CSC 3150 Assignment 1 Report

Cai, Zhao 120090679

Introduction

This project implements multi-process programming in both user mode and kernel mode, which corresponds to Program 1 and Program 2, respectively. **Program 1** implements the functionality to fork a child in user mode, execute the test program, and that the parent process waits for the child process to terminate as well as to handle different signals returned by the child process. **Program 2** allows us to modify the kernel and implement the task in Program 1 in kernel mode.

Environment

The environment of running my programs is as follows:

- **Kernel version**: 5.10.147
- **OS version**: Ubuntu 16.04.7 LTS
- **gcc version**: 5.4.0

The original kernel version is 4.4.0-210, and in order to conform to the assignment requirement, I update the kernel as the following steps:

1. Download Linux-5.10.147 source code from kernel archives.
2. Install dependency and development tools by the command:
   `sudo apt-get install libncurses-dev gawk flex bison openssl libssl-dev dkms libelf-dev libudev-dev libpci-dev libiberty-dev autoconf llvm dwarves`

   When I first copied the command from the tutorial file, I got the warning "Unable to locate package". It was due to the extra space between "libudev-" and "dev", and no such dependency exists.
3. Extract the source file to /home/seed/work
4. Copy config from /boot to /home/seed/work/linux-5.10.147
5. Start configuration as the root account by the command `make menuconfig`
6. Build kernel image and modules by the command `make -j$(nproc)`. As it first came out that there's no enough space to build, I enlarged the memory via VirtualBox and the core number, then it worked.
7. Installl kernel modules `make modules_install` and kernel `make install`.
8. Reboot

Program Design

Program 1

Here is the flow chart of Program 1:

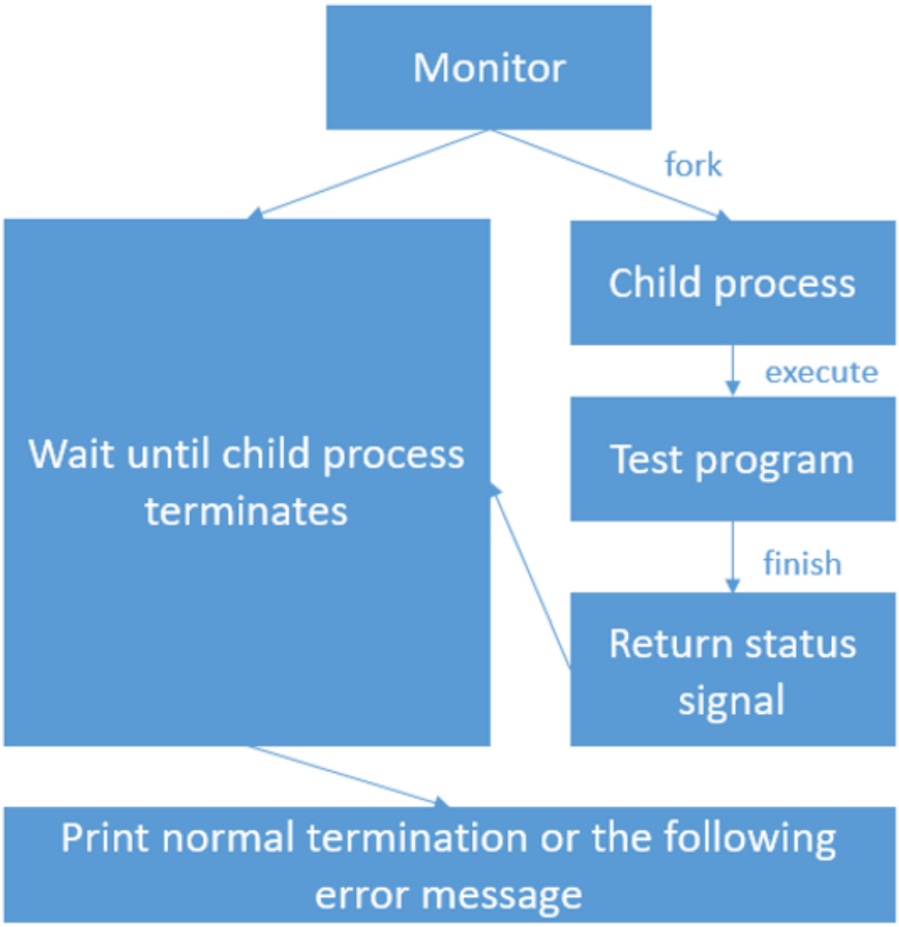

Figure 1.1: Flowchat for Program 1

For This problem, we need to fork the process in user mode. Following the flow chart, we may want to implement in this way:

1. Use `fork()` function to fork a identical children process but with different logical memory.
2. For the child process, we could use `execve()` function to execute the test program given the file name.
3. At the same time for the parent process, it should wait for the child process until it terminates and returns certain siganls back. We could use `wait_pid()` function, or `wait()`, which have the same effect here.
4. To handle the different sent-back signals, we could use `switch-case` statement to specify the given 15 cases.

Output Screenshots:

```
vagrant@csc3150:~/csc3150/program1$ ./program1 ./abort
Process start to fork
I'm the Parent Process, my pid = 12181
I'm the Child Process, my pid = 12182
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
```

```
vagrant@csc3150:~/csc3150/program1$ ./program1 ./illegal_instr
Process start to fork
I'm the Parent Process, my pid = 12584
I'm the Child Process, my pid = 12585
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGILL program

Parent process receives SIGCHLD signal
child process get SIGILL signal
```

```
vagrant@csc3150:~/csc3150/program1$ ./program1 ./normal
Process start to fork
I'm the Parent Process, my pid = 12753
I'm the Child Process, my pid = 12754
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
```

```
vagrant@csc3150:~/csc3150/program1$ ./program1 ./stop
Process start to fork
I'm the Parent Process, my pid = 12909
I'm the Child Process, my pid = 12910
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
```

The "pid" and "ppid" printed are not next to each other, this may due to the execution time of individually executing parent process and child process.

Program 2

Here is the flow chart of Program 2:

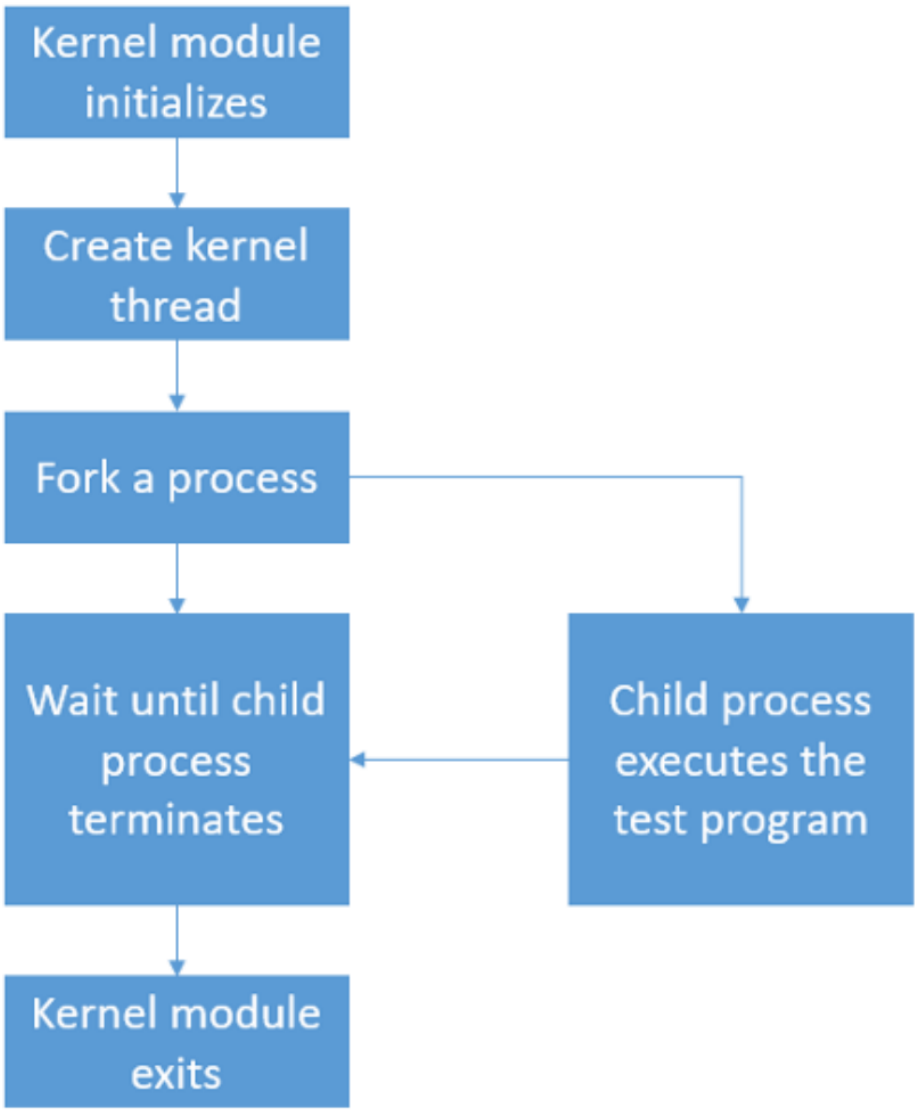

Figure 1.2: Flowchat for Program 2

For this problem, we need to fork the process in the kernel mode. Since Program 2 has the somehow the same idea of Program 1, here I would first have these functions explained:

- `program_init()` function is used to initialize the kernel and create the kernel thread.
- my_exec() is used to test the kernel program, in which the core function is `do_execve()` that execute the .

- `my_wait()`function is used to wait for the child process to terminate and send return certain signal to the parent, in which the core function is `do_wait()`.Within this function, struct `wait_opts` is constructed and passed to `do_wait` as a parameter.
- Instead of do_fork() of previous version, `kernel_clone()` is used to allocate a new process resources, which corresponds to the `fork()` function in user mode.

Compared with Program 1, what we should take care of in the operations in kernel mode is:

- Before we can use functions like `do_wait` and `do`, we should `EXPORT_SYMBOL` in the kernel source file. After that, we shoudl re-compile the kernel as how we did in the update kernel steps.
- As we cannot use the macro *WIFEXITED*, `WEXITSTATUS`, `WIFSIGNALED`, `WTERMSIG` and `WIFSTOPPED`, we should look into how these macros are defined:

```
/* Nonzero if STATUS indicates termination by a signal.  */
#define __WIFSIGNALED(status) \
  (((signed char) (((status) & 0x7f) + 1) >> 1) > 0)

/* Nonzero if STATUS indicates the child is stopped.  */
#define __WIFSTOPPED(status) (((status) & 0xff) == 0x7f)
```

We know that the signal number is the lower senven bits of the return status, and use this definition we could also build the "stop" return signal in this way. We could explore out the relationship and use `switch-case` statement. For example, 4991 for "stop" correspond to "19", and we can directly use 19 in the "case".

Output Screenshots:

```
[191848.643815] [program2] : Module_init Cai_Zhao 120090679
[191848.686200] [program2] : module_init create kthread start
[191848.729044] [program2] : module_init kthread start
[191848.755353] [program2] : The child process has pid = 13433
[191848.755458] [program2] : child process
[191848.786420] [program2] : This is the parent process, pid = 13432
[191848.935186] [program2] : get SIGABRT signal
[191848.951560] [program2] : child process aborted
[191848.970152] [program2] : The return signal is 6
[191851.362811] [program2] : Module_exit
```

```
[192197.972000] [program2] : Module_init Cai_Zhao 120090679
[192198.009160] [program2] : module_init create kthread start
[192198.040803] [program2] : module_init kthread start
[192198.056879] [program2] : The child process has pid = 16009
[192198.056912] [program2] : child process
[192198.063555] [program2] : This is the parent process, pid = 16008
[192198.206638] [program2] : get SIGILL signal
[192198.229060] [program2] : child process has illegal instruction error
[192198.269420] [program2] : The return signal is 4
[192199.443649] [program2] : Module_exit
```

```
[192387.844258] [program2] : Module_init Cai_Zhao 120090679
[192387.878203] [program2] : module_init create kthread start
[192387.897911] [program2] : module_init kthread start
[192387.917398] [program2] : The child process has pid = 17667
[192387.917415] [program2] : child process
[192387.944544] [program2] : This is the parent process, pid = 17666
[192387.981794] [program2] : child process exit normally
[192388.017027] [program2] : The return signal is 0
[192389.444237] [program2] : Module_exit
```

```
[192532.868177] [program2] : Module_init Cai_Zhao 120090679
[192532.901287] [program2] : module_init create kthread start
[192532.935670] [program2] : module_init kthread start
[192532.955027] [program2] : The child process has pid = 19356
[192532.955079] [program2] : child process
[192532.955852] [program2] : This is the parent process, pid = 19355
[192532.965665] [program2] : get SIGSTOP signal
[192532.983047] [program2] : child process stoped
[192533.005804] [program2] : The return signal is 19
[192534.595141] [program2] : Module_exit
```