Report for assignment 1

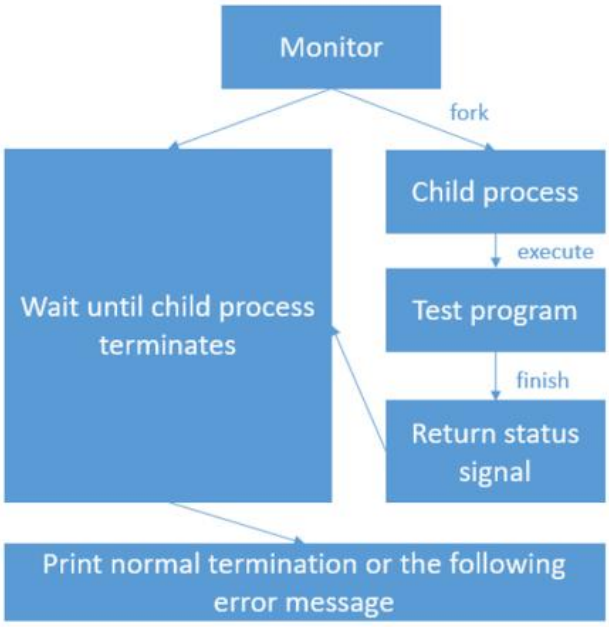
A. Name and Student ID

Name: 施若兰（Shi Ruolan）

Student ID: 120090757

B. How did you design your program?

1. **Task 1(Under User mode):**



a) **Use function() to create a child process and use a variable "pod" to get the return value of fork()**

```
pid = fork();
```

b) **Use the value of variable "pid" to distinguish the parent process and child process, and to determine whether the creation of a child process was successful.**

- **fork() returns -1, the creation of a child process was unsuccessful.**
- **fork() returns a zero to the newly created child process.**
- **fork() returns a positive value, the process ID of the child process,**

to the parent.

c) For child process:

- It calls function getpid() to get the pid of this child process.
- it calls function execve() to execute an executable file which passes in as the first parameter of the program.

```
execve(arg[0], arg, NULL);
```

d) For parent program:

- It waits until its child process terminated by waitpid().
- I choose WUNTRACED to be the argument for parameter options. The advantage of WUNTRACED is that it can suspends the execution of the calling process until one of the processes in the waiting collection becomes terminated or is stopped. The returned PID is the PID of the terminated or stopped child process that caused the return. The default behavior is to return only terminated child processes. This option is useful when you want to check for terminated and stopped child processes.
- Since signal returned is all defined in < signal.h> according to order, I establish an array to emulate the signal.

```
char *siganls[32] = {
    "zero",    "SIGHUP",    "SIGINT",
    "SIGQUIT", "SIGILL",    "SIGTRAP",
    "SIGABRT", "SIGBUS",    "SIGFPE",
    "SIGKILL", "SIGUSR1",   "SIGSEGV",
    "SIGUSR2", "SIGPIPE",   "SIGALRM",
    "SIGTERM", "SIGSTKFLT", "SIGCHLD",
    "SIGCONT", "SIGSTOP",   "SIGTSTP",
    "SIGTTIN", "SIGTTOU",   "SIGURG",
    "SIGXCPU", "SIGXFSZ",   "SIGVTALRM",
    "SIGPROF", "SIGWINCH",  "SIGIO",
    "SIGPWR",  "SIGSYS"
};
```

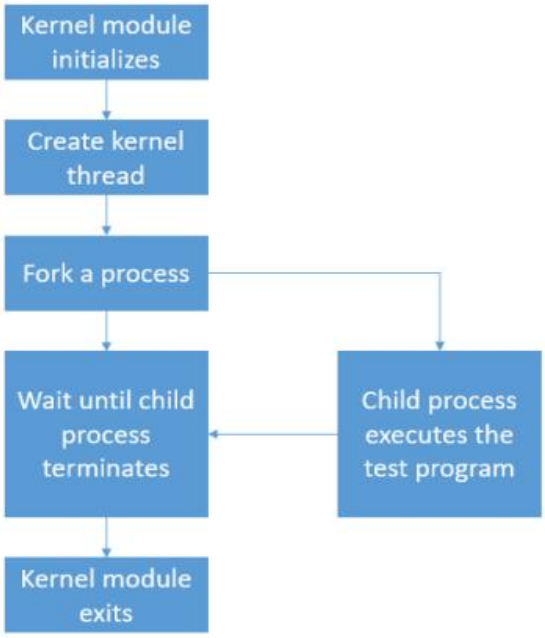
- It uses functions int WIFEXITED (int status), int WIFSIGNALED (int status), int WIFSTOPPED (int status) to check the return status signal of child process.
  - WIFEXITED (int status) == 1: normal termination
  - WIFSTOPPED (int status) == 1: SIGSTOP signal causes

process stopped

- WIFSIGNALED(int status)==1: signal causes process termination

e) Print normal termination or the error message with causing signal.

2. Task 2 (Under kernel mode)



Procedure:

Modify kernel -> Compile Kernel -> Install and Launch into New Kernel -> Code the Module -> Compile and Install the Module- > Uninstall the module

a) Set up environment

b) Define a function static int __init program2_init(void) to initialize module as well as contains the following work.

c) Use function kernel_thread() to create a kernel thread, which serves as the parent process.

d) The parent process fork() a kernel child using my_fork(), a

function defined by me.

e) my_fork() mainly serves as fork() in user mode. my_fork() will fork() a child process within it using kernel_thread().

f) In the child process, it will execute the function my_execve() which is defined by me. And my_execve() mainly serves as execve() in user mode. Within my_execve(), it will call do_execve() to execute the wanting file in the path we set(/tmp/test for this assignment). And to pass the filename of file that we need to execute, we call struct filename *getname_kernel(const char *filename);

g) The parent will wait until the child process terminates or stopped using my_wait() defined by me. my_wait() mainly serves as waitpid() in user mode.

- In my_wait(), it calls do_wait() to get the exit status of child process. Since signal returned is all defined in < signal.h> according to order, I define a function to get the signal name.

```
static const char *signal_name(long signal)
{
    char *siganls[32] = { "zero",      "SIGHUP",  "SIGINT",    "SIGQUIT",
                          "SIGILL",    "SIGTRAP", "SIGABRT",   "SIGBUS",
                          "SIGFPE",    "SIGKILL", "SIGUSR1",   "SIGSEGV",
                          "SIGUSR2",   "SIGPIPE", "SIGALRM",   "SIGTERM",
                          "SIGSTKFLT", "SIGCHLD", "SIGCONT",   "SIGSTOP",
                          "SIGTSTP",   "SIGTTIN", "SIGTTOU",   "SIGURG",
                          "SIGXCPU",   "SIGXFSZ", "SIGVTALRM", "SIGPROF",
                          "SIGWINCH",  "SIGIO",   "SIGPWR",    "SIGSYS" };
    return siganls[signal];
}
```

h) Function static void __exit program2_exit(void) will be running when remove the inserted module.

TIPS1: Relation between modes and functions

User mode	kernel mode
fork()	kernel_thread()

wait() **my_wait()(with do_wait() inside)**

execve() **my_ execve()(with do_ execve())**

inside)

TIPS2:

Use self-defined macro to mimic WIFEXITED (int status), int WIFSIGNALED (int status), int WIFSTOPPED (int status) to check signal status.

```
#define __WEXITSTATUS(status) (((status)&0xff00) >> 8)

/* If WIFSIGNALED(STATUS), the terminating signal.  */
#define __WTERMSIG(status) ((status)&0x7f)

/* If WIFSTOPPED(STATUS), the signal that stopped the child.  */
#define __WSTOPSIG(status) __WEXITSTATUS(status)

/* Nonzero if STATUS indicates normal termination.  */
#define __WIFEXITED(status) (__WTERMSIG(status) == 0)

/* 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)
```

C: How did you set up your development environment, including how to compile model?

Modify the kernel

- **Add EXPORT_SYMBOL() to the function we need to use in program2.c**
- **If there is "static" prefix for the function, delete it.**
- **Repeat part of the process of compiling kernel**

(Starts from $make –j$(nproc) to reboot)

Compile the new kernel

1. **Check what kernel version it is: uname -r**

   -> linux-4.4.x

   -> need to compile kernel whose version is linux-5.10.x

2. **download the mirror of new kernel:**

   wget https://mirror.tuna.tsinghua.edu.cn/kernel/v5.x/linux-5.10.5

3. **Install Dependency and development tools:**

   sudo apt-get install libncurses-dev gawk flex bison openssl libssl-dev dkms libelf-dev libudev-dev libpci-dev libiberty-dev autoconf llvm dwarves

4. **Extract the source file to /home/vagrant/**

   cp linux-5.10.5.tar.gz /home/ vagrant/

   sudo tar xvf /home/ vagrant/linux-5.10.5.tar.gz

   cp /boot/config /home/ vagrant/linux-5.10.5/.config

5. **Login root account and go to kernel source directory:**

   sudo su

   cd /home/ vagrant/linux-5.10.5/

6. **Clean previous setting and start configuration:**

   $make mrproper

   $make clean

   $make menuconfig (save the .config file and exit)

7. **Build kernel Image and modules:**

   $make –j$(nproc)

8. **Install kernel modules:**

   $make modules_install

9. **Install kernel:**

$make install

10. **Reboot to load new kernel:** **$reboot**

D. Screenshot of program output

Task 1

1.

```
vagrant@csc3150:~/csc3150/program1$ ./program1 ./abort
Process start to fork
I'm the Parent Process, my pid = 10528
I'm the Child Process, my pid = 10529
------------CHILD PROCESS START------------
This is the SIGABRT program

Parent process receives SIGCHLD signal
Child Process get SIGABRT Signal.
```

2.

```
vagrant@csc3150:~/csc3150/program1$ ./program1 ./alarm
Process start to fork
I'm the Parent Process, my pid = 10557
I'm the Child Process, my pid = 10558
------------CHILD PROCESS START------------
This is the SIGALRM program

Parent process receives SIGCHLD signal
Child Process get SIGALRM Signal.
```

3.

```
vagrant@csc3150:~/csc3150/program1$ ./program1 ./bus
Process start to fork
I'm the Child Process, my pid = 10624
I'm the Parent Process, my pid = 10623
------------CHILD PROCESS START------------
This is the SIGBUS program

Parent process receives SIGCHLD signal
Child Process get SIGBUS Signal.
```

4.

```
vagrant@csc3150:~/csc3150/program1$ ./program1 ./floating
Process start to fork
I'm the Parent Process, my pid = 10679
I'm the Child Process, my pid = 10680
------------CHILD PROCESS START------------
This is the SIGFPE program

Parent process receives SIGCHLD signal
Child Process get SIGFPE Signal.
```

5.

```
vagrant@csc3150:~/csc3150/program1$ ./program1 ./hangup
Process start to fork
I'm the Parent Process, my pid = 10733
I'm the Child Process, my pid = 10734
------------CHILD PROCESS START------------
This is the SIGHUP program

Parent process receives SIGCHLD signal
Child Process get SIGHUP Signal.
```

6.

```
vagrant@csc3150:~/csc3150/program1$ ./program1 ./illegal_instr
Process start to fork
I'm the Parent Process, my pid = 10797
I'm the Child Process, my pid = 10798
------------CHILD PROCESS START------------
This is the SIGILL program

Parent process receives SIGCHLD signal
Child Process get SIGILL Signal.
```

7.

```
vagrant@csc3150:~/csc3150/program1$ ./program1 ./interrupt
Process start to fork
I'm the Child Process, my pid = 10828
I'm the Parent Process, my pid = 10827
------------CHILD PROCESS START------------
This is the SIGINT program

Parent process receives SIGCHLD signal
Child Process get SIGINT Signal.
```

8.

```
vagrant@csc3150:~/csc3150/program1$ ./program1 ./kill
Process start to fork
I'm the Parent Process, my pid = 10867
I'm the Child Process, my pid = 10868
------------CHILD PROCESS START------------
This is the SIGKILL program

Parent process receives SIGCHLD signal
Child Process get SIGKILL Signal.
```

9.

```
vagrant@csc3150:~/csc3150/program1$ ./program1 ./normal
Process start to fork
I'm the Parent Process, my pid = 10934
I'm the Child Process, my pid = 10935
------------CHILD PROCESS START------------
This is the normal program

------------CHILD PROCESS END------------
Parent process receives SIGCHLD signal
Normal termination with EXIT STATUS = 0
```

10.

```
vagrant@csc3150:~/csc3150/program1$ ./program1 ./pipe
Process start to fork
I'm the Parent Process, my pid = 10983
I'm the Child Process, my pid = 10984
------------CHILD PROCESS START------------
This is the SIGPIPE program

Parent process receives SIGCHLD signal
Child Process get SIGPIPE Signal.
```

11.

```
vagrant@csc3150:~/csc3150/program1$ ./program1 ./quit
Process start to fork
I'm the Parent Process, my pid = 11020
I'm the Child Process, my pid = 11021
------------CHILD PROCESS START------------
This is the SIGQUIT program

Parent process receives SIGCHLD signal
Child Process get SIGQUIT Signal.
```

12.

```
vagrant@csc3150:~/csc3150/program1$ ./program1 ./segment_fault
Process start to fork
I'm the Child Process, my pid = 11078
I'm the Parent Process, my pid = 11077
------------CHILD PROCESS START------------
This is the SIGSEGV program

Parent process receives SIGCHLD signal
Child Process get SIGSEGV Signal.
```

13.

```
vagrant@csc3150:~/csc3150/program1$ ./program1 ./stop
Process start to fork
I'm the Parent Process, my pid = 11137
I'm the Child Process, my pid = 11138
------------CHILD PROCESS START------------
This is the SIGSTOP program

Parent process receives SIGCHLD signal
Child Process Was Stoppped By SIGSTOP.
```

14.

```
vagrant@csc3150:~/csc3150/program1$ ./program1 ./terminate
Process start to fork
I'm the Parent Process, my pid = 11183
I'm the Child Process, my pid = 11184
------------CHILD PROCESS START------------
This is the SIGTERM program

Parent process receives SIGCHLD signal
Child Process get SIGTERM Signal.
```

15.

```
vagrant@csc3150:~/csc3150/program1$ ./program1 ./trap
Process start to fork
I'm the Parent Process, my pid = 11243
I'm the Child Process, my pid = 11244
------------CHILD PROCESS START------------
This is the SIGTRAP program

Parent process receives SIGCHLD signal
Child Process get SIGTRAP Signal.
```

Task 2:

1. SIGBUS

```
root@csc3150:/home/vagrant/csc3150/program2# make
make -C /lib/modules/5.10.5/build M=/home/vagrant/csc3150/program2 modules
make[1]: Entering directory '/home/seed/work/linux-5.10.5'
make[1]: Leaving directory '/home/seed/work/linux-5.10.5'
root@csc3150:/home/vagrant/csc3150/program2# insmod program2.ko
root@csc3150:/home/vagrant/csc3150/program2# rmmod program2.ko
root@csc3150:/home/vagrant/csc3150/program2# dmesg | grep program2
[17616.328587] [program2] : Module_exit
[17630.568389] [program2] : module_init {施若兰} {120090757}
[17630.568392] [program2] : module_init create kthread start
[17630.570974] [program2] : module_init kthread start
[17630.573437] [program2] : The child process has pid = 14251
[17630.573440] [program2] : This is the parent process, pid = 14250
[17630.573441] [program2] : child process
[17630.744725] [program2] : get SIGBUS signal
[17630.744727] [program2] : child process terminated
[17630.744728] [program2] : The return signal is 7
[17633.029335] [program2] : Module_exit
```

2. SIGKILL

```
root@csc3150:/home/vagrant/csc3150/program2# gcc -o test test.c
root@csc3150:/home/vagrant/csc3150/program2# make
make -C /lib/modules/5.10.5/build M=/home/vagrant/csc3150/program2 modules
make[1]: Entering directory '/home/seed/work/linux-5.10.5'
make[1]: Leaving directory '/home/seed/work/linux-5.10.5'
root@csc3150:/home/vagrant/csc3150/program2# insmod program2.ko
root@csc3150:/home/vagrant/csc3150/program2# rmmod program2.ko
root@csc3150:/home/vagrant/csc3150/program2# dmesg | grep program2
[17752.988269] [program2] : module_init {施若兰} {120090757}
[17752.988271] [program2] : module_init create kthread start
[17752.988879] [program2] : module_init kthread start
[17752.989392] [program2] : The child process has pid = 14836
[17752.989394] [program2] : This is the parent process, pid = 14835
[17752.989394] [program2] : child process
[17752.994440] [program2] : get SIGKILL signal
[17752.994441] [program2] : child process terminated
[17752.994442] [program2] : The return signal is 9
[17756.258352] [program2] : Module_exit
```

3. Normal

```
root@csc3150:/home/vagrant/csc3150/program2# make
make -C /lib/modules/5.10.5/build M=/home/vagrant/csc3150/program2 modules
make[1]: Entering directory '/home/seed/work/linux-5.10.5'
^[[Amake[1]: Leaving directory '/home/seed/work/linux-5.10.5'
root@csc3150:/home/vagrant/csc3150/program2# insmod program2.ko
root@csc3150:/home/vagrant/csc3150/program2# rmmod program2.ko
root@csc3150:/home/vagrant/csc3150/program2# dmesg | grep program2
[17871.526002] [program2] : module_init {施若兰} {120090757}
[17871.526005] [program2] : module_init create kthread start
[17871.526700] [program2] : module_init kthread start
[17871.527043] [program2] : The child process has pid = 15328
[17871.527045] [program2] : This is the parent process, pid = 15327
[17871.527046] [program2] : child process
[17875.006377] [program2] : Module_exit
```

4. SIGTRAP

```
root@csc3150:/home/vagrant/csc3150/program2# gcc -o test test.c
root@csc3150:/home/vagrant/csc3150/program2# make
make -C /lib/modules/5.10.5/build M=/home/vagrant/csc3150/program2 modules
make[1]: Entering directory '/home/seed/work/linux-5.10.5'
make[1]: Leaving directory '/home/seed/work/linux-5.10.5'
root@csc3150:/home/vagrant/csc3150/program2# insmod program2.ko
root@csc3150:/home/vagrant/csc3150/program2# rmmod program2.ko
root@csc3150:/home/vagrant/csc3150/program2# dmesg | grep program2
[17986.864318] [program2] : module_init {施若兰} {120090757}
[17986.920907] [program2] : module_init create kthread start
[17986.967519] [program2] : module_init kthread start
[17986.999270] [program2] : The child process has pid = 15741
[17987.011872] [program2] : This is the parent process, pid = 15740
[17987.049724] [program2] : child process
[17987.187571] [program2] : get SIGTRAP signal
[17987.212841] [program2] : child process terminated
[17987.255124] [program2] : The return signal is 5
[17989.900551] [program2] : Module_exit
```

E. What did learn from the tasks?

1. **I learned the concrete difference between user mode and kernel mode. The User mode is normal mode where the process has limited access. While the Kernel mode is the privileged mode where the process has unrestricted access to system resources like hardware, memory, etc.**
2. **I learned how to compile a new kernel based on the configuration of old kernel.**
3. **I learned the usage of kernel modules and how powerful they are.**
4. **I learned how to write my own kernel module based on the need and insert as well as remove it to the kernel.**
5. **I gained more knowledges on how parent and child process work as well as the signal transmission from child process to parent process.**
6. **I learned how wait() and do_wait() function deal with signal and its transmission.**
7. **I learned how execve() and do_execve() function execute a file.**
8. **I understood better the difference between user mode function and kernel mode function. Through the reading of some kernel function source code, I had better an understanding of how they perform their own duties.**