# CSC3150 Assignment 1 Report

Jiarui Chen 120090361

2022.10.09

# 1. Development Environment

Linux Distribution

```
root@csc3150:/home/vagrant/csc3150# cat /etc/issue
Ubuntu 16.04.7 LTS \n \l
```

Linux Kernel Version

```
root@csc3150:/home/vagrant/csc3150# uname -r
5.10.147
```

GCC Version

```
gcc version 5.4.0 20160609 (Ubuntu 5.4.0-6ubuntu1~16.04.12)
```

## 2. Program Design

### 2.1. Task 1

We design the following parts to implement program 1:

i. Use *fork* function to create a child process and returns a *pid*.

ii. Use the value of *pid* to distinguish the child process and parent process.

iii. In child process, use *execve* function to execute test program.

iv. In parent process, use *waitpid* function to get the return signal from child process. We set the third parameter as WUNTRACED to wait child process to terminate or stop.

v. After child process terminated or stopped, we use following functions to analyze the signal *status*:

   a) *WIFEXITED*, which is used to detect whether child process terminates normally. If it is, output the signal *WEXITSTATUS(status)*.

   b) *WIFSIGNALED*, which is used to detect the abnormal terminal signals. The output signal is *WTERMSIG(status)*.

   c) *WIFSTOPPED*, which is used to detect the stop signal.

## 2.2. Task 2

We design the following parts to implement program 2:

i. To access the struct *wait_opts*, we need to copy the declaration of *wait_opts* in Linux kernel source code.

ii. To access the kernel functions *kernel_clone*, *do_wait*, *do_exec* and *getname_kernel*, we need to add “EXPORT_SYMBOL(func_name)” expressions to Linux kernel source file, and add “extern” declarations in program2.c.

iii. In *program2_init*, create a kernel thread to run *my_fork* function by creating a task struct *task* by *kthread_create*. If the thread is ok, use *wake_up_process* to wake up new thread.

iv. In *my_fork* function, create a *kernel_clone_args* struct *kargs*. We set *kargs.exit_signal* to *SIGCHLD*, and set *kargs.stack* to *&my_execve* in order to run *my_execve* function. Use *kernel_clone* to create a child process which execute the test program and returns *pid*.

v. In *my_execve* function, use *getname_kernel* to get filename and use *do_execve* to execute test program.

vi. In *my_wait* function, we create struct *wait_opts*, and set *wait_opts.wo_flags* to *WEXITED|WSTOPPED* to detect the terminate or stop signal from child process. Use *do_wait* function to wait for signal from child process.

vii. In *my_signal_handle*, use *switch* syntax to handle the signal get from *my_wait* function and print out

# 3. Execution

## 3.1. Task 1

We execute program1 with following steps:

i. Get access to program1 folder by cd command: cd $(path)/source/program1

ii. Compile all c files by make command: make

iii. Run program1 executable file by: ./program1 ./$(test file), for example: ./program1 ./normal

iv. Clean the executable file by: make clean

```
root@csc3150:/home/vagrant/csc3150/Assignment_1_120090361# cd source/program1
root@csc3150:/home/vagrant/csc3150/Assignment_1_120090361/source/program1# make
cc -o abort abort.c
cc -o alarm alarm.c
cc -o bus bus.c
cc -o floating floating.c
cc -o hangup hangup.c
cc -o illegal_instr illegal_instr.c
cc -o interrupt interrupt.c
cc -o kill kill.c
cc -o normal normal.c
cc -o pipe pipe.c
cc -o program1 program1.c
cc -o quit quit.c
cc -o segment_fault segment_fault.c
cc -o stop stop.c
cc -o terminate terminate.c
cc -o trap trap.c
root@csc3150:/home/vagrant/csc3150/Assignment_1_120090361/source/program1# ./program1 ./normal
Process start to fork
I'm the Parent Process, my pid = 11643
I'm the Child Process, my pid = 11644
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
root@csc3150:/home/vagrant/csc3150/Assignment_1_120090361/source/program1# make clean
rm abort alarm bus floating hangup illegal_instr interrupt kill normal pipe program1 quit segment_fault stop terminate trap
```

**Example execution of program 1**

## 3.2. Task 2

We execute program2 with following steps:

i. Get access to program2 folder by cd command: cd $(path)/source/program2

ii. Compile kernel object by make command: make

iii. Enter root mode by: sudo su

iv. Insert kernel object by: insmod program2.ko

v. Remove kernel object by: rmmod program2.ko

vi. Print kernel log by: dmesg | tail -n *x* (last *x* lines of log)

vii. Clean the kernel files by: make clean

```
root@csc3150:/home/vagrant/csc3150/Assignment_1_120090361# cd source/program2
root@csc3150:/home/vagrant/csc3150/Assignment_1_120090361/source/program2# make
make -C /lib/modules/5.10.147/build M=/home/vagrant/csc3150/Assignment_1_120090361/source/program2 modules
make[1]: Entering directory '/home/seed/work/linux-5.10.147'
  CC [M]  /home/vagrant/csc3150/Assignment_1_120090361/source/program2/program2.o
  MODPOST /home/vagrant/csc3150/Assignment_1_120090361/source/program2/Module.symvers
  CC [M]  /home/vagrant/csc3150/Assignment_1_120090361/source/program2/program2.mod.o
  LD [M]  /home/vagrant/csc3150/Assignment_1_120090361/source/program2/program2.ko
make[1]: Leaving directory '/home/seed/work/linux-5.10.147'
root@csc3150:/home/vagrant/csc3150/Assignment_1_120090361/source/program2# insmod program2.ko
root@csc3150:/home/vagrant/csc3150/Assignment_1_120090361/source/program2# rmmod program2.ko
root@csc3150:/home/vagrant/csc3150/Assignment_1_120090361/source/program2# dmesg | tail -n 10
[65076.404800] [program2] : Module_init {Jiarui Chen} {120090361}
[65076.405798] [program2] : Module_init create kthread start
[65076.406879] [program2] : Module_init kthread start
[65076.432192] [program2] : The child process has pid = 24145
[65076.432977] [program2] : This is the parent process, pid = 24144
[65076.433800] [program2] : Child process
[65076.468157] [program2] : Get SIGSTOP signal
[65076.474591] [program2] : Child process stopped
[65076.521590] [program2] : The return signal is 19
[65081.516982] [program2] : Module_exit
root@csc3150:/home/vagrant/csc3150/Assignment_1_120090361/source/program2# make clean
make -C /lib/modules/5.10.147/build M=/home/vagrant/csc3150/Assignment_1_120090361/source/program2 clean
make[1]: Entering directory '/home/seed/work/linux-5.10.147'
  CLEAN   /home/vagrant/csc3150/Assignment_1_120090361/source/program2/Module.symvers
make[1]: Leaving directory '/home/seed/work/linux-5.10.147'
```

**Example execution of program 2**

**(Since it is already in root mode, we don't need to do sudo su)**

# 4. Sample Output

## 4.1. Task 1

### i. The output of normal termination

```
root@csc3150:/home/vagrant/csc3150/Assignment_1_120090361/source/program1# ./program1 ./normal
Process start to fork
I'm the Parent Process, my pid = 14453
I'm the Child Process, my pid = 14454
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
```

### ii. The output of receiving SIGTERM

```
root@csc3150:/home/vagrant/csc3150/Assignment_1_120090361/source/program1# ./program1 ./terminate
Process start to fork
I'm the Parent Process, my pid = 14573
I'm the Child Process, my pid = 14574
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGTERM program

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

### iii. The output of being stopped

```
root@csc3150:/home/vagrant/csc3150/Assignment_1_120090361/source/program1# ./program1 ./stop
Process start to fork
I'm the Parent Process, my pid = 14679
I'm the Child Process, my pid = 14680
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
```

## 4.2. Task 2

### i. The output of normal termination

```
root@csc3150:/home/vagrant/csc3150/Assignment_1_120090361/source/program2# dmesg | tail -n 10
[65213.508772] [program2] : Module_init {Jiarui Chen} {120090361}
[65213.516131] [program2] : Module_init create kthread start
[65213.516751] [program2] : Module_init kthread start
[65213.558138] [program2] : The child process has pid = 24961
[65213.558877] [program2] : This is the parent process, pid = 24960
[65213.561047] [program2] : Child process
[65218.563390] [program2] : Normal termination
[65218.599989] [program2] : Child process stopped
[65218.647828] [program2] : The return signal is 100
[65220.139835] [program2] : Module_exit
```

### ii. The output of receiving SIGTERM

```
root@csc3150:/home/vagrant/csc3150/Assignment_1_120090361/source/program2# dmesg | tail -n 10
[64928.987926] [program2] : Module_init {Jiarui Chen} {120090361}
[64928.995458] [program2] : Module_init create kthread start
[64929.003867] [program2] : Module_init kthread start
[64929.011306] [program2] : The child process has pid = 23269
[64929.012006] [program2] : This is the parent process, pid = 23268
[64929.012466] [program2] : Child process
[64929.025923] [program2] : Get SIGTERM signal
[64929.038670] [program2] : Child process terminated
[64929.040366] [program2] : The return signal is 15
[64930.756390] [program2] : Module_exit
```

### iii. The output of being stopped

```
root@csc3150:/home/vagrant/csc3150/Assignment_1_120090361/source/program2# dmesg | tail -n 10
[65076.404800] [program2] : Module_init {Jiarui Chen} {120090361}
[65076.405798] [program2] : Module_init create kthread start
[65076.406879] [program2] : Module_init kthread start
[65076.432192] [program2] : The child process has pid = 24145
[65076.432977] [program2] : This is the parent process, pid = 24144
[65076.433800] [program2] : Child process
[65076.468157] [program2] : Get SIGSTOP signal
[65076.474591] [program2] : Child process stopped
[65076.521590] [program2] : The return signal is 19
[65081.516982] [program2] : Module_exit
```

# 5. Conclusion

## 5.1. Task 1

i. In task 1, we use *fork* function to create a process, which returns a *pid* (process id). The process id helps us to distinguish the status of the process. If pid is less than 0, then fork failed; if it is equal to 0, then the process is the child process; if it is larger than 0, then the process is the parent process.

ii. We use *wait_pid* to wait the signal from child process. We set the third parameter to WUNTRACED, which can detect the termination and stop status of child process.

iii. We use WIFEXITED, WIFSIGNALED, WIFSTOPPED to detect the type of termination or stop, and WTERSIG to analyze the abnormal termination type.

## 5.2. Task 2

i. In task 2, we use EXPORT_SYMBOL to export functions from Linux kernel source code, and we need to write *extern* declaration in order to use them.

ii. In my_wait function, we set wait_opts *wo.flags* to WEXITED|WSTOPPED, which can detect the termination and stop status.

iii. Linux returns a 16-bits exit status, which stores in *wo.stat*. The lower 8 bits of code is exit signal, and the higher 8 bits of code is exit code. In order to get exit signal, Linux kernel source code define WTERSIG as signal&0x7f, and define WIFSTOPPED as signal&0xff==0x7f. So, in function my_signal_handle, we get status of normal termination by status>>8, get abnormal termination signal by status&0x7f, and deal with stop by directly output the signal 19.