CSC3150 Assignment 1 Report

Student name: Sun Yuxuan

Student ID: 121090502

Date: 7/10/2023

1. Design

a. Overview:

Assignment 1 require us to complete two programs in user mode and kernel mode and a bonus program(I didn’ t complete this.). First, we need to set up virtual machine. The environment will be introduced later.

b. Program1

In **user mode,** program1 can fork a child process and execute the input program in the child process. Finally, the parent process gets the return signal from child process.

There are many APIs in user mode which can help us do this program easier. There is fork(), execv(), waitpid(), getpid() to make up the procedures of this program. There are also WIFEXITED(), WEXITSTATUS(), WIFSIGNALED(), WTERMSIG(), WIFSTOPPED() to help us to get the return value of the child process. The three APIs below are following the order of the program.

- **Fork**(): fork() returns a pid for parent and child process. To spare the child process from the parent process, we need to write an **if structure**.

```
if (pid == -1)
{
    perror("not fork,pid = -1\n");
}
/* execute test program */
else
{
    if (pid == 0)
    {
        // child process
    }
    /* wait for child process terminates */
    else
    {
        //parent process
    }
}
```

In this structure, the fork returns -1 if the child process fails to start. **If** the child successfully starts, we are now in the first **else** part and the fork will return two pid. The first is the pid of parent and other is the pid of child. **If** the pid == 0, we write child process here. Then, we write parent process in the **else** part.

- **Execv**(): exec is a functionality of an operating system that runs an executable file in the context of an already existing process, replacing the previous executable. In this program, I use execv() to execute the program that have been input. V in execv() means **vector**. I use

execv(arg[0], arg) for execute file function. **Arg[0]** means ./testfile_name, and **arg** is the input arguments that should be input into the testfile. This function will finally run the testfile in the child process.

- **Waitpid**(): waitpid(pid, &status, WUNTRACED) This function can get the return status of the specific pid process. In our program, it will receive the return value of the child process. We can identify the return value of normal exit and signals exit. But **SIGSTOP** won't raise a **SIGSTOP** to the parent process. To receive the signal of the **SIGSTOP**, we have **WUNTRACED**, which will return the SIGSTOP to parent process when the child process stops.

Evaluate child process's status (zero or non-zero)
- int **WIFEXITED** (int status)
- int **WIFSIGNALED** (int status)
- int **WIFSTOPPED** (int status)

Evaluate child process's returned value of status
- int **WEXITSTATUS** (int status)
- int **WTERMSIG** (int status)
- int **WSTOPSIG** (int status)

Finally, we can identify any signal from number **0 to 15** (tests provided in the program source code) and the stop signal.

That's all for program 1, the output of my program is in the bottom of the report. The program first starts to **fork**, then the child process starts, and the parent process **wait**. After the child process ends, the parent process gets the return value(**signal**). After identifying, the parent process will **print** out the signal that the child process raises.

c. Program2

Program2 requires us to write a program that creates a module in **kernel mode** and endows it with the capability to create and run child processes. The environment setting will be introduced in the Environment part later.

Program2 have **4 APIs** that can help us a lot to complete the task. I will start from introduce these four functions.

```
/*external functions*/
extern int do_execve(struct filename *filename,
                     const char __user *const __user *__argv,
                     const char __user *const __user *__envp);
extern long do_wait(struct wait_opts *wo);
struct filename *getname_kernel(const char *filename);
extern pid_t kernel_clone(struct kernel_clone_args *kargs);
```

External functions

- **pid_t kernel_clone(){}**

  This function will use **get_task_pid()** to get and assign a new pid to child process. This child process will be create by **copy_process()**. In **copy_process**, it calls **dup_task_struct()**, which creates a new kernel stack, thread_info structure, and task_struct for the new process. Then the function uses **wake_up_new_task()** to wake up the child process and insert the child process to the running queue.

  Arguments:

```
struct kernel_clone_args args = {
    .flags = SIGCHLD,
    .pidfd = NULL,
    .child_tid = NULL,
    .parent_tid = NULL,
    .exit_signal = SIGCHLD,
    .stack = my_exec_address,
    .stack_size = 0,
    .tls = 0,
```

  flags: How to clone the child process. When executing fork(), it is set as SIGCHILD.

  stack: Specifies the location of the stack used by the child process. In this program, it is &my_exec.

  stack_size: Normally set as 0 because it is unused.

  parent_tid: Used for clone() to point to user space memory in parent process address space. It is set as NULL when executing fork();

  child_tid: Used for clone() to point to user space memory in child process address space. It is set as NULL when executing fork();

  tls: Set thread local storage. In this program, set it to 0.

- **int do_execve (){}**

  This function receive three input, which is filename, argv and envp in order. Then it use the address of **envp** to set the environment. And input the **argv** into the executable file, which name is **filename**.

  Arguments:

  Filename(struct) : Filename of the executable file.

  Argv: The arguments for executing the file.

  Envp: System environment variable set.

  - **struct** filename *getname_kernel(){}

- **long do_wait (){}**

  This function need the support of the exit.ko module. When execute this function, it will first create a wait_ops structure and add it into wait queue by calling add_wait_queue. Then it will Use set_current_state to update state as TASK_INTERRUPTIBLE or TASK_RUNNING. When child process terminates, it calls wake_up_parent. Then parent process will go to repeat scanning, so that it can get return of child process' termination.

  - **struct** wait_opts:

```
struct wait_opts
{
    enum pid_type wo_type;          // It is defined in '/include/linux/pid.h'.
    int wo_flags;                   // Wait options. (0, WNOHANG, WEXITED, etc.)
    struct pid *wo_pid;             // Kernel's internal notion of a process iden
    struct siginfo *wo_info;        // Singal information.
    int wo_stat;                    // Child process's termination status
    struct rusage *wo_rusage;       // Resource usage
    wait_queue_entry_t child_wait; // Task wait queue
    int notask_error;
};
```

Arguments:

wo_type: It is defined in "/include/linux/pid.h".

wo_flags: Wait options. (0, WNOHANG, WEXITED, etc.)

*wo_pid: Kernel's internal notion of a process identifier. "Find_get_pid()"

*wo_info: Singal information.

*wo_stat: Child process' s termination status

*wo_rusage: Resource usage

child_wait: Task wait queue.

notask_error: Check error.

```
struct wait_opts wo;
struct pid *wo_pid = NULL;
enum pid_type type;
type = PIDTYPE_PID;
wo_pid = find_get_pid(pid);

wo.wo_type = type;
wo.wo_pid = wo_pid;
wo.wo_flags = WUNTRACED | WEXITED;
wo.wo_info = NULL;
wo.wo_stat = status;
wo.wo_rusage = NULL;
```

To init a argument wo whose type is wait_ops, we need to:

- **Set *wo_pid to NULL.** No value at first.
- **Set wo_type to PIDTYPE_PID.** The classical pid type.
- **Set wo_pid to find_get_pid(pid).** Use number of pid to get "pidtype" pid.
- **Set wo_flags to WUNTRACED | WEXITED.** Get the return from stopped/exited process.
- **Set wo_info to NULL.** The additional information for child process is NULL.
- **Set wo_stat to status.** 0(normal) at first, this is the status of child process, it use int to represent signal. (such as 1=SIGHUP, 2=SIGINT, ….)
- **Set wo_rusage to NULL.** Not concerned about resource usage information of child processes

Other structures:

- **Kernel_thread:**

This function is the core of a kernel object. After executing the line: "task = kthread_create (&my_fork, NULL, "MyThread")" , the task has been created. But at this moment, it won't execute. The process continue. When the kernel_clone execute, the function wake_up_process() will wake up the task and the child process start to execute.

  - **static** struct task_struct *task;
  - int **wake_up_process** (struct task_struct * p);
  - may use kthread_run instead of kthread_create & wake_up_process()

Program 2 design

- **My_fork():**

After kernel_create and wake_up_process, the task start to process. The program first come to my_fork(). My_fork will first initiate all the

arguments to the default value.

```
struct k_sigaction *k_action = &current->sighand->action[0];
for (i = 0; i < _NSIG; i++)
{
    k_action->sa.sa_handler = SIG_DFL;
    k_action->sa.sa_flags = 0;
    k_action->sa.sa_restorer = NULL;
    sigemptyset(&k_action->sa.sa_mask);
    k_action++;
}
```

K_action: Define a pointer *k_action to k_sigaction and initialize it as the address of the first element in the current process's signal handler list.

Sa.sa_handler: Set to default(SIG_DFL)

Sa.sa_flags: There are no special flags or options associated with this signal handler.

Sa.sa_restorer: NULL indicates that there is no need to restore the program.

Sa.sa_mask: Empty indicates that other signals will not be blocked when processing this signal.

This structure init ensures that all signals have a consistent initial states.

Kernel_clone()

```
pid = kernel_clone(&args);
```

Then, the program start to clone process.

```
1  if (pid == -1)
2  {
3      perror("not fork,pid = -1\n");
4  }
5  /* execute test program */
6  else
7  {
8      if (pid == 0)
9      {
10         // child process
11     }
12     /* wait for child process terminates */
13     else
14     {
15         //parent process
16     }
17 }
```

Just like Program1, the program2 will get two pids, one for parent and one for child. Let' s look at parent process first. It will come to my_wait().

My_wait() initial the wait_ops and then execute the external function "

do_wait() ", do_wait() will return the status number to wo.wo_stat. in this program, we just need to identify 0-15&19, so I use wo.wo_stat&0xf to get the last 4 bits of the status(32bits in total). Convent the last 4 bits

```
static char *childsignal[] = {NULL,
                              "SIGHUP", "SIGINT", "SIGQUIT", "SIGILL", "SIGTRAP",
                              "SIGABRT", "SIGBUS", "SIGFPE", "SIGKILL", "SIGUSR1",
                              "SIGSEGV", "SIGUSR2", "SIGPIPE", "SIGALRM", "SIGTERM"};
```

to decimal. I write a list of signal and their status numbers.

There is an if structure to identify the signal that the child raise.

```
if ((wo.wo_stat & 0xf) == 0)···
else if ((wo.wo_stat & 0xf) == 1)···
else if ((wo.wo_stat & 0xf) == 2)···
else if ((wo.wo_stat & 0xf) == 3)···
else if ((wo.wo_stat & 0xf) == 4)···
else if ((wo.wo_stat & 0xf) == 5)···
else if ((wo.wo_stat & 0xf) == 6)···
else if ((wo.wo_stat & 0xf) == 7)···
else if ((wo.wo_stat & 0xf) == 8)···
else if ((wo.wo_stat & 0xf) == 9)···
else if ((wo.wo_stat & 0xf) == 11)···
else if ((wo.wo_stat & 0xf) == 13)···
else if ((wo.wo_stat & 0xf) == 14)···
else if ((wo.wo_stat & 0x7f) == 15)···
else if (((wo.wo_stat >> 8) & 0x7f) == 19)···
else···
```

And now, we just need to wait the child process to come to end and raise the signal.

My_exec():

For child process part, it will execute "my_exec()". This function's core is "do_execve" which usually receives 3 inputs, filename, argument and environment. For this program, I first define a argv to be the path of the test file. Then I use function getname_kernel to get the filename through path.

Finally, from the start of initial module, to the end of the module, I think it's clear.

```
module_init(program2_init);
module_exit(program2_exit);
```

2. Environment

Ubuntu: 16.04.7 LTS

```
vagrant@csc3150:~/CSC3150/source/program2$ cat /etc/os-release
NAME="Ubuntu"
VERSION="16.04.7 LTS (Xenial Xerus)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 16.04.7 LTS"
VERSION_ID="16.04"
HOME_URL="http://www.ubuntu.com/"
SUPPORT_URL="http://help.ubuntu.com/"
BUG_REPORT_URL="http://bugs.launchpad.net/ubuntu/"
VERSION_CODENAME=xenial
UBUNTU_CODENAME=xenial
```

Linux Kernel: 5.10.197

```
vagrant@csc3150:~/CSC3150/source/program2$ uname -a
Linux csc3150 5.10.197 #3 SMP Fri Oct 6 06:33:45 UTC 2023 x86_64 x86_64 x86_64 GNU/Linux
vagrant@csc3150:~/CSC3150/source/program2$
```

Gcc: 5.4.0

```
vagrant@csc3150:~/CSC3150/source/program2$ gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/lib/gcc/x86_64-linux-gnu/5/lto-wrapper
Target: x86_64-linux-gnu
Configured with: ../src/configure -v --with-pkgversion='Ubuntu 5.4.0-6ubuntu1~16.04.12' --
+ --prefix=/usr --program-suffix=-5 --enable-shared --enable-linker-build-id --libexecdir=
nable-clocale=gnu --enable-libstdcxx-debug --enable-libstdcxx-time=yes --with-default-libs
-zlib --disable-browser-plugin --enable-java-awt=gtk --enable-gtk-cairo --with-java-home=/
d64 --with-jvm-jar-dir=/usr/lib/jvm-exports/java-1.5.0-gcj-5-amd64 --with-arch-directory=a
h-arch-32=i686 --with-abi=m64 --with-multilib-list=m32,m64,mx32 --enable-multilib --with-t
nu
Thread model: posix
gcc version 5.4.0 20160609 (Ubuntu 5.4.0-6ubuntu1~16.04.12)
```

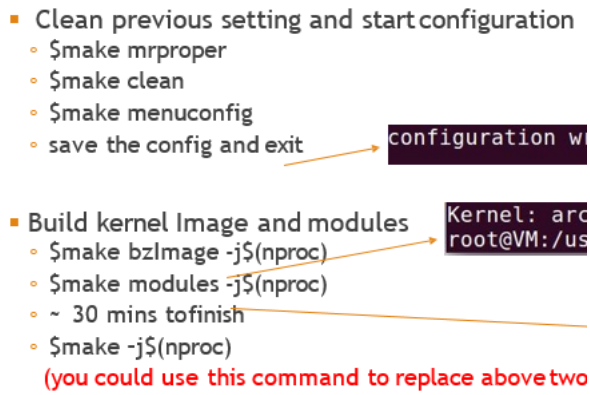
a. Project1 set up:

Upgrade the kernel from 4.4.0 to 5.10.197 (any version of 5.10.x):

First download kernel source from web. After install necessary tools, copy the Linux kernel to /home/seed/work.

- Download source code from
  - http://www.kernel.org
  - mirro: https://mirror.tuna.tsinghua.edu.cn/kernel/v5.x/
- 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
- Extract the source file to /home/seed/work
  - cp KERNEL_FILE.tar.xz /home/seed/work
  - cd /home/seed/work
  - $sudo tar xvf KERNEL_FILE.tar.xz

Then copy config from /boot to /home/seed/work/linux_kernel. Clean up the config before and set the new config. After do "make bzImage" and "make modules" and install them, reboot the VM.



Then the kernel is set to be 5.10. We can check by "uname -r".

This is the process of upgrade the kernel.

b. **Project2 set up**:

Because of the use of extern functions in Linux, we should use "export symbol" to import (clarify) the functions in another module. And we need to:

1. Find the module which have the functions we need.

Hints:

- Use "_do_fork" to fork a new process. (/kernel/fork.c)
- Use "do_execve" to execute the test program. (/fs/exec.c)
- Use "getname" to get filename. (/fs/namei.c)
- Use "do_wait" to wait for child process' termination status. (/kernel/exit.c)

The function _do_fork() is not in kernel so I change it to kernel_clone(). They are similar.

2. Use vim or nano to add "EXPORT_SYMBOL(function_name)". Don't forget use sudo to obtain permission to edit documents.

3. Starting from "make bzImage", to "reboot". After rebooting, the kernel is set to the situation we want.

```
/*external functions*/
extern int do_execve(struct filename *filename,
                     const char __user *const __user *__argv,
                     const char __user *const __user *__envp);
extern long do_wait(struct wait_opts *wo);
struct filename *getname_kernel(const char *filename);
extern pid_t kernel_clone(struct kernel_clone_args *kargs);
```

3. What I learnt

a. How to **set up virtual machine** in VirtualBox and connect it to vscode.

I got to know how to use powershell to communicate with VirtualBox through Vagrant.

I searched about the use of Vagrantfile.

I learn vagrant in powershell. (vagrant destroy/ssh-config/init/reload)

I got to know how to use vscode plugin SSH Remote.

I can now fix the config of ssh.

The file known_hosts is familiar with me.

I get the knowledge about rsa publickey and asymmetric encryption.

I get through a big problem to me with VM:

https://piazza.com/class/lmixo39osgi7f5/post/20

I help two classmates who have the same problem with me to overcome.

b. How to **update and write in** the kernel of linux.

I can now do easy kernel update and configure.

I can use vim or nano to edit source code of kernel.

I know why and how to export_symbol().

I know the necessary of Permission denied mostly caused by not "sudo".

c. How to **insert and remove module**.

I can now write easy kernel object and execute it.

I can now write makefile of kernel object.

I can now check the installed modules.

d. How to create **child process**

I can create child process through user/kernel mode.

I get to know the meaning of 15 kinds of signal.

I get to know the use of raise and wait.

e. How to work

I get to know how to work with forum. I posted on the forum and received help from it. After completing the code, I answered a few questions on the forum. I get to know the importance of communication with others. What's more, I gain self-learning skills. I get the passion for coding again!

Finally, thanks all the TAs and USTFs for answering our questions. It really helps me a lot!

4. How to execute & outputs

PROGRAM1:

Cd to /program1 ->make-> ./program1 ./testfile_name -> check the output in shell

```
vagrant@csc3150:~/CSC3150$ cd source/program1
vagrant@csc3150:~/CSC3150/source/program1$ make
cc -o program1 program1.c
vagrant@csc3150:~/CSC3150/source/program1$ ./program1 ./normal
Process start to fork
I'm the Parent Process, my pid = 2246
I'm the Child Process, my pid = 2247
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/source/program1$ ./program1 ./abort
Process start to fork
I'm the Parent Process, my pid = 2253
I'm the Child Process, my pid = 2254
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/source/program1$ ./program1 ./stop
Process start to fork
I'm the Parent Process, my pid = 2258
I'm the Child Process, my pid = 2259
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
```

PROGRAM2: (need first update the kernel)

Cd into /program2 -> make ->gcc -o test test.c -> sudo insmod program2.ko

-> sudo rmmod program2 ->dmesg

Init:

```
vagrant@csc3150:~/CSC3150/source/program2$ make
make -C /lib/modules/5.10.197/build M=/home/vagrant/CSC3150/source/program2 modules
make[1]: Entering directory '/home/seed/work/linux-5.10.197'
  CC [M]  /home/vagrant/CSC3150/source/program2/program2.o
/home/vagrant/CSC3150/source/program2/program2.c: In function ‘my_fork’:
/home/vagrant/CSC3150/source/program2/program2.c:200:24: warning: cast from pointer
to integer of different size [-Wpointer-to-int-cast]
  int my_exec_address = (int)&my_exec;
                        ^
  MODPOST /home/vagrant/CSC3150/source/program2/Module.symvers
  CC [M]  /home/vagrant/CSC3150/source/program2/program2.mod.o
  LD [M]  /home/vagrant/CSC3150/source/program2/program2.ko
make[1]: Leaving directory '/home/seed/work/linux-5.10.197'
vagrant@csc3150:~/CSC3150/source/program2$ gcc -o test test.c
vagrant@csc3150:~/CSC3150/source/program2$ sudo insmod program2.ko
vagrant@csc3150:~/CSC3150/source/program2$ sudo rmmod program2
vagrant@csc3150:~/CSC3150/source/program2$ dmesg
```

Test1-15:

```
[ 2235.917559] [program2] : module_init
[ 2235.917561] [program2] : module_init create kthread start
[ 2235.917783] [program2] : module_init kthread start
[ 2235.917841] [program2] : The child process has pid= 5907
[ 2235.917842] [program2] : This is the parent process, pid= 5906
[ 2235.917970] [program2] : child process
[ 2235.918876] [program2] : get SIGHUP signal
[ 2235.918877] [program2] : child process is hung up
[ 2235.918877] [program2] : The return signal is 1
[ 2238.194425] [program2] : module_exit./my
[ 2264.835836] [program2] : module_init
[ 2264.835838] [program2] : module_init create kthread start
[ 2264.836102] [program2] : module_init kthread start
[ 2264.836220] [program2] : The child process has pid= 6004
[ 2264.836220] [program2] : This is the parent process, pid= 6003
[ 2264.836307] [program2] : child process
[ 2264.836735] [program2] : get SIGINT signal
[ 2264.836737] [program2] : child process interrupt
[ 2264.836738] [program2] : The return signal is 2
[ 2267.801347] [program2] : module_exit./my
```

```
[ 2291.333286] [program2] : module_init
[ 2291.333288] [program2] : module_init create kthread start
[ 2291.333625] [program2] : module_init kthread start
[ 2291.333672] [program2] : The child process has pid= 6091
[ 2291.333672] [program2] : This is the parent process, pid= 6090
[ 2291.333702] [program2] : child process
[ 2291.408106] [program2] : get SIGQUIT signal
[ 2291.408107] [program2] : child process quit
[ 2291.408107] [program2] : The return signal is 3
[ 2292.894929] [program2] : module_exit./my
```

```
[ 2343.684278] [program2] : module_init
[ 2343.684279] [program2] : module_init create kthread start
[ 2343.684418] [program2] : module_init kthread start
[ 2343.684444] [program2] : The child process has pid= 6222
[ 2343.684445] [program2] : This is the parent process, pid= 6221
[ 2343.684566] [program2] : child process
[ 2343.751114] [program2] : get SIGILL signal
[ 2343.751115] [program2] : child process has an illegal instruction error
[ 2343.751115] [program2] : The return signal is 4
[ 2346.474611] [program2] : module_exit./my
```

```
[ 2309.470647] [program2] : module_init
[ 2309.470648] [program2] : module_init create kthread start
[ 2309.470856] [program2] : module_init kthread start
[ 2309.470882] [program2] : The child process has pid= 6155
[ 2309.470883] [program2] : This is the parent process, pid= 6154
[ 2309.471013] [program2] : child process
[ 2309.548036] [program2] : get SIGTRAP signal
[ 2309.548037] [program2] : child process has a trap error
[ 2309.548037] [program2] : The return signal is 5
[ 2313.199448] [program2] : module_exit./my
```

```
[ 2367.940288] [program2] : module_init
[ 2367.940289] [program2] : module_init create kthread start
[ 2367.940553] [program2] : module_init kthread start
[ 2367.940713] [program2] : The child process has pid= 6307
[ 2367.940714] [program2] : This is the parent process, pid= 6306
[ 2367.940736] [program2] : child process
[ 2368.003962] [program2] : get SIGABRT signal
[ 2368.003963] [program2] : child process has an abort error
[ 2368.003963] [program2] : The return signal is 6
[ 2369.864123] [program2] : module_exit./my
```

```
[ 2397.515017] [program2] : module_init
[ 2397.515018] [program2] : module_init create kthread start
[ 2397.515377] [program2] : module_init kthread start
[ 2397.515505] [program2] : The child process has pid= 6363
[ 2397.515506] [program2] : This is the parent process, pid= 6362
[ 2397.515743] [program2] : child process
[ 2397.583907] [program2] : get SIGBUS signal
[ 2397.583908] [program2] : child process has a bus error
[ 2397.583908] [program2] : The return signal is 7
[ 2399.298398] [program2] : module_exit./my
```

```
[ 2418.455616] [program2] : module_init
[ 2418.455618] [program2] : module_init create kthread start
[ 2418.455686] [program2] : module_init kthread start
[ 2418.455750] [program2] : The child process has pid= 6439
[ 2418.455752] [program2] : This is the parent process, pid= 6438
[ 2418.455940] [program2] : child process
[ 2418.522230] [program2] : get SIGFPE signal
[ 2418.522231] [program2] : child process has a float error
[ 2418.522232] [program2] : The return signal is 8
[ 2422.570258] [program2] : module_exit./my
```

```
[ 2441.487286] [program2] : module_init
[ 2441.487287] [program2] : module_init create kthread start
[ 2441.487420] [program2] : module_init kthread start
[ 2441.487450] [program2] : The child process has pid= 6495
[ 2441.487451] [program2] : This is the parent process, pid= 6494
[ 2441.487559] [program2] : child process
[ 2441.487992] [program2] : get SIGKILL signal
[ 2441.487994] [program2] : child process is killed
[ 2441.487995] [program2] : The return signal is 9
[ 2444.465901] [program2] : module_exit./my
```

```
[ 2471.933268] [program2] : module_init
[ 2471.933270] [program2] : module_init create kthread start
[ 2471.933540] [program2] : module_init kthread start
[ 2471.933656] [program2] : The child process has pid= 6549
[ 2471.933657] [program2] : This is the parent process, pid= 6548
[ 2471.933739] [program2] : child process
[ 2471.996525] [program2] : get SIGSEGV signal
[ 2471.996526] [program2] : child process has a segmentation fault error
[ 2471.996526] [program2] : The return signal is 11
[ 2474.578688] [program2] : module_exit./my
```

```
[ 1071.464094] [program2] : module_init
[ 1071.464097] [program2] : module_init create kthread start
[ 1071.464384] [program2] : module_init kthread start
[ 1071.464509] [program2] : The child process has pid= 3188
[ 1071.464511] [program2] : This is the parent process, pid= 3187
[ 1071.464561] [program2] : child process
[ 1071.465430] [program2] : get SIGPIPE signal
[ 1071.465432] [program2] : child process has a pipe error
[ 1071.465433] [program2] : The return signal is 13
[ 1072.979785] [program2] : module_exit./my
```

```
[ 1144.576817] [program2] : module_init
[ 1144.576820] [program2] : module_init create kthread start
[ 1144.577218] [program2] : module_init kthread start
[ 1144.577348] [program2] : The child process has pid= 3646
[ 1144.577351] [program2] : This is the parent process, pid= 3645
[ 1144.577540] [program2] : child process
[ 1144.578697] [program2] : get SIGALARM signal
[ 1144.578700] [program2] : child process has a alarm error
[ 1144.578701] [program2] : The return signal is 14
[ 1146.820232] [program2] : module_exit./my
```

```
[ 1164.224124] [program2] : module_init
[ 1164.224127] [program2] : module_init create kthread start
[ 1164.224358] [program2] : module_init kthread start
[ 1164.224504] [program2] : The child process has pid= 3680
[ 1164.224507] [program2] : This is the parent process, pid= 3679
[ 1164.224520] [program2] : child process
[ 1164.225712] [program2] : get SIGTERM signal
[ 1164.225714] [program2] : child process terminated
[ 1164.225715] [program2] : The return signal is 15
[ 1165.454871] [program2] : module_exit./my
```

```
[ 1925.712580] [program2] : module_init
[ 1925.715668] [program2] : module_init create kthread start
[ 1925.725699] [program2] : module_init kthread start
[ 1925.729527] [program2] : The child process has pid= 5340
[ 1925.732544] [program2] : This is the parent process, pid= 5339
[ 1925.734792] [program2] : child process
[ 1927.731100] [program2] : Normal termination with EXIT STATUS = 0
[ 1927.733088] [program2] : The return signal is 0
[ 1928.056342] [program2] : module_exit./my
```

```
[ 1255.025615] [program2] : module_init
[ 1255.025619] [program2] : module_init create kthread start
[ 1255.025889] [program2] : module_init kthread start
[ 1255.026025] [program2] : The child process has pid= 3726
[ 1255.026028] [program2] : This is the parent process, pid= 3725
[ 1255.026090] [program2] : child process
[ 1255.027153] [program2] : get SIGSTOP signal
[ 1255.027158] [program2] : The return signal is 19
[ 1256.499576] [program2] : module_exit./my
```