CSC3150 Assignment1 Report

Ziqi Gao 高梓骐

118010077

Task 1:

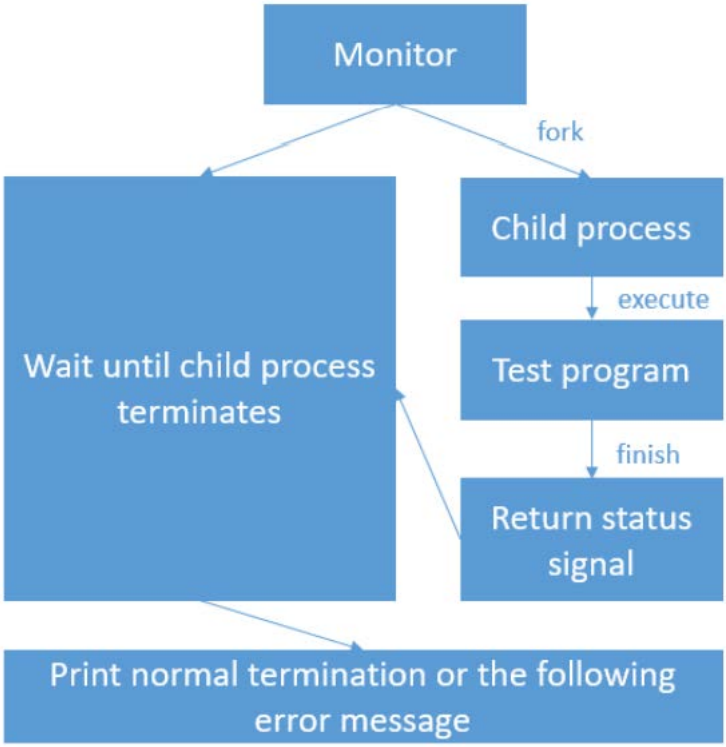
1.1 Introduction:

Versions of OS: Ubuntu 16.04.2 LTS (Xenial)

Kernel-Version: 4.10.14

In this task, a program ("program1.c") will implement the functions given by the interface from the operating system **in the user mode** like the flow chart below:

- Fork a child process to execute the test program.
- The child process will execute a test program that will send a **SIGCHLD** signal to the parent process.
- The parent process will wait the child process and capture the signals from its child process. After that, the parent process will print out the termination/stop or some other child process information by checking the signal.



1.2 Basic Design:

The program's design is in accordance with the flow chart and call the corresponding functions to finish the task. Fork function will generate a child process and return the 0 pid to the child process and return the pid of the child

process in the parent process. Then, the details in each process will be introduced.

Parent Process: *waitpid/wait* function will suspend the parent process until one of its children terminates, or when it receives some special signals. Also, those two functions will free the memory space of the exited process. This program use *waitpid* instead of wait because *waitpid* can have more setting options to handle the stop signal.

waitpid has three different option flags: Among them, the WUNTRACED option allows the parent to be returned from *waitpid()* if a child gets stopped as well as exiting or being killed. Since we need to deal with SIGSTOP, we need to set WUNTRACED.

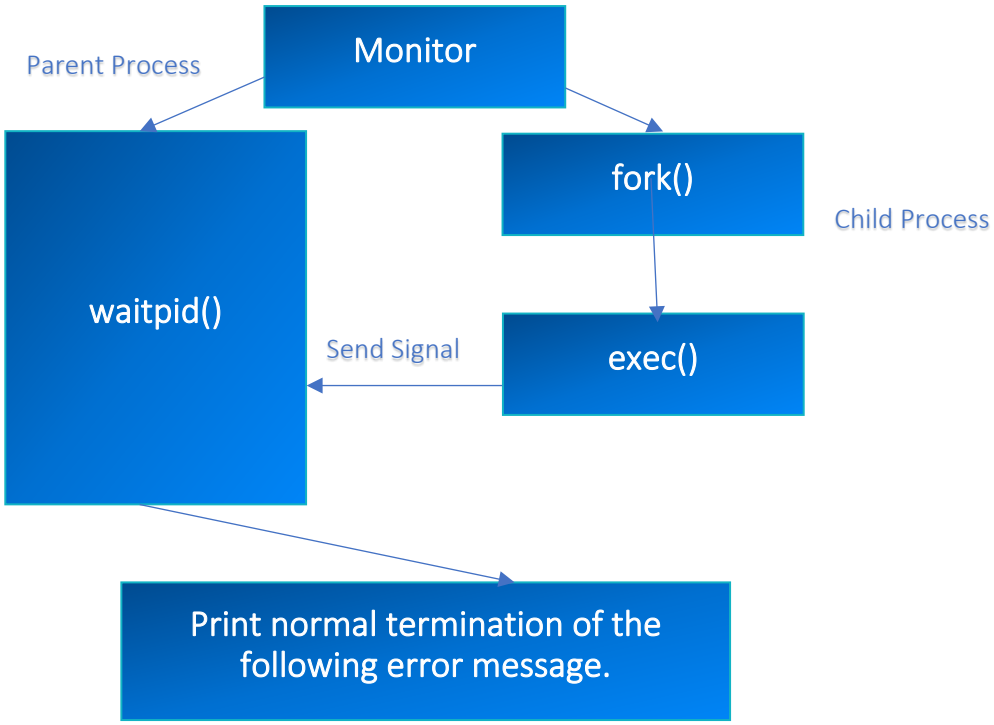
Child Process: It will call the exec function to run an executable file in the context of an already existing child process and replace the previous executable.

The three arguments of this exec function are:

*argv[0]: Path of the executed program

* arg: Array of pointers to strings passed to the new program as command-line arguments

* envp[]: Environment variable of the executed program



1.3 Execution:

1) build and generate the executable files using "make".

```
[10/10/20]seed@VM:~/.../program1$ make
cc -o abort abort.c
cc -o alarm alarm.c
cc -o bus bus.c
cc -o floating floating.c
cc -o hangup hangup.c
cc -o illegal_instr illegal_instr.c
cc -o interrupt interrupt.c
cc -o kill kill.c
cc -o normal normal.c
cc -o pipe pipe.c
cc -o program1 program1.c
cc -o quit quit.c
cc -o segment_fault segment_fault.c
cc -o stop stop.c
cc -o terminate terminate.c
cc -o trap trap.c
```

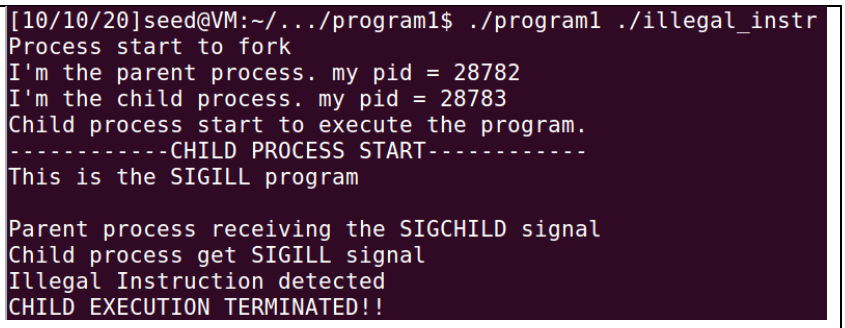
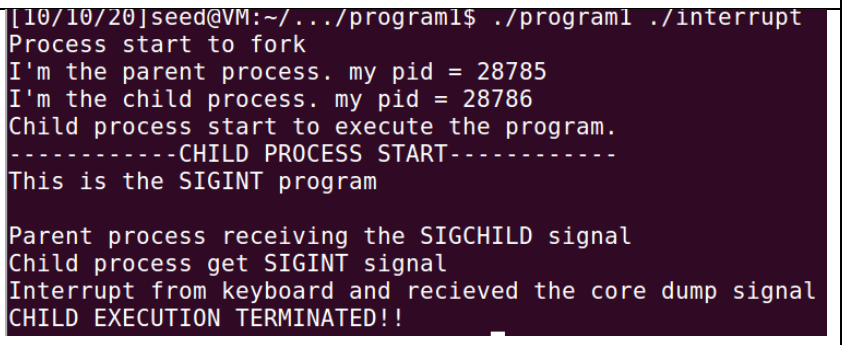
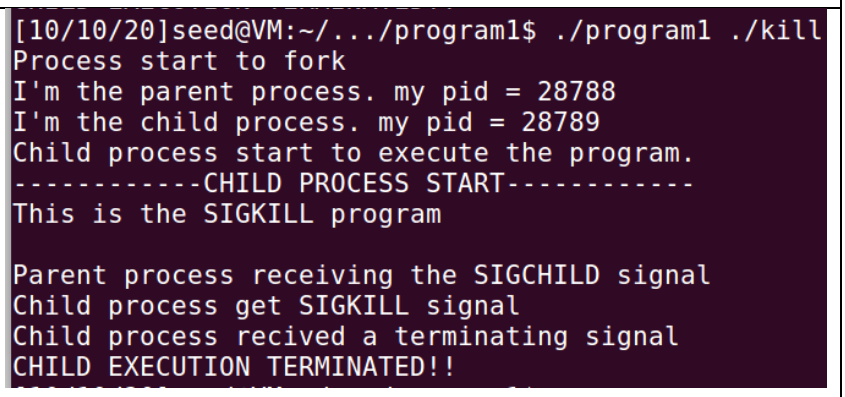
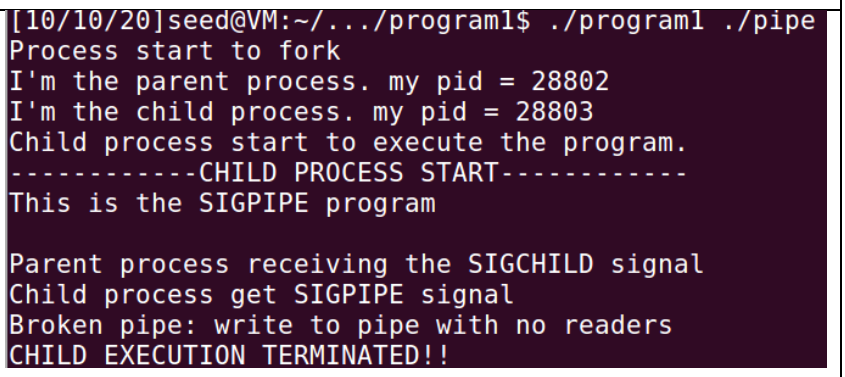
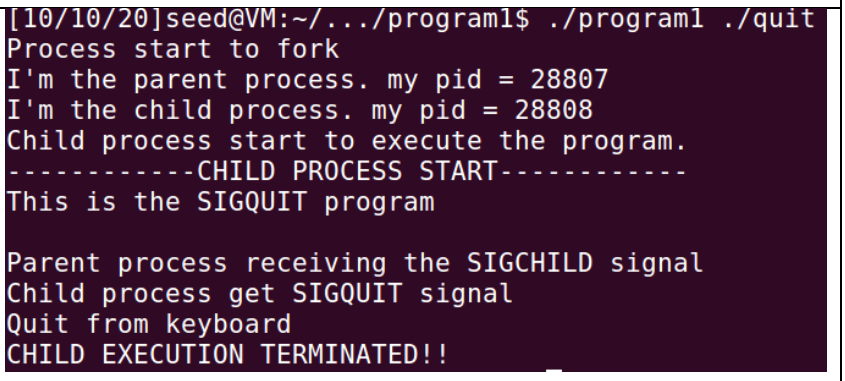
2) Run the program1 executable file and another program using "./program1 abort (or some other programs)"

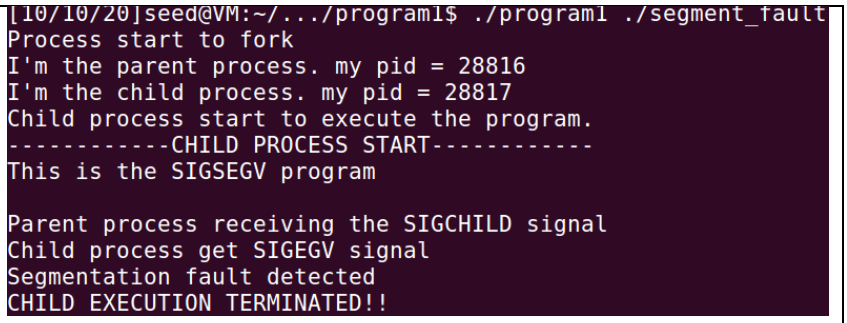
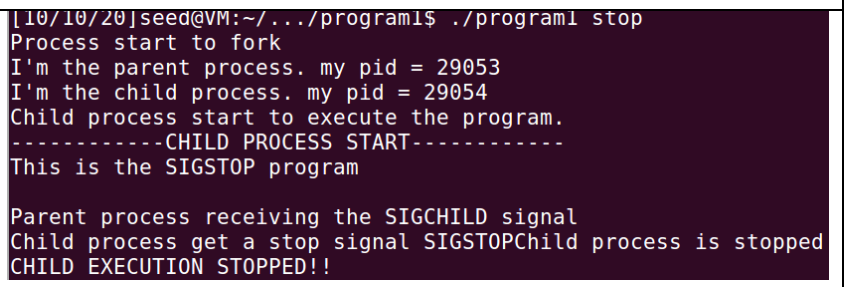
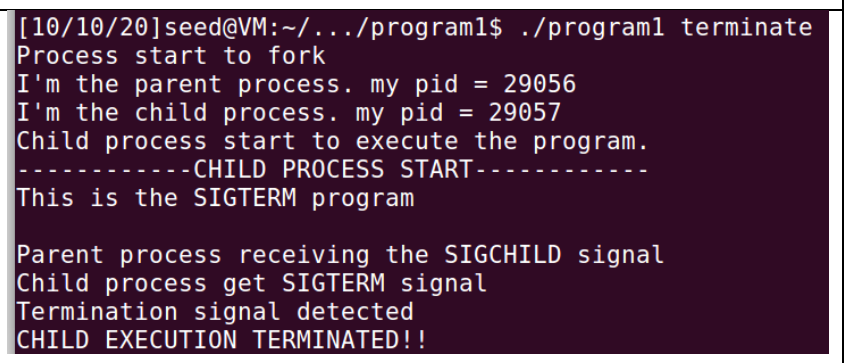
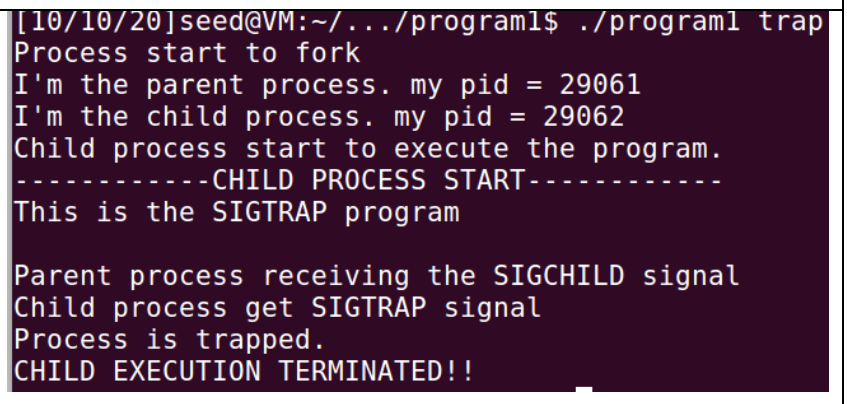
```
[10/10/20]seed@VM:~/.../program1$ ./program1 abort
Process start to fork
I'm the parent process. my pid = 28610
I'm the child process. my pid = 28611
Child process start to execute the program.
------------CHILD PROCESS START------------
This is the SIGABRT program

Parent process receiving the SIGCHILD signal
Child process get SIGABRT signal
Child process is aborted by abort signal
CHILD EXECUTION FAILED!!
```

1.4 Result:

./program1 normal	[10/10/20]seed@VM:~/.../program1$ ./program1 normal Process start to fork I'm the parent process. my pid = 28629 I'm the child process. my pid = 28630 Child process start to execute the program. ------------CHILD PROCESS START------------ This is the normal program ------------CHILD PROCESS END------------ Parent process receiving the SIGCHILD signal Normal termination with EXIT STATUS = 0

./program1 abort	[10/10/20]seed@VM:~/.../program1$ ./program1 ./abort Process start to fork I'm the parent process. my pid = 28664 I'm the child process. my pid = 28665 Child process start to execute the program. ------------CHILD PROCESS START------------ This is the SIGABRT program Parent process receiving the SIGCHILD signal Child process get SIGABRT signal Child process is aborted by abort signal CHILD EXECUTION FAILED!!
./program1 alarm	[10/10/20]seed@VM:~/.../program1$ ./program1 ./alarm Process start to fork I'm the parent process. my pid = 28740 I'm the child process. my pid = 28741 Child process start to execute the program. ------------CHILD PROCESS START------------ This is the SIGALRM program Parent process receiving the SIGCHILD signal Child process get SIGALRM signal Timer signal detected CHILD EXECUTION OVERTIME!!
./program1 bus	[10/10/20]seed@VM:~/.../program1$ ./program1 ./bus Process start to fork I'm the parent process. my pid = 28743 I'm the child process. my pid = 28744 Child process start to execute the program. ------------CHILD PROCESS START------------ This is the SIGBUS program Parent process receiving the SIGCHILD signal Child process get SIGBUS signal Bad memory access detected CHILD EXECUTION TERMINATED!!
./program1 floating	[10/10/20]seed@VM:~/.../program1$ ./program1 ./floating Process start to fork I'm the parent process. my pid = 28748 I'm the child process. my pid = 28749 Child process start to execute the program. ------------CHILD PROCESS START------------ This is the SIGFPE program Parent process receiving the SIGCHILD signal Child process get SIGFPE signal Floating point exception detected CHILD EXECUTION TERMINATED!!
./program1 hangup	[10/10/20]seed@VM:~/.../program1$ ./program1 ./hangup Process start to fork I'm the parent process. my pid = 28751 I'm the child process. my pid = 28752 Child process start to execute the program. ------------CHILD PROCESS START------------ This is the SIGHUP program Parent process receiving the SIGCHILD signal Child process get SIGHUP signal Hangup detected on controlling terminal CHILD EXECUTION TERMINATED!!

./program1 illegal_instr	
./program1 interrupt	
./program1 kill	
./program1 pipe	
./program1 quit	

./program1 segment_fault	
./program1 stop	
./program1 terminate	
./program1 trap	

1.5 What I learned:

In this task, I learned some knowledge about the user interface given by the Linux operating system. Also, this was my first time to use Makefile. I studied some syntax about it. And I got to know how the Makefile helped to compile my source code.

In addition, I knew some information about **process, pid, and signa**l. And I found man page was especially helpful when I was dealing with the signal.

However, although I finished this project, I did not quite understand what happened in the kernel mode. It just seemed I used some standard c library to do something without knowing the execution behind the interface.

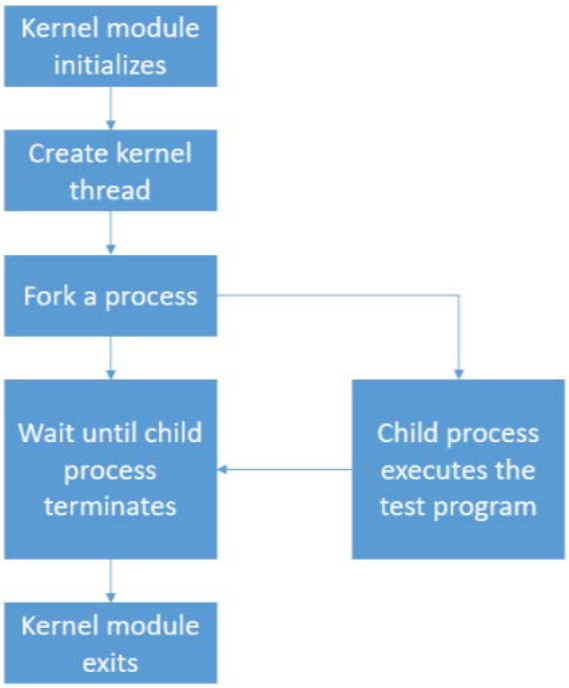
Task 2:

2.1 Introduction:

Versions of OS: Ubuntu 16.04.2 LTS (Xenial)

Kernel Version: 4.10.14 (with some modified kernel modules)

In this task, a program ("program2.c") will implement the functions using the Linux kernel functions **in the kernel mode** like the flow chart below:



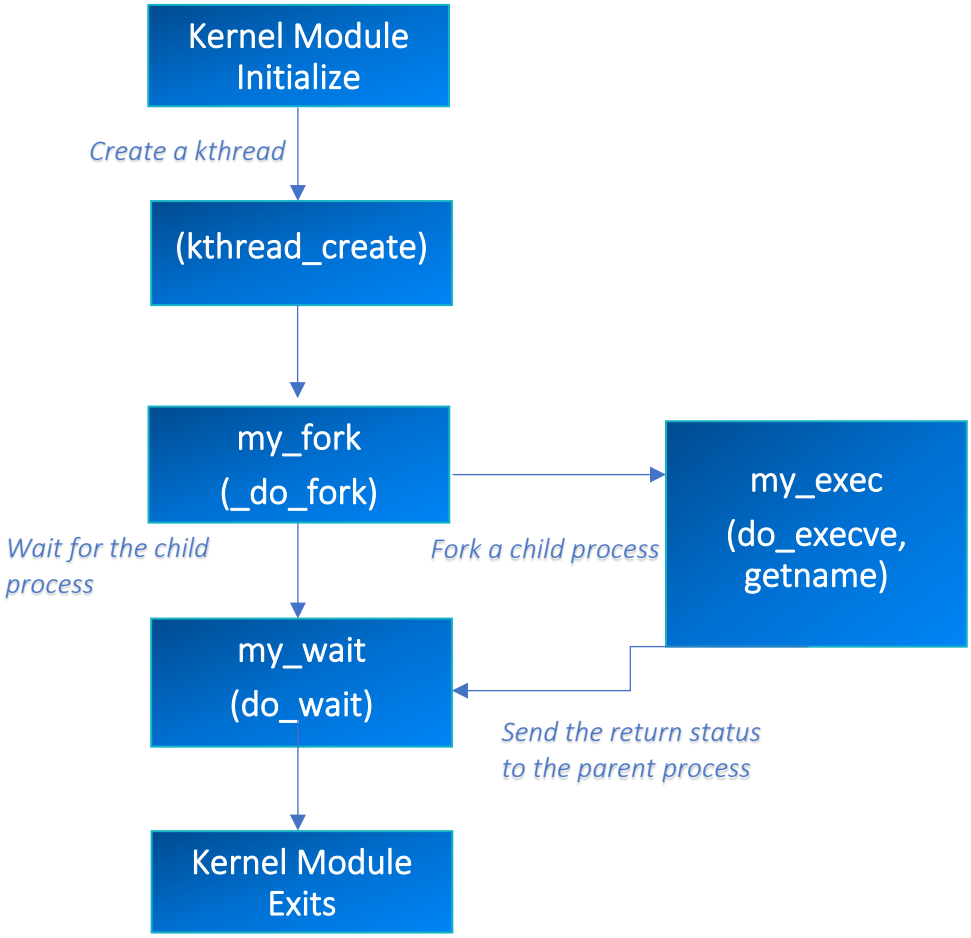
- Create a kernel thread (A process created by the kernel. It keeps track of what all is running on the system, how much of which resources are allocated to what process, and to do the scheduling. Also, kernel thread cannot get to the user mode address space).
- Fork a child process to execute the test program.
- Wait for the child process terminates.
- Catch the signals and print out the corresponding information.

2.2 Basic Design:

Since we are going to build and compile an extra kernel module to run our test program, we need to use the functions in the kernel instead of the interface given by the operating system. That is, I cannot use the functions (*fork, waitpid*, exec) and some macros like WIFSIGNALED in program1. However, some Linux kernel functions have been encapsulated, requiring us to modify the kernel source code to export those kernel functions. The relevant steps will be introduced later.

The following charts introduces the functions used in this module. Functions in the parenthesis are the relevant kernel functions.

Here is a flow chart of functions and brief introductions to each function:



(1) kthread_create: Create a kernel thread to run a certain function. The thread will be stopped at first. And we need to call *wake_up_process* to start it. This thread will be stopped if the threadfn call *do_exit.*

(2) my_fork: This function will fork a child process using *_do_fork* and print pid of the parent and child process.

_do_fork: Copy the parent process' information and allocate resources to the new child process.

(3) my_wait: This function will call *do_wait* function to wait for the execution of the child process and catch the return status. Then, using the macros about signal, it will get the signal from the return status from the child process.

do_wait: This function will receive an options structure to set when the parent process continues to execute. In this program, one should set the waiting flags to WEXITED | WUNTRACED to not wait for both the stopped or terminated child process and get back to the parent process. This function will also get a return status that can be interpreted as signals if the child process is terminated or stopped.

(4) my_exec: This function will set arguments for the *do_exec* function. The child process will then run an executable file in the context of an already existing child process and replace the previous executable.

getname: It copies the file name from user space to kernel space.

do_execve: Run an executable file in the context of an already existing child process and replace the previous executable. It will return a variable. If this variable is 0, the execution is successful.

(5) Signal Macros: Some macros are defined at the beginning of my program. Those macros are very similar to the macros in the GNU C library. Those macros are especially helpful in dealing with stop signal using some bitwise operations.

2.3 Modify and Compile the Kernel:

Those steps are so important that the program2.ko cannot use some encapsuled kernel functions without modifying the kernel modules. To make the program2 module works correctly, we need to modify four kernel files. What we need to modify is to add *EXPORT_SYMBOL* just after the definition of each function we need.

(1) getname: In /home/seed/work/linux-4.10.14/fs/namei.c

(2) _do_fork: In /home/seed/work/linux-4.10.14/kernel/fork.c

(3) do_wait: In /home/seed/work/linux-4.10.14/kernel/exit.c

(4) do_execve: In /home/seed/work/linux-4.10.14/fs/exec.c

After modifying the source files, we need to recompile the kernel modules and reboot the system.

(1) $make bzImage

(2) $make modules_install

(3) $make install

(4) Reboot the system.

2.4 Execution:

(1) Change to root user is more convenient.

```
[10/11/20]seed@VM:~/.../linux-4.10.14$ sudo su
root@VM:/home/seed/work/linux-4.10.14#
```

(2) Compile the executable test file: The test file 's directory is set as /home/seed/work/assignment1/source/program2/test. Please be sure that the test file's directory is correct and executable.

(3) Use the Makefile and $make to build the module:

```
root@VM:/home/seed/exp3150/program2# make
make -C /lib/modules/4.10.14/build M=/home/seed/exp3150/program2 modules
make[1]: Entering directory '/home/seed/work/linux-4.10.14'
  CC [M]  /home/seed/exp3150/program2/program2.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/seed/exp3150/program2/program2.mod.o
  LD [M]  /home/seed/exp3150/program2/program2.ko
make[1]: Leaving directory '/home/seed/work/linux-4.10.14'
```

(4) Use $insmod program2.ko to insert the module to the kernel:

```
root@VM:/home/seed/exp3150/program2# insmod program2.ko
root@VM:/home/seed/exp3150/program2#
```

(5) Use $rmmod program2.ko to remove the module from the kernel:

```
root@VM:/home/seed/exp3150/program2# rmmod program2.ko
root@VM:/home/seed/exp3150/program2#
```

(6) Use $dmesg | tail -n 10 to show the outputs in the kernel message buffer.

```
root@VM:/home/seed/exp3150/program2# dmesg | tail -n 10
[40420.684916] [program2] : Module_exit
[44046.695250] [program2] : Module_init
[44046.695251] [program2] : Module_init create kthread start
[44046.695320] [program2] : Module_init kthread start
[44046.695760] [program2] : The child process has pid = 3155
[44046.695760] [program2] : This is the parent process. pid = 3154
[44046.695812] [program2] : child process
[44046.696011] [program2] : The return signal is 19
[44046.696011] [program2] : get SIGSTOP signal
[44046.696012] [program2] : Child process is stopped
root@VM:/home/seed/exp3150/program2#
```

2.5 Result:

I copy the test program from program1 to test my program2 module.

Normal	[44572.382113] [program2] : Module_init [44572.382114] [program2] : Module_init create kthread s [44572.382127] [program2] : Module_init kthread start [44572.386078] [program2] : The child process has pid = [44572.386079] [program2] : This is the parent process. [44572.386244] [program2] : child process [44572.386560] [program2] : Normal termination with EXI
Abort	[44618.266129] [program2] : Module_init [44618.266130] [program2] : Module_init create kthread start [44618.266142] [program2] : Module_init kthread start [44618.266380] [program2] : The child process has pid = 6330 [44618.266380] [program2] : This is the parent process. pid = 6329 [44618.266442] [program2] : child process [44618.267041] [program2] : get SIGABRT signal [44618.267041] [program2] : Abort signal is detected

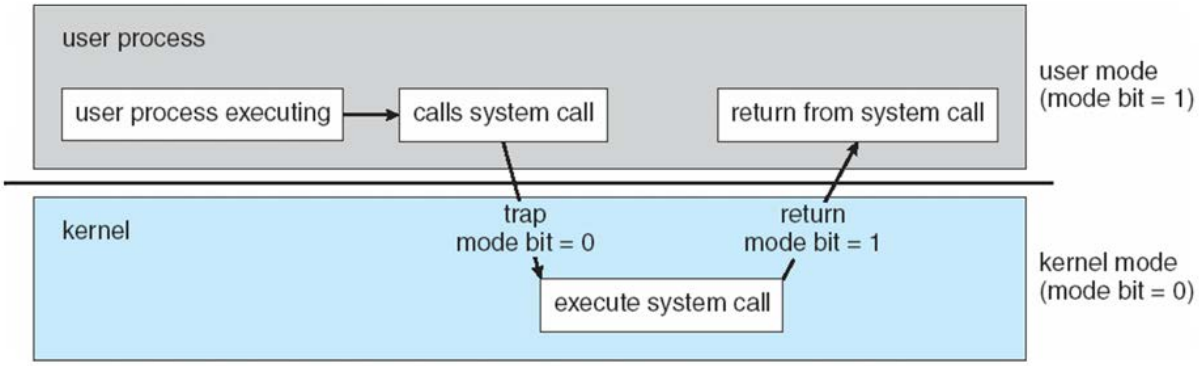
Alarm(Signal occurs after rmmod)	[44819.461744] [program2] : Module_init [44819.461745] [program2] : Module_init create kthread start [44819.461821] [program2] : Module_init kthread start [44819.461828] [program2] : The child process has pid = 9779 [44819.461829] [program2] : This is the parent process. pid = 9778 [44819.466841] [program2] : child process [44821.467258] [program2] : get SIGALRM signal [44821.467259] [program2] : Timer signal detected [44821.467259] [program2] : The return signal is 14
Bus	[44739.389465] [program2] : Module_init [44739.389466] [program2] : Module_init create kthread start [44739.390248] [program2] : Module_init kthread start [44739.395759] [program2] : The child process has pid = 8475 [44739.395760] [program2] : This is the parent process. pid = 8472 [44739.395942] [program2] : child process [44739.396321] [program2] : get SIGBUS signal [44739.396323] [program2] : Bad memory access detected
Floating	[44884.232939] [program2] : Module_init [44884.232940] [program2] : Module_init create kthread start [44884.232954] [program2] : Module_init kthread start [44884.238531] [program2] : The child process has pid = 10506 [44884.238533] [program2] : This is the parent process. pid = 10503 [44884.238649] [program2] : child process [44884.238934] [program2] : get SIGFPE signal [44884.238944] [program2] : Floating point exception detected
Hangup	[44918.612172] [program2] : Module_init [44918.612172] [program2] : Module_init create kthread start [44918.612185] [program2] : Module_init kthread start [44918.615666] [program2] : The child process has pid = 11214 [44918.615666] [program2] : This is the parent process. pid = 11211 [44918.619021] [program2] : child process [44918.619275] [program2] : get SIGHUP signal [44918.619276] [program2] : Hangup detected on controlling terminal
Illegal_instr	[44938.594714] [program2] : Module_init [44938.594714] [program2] : Module_init create kthread start [44938.594981] [program2] : Module_init kthread start [44938.595289] [program2] : The child process has pid = 11920 [44938.595290] [program2] : This is the parent process. pid = 11919 [44938.595349] [program2] : child process [44938.595602] [program2] : get SIGILL signal [44938.595602] [program2] : Illegal Instruction detected
Interrupt	[44976.701868] [program2] : Module_init [44976.701869] [program2] : Module_init create kthread start [44976.704666] [program2] : Module_init kthread start [44976.707354] [program2] : The child process has pid = 12631 [44976.707356] [program2] : This is the parent process. pid = 12627 [44976.707423] [program2] : child process [44976.707812] [program2] : get SIGINT signal [44976.707814] [program2] : Interrupt from keyboard and recieved the core dump signal
Kill	[45051.376479] [program2] : Module_init [45051.376480] [program2] : Module_init create kthread start [45051.376561] [program2] : Module_init kthread start [45051.376569] [program2] : The child process has pid = 13353 [45051.376570] [program2] : This is the parent process. pid = 13352 [45051.378216] [program2] : child process [45051.378727] [program2] : get SIGKILL signal [45051.378727] [program2] : Child process recived a terminating signal

Pipe	[45093.977839] [program2] : Module_init [45093.977840] [program2] : Module_init create kthread start [45093.981210] [program2] : Module_init kthread start [45093.984657] [program2] : The child process has pid = 14056 [45093.984659] [program2] : This is the parent process. pid = 14052 [45093.984798] [program2] : child process [45093.985377] [program2] : get SIGPIPE signal [45093.985380] [program2] : Broken pipe: write to pipe with no readers
Quit	[45113.777134] [program2] : Module_init [45113.777138] [program2] : Module_init create kthread start [45113.777384] [program2] : Module_init kthread start [45113.777415] [program2] : The child process has pid = 14748 [45113.777415] [program2] : This is the parent process. pid = 14747 [45113.778340] [program2] : child process [45113.792726] [program2] : get SIGQUIT signal [45113.792727] [program2] : Quit from keyboard
Segment_fault	[45133.076732] [program2] : Module_init [45133.076733] [program2] : Module_init create kthread start [45133.082147] [program2] : Module_init kthread start [45133.082686] [program2] : The child process has pid = 15450 [45133.082687] [program2] : This is the parent process. pid = 15449 [45133.083365] [program2] : child process [45133.083871] [program2] : get SIGSEGV signal [45133.083873] [program2] : Segmentation fault detected
Stop	[45154.587433] [program2] : Module_init [45154.587449] [program2] : Module_init create kthread start [45154.592024] [program2] : Module_init kthread start [45154.592165] [program2] : The child process has pid = 16157 [45154.592167] [program2] : This is the parent process. pid = 16156 [45154.592641] [program2] : child process [45154.595200] [program2] : get SIGSTOP signal [45154.595203] [program2] : Child process is stopped
Terminate	[45171.558963] [program2] : Module_init [45171.558964] [program2] : Module_init create kthread start [45171.561380] [program2] : Module_init kthread start [45171.561441] [program2] : The child process has pid = 16856 [45171.561441] [program2] : This is the parent process. pid = 16855 [45171.561664] [program2] : child process [45171.562338] [program2] : get SIGTERM signal [45171.562339] [program2] : Termination signal detected
Trap	[45190.861946] [program2] : Module_init [45190.861948] [program2] : Module_init create kthread start [45190.861979] [program2] : Module_init kthread start [45190.867191] [program2] : The child process has pid = 17560 [45190.867192] [program2] : This is the parent process. pid = 17557 [45190.868588] [program2] : child process [45190.869075] [program2] : get SIGTRAP signal [45190.869076] [program2] : Process is trapped.

2.6 What I Learned:

I compare this task with task1 because both are doing almost the same things, which is obvious by looking at the flow charts of those two tasks. However, in task2, we are in the kernel mode. I am unfamiliar with kernel mode so I tried to use some operating syscall interface just like what I did in task1. This is definitely not a good idea. Actually, when the user calls a function like *fork*, the operating system will receive the syscall from the user and call *_do_fork* function in the kernel mode to do the corresponding work. That is, in the

task2 we are already in the kernel mode; we just need to call the kernel functions.



Also, to finish the task, I learned a little bit about how to modify the kernel source file and recompile the kernel. Although I failed two or three times when I modify the kernel and meet some terrible problems that I have no idea to solve, it is a very good experience.

I still need to read some books about the operating system and the Linux kernel to be prepared for the following challenges.