Report for CSC3150 Assignment 1

Yang Liang 120090874

1. Program Design Methodology

1.1 Design Methodology for Task 1

In this task, we are required to fork a child process to execute the test programs provided, and utilize the *wait* or *waitpid* function to require the parent process to wait for the child's execution. Whenever the child process terminates, it will send signal SIGCHLD to the parent signal to symbolize the termination, then we print out how the child process terminated and what signals were raised.

We mainly need to focus on how to distinguish and tackle the execution of the parent and child process, respectively. When we implement the function *fork()* to create a new process, the process created by fork is an exact clone of the original parent process, except the process ID, then both process will start executing the program simultaneously from the next statement following the *fork()* call. *fork()* function has 2 return values, it returns a 0 to the newly created child process, and it returns a positive values, namely the process ID of the child process to the parent, since the process ID is organized in a linked-list like structure. Hence, when we assign the return value of *fork()* function to pid, i.e., *pid = fork()*, we get 2 different values for the variable *pid* in the 2 processes, this property for *fork()* can be utilized to distinguish and manage the 2 processes.

We then organize them by an *if…else…* structure, when the child and parent processes simultaneously execute the remaining part of the program, the part for *if (pid == 0)* is where the child process need to execute, it includes executing another test file by the function *execve()*. Worth mentioning, *execve()* will replace the original remaining processes after it by the execution of the new program we assign in the parameter list. The part for *else…* is the part where the parent process will execute. It calls the *wait()* function and obtains the signal from the child process. Generally speaking, the child process has 4 execution states, one is normal exit, one is abnormally terminated by signals, one is stopped, and the last one is continued, these 4 states can be captured and interpreted by the parent process using the macros WIFEXITED, WIFSIGNALED, WIFSTOPPED and WIFCONTINUED, respectively. Correspondingly, in each status, the signals

from the child process can be evaluated by some other macros such as WTERMSIG, WEXITSTATUS and WSTOPSIG, which returns the numerical values of the signals, for example, the return value 3 means the signal SIGQUIT, since there is a one-to-one mapping relationship between signals and the numbers. The corresponding relationship and the information of signals are shown in the figure attached. In this way, we are able to interpret what signals are sent.

#	Signal name	Default action	Comment	POSIX
1	SIGHUP	Terminate	Hang up controlling terminal or process	Yes
2	SIGINT	Terminate	Interrupt from keyboard	Yes
3	SIGQUIT	Dump	Quit from keyboard	Yes
4	SIGILL	Dump	Illegal instruction	Yes
5	SIGTRAP	Dump	Breakpoint for debugging	No
6	SIGABRT	Dump	Abnormal termination	Yes
6	SIGIOT	Dump	Equivalent to SIGABRT	No
7	SIGBUS	Dump	Bus error	No
8	SIGFPE	Dump	Floating-point exception	Yes
9	SIGKILL	Terminate	Forced-process termination	Yes
10	SIGUSR1	Terminate	Available to processes	Yes
11	SIGSEGV	Dump	Invalid memory reference	Yes
12	SIGUSR2	Terminate	Available to processes	Yes
13	SIGPIPE	Terminate	Write to pipe with no readers	Yes
14	SIGALRM	Terminate	Real-timerclock	Yes
15	SIGTERM	Terminate	Process termination	Yes
16	SIGSTKFLT	Terminate	Coprocessor stack error	No
17	SIGCHLD	Ignore	Child process stopped or terminated, or got signal if traced	Yes
18	SIGCONT	Continue	Resume execution, if stopped	Yes
19	SIGSTOP	Stop	Stop process execution	Yes
20	SIGTSTP	Stop	Stop process issued from tty	Yes
21	SIGTTIN	Stop	Background process requires input	Yes
22	SIGTTOU	Stop	Background process requires output	Yes
23	SIGURG	Ignore	Urgent condition on socket	No
24	SIGXCPU	Dump	CPU time limit exceeded	No
25	SIGXFSZ	Dump	File size limit exceeded	No
26	SIGVTALRM	Terminate	Virtual timer clock	No
27	SIGPROF	Terminate	Profile timer clock	No
28	SIGWINCH	Ignore	Window resizing	No
29	SIGIO	Terminate	I/O now possible	No
29	SIGPOLL	Terminate	Equivalent to SIGIO	No
30	SIGPWR	Terminate	Power supply failure	No

Figure 1. Information for some signals

1.2 Design Methodology for Task2

This task can be viewed as the kernel mode version of task1, where we are required to implement the same process creation flow using the kernel functions. Similarly, we need to manage the child process and the parent process in the *my_fork()* function we defined. The child process will mainly be executed by the *kernel_clone()* function. This kernel function executes step by step in the process creation procedures, such as assign an available PID to the new task, duplicates the content in the parent process to the child process and create spaces for the child process, then lastly insert the child process to the wait_queue. In the child process, we need to execute the test programs by the kernel function *do_execve()* and *getname_kernel()*. The parent

process will mainly execute the *my_wait()* function we define, where we wait for the completion of child process by invoking the kernel function *do_wait()*, and then analyzes the return signals from the child process. The *do_wait()* function generally updates the status of the processes, and activate the parent process when the child process terminates. Analogous to task 1, the return signals are stored in the *wo_stat* member of the *wait_opts* structure defined in the kernel as an integer. Again, we utilize the one-to-one relationship between signals and the integer return value from *wo_stat* to interpret what signals are raised in the child process.

Worth mentioning, since we need to invoke some kernel functions in the above process, before we use these APIs, we need to add the EXPORT_SYMBOL(*kernel_function_name*) after every kernel function we need to invoke in the kernel source code. Since the kernel source code is modified by us (even a little bit), we then need to compile the kernel again so as to rebuilt and save these changes, in order that they could be used in our own modules. The way for EXPORT_SYMBOL is shown in the below figure.

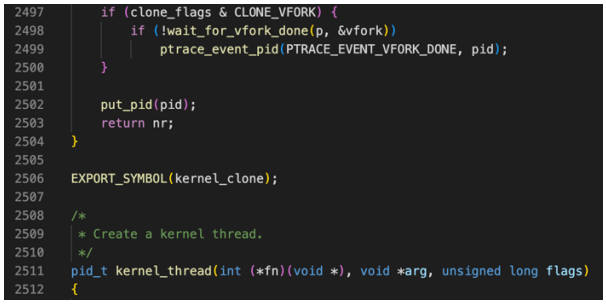

Figure 2. EXPORT_SYMBOL

2. Environment Configuration

2.1 Linux Environment on Virtual Machine Set Up

In order not to crash the operation system on our own computer, we need to run the program on the Virtual Machine.

The operating system on my own computer is Mac OS Big Sur 11.1, run on the Intel/x86 structure. The detailed steps are shown below.

1. Install VirtualBox and Vagrant, reboot the machine after installation.
2. Set up a directory for csc3150 on the desktop by the *mkdir -p* command.
3. Use the *cd* command to the directory we set up in the last step, then execute in terminal

vagrant init cyzhu/csc3150.

4. Execute *vagrant up*, then wait for downloading the system image, the virtualbox window may pop up automatically.
5. Now the Virtual Machine is set up, and we can set up the *ssh* to connect to the VM, by executing the command *mkdir -p ~/.ssh && vagrant ssh-config >> ~/.ssh/config* in terminal. Now we are able to connect to VM in terminal with *ssh default*.

2.2 Kernel Compilation

The Linux kernel version required for this assignment is *5.10.x*. Since we need to invoke some kernel functions, we need to update the original Linux kernel version (which is 4.4.0-210) to *5.10.x*. and update it in our virtual machine. The kernel compilation step is as belows.

1. Download the kernel source code from http://www.kernel.org. The version I choose is 5.10.146.
2. Install dependency and development tools in the terminal with the linux 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
3. Extract the kernel file we downloaded to a path we created, the path I created is under the *csc3150* folder *~csc3150/seed/*. Use the *cd* command to this directory and unzip the kernel package with the command

   $sudo tar xvf linux-5.10.146.tar.xz
4. Copy the config file from */boot* to the *.config* in our newly-downloaded kernel file *~csc3150/seed/linux-5.10.146/.config*
5. Grant the root permission by *sudo su* command and again, in the same directory *~csc3150/seed/linux-5.10.146*. We then start to load the configuration to the .config by the *make menuconfig* command, in the interface, we load the configuration, save it and exit.
6. We then start to build the kernel images and modules, to update the Linux kernel to 5.10.146 version. First, we need to install *bc* tools with command

   sudo apt install bc

After that, we start the building process with the commands

make bzImage -j$(nproc)

make modules -j$(nproc)

these 2 steps may take approximately 30 minutes to 1 hour.

7. Lastly, install the kernel modules by command

   make modules_install

   install the kernel by

   make install

   After finishing the command above, we type the command *reboot* to restart, load and update the new kernel.

8. To check whether the kernel has been compiled and updated successfully, we reconnect to the virtual machine and went into the VM environment, type the command

   uname -r

   to check whether the version has been updated to version 5.10.146.

Please note that, in task2 we need to use EXPORT_SYMBOL to modify some of the kernel source codes. Therefore, before using them in our modules, we need to compile the kernel again to update the changes every time we are modifying the kernel source code. To save our time, we can start from step 6 to only rebuild the update modules.

When inserting the kernel module, we first need to use the *sudo su* and *cd* command to grant root permission and direct to the directory where the program belongs to. After that, we use the *make* command to compile and generate the .ko file as the kernel module. To insert the module, We need to type in the command

insmod program2.ko

to view the execution states and the outputs, we can use the *dmesg* commands to view the kernel log. To remove the module, we can utilize the command

rmmod program2.ko

3. What is Learned from the Assignment

Generally speaking, we perform process creation in both the user mode and kernel mode. In task1, we understand the workflow of the management of parent process and child process in the user mode, especially the functionality of 2 main functions: *fork()*, which generates 2 different return values for the 2 processes, and *wait()*, which enables the parent process to wait for the child process. In the practical invoke of the *wait()* and *waitpid()*, I obtain deeper learning on the meanings of its parameters and some macros such as WIFEXITED.

In task2, we implement the fork process in kernel mode, where we invoked the kernel functions *kernel_clone()*, *do_wait()* and *do_execve()*. During the programming design process, we went through the source code of kernel functions, then we are able to develop a more fundamental understanding on the process creation and process managemen. Moreover, when compiling the kernels, we are more familiar with the Linux command line syntax.

4. Screenshots of the kernel outputs

We will test our programs in task1 and task2 based on the 15 different signal cases provided in task1.

4.1 Outputs in Task1

The output will display the workflow of the fork process, including the execution of the test program, and the detailed signal that cause the termination of child process.

```
vagrant@csc3150:~/csc3150/Assignment_1_120090874/source/program1$ ./program1 ./normal
Process start to fork
I'm the parent process, my pid = 1839
I'm the child process, my pid = 1840
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 = 100
```

```
vagrant@csc3150:~/csc3150/Assignment_1_120090874/source/program1$ ./program1 ./abort
Process start to fork
I'm the parent process, my pid = 1886
I'm the child process, my pid = 1887
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/Assignment_1_120090874/source/program1$
```

```
vagrant@csc3150:~/csc3150/Assignment_1_120090874/source/program1$ ./program1 ./alarm
Process start to fork
I'm the parent process, my pid = 1928
I'm the child process, my pid = 1929
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
vagrant@csc3150:~/csc3150/Assignment_1_120090874/source/program1$
```

```
vagrant@csc3150:~/csc3150/Assignment_1_120090874/source/program1$ ./program1 ./bus
Process start to fork
I'm the parent process, my pid = 1982
I'm the child process, my pid = 1983
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGBUS program

Parent process receives SIGCHLD signal
child process get SIGBUS signal
vagrant@csc3150:~/csc3150/Assignment_1_120090874/source/program1$
```

```
vagrant@csc3150:~/csc3150/Assignment_1_120090874/source/program1$ ./program1 ./floating
Process start to fork
I'm the parent process, my pid = 2033
I'm the child process, my pid = 2034
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGFPE program

Parent process receives SIGCHLD signal
child process get SIGFPE signal
vagrant@csc3150:~/csc3150/Assignment_1_120090874/source/program1$
```

```
vagrant@csc3150:~/csc3150/Assignment_1_120090874/source/program1$ ./program1 ./hangup
Process start to fork
I'm the parent process, my pid = 2117
I'm the child process, my pid = 2118
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
vagrant@csc3150:~/csc3150/Assignment_1_120090874/source/program1$
```

```
vagrant@csc3150:~/csc3150/Assignment_1_120090874/source/program1$ ./program1 ./illegal_instr
Process start to fork
I'm the parent process, my pid = 2170
I'm the child process, my pid = 2171
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/Assignment_1_120090874/source/program1$
```

```
vagrant@csc3150:~/csc3150/Assignment_1_120090874/source/program1$ ./program1 ./interrupt
Process start to fork
I'm the parent process, my pid = 2218
I'm the child process, my pid = 2219
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGINT program

Parent process receives SIGCHLD signal
child process get SIGINT signal
vagrant@csc3150:~/csc3150/Assignment_1_120090874/source/program1$
```

```
vagrant@csc3150:~/csc3150/Assignment_1_120090874/source/program1$ ./program1 ./kill
Process start to fork
I'm the parent process, my pid = 2259
I'm the child process, my pid = 2260
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGKILL program

Parent process receives SIGCHLD signal
child process get SIGKILL signal
vagrant@csc3150:~/csc3150/Assignment_1_120090874/source/program1$
```

```
vagrant@csc3150:~/csc3150/Assignment_1_120090874/source/program1$ ./program1 ./pipe
Process start to fork
I'm the parent process, my pid = 2309
I'm the child process, my pid = 2310
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGPIPE program

Parent process receives SIGCHLD signal
child process get SIGPIPE signal
vagrant@csc3150:~/csc3150/Assignment_1_120090874/source/program1$
```

```
vagrant@csc3150:~/csc3150/Assignment_1_120090874/source/program1$ ./program1 ./quit
Process start to fork
I'm the child process, my pid = 2357
Child process start to execute test program:
I'm the parent process, my pid = 2356
------------CHILD PROCESS START------------
This is the SIGQUIT program

Parent process receives SIGCHLD signal
child process get SIGQUIT signal
vagrant@csc3150:~/csc3150/Assignment_1_120090874/source/program1$
```

```
vagrant@csc3150:~/csc3150/Assignment_1_120090874/source/program1$ ./program1 ./segment_fault
Process start to fork
I'm the parent process, my pid = 2422
I'm the child process, my pid = 2423
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGSEGV program

Parent process receives SIGCHLD signal
child process get SIGSEGV signal
vagrant@csc3150:~/csc3150/Assignment_1_120090874/source/program1$
```

```
vagrant@csc3150:~/csc3150/Assignment_1_120090874/source/program1$ ./program1 ./stop
Process start to fork
I'm the parent process, my pid = 2461
I'm the child process, my pid = 2462
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
vagrant@csc3150:~/csc3150/Assignment_1_120090874/source/program1$
```

```
vagrant@csc3150:~/csc3150/Assignment_1_120090874/source/program1$ ./program1 ./terminate
Process start to fork
I'm the parent process, my pid = 2499
I'm the child process, my pid = 2500
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
vagrant@csc3150:~/csc3150/Assignment_1_120090874/source/program1$
```

```
vagrant@csc3150:~/csc3150/Assignment_1_120090874/source/program1$ ./program1 ./trap
Process start to fork
I'm the parent process, my pid = 2525
I'm the child process, my pid = 2526
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGTRAP program

Parent process receives SIGCHLD signal
child process get SIGTRAP signal
vagrant@csc3150:~/csc3150/Assignment_1_120090874/source/program1$
```

4.2 Outputs in task2

The output will display the workflow of the fork process in the kernel log, including the execution of the test program, and the detailed signal raised by the child process, **the detailed reason that cause the child process to terminate (the reason is written according to the "*comment*" column in *figure 1* attached in section 1.1 as a reference**), and the detailed return signal number.

```
[ 1777.407527] [program2] : Module_init {Yang Liang} {120090874}
[ 1777.407530] [program2] : Module_init create kthread start
[ 1777.407693] [program2] : module_init kthread start
[ 1777.408986] [program2] : The child process has pid = 2712
[ 1777.408989] [program2] : This is the parent process, pid = 2711
[ 1777.408991] [program2] : Child process
[ 1777.599033] [program2] : get SIGBUS signal
[ 1777.599034] [program2] : child process has bus error
[ 1777.599035] [program2] : The return signal is 7
[ 1783.646219] [program2] : Module_exit
```

```
[ 1918.154932] [program2] : Module_init {Yang Liang} {120090874}
[ 1918.154934] [program2] : Module_init create kthread start
[ 1918.155031] [program2] : module_init kthread start
[ 1918.155086] [program2] : The child process has pid = 2871
[ 1918.155087] [program2] : This is the parent process, pid = 2870
[ 1918.155102] [program2] : Child process
[ 1918.294041] [program2] : get SIGABRT signal
[ 1918.294043] [program2] : child process is aborted
[ 1918.294044] [program2] : The return signal is 6
[ 1921.683101] [program2] : Module_exit
root@csc3150:/home/vagrant/csc3150/Assignment_1_120090874/source/program2#
```

```
[ 1976.356966] [program2] : Module_init {Yang Liang} {120090874}
[ 1976.356968] [program2] : Module_init create kthread start
[ 1976.357124] [program2] : module_init kthread start
[ 1976.357199] [program2] : The child process has pid = 2935
[ 1976.357200] [program2] : This is the parent process, pid = 2934
[ 1976.357203] [program2] : Child process
[ 1978.379289] [program2] : get SIGALRM signal
[ 1978.379292] [program2] : child process is alarmed
[ 1978.379293] [program2] : The return signal is 14
[ 1979.896105] [program2] : Module_exit
root@csc3150:/home/vagrant/csc3150/Assignment_1_120090874/source/program2#
```

```
[ 2155.939167] [program2] : Module_init {Yang Liang} {120090874}
[ 2155.939169] [program2] : Module_init create kthread start
[ 2155.939253] [program2] : module_init kthread start
[ 2155.939959] [program2] : The child process has pid = 3033
[ 2155.939962] [program2] : This is the parent process, pid = 3031
[ 2155.939966] [program2] : Child process
[ 2156.092351] [program2] : get SIGFPE signal
[ 2156.092353] [program2] : child process has floating-point exception
[ 2156.092354] [program2] : The return signal is 8
[ 2159.616450] [program2] : Module_exit
root@csc3150:/home/vagrant/csc3150/Assignment_1_120090874/source/program2#
```

```
[ 2262.742381] [program2] : Module_init {Yang Liang} {120090874}
[ 2262.742384] [program2] : Module_init create kthread start
[ 2262.742746] [program2] : module_init kthread start
[ 2262.742845] [program2] : The child process has pid = 3193
[ 2262.742845] [program2] : This is the parent process, pid = 3192
[ 2262.742909] [program2] : Child process
[ 2262.744532] [program2] : get SIGHUP signal
[ 2262.744534] [program2] : child process hangs up controlling terminal or process
[ 2262.744535] [program2] : The return signal is 1
[ 2266.678286] [program2] : Module_exit
root@csc3150:/home/vagrant/csc3150/Assignment_1_120090874/source/program2#
```

```
[ 2373.336642] [program2] : Module_init {Yang Liang} {120090874}
[ 2373.336645] [program2] : Module_init create kthread start
[ 2373.336780] [program2] : module_init kthread start
[ 2373.336920] [program2] : The child process has pid = 3276
[ 2373.336922] [program2] : This is the parent process, pid = 3275
[ 2373.336926] [program2] : Child process
[ 2373.461017] [program2] : get SIGILL signal
[ 2373.461018] [program2] : child process has illegal instruction
[ 2373.461019] [program2] : The return signal is 4
[ 2376.837805] [program2] : Module_exit
root@csc3150:/home/vagrant/csc3150/Assignment_1_120090874/source/program2#
```

```
[ 2440.172388] [program2] : Module_init {Yang Liang} {120090874}
[ 2440.172390] [program2] : Module_init create kthread start
[ 2440.172499] [program2] : module_init kthread start
[ 2440.172605] [program2] : The child process has pid = 3381
[ 2440.172607] [program2] : This is the parent process, pid = 3380
[ 2440.172611] [program2] : Child process
[ 2440.173104] [program2] : get SIGINT signal
[ 2440.173105] [program2] : child process has interrupt from keyboard
[ 2440.173106] [program2] : The return signal is 2
[ 2445.398396] [program2] : Module_exit
root@csc3150:/home/vagrant/csc3150/Assignment_1_120090874/source/program2#
```

```
[ 2501.505999] [program2] : Module_init {Yang Liang} {120090874}
[ 2501.506001] [program2] : Module_init create kthread start
[ 2501.506075] [program2] : module_init kthread start
[ 2501.506687] [program2] : The child process has pid = 3467
[ 2501.506688] [program2] : This is the parent process, pid = 3465
[ 2501.506690] [program2] : Child process
[ 2501.507612] [program2] : get SIGKILL signal
[ 2501.507614] [program2] : child process has forced-process termination
[ 2501.507614] [program2] : The return signal is 9
[ 2504.570331] [program2] : Module_exit
root@csc3150:/home/vagrant/csc3150/Assignment_1_120090874/source/program2#
```

```
[ 3007.482067] [program2] : Module_init {Yang Liang} {120090874}
[ 3007.482069] [program2] : Module_init create kthread start
[ 3007.482278] [program2] : module_init kthread start
[ 3007.482330] [program2] : The child process has pid = 5163
[ 3007.482331] [program2] : This is the parent process, pid = 5162
[ 3007.482372] [program2] : Child process
[ 3007.482995] [program2] : get SIGPIPE signal
[ 3007.482997] [program2] : child process writes to pipe with no readers
[ 3007.482998] [program2] : The return signal is 13
[ 3010.784129] [program2] : Module_exit
root@csc3150:/home/vagrant/csc3150/Assignment_1_120090874/source/program2#
```

```
[ 3095.282824] [program2] : Module_init {Yang Liang} {120090874}
[ 3095.282826] [program2] : Module_init create kthread start
[ 3095.283063] [program2] : module_init kthread start
[ 3095.283192] [program2] : The child process has pid = 5309
[ 3095.283194] [program2] : This is the parent process, pid = 5308
[ 3095.283198] [program2] : Child process
[ 3095.393676] [program2] : get SIGQUIT signal
[ 3095.393678] [program2] : child process quits from keyboard
[ 3095.393730] [program2] : The return signal is 3
[ 3100.223331] [program2] : Module_exit
root@csc3150:/home/vagrant/csc3150/Assignment_1_120090874/source/program2#
```

```
[ 3140.662288] [program2] : Module_init {Yang Liang} {120090874}
[ 3140.662290] [program2] : Module_init create kthread start
[ 3140.662698] [program2] : module_init kthread start
[ 3140.663224] [program2] : The child process has pid = 5396
[ 3140.663226] [program2] : This is the parent process, pid = 5394
[ 3140.663228] [program2] : Child process
[ 3140.787238] [program2] : get SIGSEGV signal
[ 3140.787239] [program2] : child process has illegal memory reference
[ 3140.787241] [program2] : The return signal is 11
[ 3145.338797] [program2] : Module_exit
root@csc3150:/home/vagrant/csc3150/Assignment_1_120090874/source/program2#
```

```
[ 3214.733812] [program2] : Module_init {Yang Liang} {120090874}
[ 3214.733814] [program2] : Module_init create kthread start
[ 3214.733967] [program2] : module_init kthread start
[ 3214.734299] [program2] : The child process has pid = 5478
[ 3214.734301] [program2] : This is the parent process, pid = 5477
[ 3214.734304] [program2] : Child process
[ 3214.734694] [program2] : get SIGSTOP signal
[ 3214.734695] [program2] : child process has stopped process execution
[ 3214.734696] [program2] : The return signal is 19
[ 3218.926308] [program2] : Module_exit
root@csc3150:/home/vagrant/csc3150/Assignment_1_120090874/source/program2#
```

```
[ 3261.474249] [program2] : Module_init {Yang Liang} {120090874}
[ 3261.474251] [program2] : Module_init create kthread start
[ 3261.474387] [program2] : module_init kthread start
[ 3261.474425] [program2] : The child process has pid = 5646
[ 3261.474427] [program2] : This is the parent process, pid = 5645
[ 3261.474514] [program2] : Child process
[ 3261.475691] [program2] : get SIGTERM signal
[ 3261.475693] [program2] : child process has process termination
[ 3261.475694] [program2] : The return signal is 15
[ 3264.943892] [program2] : Module_exit
root@csc3150:/home/vagrant/csc3150/Assignment_1_120090874/source/program2#
```

```
[ 3313.929280] [program2] : Module_init {Yang Liang} {120090874}
[ 3313.929282] [program2] : Module_init create kthread start
[ 3313.929605] [program2] : module_init kthread start
[ 3313.929918] [program2] : The child process has pid = 5727
[ 3313.929919] [program2] : This is the parent process, pid = 5726
[ 3313.929941] [program2] : Child process
[ 3314.038623] [program2] : get SIGTRAP signal
[ 3314.038625] [program2] : child process has breakpoint for debugging
[ 3314.038626] [program2] : The return signal is 5
[ 3318.312975] [program2] : Module_exit
root@csc3150:/home/vagrant/csc3150/Assignment_1_120090874/source/program2#
```

```
[ 3422.440776] [program2] : Module_init {Yang Liang} {120090874}
[ 3422.440778] [program2] : Module_init create kthread start
[ 3422.440811] [program2] : module_init kthread start
[ 3422.441230] [program2] : The child process has pid = 5870
[ 3422.441232] [program2] : This is the parent process, pid = 5869
[ 3422.441235] [program2] : Child process
[ 3427.450231] [program2] : Normal termination
[ 3427.450235] [program2] : The return signal is 0
[ 3427.633607] [program2] : Module_exit
root@csc3150:/home/vagrant/csc3150/Assignment_1_120090874/source/program2#
```

--- End of Report ---