CSC3150 Assignment 1

Homework Requirements

Environment

- **WARNING!!!** Before starting on this assignment, make sure you have set up your VM properly. We would test all students' homework using the following environment. You can type the following command in terminal on your VM to see if your configuration matches the test environment. If not, you are still good to go, but please try to test your program with the following environment for at least once. Because you may be able to run your program on your environment, but not on TAs' environment, causng inconvenience or even grade deduction.

  If you follow the tutorials then your VM setting should be fine, though verify your environment again is recommended.

- **Linux Version**

```
main@ubuntu:/$ cat /etc/issue
Ubuntu 16.04.5 LTS \n \l
```

- **Linux Kernel Version**

```
main@ubuntu:/$ uname -r
4.10.14
```

- **GCC Version**

```
main@ubuntu:/$ gcc --version
gcc (Ubuntu 5.4.0-6ubuntu1~16.04.10) 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.
```

Submission

- **Due on: 23:59, 11 Oct 2020**
- Please note that, teaching assistants may ask you to explain the meaning of your program, to ensure that the codes are indeed written by yourself. Please also note that we would check whether your program is too similar to your fellow students' code using plagiarism detectors.
- Violation against the format requirements will lead to grade deduction.

Here is the format guide. The project structure is illustrated as below. You can also use `tree` command to check if your structure is fine. Structure mismatch would cause grade deduction.

```
main@ubuntu:~/Desktop/Assignment_1_<student_id>$ tree
```

```
.
├── report.pdf
└── source
    ├── bonus
    │   ├── Makefile
    │   ├── myfork.c
    │   └── <other_files>
    ├── program1
    │   ├── abort.c
    │   ├── alarm.c
    │   ├── bus.c
    │   ├── floating.c
    │   ├── hangup.c
    │   ├── illegal_instr.c
    │   ├── interrupt.c
    │   ├── kill.c
    │   ├── Makefile
    │   ├── normal.c
    │   ├── pipe.c
    │   ├── program1.c
    │   ├── quit.c
    │   ├── segment_fault.c
    │   ├── stop.c
    │   ├── terminate.c
    │   └── trap.c
    └── program2
        ├── Makefile
        ├── program2.c
        └── test.c

4 directories, <your_file_num> files
```

Please compress all files in the file structure root folder into a single zip file and **name it using your student id as the code showing below and above, for example, Assignment_1_118010001.zip.** The report should be submitted in the format of **pdf**, together with your source code. Format mismatch would cause grade deduction. Here is the sample step for compress your code.

```
main@ubuntu:~/Desktop$ zip -q -r Assignment_1_<student_id>.zip
Assignment_1_<student_id>
main@ubuntu:~/Desktop$ ls
Assignment_1_<student_id>                Assignment_1_<student_id>.zip
```

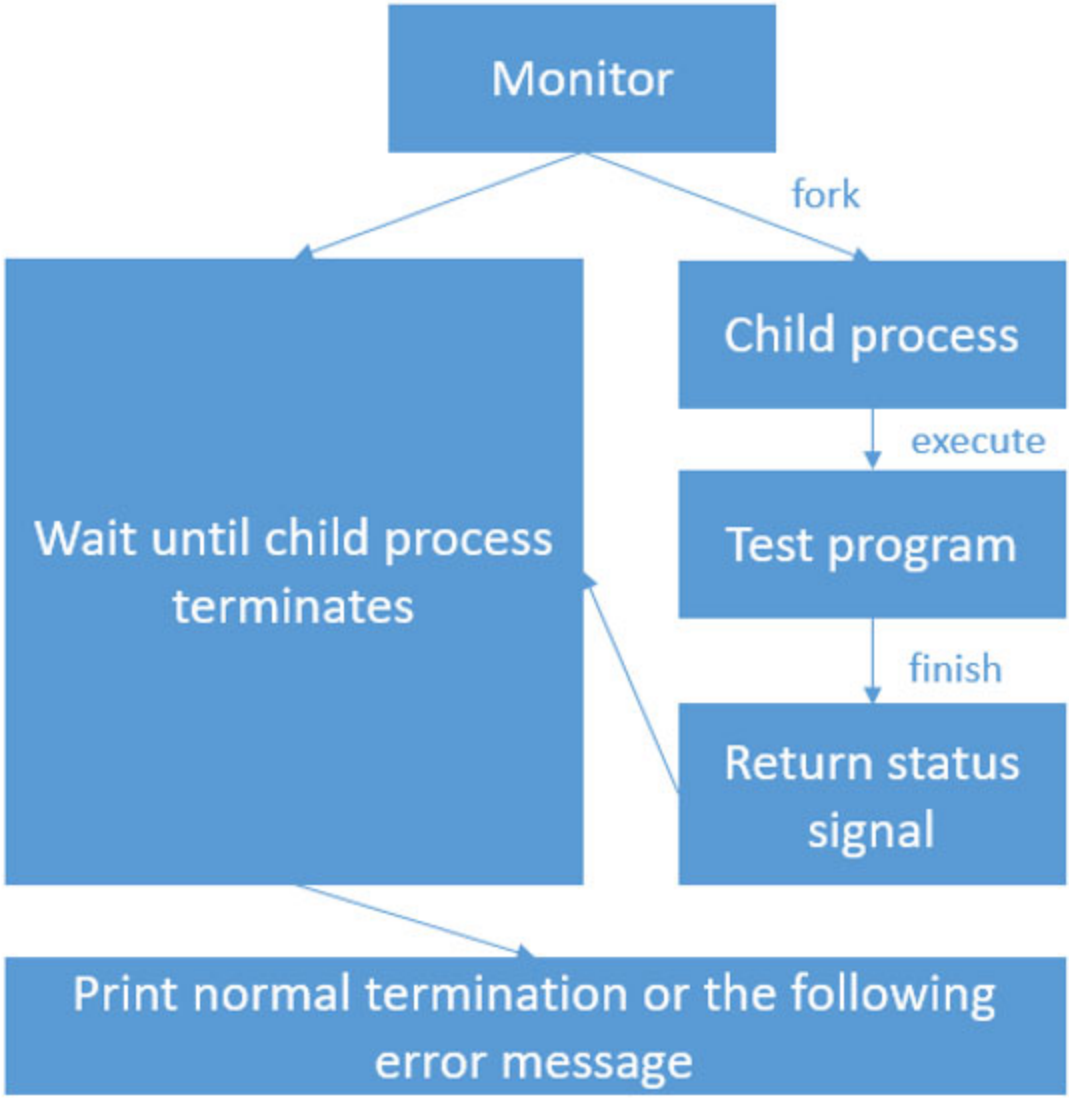
Task 1 (30 points)

In this task, you should write a program (`program1.c`) that implement the functions below:

- In user mode, fork a child process to execute the test program. (10 points)
- When child process finish execution, the parent process will receive the SIGCHLD signal by wait() function. (5 points)
- There are 15 test programs provided. 1 is for normal termination, and the rest are exception cases. Please use these test programs as your executing programs.

- The termination information of child process should be print out. If normal termination, print normal termination and exit status. If not, print out how did the child process terminates and what signal was raised in child process. (15 points)

The main flow chart for Task 1 is:



Demo outputs:

- Demo output for normal termination:

```
main@ubuntu:~/Desktop/Assignment_1_example/source/program1$ ./program1
./normal
Process start to fork
I'm the Parent Process, my pid = 4903
I'm the Child Process, my pid = 4904
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
```

- Demo output for signaled abort:

```
main@ubuntu:~/Desktop/Assignment_1_example/source/program1$ ./program1
./abort
Process start to fork
I'm the Parent Process, my pid = 4908
I'm the Child Process, my pid = 4909
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGABRT program

Parent process receives SIGCHLD signal
```

- Demo output for stopped:

```
main@ubuntu:~/Desktop/Assignment_1_example/source/program1$ ./program1
./stop
Process start to fork
I'm the Parent Process, my pid = 4931
I'm the Child Process, my pid = 4932
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGSTOP program

Parent process receives SIGCHLD signal
CHILD PROCESS STOPPED
```

Task 2 (60 points)

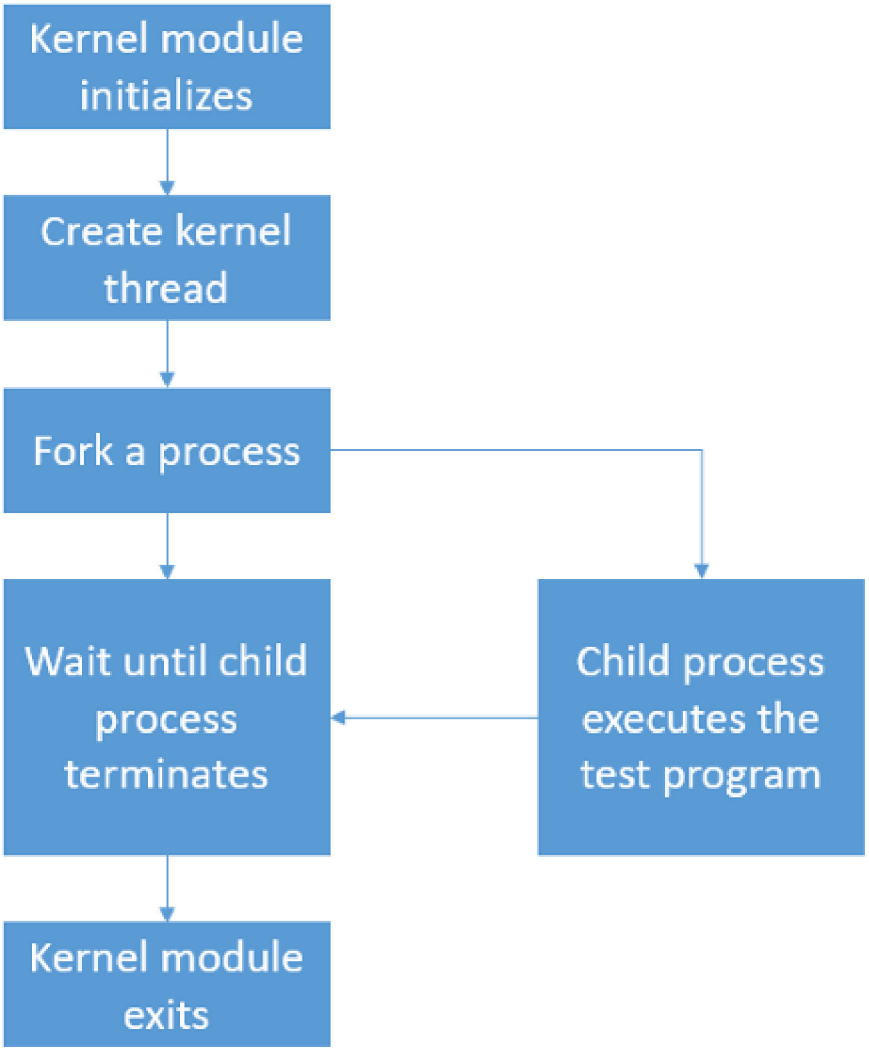
In this task, a template (“program2.c”) is provided. Within the template, please implement the functions below:

- When program2.ko being initialized, create a kernel thread and run my_fork function. (10 points)
- Within my_fork, fork a process to execute the test program. (10 points)
- The parent process will wait until child process terminates. (10 points)
- Print out the process id for both parent and child process. (5 points)
- Within this test program, it will raise signal. The signal could be caught and related message should be printed out in kernel log. (10 points)
- Follow the hints below to implement your function. If the function is non-static, you should firstly export this symbol so that it could be used in your own kernel module. After that, you should compile the kernel source code and install it. (Kernel compile: 15 points)

  Hints:

  - Use “_do_fork” to fork a new process. (/kernel/fork.c)
  - Use “do_execve” to execute the test program. (/fs/exec.c)
  - Use “getname” to get filename. (/fs/namei.c)
  - Use “do_wait” to wait for child process’ termination status. (/kernel/exit.c)

The main flow chart for Task 2 is:



Demo output:

```
[ 3769.385776] [program2] : module_init
[ 3769.385777] [program2] : module_init create kthread start
[ 3769.385777] [program2] : module_init kthread start
[ 3769.389787] [program2] : The child process has pid = 2914
[ 3769.389793] [program2] : This is the parent process, pid = 2912
[ 3769.391602] [program2] : child process
[ 3769.391604] [program2] : get SIGTERM signal
[ 3769.391605] [program2] : child process terminated
[ 3769.391605] [program2] : The return signal is 15
[ 3773.346070] [program2] : module_exit./my
```

Bonus Task (10 pionts)

In this task, you should create an executable file named “myfork” to implement the functions below:

- Execute “./myfork test_program” will show the signal message that child process executes test_program. (3 points)

- We can add multiple executable files as the argument of myfork, like "./myfork test_program1 test_program2 test_program3". From second argument, the queue indicates the relationship of parent and child. The proceeding one is parent, and the tailing one is child. For example: test_program_1 is parent of test_program_2, and test_program_2 is the parent of test_program_3. (3 points)
- Print out a process tree, which can indicate the relationship of the test programs.
- For example: 1 -> 2 -> 3 (1/2/3 is the process ID) (4 points)

Demo output:

```
main@ubuntu:~/Desktop/Assignment_1_117010033/source/bonus$ ./myfork hangup
normal8 alarm
------------CHILD PROCESS START------------
This is the SIGALRM program

This is normal8 program
------------CHILD PROCESS START------------
This is the SIGHUP program



---
Process tree: 5712->5713->5714->5715
Child process 5715 of parent process 5714 is terminated by signal 14 (Alarm
clock)
Child process 5714 of parent process 5713 terminated normally with exit code 0
Child process 5713 of parent process 5712 is terminated by signal 1 (Hangup)
Myfork process (5712) terminated normally
```

Report (10 points)

Write a report for your assignment, which should include main information as below:

- Your name and student id.
- How did you design your program?
- The environment of running your program. (E.g., version of OS and kernel)
- The steps to execute your program.
- Screenshot of your program output.
- What did you learn from the tasks?

Grading rules

Here is a sample grading scheme. Differ from the points specified above, this is the general guide when TA's grading.

Completion	Marks
Bonus	10 points
Report	10 points
Completed with good quality	80 ~ 90
Completed accurately	80 +
Fully Submitted (compile successfully)	60 +
Partial submitted	0 ~ 60
No submission	0
Late submission	Not allowed