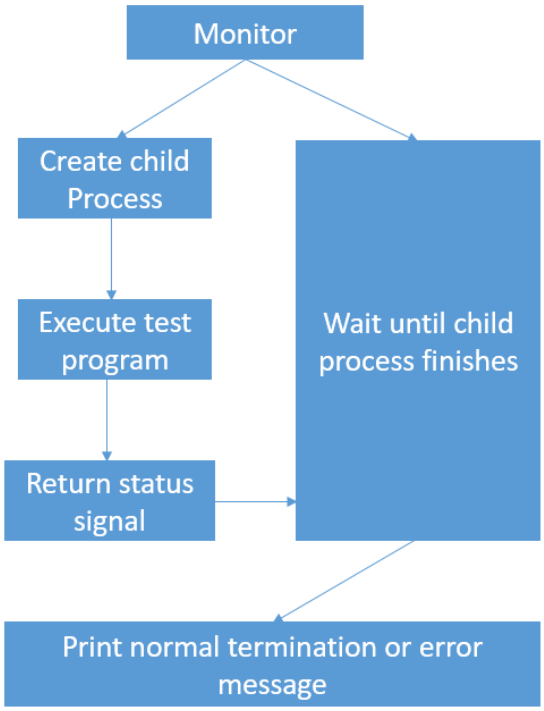
CSC3150 Assignment 1

Task 1 (30 points)

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

- 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 program.
- 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 output for normal termination:

```
[09/26/18]seed@VM:~/.../program1$ ./program1 ./normal
Process start to fork
I'm the parent process, my pid = 2418
I'm the child process, my pid = 2419
Child process start to execute the program
------------CHILD PROCESS START------------
This is the normal program

------------CHILD PROCESS END------------
Parent process receiving the SIGCHLD signal
Normal termination with EXIT STATUS = 0
[09/26/18]seed@VM:~/.../program1$
```

- Demo output for signaled termination:

```
[09/26/18]seed@VM:~/.../program1$ ./program1 ./terminate
Process start to fork
I'm the parent process, my pid = 2513
I'm the child process, my pid = 2514
Child process start to execute the program
------------CHILD PROCESS START------------
This is the SIGTERM program

Parent process receiving the SIGCHLD signal
child process get SIGTERM signal
child process is terminated by termination signal
CHILD EXECUTION FAILED!!
[09/26/18]seed@VM:~/.../program1$
```

- Demo output for stopped:

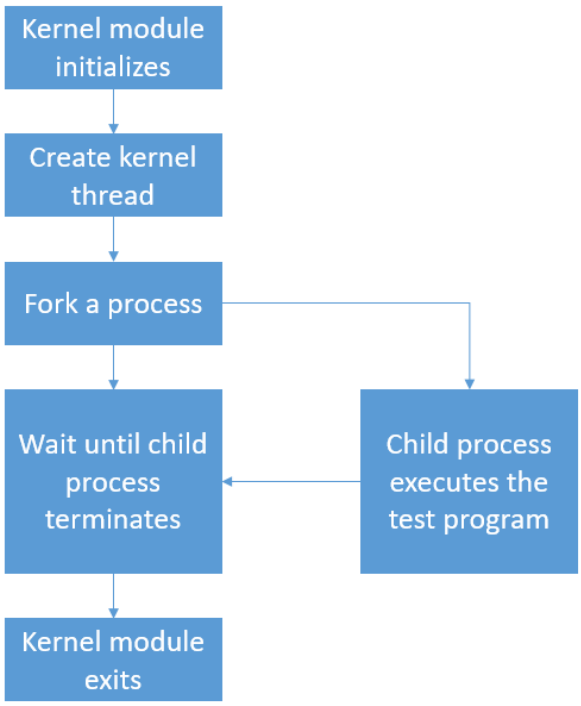
```
[09/26/18]seed@VM:~/.../program1$ ./program1 ./stop
Process start to fork
I'm the parent process, my pid = 3129
I'm the child process, my pid = 3130
Child process start to execute the program
------------CHILD PROCESS START------------
This is the SIGSTOP program

Parent process receiving the SIGCHLD signal
child process get SIGSTOP signal
child process stopped
CHILD PROCESS STOPPED
[09/26/18]seed@VM:~/.../program1$
```

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:
  1) Use "_do_fork" to fork a new process. (/kernel/fork.c)
  2) Use "do_execve" to execute the test program. (/fs/exec.c)
  3) Use "getname" to get filename. (/fs/namei.c)
  4) Use "do_wait" to wait for child process' termination status. (/kernel/exit.c)
- The main flow chart for Task 2 is:



- Demo output:

```
[  817.172447] [program2] : module_init
[  817.172450] [program2] : module_init create kthread start
[  817.172910] [program2] : module_init kthread start
[  817.176161] [program2] : The child process has pid = 6185
[  817.176163] [program2] : This is the parent process,pid = 6183
[  817.180450] [program2] : The child process is core-dumped
[  817.180452] [program2] : The return signal number = 126
[  817.180453] [program2] : child process
[  817.180454] get signal, signal number = 126
[  823.254158] [program2] : module_exit
[09/25/18]seed@VM:~/.../program2$
```

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 execute 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

```
[09/26/18]seed@VM:~/.../bonus$ ./myfork hangup normal8 trap
------------CHILD PROCESS START------------
This is the SIGTRAP program

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

the process tree : 2375->2376->2377->2378
The child process(pid=2378) of parent process(pid=2377) is terminated by signal
Its signal number = 5
child process get SIGTRAP signal
child process reach a breakpoint

The child process(pid=2377) of parent process(pid=2376) has normal execution
Its exit status = 0

The child process(pid=2376) of parent process(pid=2375) is terminated by signal
Its signal number = 1
child process get SIGHUP signal
child process is hung up


Myfork process(pid=2375) execute normally
[09/26/18]seed@VM:~/.../bonus$
```

Report (10 points)

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

- How did you design your program?
- The steps to execute your program.
- Screenshot of your program output.
- What did you learn from the tasks?

Submission

- Please submit the file as package with directory structure as below:
  - **Assignment_1_(Student ID)**
    - Source
      - Task 1
      - Task 2
      - Bonus
    - Report
- Due date: Oct 11, 2018

Grading rules

Completion	Marks
Bonus	0 ~ 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