CSC3150 Assignment 1

Homework Requirements

Environment

WARNING!!! Before starting on this assignment, make sure you have set up your VM following the instructions in tutorial 1 or meet the following conditions. We would test all students' homework using the following environment. For the programs unable to run on TA's environment, an offline meeting is a must.

- **Linux Distribution**: Ubuntu 16.04-22.04
- **Linux Kernel Version**: Target version: 5.15.x
- **GCC Version**: 4.9 above (use `gcc -v` to get it)
- **Makefile**: Please write makefile to compile and install your program in this course. We only use makefile to test your program when we grade (If not used, this program will have a score of 0, and it is not acceptable to use your own computer to run).

Submission

Due on: 23:59, 8 Oct 2025

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.

Late submission: A late submission within 15 minutes will not induce any penalty on your grades. But 00:16 am-1:00 am: Reduced by 10%; 1:01 am-2:00 am: Reduced by 20%; 2:01 am-3:00 am: Reduced by 30% and so on.

Example: Li Hua submit a perfect attempt of hw1 on 2:10 am. He will get `(100+10 (bonus))*0.7=77` points for his hw1.

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_122010001$ 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_122010001.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:

bash

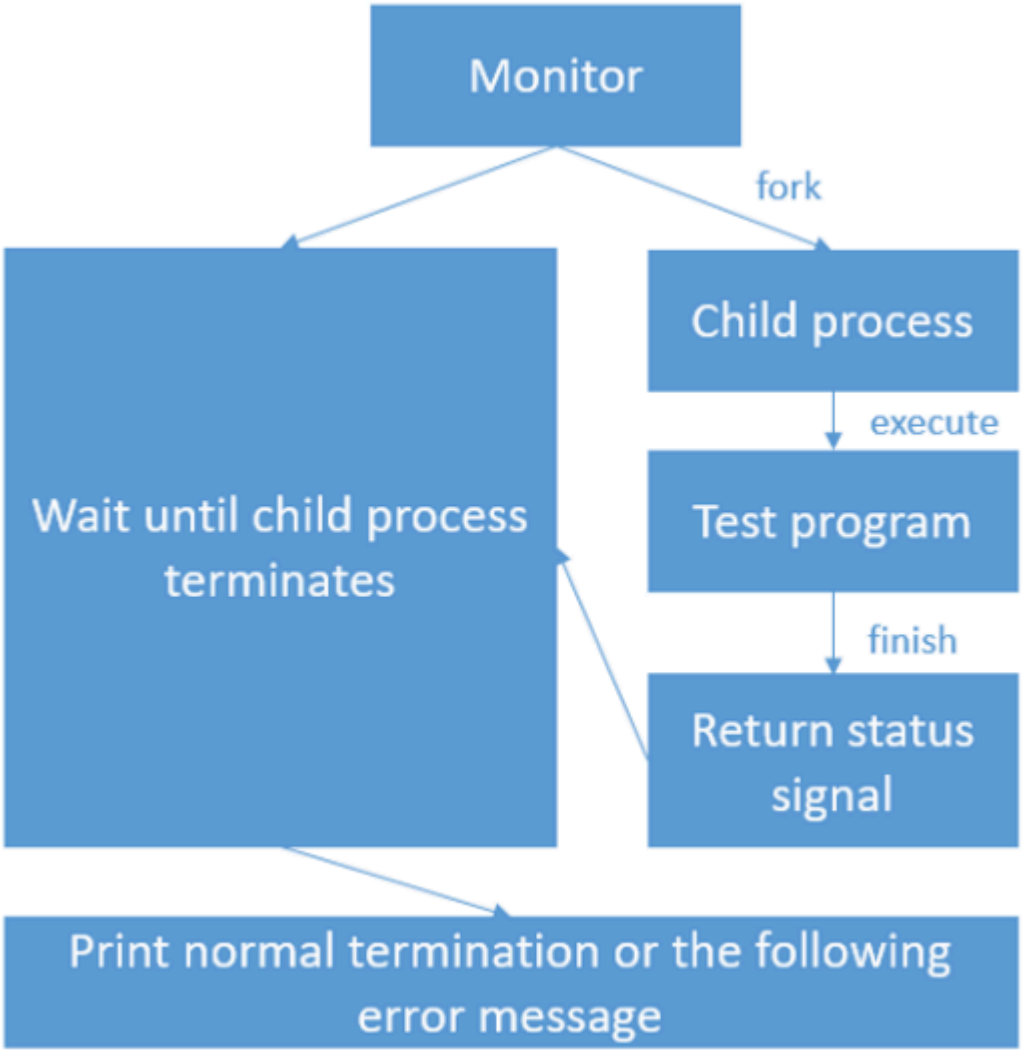
```
main@ubuntu:~/Desktop$ zip -q -r Assignment_1_122010001.zip
Assignment_1_122010001
```

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
child process get SIGABRT 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 get SIGSTOP signal
```

Task 2 (60 points)

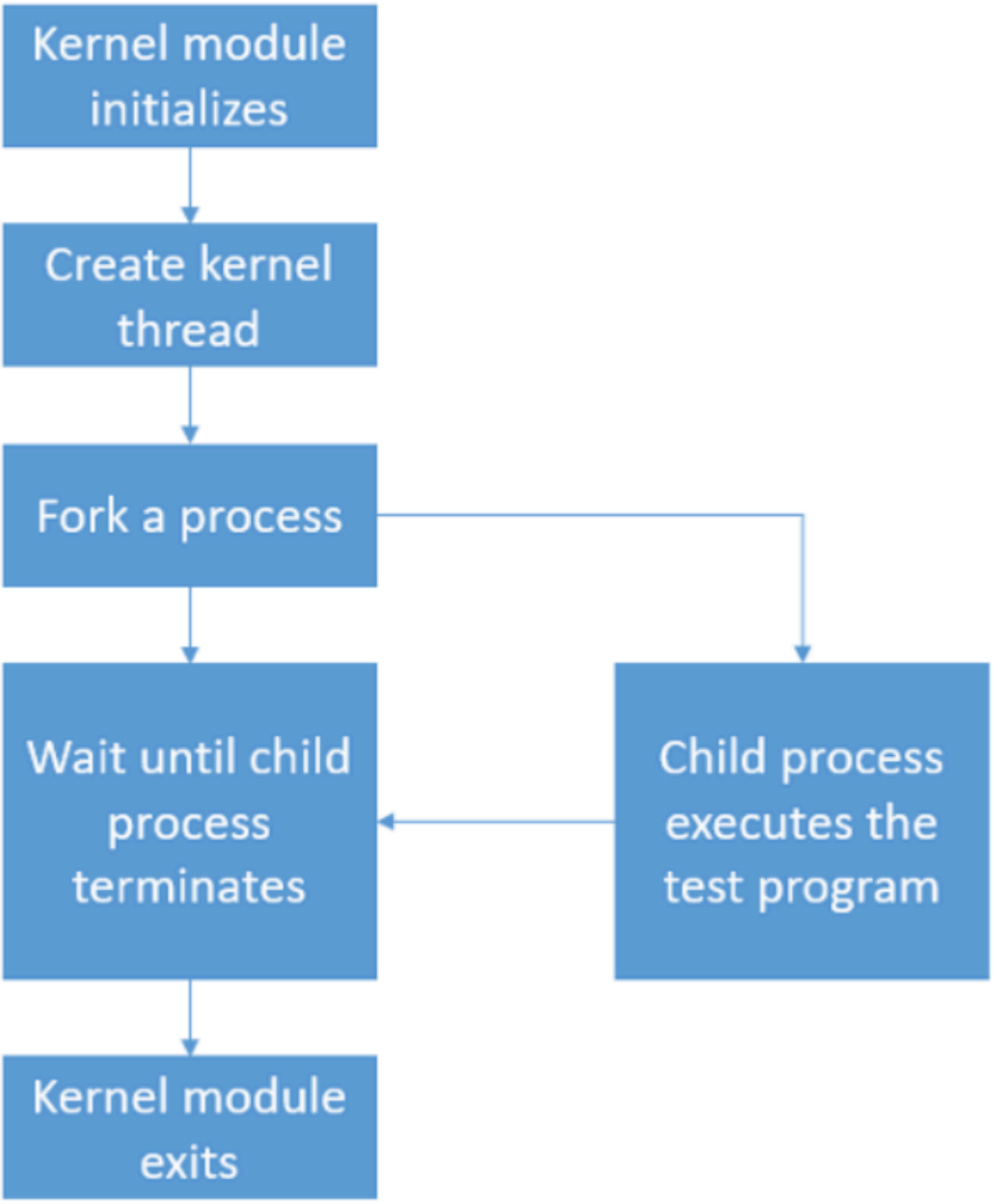
In this task, a template (“program2.c”) is provided. Within the template, please implement the functions below:

- 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
```

Note: the variable path in `my_exec()` in `program2.c` should be `/tmp/test` before submitting your homework, otherwise your grade would be influenced seriously.

Bonus Task (10 points)

There is a linux command, called `pstree`, which could print out the process tree of linux.

```
pstree
systemd─┬─NetworkManager───2*[{NetworkManager}]
 ├─abrt-watch-log
 ├─abrtd
 ├─agetty
 ├─atd
 ├─auditd───{auditd}
 ├─blackbox_export─┬─run_blackbox_ex───tee
 | └─75*[{blackbox_export}]
```

In this task, we need to create a file to implement this function and the file name is `pstree.c`.There are many options of `pstree` . You can use `man pstree` to discover it. Before implementing this program, I suggest that use this command by yourself to see how it works.

Grading criteria:

- The program can output a process tree as above. (5 points)
- You will receive one point for each option implemented. So if you want to get 10 points in bonus, you need to implement another 5 options of `pstree`.
- You cannot call the `pstree` command directly in your program.

Hints: This reference (https://en.wikipedia.org/wiki/Procfs) might help you to finish the bonus task.

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?** (4 points)
- **How to set up your development environment, including how to compile kernel?** (2 points)
- **Screenshot of your program output.** (2 points)
- What did you learn from the tasks? (2 points)

Please note that **the report will only be graded if you total score for Task 1 and Task 2 exceeds 54 points.**

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

References

- https://www.gnu.org/software/libc/manual/html_node/Process-Identification.html
- Linux source code (v5.15.10) - Bootlin Elixir Cross Referencer (This one can help you to search symbols in the kernel.)
- https://seisman.github.io/how-to-write-makefile/ (Chinese)
- https://makefiletutorial.com/