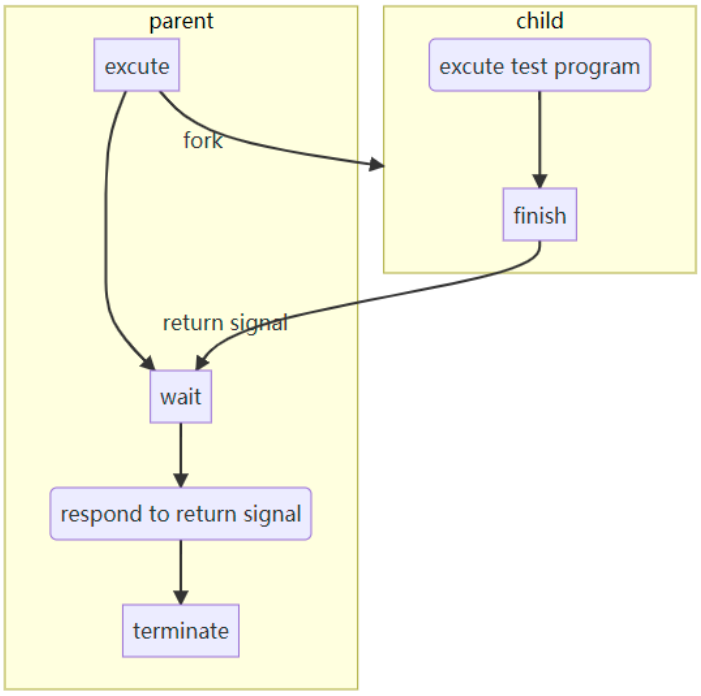
CSC3150 Assignment 1 Report

Name: Qin Peiran Student ID: 119010249

Part 1: Idea of Design

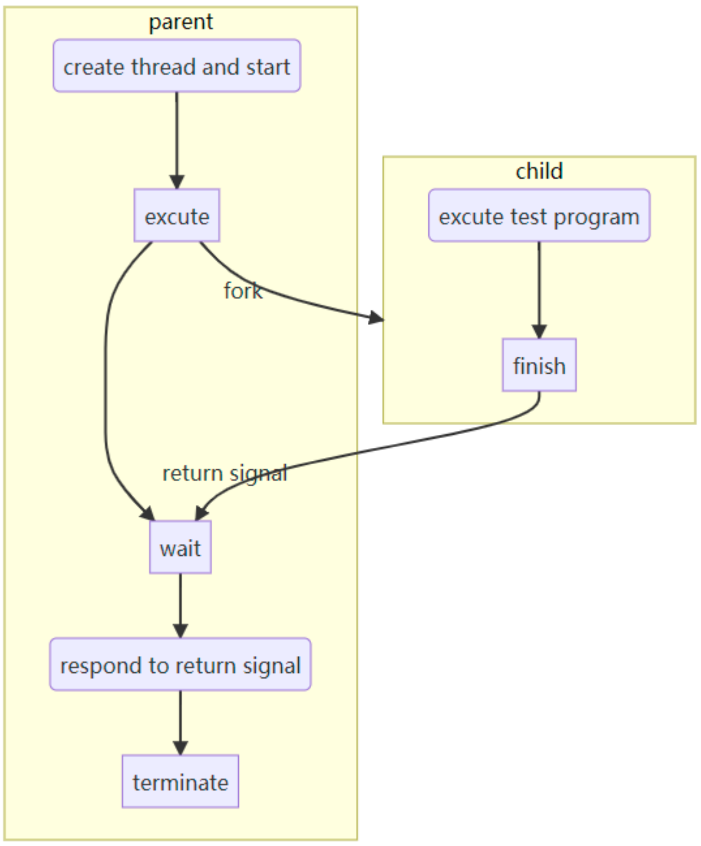
Design of task 1:

- There are mainly four components of my design.
    1. Parent process use system call `fork( )` to fork a child process
        - `fork( )` will return 0 to child process and return the pid of child process to parent, which is the basic key for following program to distinguish between two processes.
    2. Child process's executing test program.
        - implemented by system call `execve( )`
    3. Parent process's waiting for child's return status
        - implemented by `waitpid(pid, &status, WUNTRACED);` (option WUNTRACED enables it to report the return signal even if the child process is stopped)to wait until recieve the messge from specific pid of child process.
    4. Parent process evaluate the corresponding return status.
        - `WIFEXITED(status)` : will be true if child process ends normally, then use `WEXITSTATUS(status)` , to get return status (the parameter of `exit( )`) of child process
        - `WIFSIGNALED(status)` : will be true if child process ends anormally, via `WTERMSIG(status)` , it can get the return status.
        - `WIFISTOPPED(status)` : will be true if process is stopped. Via `WSTOPSIG(status)` , it can get the return status. `WIFCONTINUED(status)` : will be true if process has continued to run.



Design of task 2:

- The basic architecture is similar to task 1, the difference is that
    - it should create a kernel thread and run specific function
        - Via `kthread_create` and `wake_up_process`
    - its function should be implemented by using the externed functions in kernel(but not the api of syscalls), which should be exported and compiled the kernel again.
        - Reach to the corresponding source file of linux kernel to add `EXPORT_SYMBOL( )` statement to certain function. Then `extern` them in the program (also include some `structs` involved)
    - the whole program should be compiled as a kernel object and be inserted.
- Therefore the design steps are:
    1. Preparation: Finding the corresponding functions in the kernel source files, export and compile them.
    2. Parent process: create a thread, start it and fork a child process.
    3. Child process: execute the test program and return status.
    4. Parent process waits then respond to the return status, print the return status of child process

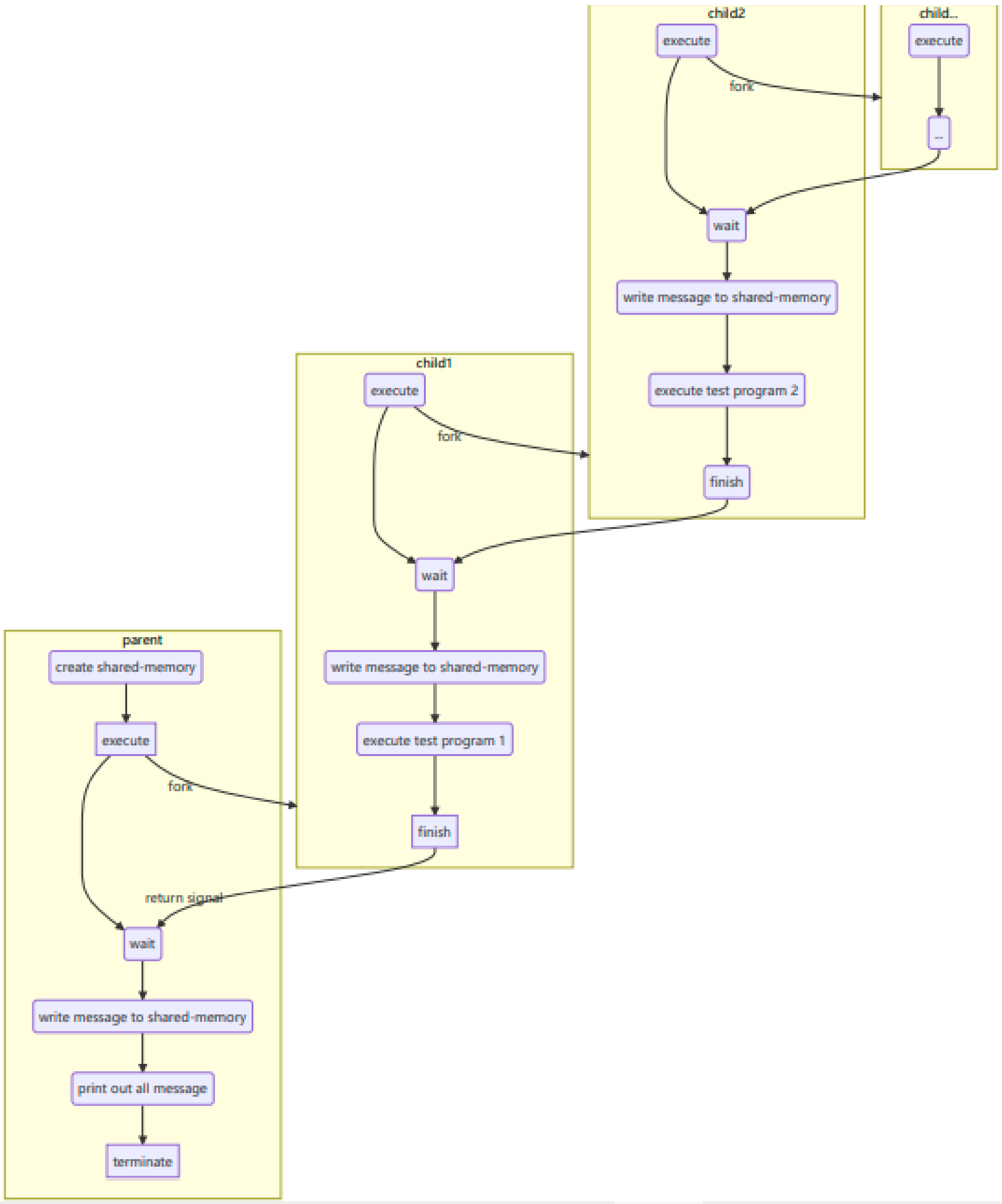


Design of Bonus

- There are two key points for designing `Bonus`:
    - Share-Memory Strategy (as mention above) : alloctate a share memory, which can be used by all the processes.
    - Recursion: Being used to create the link of processes.
- The main steps of `Bonus` are:
    1. Create a shared-memory by

- With the help of `mmap()` , which will creates a new mapping in the virtual address space. Other process created later can access this space.

2. Root(Parent) process: use `fork()` to fork a child proccess, and wait return status.
3. Child process: write its pid to shared-memory, execute the test program, call a function to create its child process then wait for its child.
4. (Keep calling recursion function until all the input test-program are assigned to child processes)
5. Parent process responds to the return status, and write the return status to share memory
6. After all the child processes return, Root (Parent) process is in charge of printing out the message in shared memory to terminal. Finally, the mapping would be released by `munmap()`



- The arrangement of shared-memory

Parent pid	Child1 status	Child1 pid	Child2 status	Child2 pid	...

Remark: Each block represent 4 bytes memory. In the end, the parent process will retrieve the blocks in order and prints them out to terminal.

Part 2: Environment

- My project adopts the recommended environment:
  - version of OS:

```
qpr@ubuntu:~/Desktop/Assignment_1_119010249$ cat /etc/issue
Ubuntu 16.04.5 LTS \n \l
```

  - version of kernel:

```
qpr@ubuntu:~/Desktop/Assignment_1_119010249$ uname -r
4.10.14
```

  - gcc version:

```
root:/ $ gcc --version
gcc (Ubuntu 5.4.0-6ubuntu1~16.04.12) 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.
```

Part 3: The Steps to Execute My Code

Task 1

- enter into the directory

```
cd Assignment_1_119010249/source/program1
```

- compile the project

```
make
```

- execute the file

```
./program1 ./FILE_NAME
```

Task 2

- enter into the directory

```
cd Assignment_1_119010249/source/program2
```

- login and get higher permission

```
sudo su
```

- compile the project

```
make
```

- Insert and remove the module

```
insmod program2.ko
rmmod program2.ko
```

- check the messge in the kernel mode

```
dmesg | tail -n 10
```

Task 3

- cd into the directory

```
cd Assignment_1_119010249/source/bonus
```

- compile the project

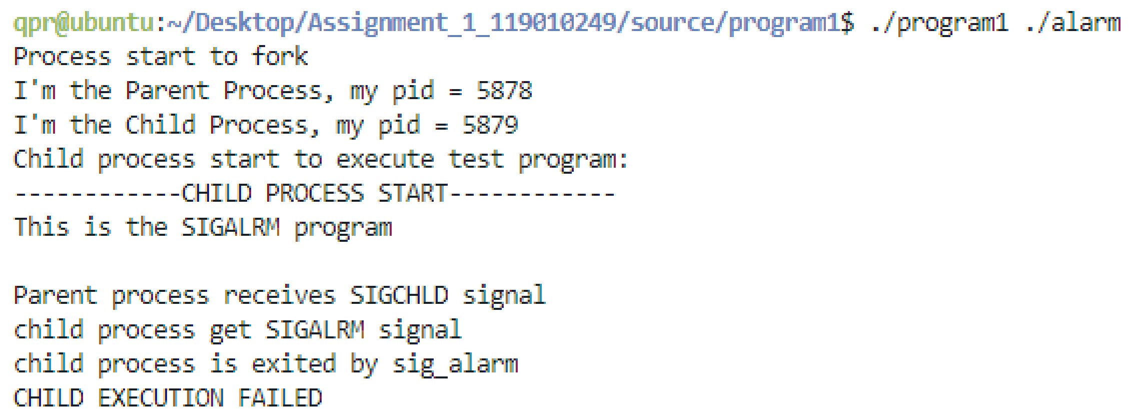
```
make
```

- execute the file

```
./myfork FILE_NAME1 FILE_NAME2
```

Part 4: Screenshot of Program Output

Sample Output of Task 1



```
qpr@ubuntu:~/Desktop/Assignment_1_119010249/source/program1$ ./program1 ./alarm
Process start to fork
I'm the Parent Process, my pid = 5878
I'm the Child Process, my pid = 5879
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGALRM program

Parent process receives SIGCHLD signal
child process get SIGALRM signal
child process is exited by sig_alarm
CHILD EXECUTION FAILED
```

```
qpr@ubuntu:~/Desktop/Assignment_1_119010249/source/program1$ ./program1 ./stop
Process start to fork
I'm the Parent Process, my pid = 10258
I'm the Child Process, my pid = 10259
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
child process stopped
CHILD PROCESS STOPPED
```

```
qpr@ubuntu:~/Desktop/Assignment_1_119010249/source/program1$ ./program1 ./hangup
Process start to fork
I'm the Parent Process, my pid = 5888
I'm the Child Process, my pid = 5889
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGHUP program

Parent process receives SIGCHLD signal
child process get SIGHUP signal
child process is hung up
CHILD EXECUTION FAILED
```

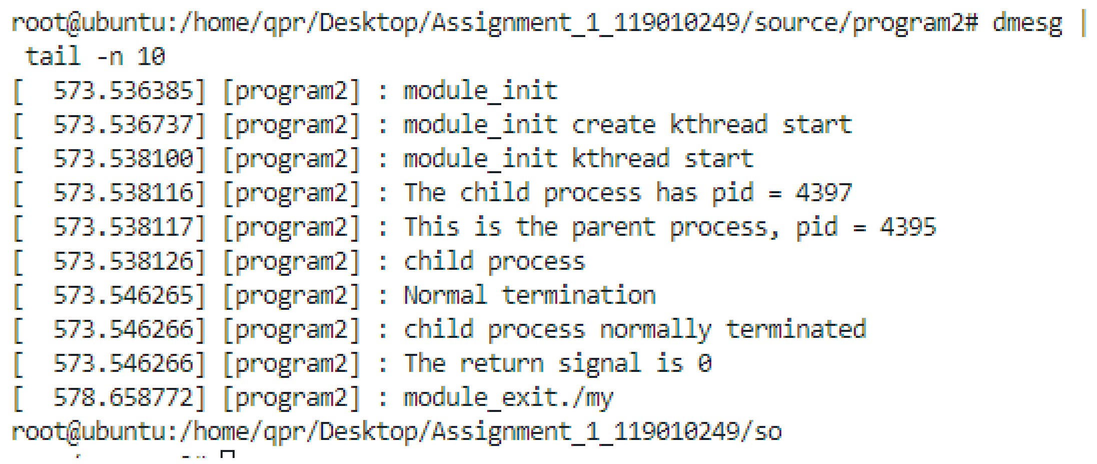
Sample Output of Task 2

- For the `test` program

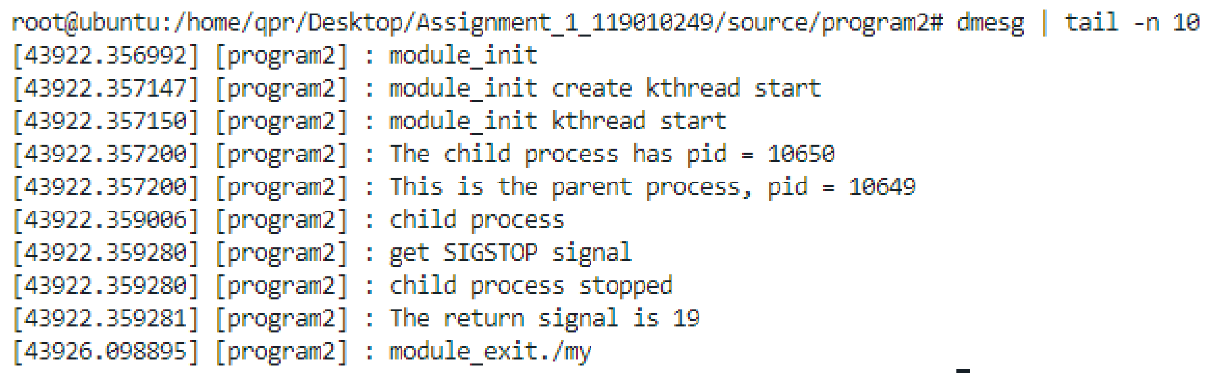
```
root@ubuntu:/home/qpr/Desktop/Assignment_1_119010249/source/program2# dmesg |
 tail -n 10
[  506.309830] [program2] : module_init
[  506.310056] [program2] : module_init create kthread start
[  506.311266] [program2] : module_init kthread start
[  506.311278] [program2] : The child process has pid = 4013
[  506.311278] [program2] : This is the parent process, pid = 4011
[  506.311290] [program2] : child process
[  506.426753] [program2] : get SIGBUS signal
[  506.426754] [program2] : child process met bus error
[  506.426755] [program2] : The return signal is 7
[  510.394887] [program2] : module_exit./my
```

- For from Task 1

```
root@ubuntu:/home/qpr/Desktop/Assignment_1_119010249/source/program2# dmesg |
 tail -n 10
[  378.318243] [program2] : module_init
[  378.318383] [program2] : module_init create kthread start
[  378.318385] [program2] : module_init kthread start
[  378.318393] [program2] : The child process has pid = 3333
[  378.318394] [program2] : This is the parent process, pid = 3332
[  378.318584] [program2] : child process
[  378.451473] [program2] : get SIGTRAP signal
[  378.451474] [program2] : child process was trapped
[  378.451475] [program2] : The return signal is 5
[  382.815387] [program2] : module_exit./my
```

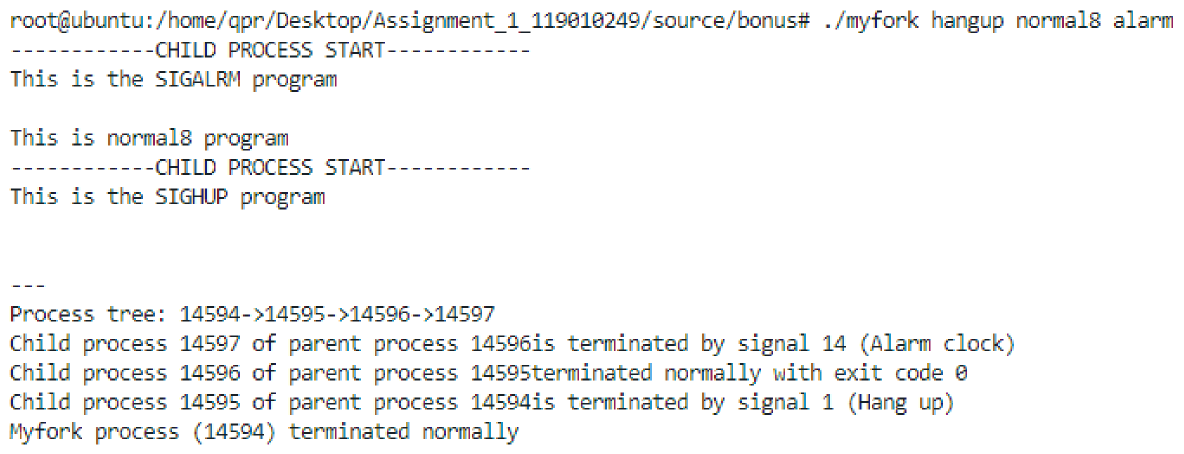


```
root@ubuntu:/home/qpr/Desktop/Assignment_1_119010249/source/program2# dmesg |
 tail -n 10
[  573.536385] [program2] : module_init
[  573.536737] [program2] : module_init create kthread start
[  573.538100] [program2] : module_init kthread start
[  573.538116] [program2] : The child process has pid = 4397
[  573.538117] [program2] : This is the parent process, pid = 4395
[  573.538126] [program2] : child process
[  573.546265] [program2] : Normal termination
[  573.546266] [program2] : child process normally terminated
[  573.546266] [program2] : The return signal is 0
[  578.658772] [program2] : module_exit./my
root@ubuntu:/home/qpr/Desktop/Assignment_1_119010249/so
```



```
root@ubuntu:/home/qpr/Desktop/Assignment_1_119010249/source/program2# dmesg | tail -n 10
[43922.356992] [program2] : module_init
[43922.357147] [program2] : module_init create kthread start
[43922.357150] [program2] : module_init kthread start
[43922.357200] [program2] : The child process has pid = 10650
[43922.357200] [program2] : This is the parent process, pid = 10649
[43922.359006] [program2] : child process
[43922.359280] [program2] : get SIGSTOP signal
[43922.359280] [program2] : child process stopped
[43922.359281] [program2] : The return signal is 19
[43926.098895] [program2] : module_exit./my
```

Sample output of Bonus



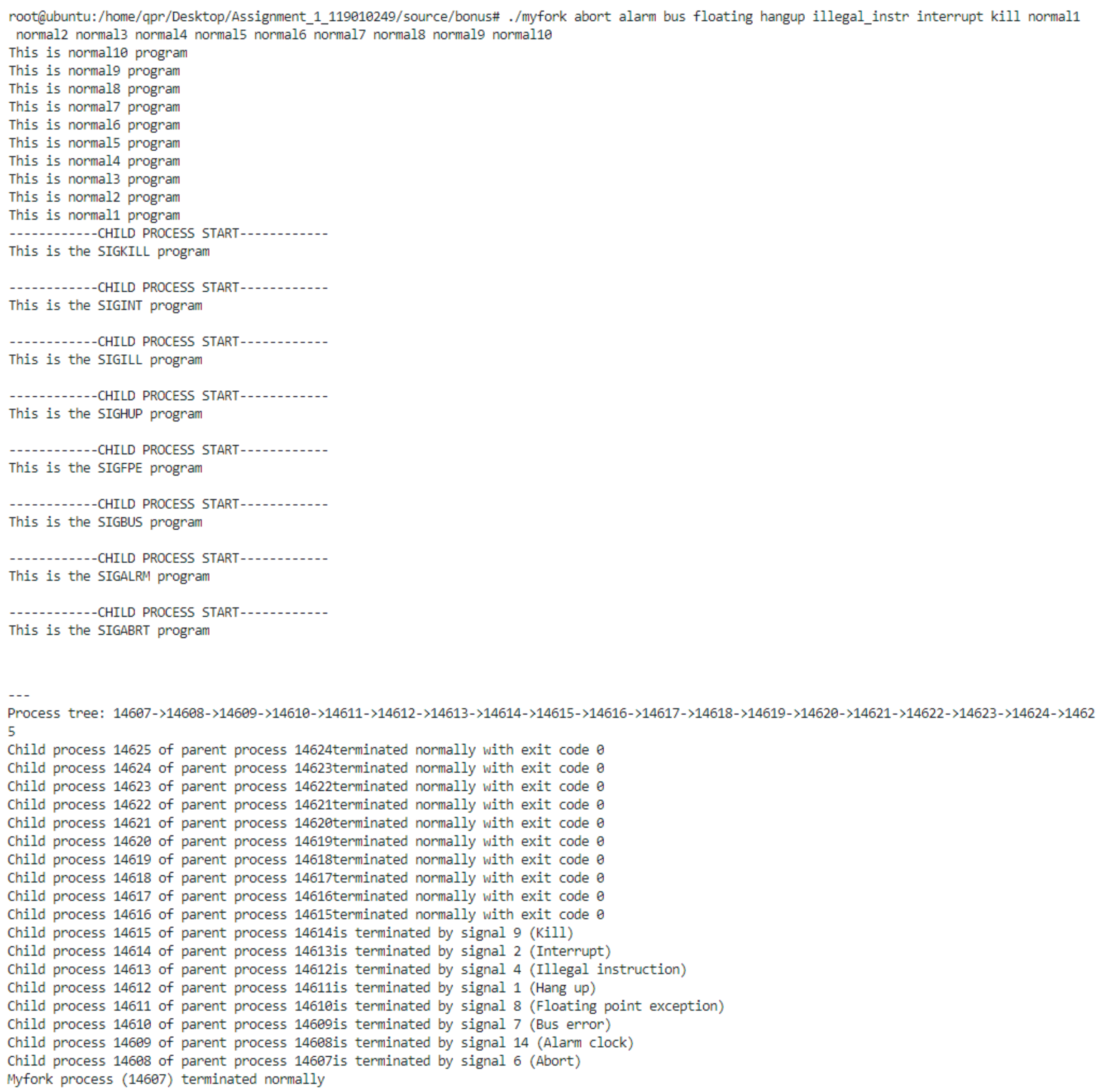
```
root@ubuntu:/home/qpr/Desktop/Assignment_1_119010249/source/bonus# ./myfork hangup normal8 alarm
------------CHILD PROCESS START------------
This is the SIGALRM program

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



---
Process tree: 14594->14595->14596->14597
Child process 14597 of parent process 14596is terminated by signal 14 (Alarm clock)
Child process 14596 of parent process 14595terminated normally with exit code 0
Child process 14595 of parent process 14594is terminated by signal 1 (Hang up)
Myfork process (14594) terminated normally
```

- input all the provided files



```
root@ubuntu:/home/qpr/Desktop/Assignment_1_119010249/source/bonus# ./myfork abort alarm bus floating hangup illegal_instr interrupt kill normal1
 normal2 normal3 normal4 normal5 normal6 normal7 normal8 normal9 normal10
This is normal10 program
This is normal9 program
This is normal8 program
This is normal7 program
This is normal6 program
This is normal5 program
This is normal4 program
This is normal3 program
This is normal2 program
This is normal1 program
------------CHILD PROCESS START------------
This is the SIGKILL program

------------CHILD PROCESS START------------
This is the SIGINT program

------------CHILD PROCESS START------------
This is the SIGILL program

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

------------CHILD PROCESS START------------
This is the SIGFPE program

------------CHILD PROCESS START------------
This is the SIGBUS program

------------CHILD PROCESS START------------
This is the SIGALRM program

------------CHILD PROCESS START------------
This is the SIGABRT program


---
Process tree: 14607->14608->14609->14610->14611->14612->14613->14614->14615->14616->14617->14618->14619->14620->14621->14622->14623->14624->1462
5
Child process 14625 of parent process 14624terminated normally with exit code 0
Child process 14624 of parent process 14623terminated normally with exit code 0
Child process 14623 of parent process 14622terminated normally with exit code 0
Child process 14622 of parent process 14621terminated normally with exit code 0
Child process 14621 of parent process 14620terminated normally with exit code 0
Child process 14620 of parent process 14619terminated normally with exit code 0
Child process 14619 of parent process 14618terminated normally with exit code 0
Child process 14618 of parent process 14617terminated normally with exit code 0
Child process 14617 of parent process 14616terminated normally with exit code 0
Child process 14616 of parent process 14615terminated normally with exit code 0
Child process 14615 of parent process 14614is terminated by signal 9 (Kill)
Child process 14614 of parent process 14613is terminated by signal 2 (Interrupt)
Child process 14613 of parent process 14612is terminated by signal 4 (Illegal instruction)
Child process 14612 of parent process 14611is terminated by signal 1 (Hang up)
Child process 14611 of parent process 14610is terminated by signal 8 (Floating point exception)
Child process 14610 of parent process 14609is terminated by signal 7 (Bus error)
Child process 14609 of parent process 14608is terminated by signal 14 (Alarm clock)
Child process 14608 of parent process 14607is terminated by signal 6 (Abort)
Myfork process (14607) terminated normally
```

Part 5: The Things I Learned from the Tasks

In general, from this project I learned sevaral things about multi-processes programming, different ways for processes to perform interprocess communicate, and some basic knowledge about kernel.

I would like to state what I learned in three main parts:

- **Comprehensive understanding about:**
  - Several systems calls related to multi-processing programming
    - `fork( )`, `execve()`, `wait( )` ...
    - In addition, investigating deeper into the kernel file, I know how the syscalls like `fork( )`, `execve( )`, `wait( )` are implemented by some further functions like `_do_fork( )` and `do_wait( )` ... This process give me a deeper knowledge about system calls. (for example, when I reading the code of `_do_fork( )`, it is found that it calls a function called `copy_process( )`, which will copy the process first)
    - When externing the functions, I wonder what's the difference between `do_fork( )` or `_do_fork( )`, because it is hard to understand why `do_fork( )` just invoke `_do_fork( )` and default setting one argument to 0. After searching the internet, it is shown

    that it is because the updated linux version's `_do_fork( )` supports to pass the parameter `tls`, which is not supported by old version `do_fork( )`. Therefore `_do_fork( )` replace `do_fork( )` in a more general way.
  - Basic knowledge about signals
    - All the tasks have to deal with the return status of child processes, which give me an understanding of several types of signal and their usage. For example, `alarm( )` send a SIGALRM signal to its caller after some time and terminate the process. Also, I learned how the parent process can reviece and evaluate the return status.
  - Kernel Programming
    - I learned about how to program a kernel object, and how to modify the kernel.
- **Design steps of multi-processes programming**
  - Through these three tasks, I get used to Single Instruction Streaming-Multiple Data Stream(SIMD) strategy and learn how to practically write a multi-processes. (use pid to assign part of code to each process)
  - Understanding of process communication:
    1. parent-child communication: child process raise signals and parents use `waitpid( )` to recieve the return status of child
    2. Shared-memory: If not using a shared-memory, all the processes's memory are distributed, which is inconvenient when frequency communication involves. Using shared-memory can benefit while passing some public message.
- **Large amount of details and multi-processes coding skills**

  Including:
  - several things I should pay attention to: e.g. how to deal with wo_flags when using do_wait( )