7 LTS** and Kernel Version used is **Linux 5.10.147**.

Problem 1

In order to complete Task 1, we must use the fork() function in user mode to establish a child process that will run a separate program while the parent process waits for it to finish. This task's design and execution are both rather simple. We distinguish the parent and child processes once fork() is invoked using their process identifiers. The return value from fork() can be used for this. Following that, it just involves invoking execve() on the specified executable to start the child process.



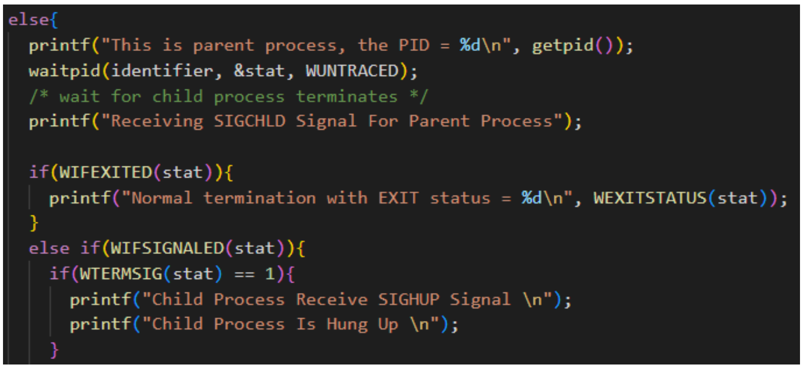
```
if(identifier != -1){
  if(identifier == 0){
    int n;
    char *arg[argc];

    n = 0;
    while(n<argc-1){
      arg[n] = argv[n+1];
      n++;
    }

    arg[argc-1] = NULL;
    printf("This is child process, the PID = %d\n", getpid());
    printf("Child start to execute test program:\n");
    execve(arg[0], arg, NULL);
  }
```

Figure 1.1 Example for getpid() and execve()

We instruct the parent process to wait until the child process has finished executing before proceeding while the child process is active. The wait() and waitpid() external function routines can be used for this. The child process will send a SIGCHLD signal to indicate its termination. The parent process will then proceed to output processing after receiving this and print out various outputs based on the exit signal received from the child process. This is assisted by the family of functions WIFEXITED(stat), WTERMSIG(stat), and WIFSTOPPED (stat).



```
else{
  printf("This is parent process, the PID = %d\n", getpid());
  waitpid(identifier, &stat, WUNTRACED);
  /* wait for child process terminates */
  printf("Receiving SIGCHLD Signal For Parent Process");

  if(WIFEXITED(stat)){
    printf("Normal termination with EXIT status = %d\n", WEXITSTATUS(stat));
  }
  else if(WIFSIGNALED(stat)){
    if(WTERMSIG(stat) == 1){
      printf("Child Process Receive SIGHUP Signal \n");
      printf("Child Process Is Hung Up \n");
    }
```

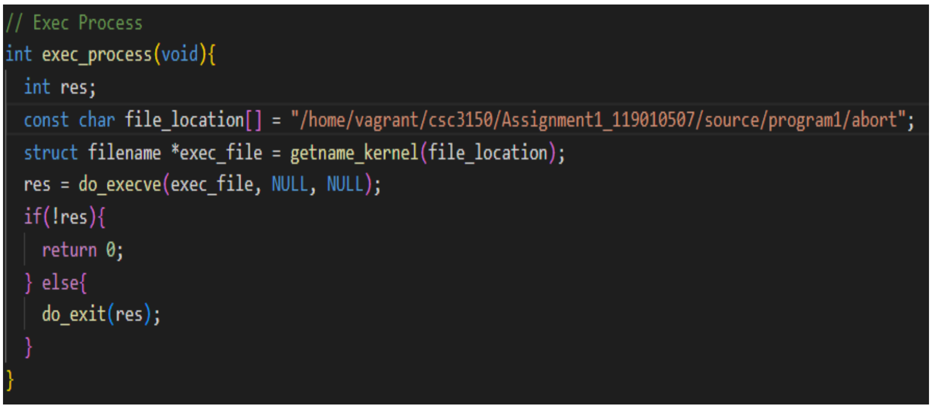
Figure 1.2 Parent Process with WIFEXITED and WTERMSIG

Problem 2

The main difference between job 2 and task 1 is that task 2 is carried out in kernel mode as opposed to user mode. This makes the work more complex since we must examine kernel function calls and modify them to export some of their functions. This entails updating the kernel version and compiling and recompiling the kernel in order to correctly handle some essential function calls.

Problem 2 Process

Create a thread to start building task. To create a thread we use a function called kthread_create(). We then use a straightforward approach by attempting to fork a child process, similar to job 1. It turned out to be more difficult to complete in kernel mode because we had to call kernel_clone(), which is located in /kernel/fork.c, to complete the task, as opposed to the simple fork() function in user mode. The settings that will be used to create kernel clone are passed as parameters to the kernel clone() function in the form of a struct object called kernel clone args. Things like its flags, stack size, exit signal, etc. are included in this. In a manner similar to the user mode fork() function, it will output the pid of the child upon success. After calling kerne_clone function we will run exec_process() to tell the kernel_clone() function about the stack information that the should run. Throughout the exec_process() we will utilize the external function that we imported which is do_execve(). This function do_execve() is the kernel version of execve(). This function is located in the /fs/exec.c. There will also be a getname_kernel() function which is located in /fs/namei.c which is called to get the path of executable before it given ti do_execve() and if it succed it will return a) value. We must additionally instruct the parent process to hold off until the child process has finished running. This is accomplished through the use of the do wait() method from kernel/exit.c in the custom wait process() function. A struct object called wait opts that is comparable to kernel clone() is passed into the do wait() function. The outcomes of the child process's exit signal can then be obtained by accessing the wait opts structure. The status member inside the wait opts structure is where this is primarily accessed. Similar to task 1, we process this member to obtain the appropriate exit signals we seek, which subsequently direct our programs' output.



```c
// Exec Process
int exec_process(void){
  int res;
  const char file_location[] = "/home/vagrant/csc3150/Assignment1_119010507/source/program1/abort";
  struct filename *exec_file = getname_kernel(file_location);
  res = do_execve(exec_file, NULL, NULL);
  if(!res){
    return 0;
  } else{
    do_exit(res);
  }
}
```

Figure 2.1 Code for Exec_process function using do_execve()

Environment Set up

The kernel needs to be updated because it was no longer appropriate for the job. This entails installing a 5.10.x Linux kernel version that is downloaded to our machine as a tar or zip file. After extracting the tar or zip file, we use a variety of make calls to compile the kernel including make mrproper, clean, etc. To be able to compile the kernel, we must first install the appropriate development tools. In order to install our upgraded kernel, we then moved and copied the .config() file from /boot to our new kernel directory. Then, by using the make clean command, we must clean up our old kernel installation. The prior built and compiled modules from our old kernel version are erased as a result. After cleaning the old kernel, we then configure the menu with make menuconfig and a GUI will pop up.

After everything is finished, we proceed with make bzImage to build kernel images and install all the modules. After the installation process is done the kernel will reboot automatically. In besides installing the kernel, we must export various functions from the kernel function calls in order to use them correctly in problem 2. In the paragraph before, each function call included a specific file location specification. They must be exported using EXPORT SYMBOL (). We rebuild the kernel after exporting these methods, reboot once more, and then move on to problem 2. Furthermore, in order to use these functions, we must first import it with the extern tag in problem 2.

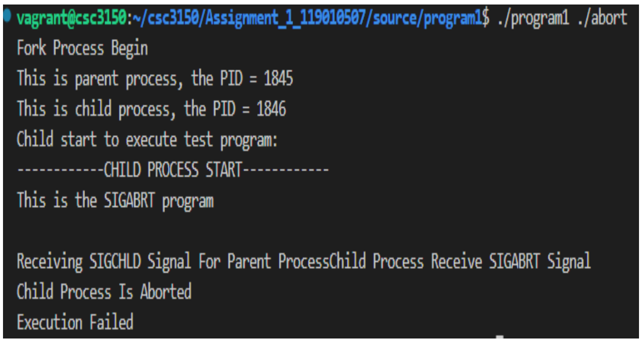
Project Summary and Learning

The assignment provided us with knowledge about the internal dynamics of several of the user mode function calls that we frequently use when programming. Additionally, this project demonstrated to us how the operating system controls several processes. The biggest difficulty is creating the ideal environment for our software before we even begin to create it. I picked up how to assemble a kernel and add and remove modules from a kernel during this procedure. After completing the two jobs, I have a clearer idea of the process. I also learn how the child process runs the program and how the parent process receives the message from its child process when I experiment with various test programs in task 1. Additionally, task 2 involves changing the kernel, and in order to do this effectively, we must read a few references.

OUTPUT SAMPLE

Problem 1 Result Output

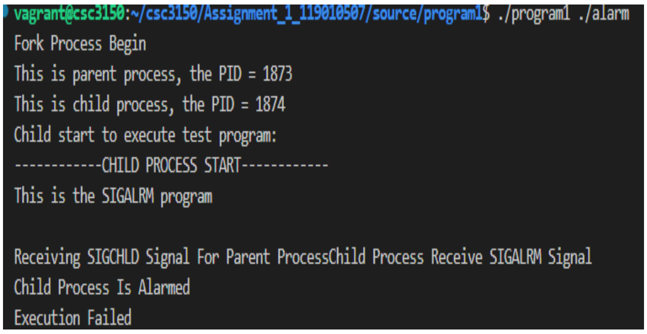
Abort



```
vagrant@csc3150:~/csc3150/Assignment_1_119010507/source/program1$ ./program1 ./abort
Fork Process Begin
This is parent process, the PID = 1845
This is child process, the PID = 1846
Child start to execute test program:
-----------CHILD PROCESS START-----------
This is the SIGABRT program

Receiving SIGCHLD Signal For Parent ProcessChild Process Receive SIGABRT Signal
Child Process Is Aborted
Execution Failed
```

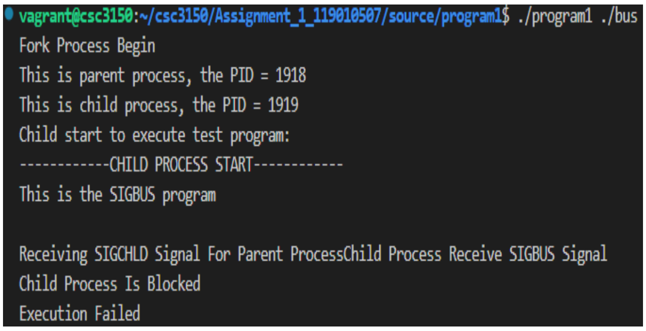
Alarm



```
vagrant@csc3150:~/csc3150/Assignment_1_119010507/source/program1$ ./program1 ./alarm
Fork Process Begin
This is parent process, the PID = 1873
This is child process, the PID = 1874
Child start to execute test program:
-----------CHILD PROCESS START-----------
This is the SIGALRM program

Receiving SIGCHLD Signal For Parent ProcessChild Process Receive SIGALRM Signal
Child Process Is Alarmed
Execution Failed
```

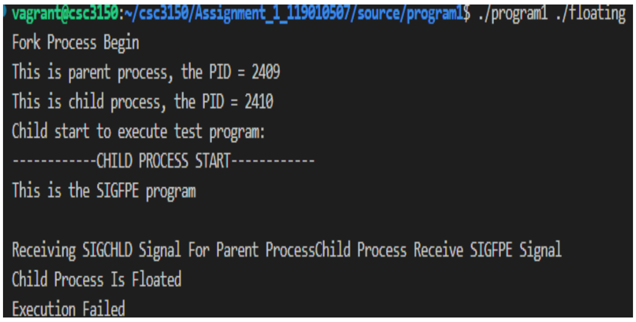
Bus



```
vagrant@csc3150:~/csc3150/Assignment_1_119010507/source/program1$ ./program1 ./bus
Fork Process Begin
This is parent process, the PID = 1918
This is child process, the PID = 1919
Child start to execute test program:
-----------CHILD PROCESS START-----------
This is the SIGBUS program

Receiving SIGCHLD Signal For Parent ProcessChild Process Receive SIGBUS Signal
Child Process Is Blocked
Execution Failed
```

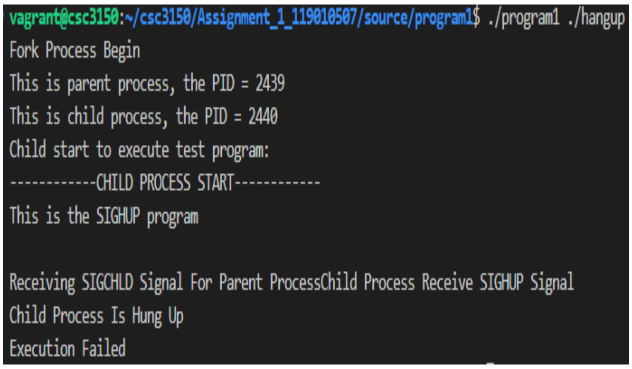
Floating



```
vagrant@csc3150:~/csc3150/Assignment_1_119010507/source/program1$ ./program1 ./floating
Fork Process Begin
This is parent process, the PID = 2409
This is child process, the PID = 2410
Child start to execute test program:
-----------CHILD PROCESS START-----------
This is the SIGFPE program

Receiving SIGCHLD Signal For Parent ProcessChild Process Receive SIGFPE Signal
Child Process Is Floated
Execution Failed
```

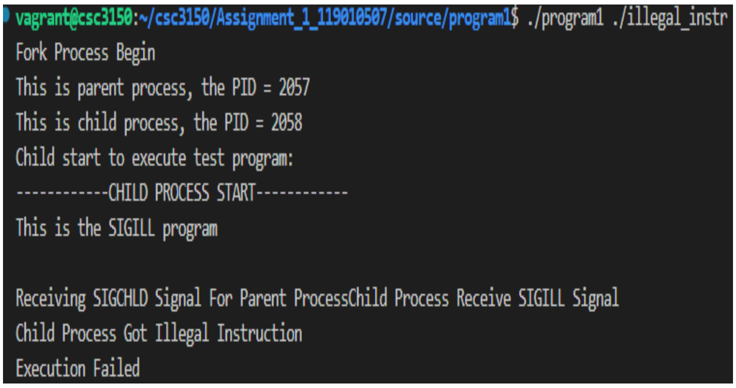
Hang up



```
vagrant@csc3150:~/csc3150/Assignment_1_119010507/source/program1$ ./program1 ./hangup
Fork Process Begin
This is parent process, the PID = 2439
This is child process, the PID = 2440
Child start to execute test program:
-----------CHILD PROCESS START-----------
This is the SIGHUP program

Receiving SIGCHLD Signal For Parent ProcessChild Process Receive SIGHUP Signal
Child Process Is Hung Up
Execution Failed
```

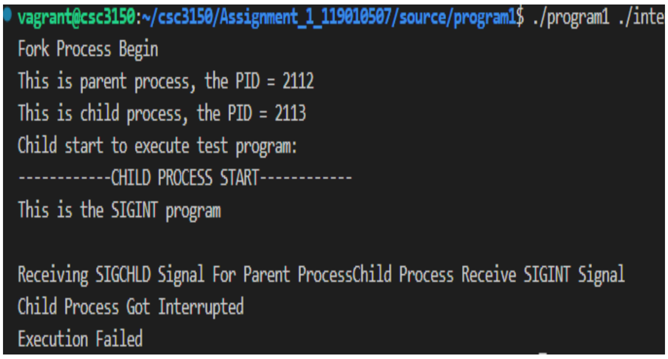
Illegal Instr



```
vagrant@csc3150:~/csc3150/Assignment_1_119010507/source/program1$ ./program1 ./illegal_instr
Fork Process Begin
This is parent process, the PID = 2057
This is child process, the PID = 2058
Child start to execute test program:
-----------CHILD PROCESS START-----------
This is the SIGILL program

Receiving SIGCHLD Signal For Parent ProcessChild Process Receive SIGILL Signal
Child Process Got Illegal Instruction
Execution Failed
```

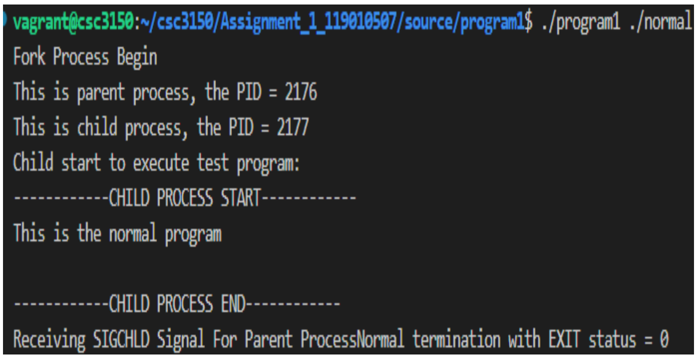
Interrupt



```
vagrant@csc3150:~/csc3150/Assignment_1_119010507/source/program1$ ./program1 ./inte
Fork Process Begin
This is parent process, the PID = 2112
This is child process, the PID = 2113
Child start to execute test program:
-----------CHILD PROCESS START-----------
This is the SIGINT program

Receiving SIGCHLD Signal For Parent ProcessChild Process Receive SIGINT Signal
Child Process Got Interrupted
Execution Failed
```

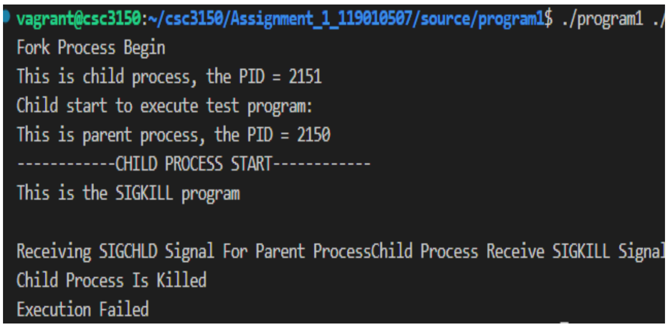
Normal



```
vagrant@csc3150:~/csc3150/Assignment_1_119010507/source/program1$ ./program1 ./normal
Fork Process Begin
This is parent process, the PID = 2176
This is child process, the PID = 2177
Child start to execute test program:
-----------CHILD PROCESS START-----------
This is the normal program

-----------CHILD PROCESS END-----------
Receiving SIGCHLD Signal For Parent ProcessNormal termination with EXIT status = 0
```

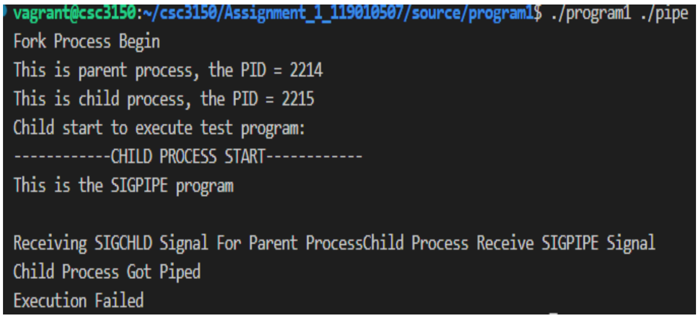
Kill



```
vagrant@csc3150:~/csc3150/Assignment_1_119010507/source/program1$ ./program1 ./
Fork Process Begin
This is child process, the PID = 2151
Child start to execute test program:
This is parent process, the PID = 2150
-----------CHILD PROCESS START-----------
This is the SIGKILL program

Receiving SIGCHLD Signal For Parent ProcessChild Process Receive SIGKILL Signal
Child Process Is Killed
Execution Failed
```

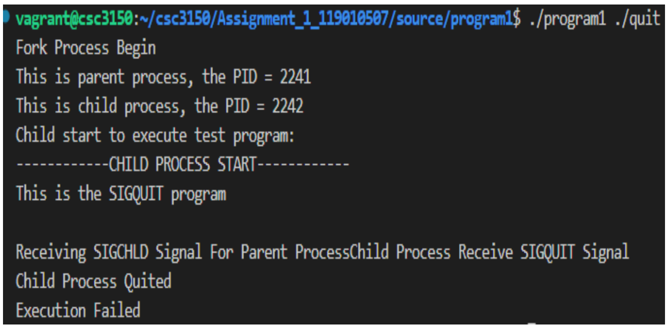
Pipe



```
vagrant@csc3150:~/csc3150/Assignment_1_119010507/source/program1$ ./program1 ./pipe
Fork Process Begin
This is parent process, the PID = 2214
This is child process, the PID = 2215
Child start to execute test program:
-----------CHILD PROCESS START-----------
This is the SIGPIPE program

Receiving SIGCHLD Signal For Parent ProcessChild Process Receive SIGPIPE Signal
Child Process Got Piped
Execution Failed
```

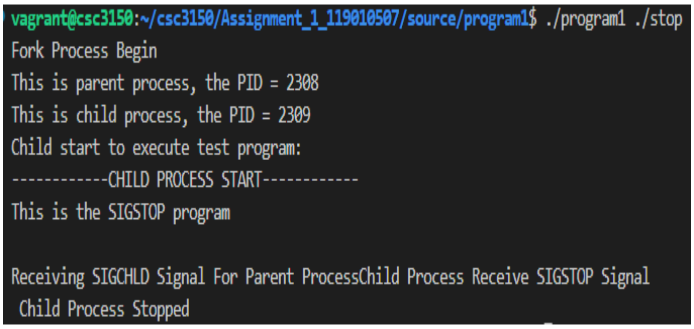
Quit



```
vagrant@csc3150:~/csc3150/Assignment_1_119010507/source/program1$ ./program1 ./quit
Fork Process Begin
This is parent process, the PID = 2241
This is child process, the PID = 2242
Child start to execute test program:
------------CHILD PROCESS START------------
This is the SIGQUIT program

Receiving SIGCHLD Signal For Parent ProcessChild Process Receive SIGQUIT Signal
Child Process Quited
Execution Failed
```

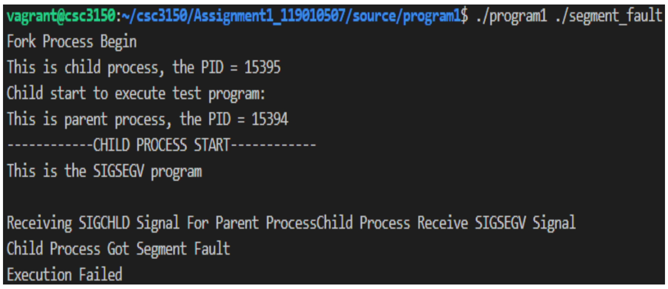
Stop



```
vagrant@csc3150:~/csc3150/Assignment_1_119010507/source/program1$ ./program1 ./stop
Fork Process Begin
This is parent process, the PID = 2308
This is child process, the PID = 2309
Child start to execute test program:
------------CHILD PROCESS START------------
This is the SIGSTOP program

Receiving SIGCHLD Signal For Parent ProcessChild Process Receive SIGSTOP Signal
 Child Process Stopped
```

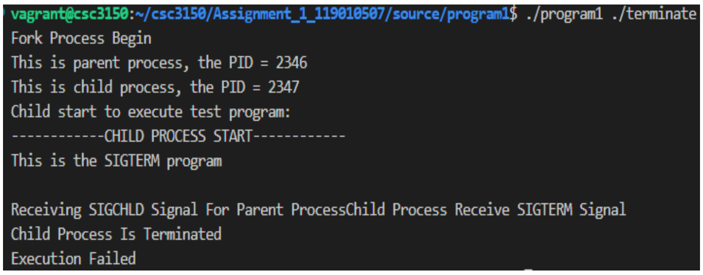
Segment Fault



```
vagrant@csc3150:~/csc3150/Assignment1_119010507/source/program1$ ./program1 ./segment_fault
Fork Process Begin
This is child process, the PID = 15395
Child start to execute test program:
This is parent process, the PID = 15394
------------CHILD PROCESS START------------
This is the SIGSEGV program

Receiving SIGCHLD Signal For Parent ProcessChild Process Receive SIGSEGV Signal
Child Process Got Segment Fault
Execution Failed
```

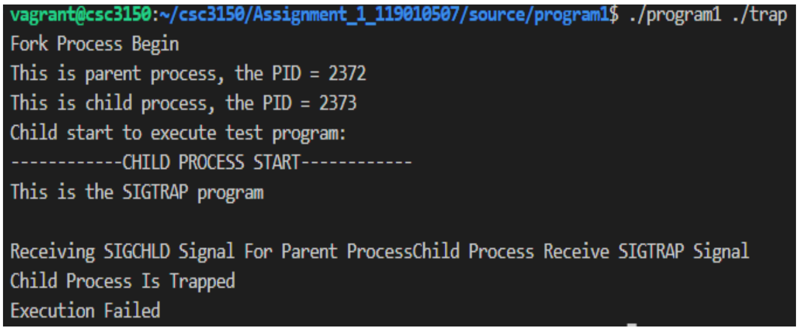
Terminate



```
vagrant@csc3150:~/csc3150/Assignment_1_119010507/source/program1$ ./program1 ./terminate
Fork Process Begin
This is parent process, the PID = 2346
This is child process, the PID = 2347
Child start to execute test program:
------------CHILD PROCESS START------------
This is the SIGTERM program

Receiving SIGCHLD Signal For Parent ProcessChild Process Receive SIGTERM Signal
Child Process Is Terminated
Execution Failed
```

Trap

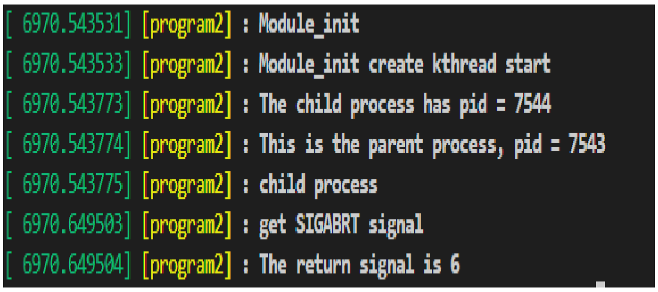


```
vagrant@csc3150:~/csc3150/Assignment_1_119010507/source/program1$ ./program1 ./trap
Fork Process Begin
This is parent process, the PID = 2372
This is child process, the PID = 2373
Child start to execute test program:
------------CHILD PROCESS START------------
This is the SIGTRAP program

Receiving SIGCHLD Signal For Parent ProcessChild Process Receive SIGTRAP Signal
Child Process Is Trapped
Execution Failed
```

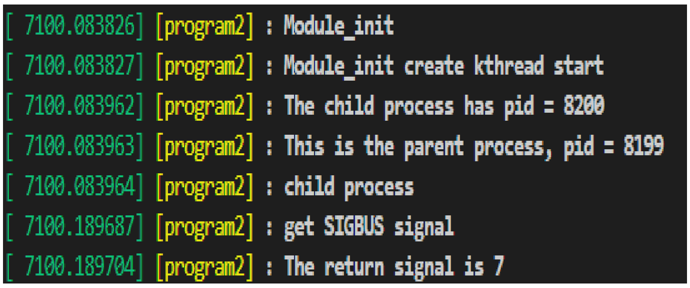
Problem 2 Result Output

Abort



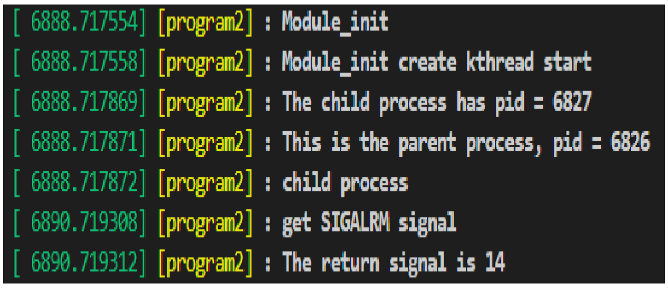
```
[ 6970.543531] [program2] : Module_init
[ 6970.543533] [program2] : Module_init create kthread start
[ 6970.543773] [program2] : The child process has pid = 7544
[ 6970.543774] [program2] : This is the parent process, pid = 7543
[ 6970.543775] [program2] : child process
[ 6970.649503] [program2] : get SIGABRT signal
[ 6970.649504] [program2] : The return signal is 6
```

Bus



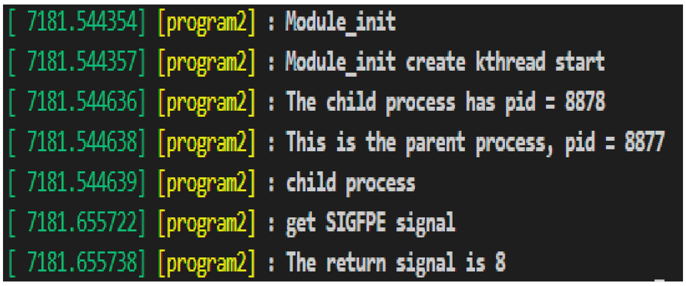
```
[ 7100.083826] [program2] : Module_init
[ 7100.083827] [program2] : Module_init create kthread start
[ 7100.083962] [program2] : The child process has pid = 8200
[ 7100.083963] [program2] : This is the parent process, pid = 8199
[ 7100.083964] [program2] : child process
[ 7100.189687] [program2] : get SIGBUS signal
[ 7100.189704] [program2] : The return signal is 7
```

Alarm



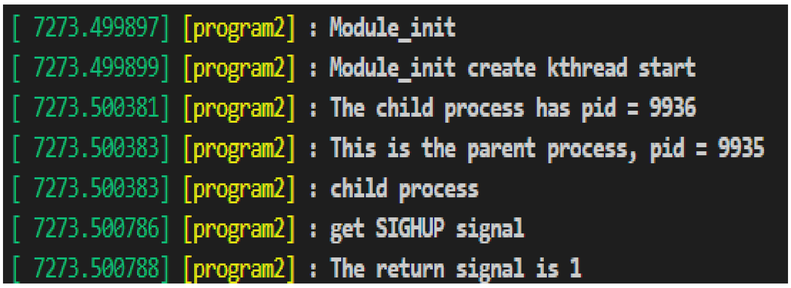
```
[ 6888.717554] [program2] : Module_init
[ 6888.717558] [program2] : Module_init create kthread start
[ 6888.717869] [program2] : The child process has pid = 6827
[ 6888.717871] [program2] : This is the parent process, pid = 6826
[ 6888.717872] [program2] : child process
[ 6890.719308] [program2] : get SIGALRM signal
[ 6890.719312] [program2] : The return signal is 14
```

Floating



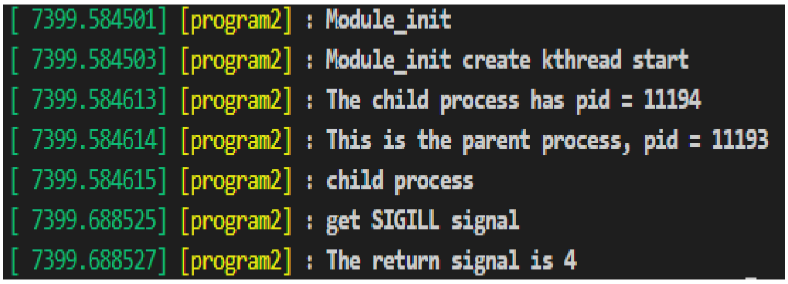
```
[ 7181.544354] [program2] : Module_init
[ 7181.544357] [program2] : Module_init create kthread start
[ 7181.544636] [program2] : The child process has pid = 8878
[ 7181.544638] [program2] : This is the parent process, pid = 8877
[ 7181.544639] [program2] : child process
[ 7181.655722] [program2] : get SIGFPE signal
[ 7181.655738] [program2] : The return signal is 8
```

Hang Up



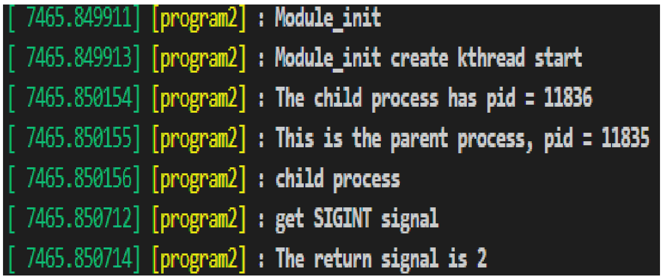
```
[ 7273.499897] [program2] : Module_init
[ 7273.499899] [program2] : Module_init create kthread start
[ 7273.500381] [program2] : The child process has pid = 9936
[ 7273.500383] [program2] : This is the parent process, pid = 9935
[ 7273.500383] [program2] : child process
[ 7273.500786] [program2] : get SIGHUP signal
[ 7273.500788] [program2] : The return signal is 1
```

Illegal Instr



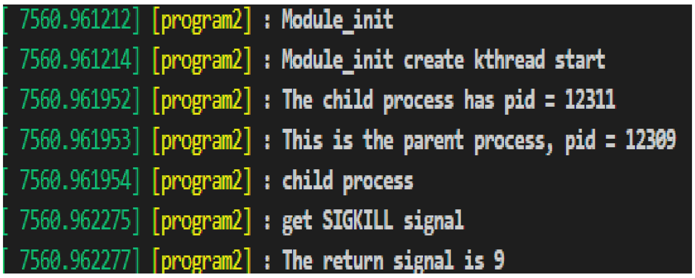
```
[ 7399.584501] [program2] : Module_init
[ 7399.584503] [program2] : Module_init create kthread start
[ 7399.584613] [program2] : The child process has pid = 11194
[ 7399.584614] [program2] : This is the parent process, pid = 11193
[ 7399.584615] [program2] : child process
[ 7399.688525] [program2] : get SIGILL signal
[ 7399.688527] [program2] : The return signal is 4
```

Interrupt



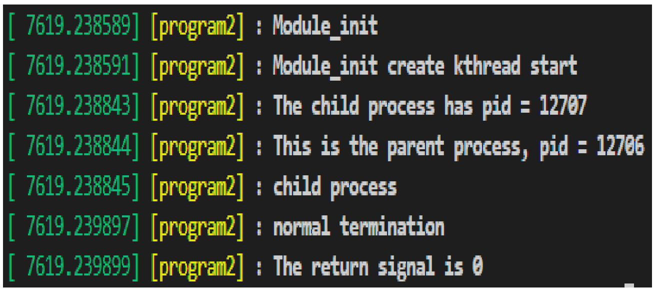
```
[ 7465.849911] [program2] : Module_init
[ 7465.849913] [program2] : Module_init create kthread start
[ 7465.850154] [program2] : The child process has pid = 11836
[ 7465.850155] [program2] : This is the parent process, pid = 11835
[ 7465.850156] [program2] : child process
[ 7465.850712] [program2] : get SIGINT signal
[ 7465.850714] [program2] : The return signal is 2
```

Kill



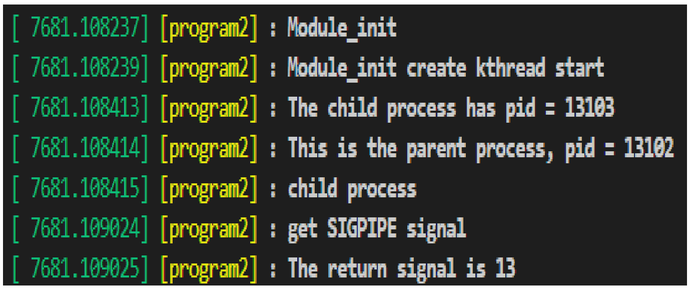
```
[ 7560.961212] [program2] : Module_init
[ 7560.961214] [program2] : Module_init create kthread start
[ 7560.961952] [program2] : The child process has pid = 12311
[ 7560.961953] [program2] : This is the parent process, pid = 12309
[ 7560.961954] [program2] : child process
[ 7560.962275] [program2] : get SIGKILL signal
[ 7560.962277] [program2] : The return signal is 9
```

Normal



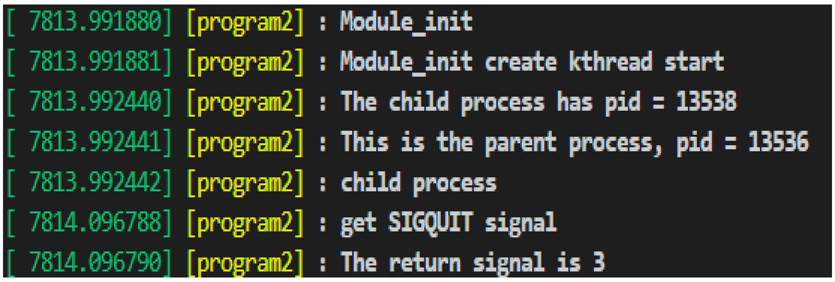
```
[ 7619.238589] [program2] : Module_init
[ 7619.238591] [program2] : Module_init create kthread start
[ 7619.238843] [program2] : The child process has pid = 12707
[ 7619.238844] [program2] : This is the parent process, pid = 12706
[ 7619.238845] [program2] : child process
[ 7619.239897] [program2] : normal termination
[ 7619.239899] [program2] : The return signal is 0
```

Pipe



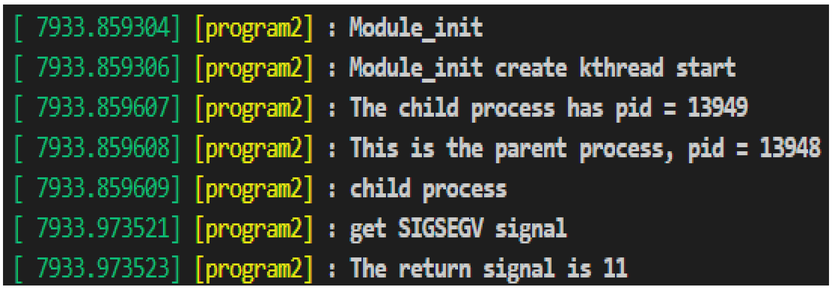
```
[ 7681.108237] [program2] : Module_init
[ 7681.108239] [program2] : Module_init create kthread start
[ 7681.108413] [program2] : The child process has pid = 13103
[ 7681.108414] [program2] : This is the parent process, pid = 13102
[ 7681.108415] [program2] : child process
[ 7681.109024] [program2] : get SIGPIPE signal
[ 7681.109025] [program2] : The return signal is 13
```

Quit



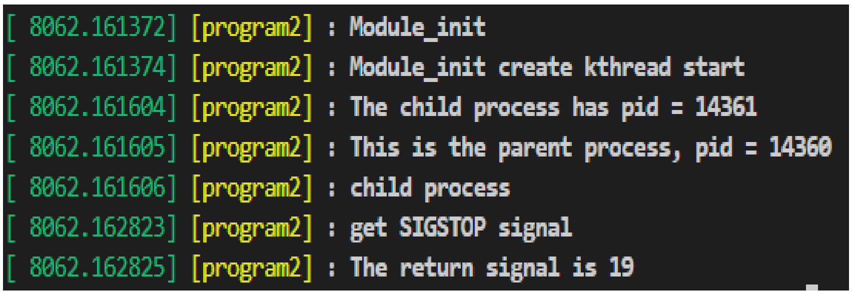
```
[ 7813.991880] [program2] : Module_init
[ 7813.991881] [program2] : Module_init create kthread start
[ 7813.992440] [program2] : The child process has pid = 13538
[ 7813.992441] [program2] : This is the parent process, pid = 13536
[ 7813.992442] [program2] : child process
[ 7814.096788] [program2] : get SIGQUIT signal
[ 7814.096790] [program2] : The return signal is 3
```

Segment Fault



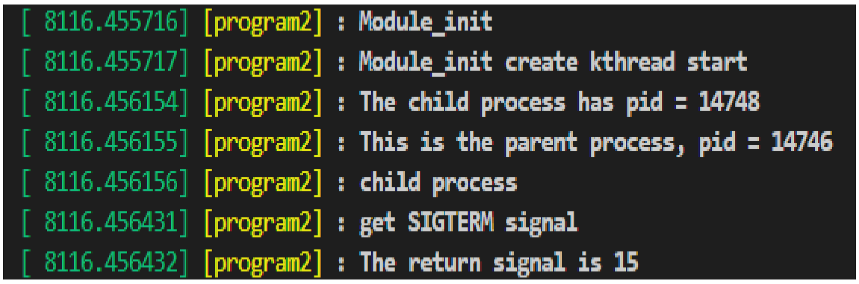
```
[ 7933.859304] [program2] : Module_init
[ 7933.859306] [program2] : Module_init create kthread start
[ 7933.859607] [program2] : The child process has pid = 13949
[ 7933.859608] [program2] : This is the parent process, pid = 13948
[ 7933.859609] [program2] : child process
[ 7933.973521] [program2] : get SIGSEGV signal
[ 7933.973523] [program2] : The return signal is 11
```

Stop



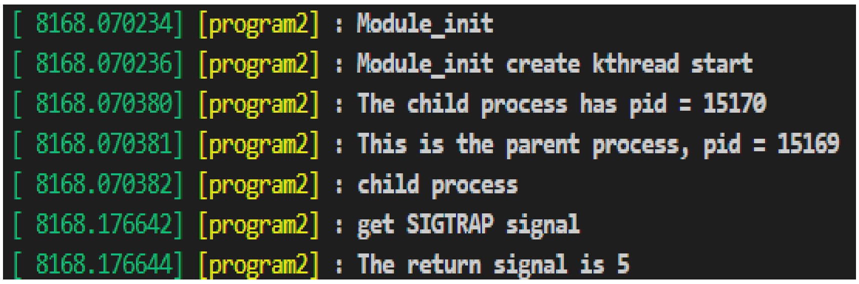
```
[ 8062.161372] [program2] : Module_init
[ 8062.161374] [program2] : Module_init create kthread start
[ 8062.161604] [program2] : The child process has pid = 14361
[ 8062.161605] [program2] : This is the parent process, pid = 14360
[ 8062.161606] [program2] : child process
[ 8062.162823] [program2] : get SIGSTOP signal
[ 8062.162825] [program2] : The return signal is 19
```

Terminate



```
[ 8116.455716] [program2] : Module_init
[ 8116.455717] [program2] : Module_init create kthread start
[ 8116.456154] [program2] : The child process has pid = 14748
[ 8116.456155] [program2] : This is the parent process, pid = 14746
[ 8116.456156] [program2] : child process
[ 8116.456431] [program2] : get SIGTERM signal
[ 8116.456432] [program2] : The return signal is 15
```

Trap



```
[ 8168.070234] [program2] : Module_init
[ 8168.070236] [program2] : Module_init create kthread start
[ 8168.070380] [program2] : The child process has pid = 15170
[ 8168.070381] [program2] : This is the parent process, pid = 15169
[ 8168.070382] [program2] : child process
[ 8168.176642] [program2] : get SIGTRAP signal
[ 8168.176644] [program2] : The return signal is 5
```