CSC3150 Assignment1 Report

120090211 Tao Chujun

Overview:

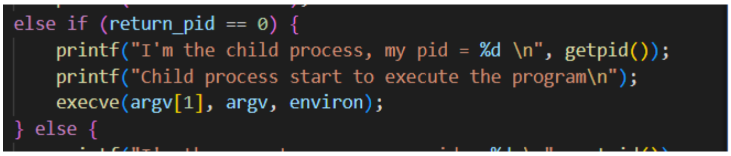
Project purpose:

This assignment is about creating processes and dealing with communication between processes by raising signals. It contains two tasks. Task 1 is to design from a user point of view. A child process is created using system call fork(). Task 2 is to design from the view of operating system, which means diving deep to the kernel space to do what operating system does whenever user uses fork() system call.

Task 1:

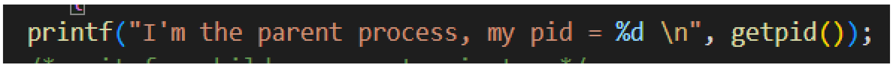
1. Design Logic

What this program does is to create a child process and let it run a certain program. There is only one main function for this program. The logic is firstly, creating a child process by fork(). The child process will copy its parent process's data. The fork function will return two times, one in the parent process, another one in the child process. For the child process, the return result will be 0. For the parent process, the return result will be the pid (process ID) of the child process. Next, the child process and the parent process will start to run concurrently. For child process, since return_pid here is 0, it will print informing statements in the terminal and will execute an existed program by calling the execve function.



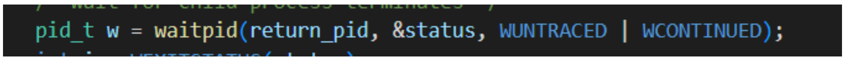
```
else if (return_pid == 0) {
    printf("I'm the child process, my pid = %d \n", getpid());
    printf("Child process start to execute the program\n");
    execve(argv[1], argv, environ);
} else {
```

For parent process, since its return_pid is not 0, it will print different message.



```
printf("I'm the parent process, my pid = %d \n", getpid());
```

Then the parent process will be blocked and will be awaked when the child process returns.

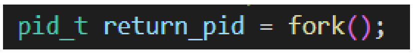


```
pid_t w = waitpid(return_pid, &status, WUNTRACED | WCONTINUED);
```

As the child process returns, the parent process will receive a SIGCHLD signal and more detailed returning status will be recorded in `int status;`.
For different returning status, the outputs will be different.

2. Functions and variables spec

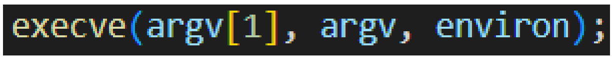
- pid_t fork(void)



```
pid_t return_pid = fork();
```

Fork function will create a child process. In parent process, the return result will be the child process pid if it is successfully created. In child process, the return result will be 0.

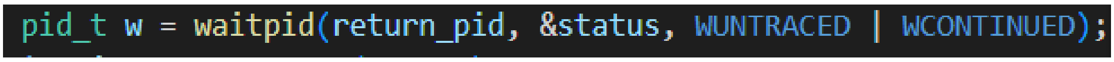
- int execve (const char *_path, char* const*_argv, char* const* envp)



```
execve(argv[1], argv, environ);
```

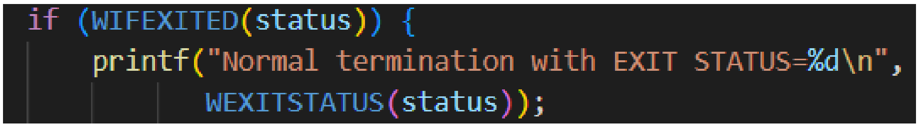
This function takes the first command line input as the executable file name and argv, environ here is for the inputs of the executable file.

- pid_t waitpid(pid_t _pid, int* _stat _loc, int _option)



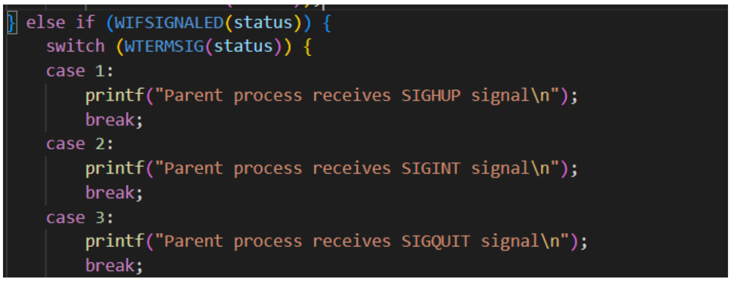
```
pid_t w = waitpid(return_pid, &status, WUNTRACED | WCONTINUED);
```

The waitpid will wait for the child process with pid until it stops or exits. The signals indicating how child process exits will be recorded in the status. As for the options, there are three kind of options to define how the waitpid function behaves.
WUNTRACED means the waitpid function will return as soon as the process exit.
WCONTINUED means the waitpid function will return if a stopped process continues with SIGCONT signal.
WIFEXITED(statloc) return TRUE if the child process exit normally. If it happens we can use WEXITSTATUS(statloc) to get the lower 8 bit of the status.



```
if (WIFEXITED(status)) {
    printf("Normal termination with EXIT STATUS=%d\n",
           WEXITSTATUS(status));
```

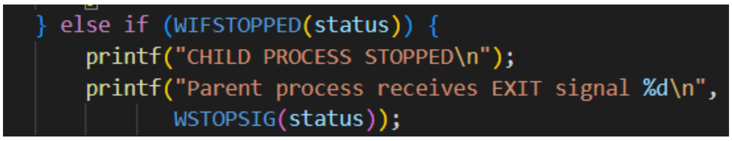
WIFSIGNALED(statloc) return TRUE if the child process raise a signal and kill itself. If it happens the WTERMSIG(statloc) will get that signal from status.



```
} else if (WIFSIGNALED(status)) {
    switch (WTERMSIG(status)) {
    case 1:
        printf("Parent process receives SIGHUP signal\n");
        break;
    case 2:
        printf("Parent process receives SIGINT signal\n");
        break;
    case 3:
        printf("Parent process receives SIGQUIT signal\n");
        break;
```

WIFSTOPPED(statloc) return TRUE if the child process stop. If it happens, the

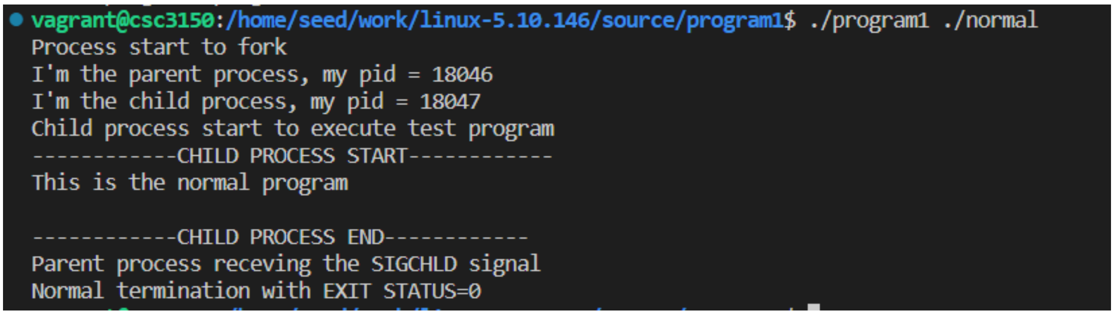
WSTOPSIG(statloc) will get the stopped signal from status.



```
} else if (WIFSTOPPED(status)) {
    printf("CHILD PROCESS STOPPED\n");
    printf("Parent process receives EXIT signal %d\n",
           WSTOPSIG(status));
```

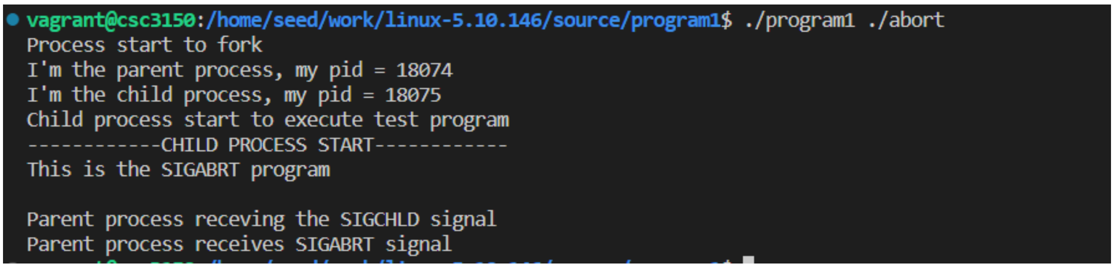
3. Sample Outputs:

Use ./program1 ./$(testing_filename) to execute the program1.



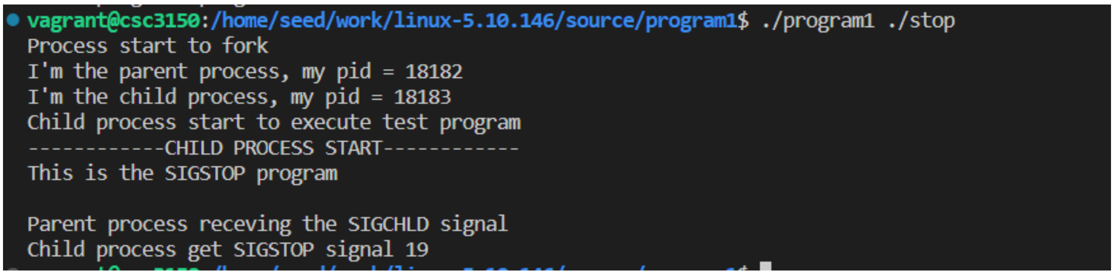
```
vagrant@csc3150:/home/seed/work/linux-5.10.146/source/program1$ ./program1 ./normal
Process start to fork
I'm the parent process, my pid = 18046
I'm the child process, my pid = 18047
Child process start to execute test program
------------CHILD PROCESS START------------
This is the normal program

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



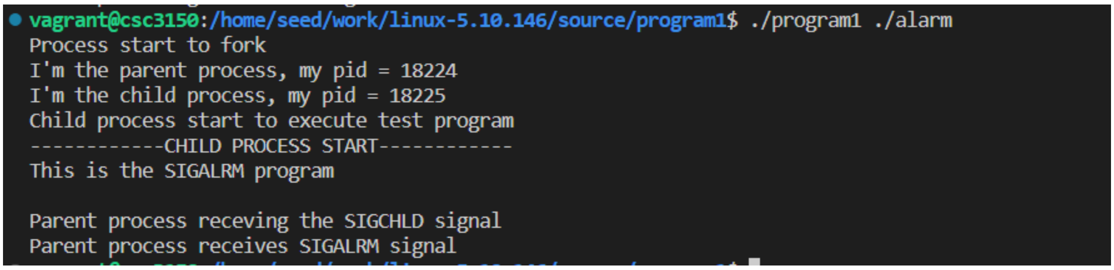
```
vagrant@csc3150:/home/seed/work/linux-5.10.146/source/program1$ ./program1 ./abort
Process start to fork
I'm the parent process, my pid = 18074
I'm the child process, my pid = 18075
Child process start to execute test program
------------CHILD PROCESS START------------
This is the SIGABRT program

Parent process receving the SIGCHLD signal
Parent process receives SIGABRT signal
```



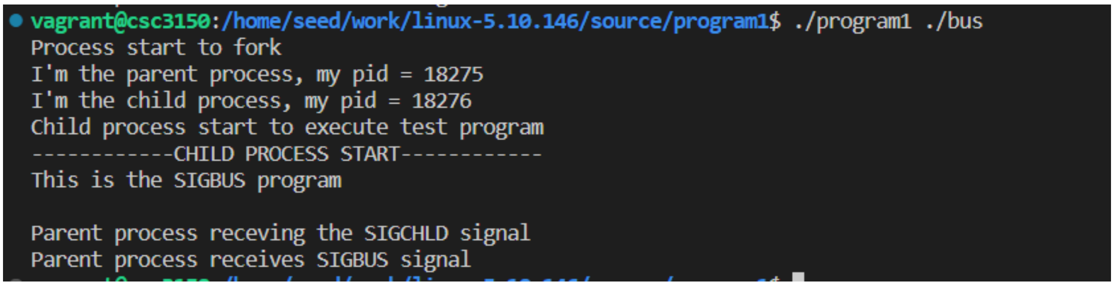
```
vagrant@csc3150:/home/seed/work/linux-5.10.146/source/program1$ ./program1 ./stop
Process start to fork
I'm the parent process, my pid = 18182
I'm the child process, my pid = 18183
Child process start to execute test program
------------CHILD PROCESS START------------
This is the SIGSTOP program

Parent process receving the SIGCHLD signal
Child process get SIGSTOP signal 19
```



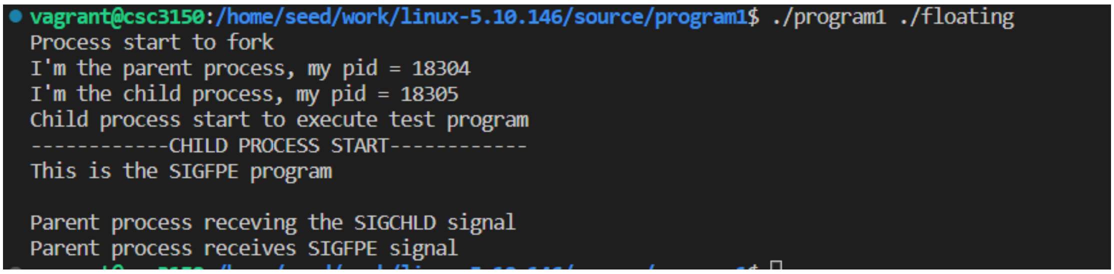
```
vagrant@csc3150:/home/seed/work/linux-5.10.146/source/program1$ ./program1 ./alarm
Process start to fork
I'm the parent process, my pid = 18224
I'm the child process, my pid = 18225
Child process start to execute test program
------------CHILD PROCESS START------------
This is the SIGALRM program

Parent process receving the SIGCHLD signal
Parent process receives SIGALRM signal
```



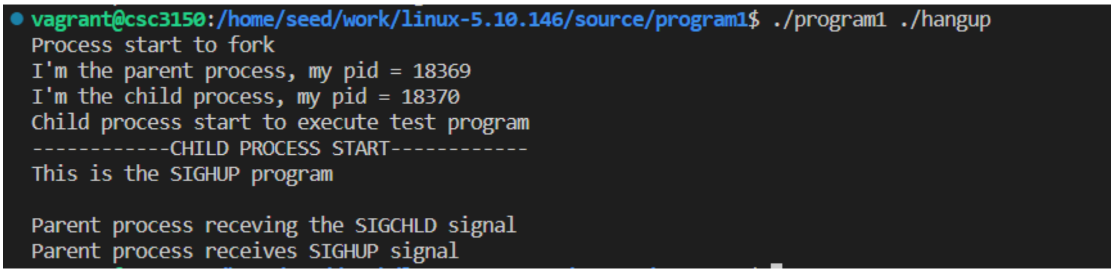
```
vagrant@csc3150:/home/seed/work/linux-5.10.146/source/program1$ ./program1 ./bus
Process start to fork
I'm the parent process, my pid = 18275
I'm the child process, my pid = 18276
Child process start to execute test program
------------CHILD PROCESS START------------
This is the SIGBUS program

Parent process receving the SIGCHLD signal
Parent process receives SIGBUS signal
```



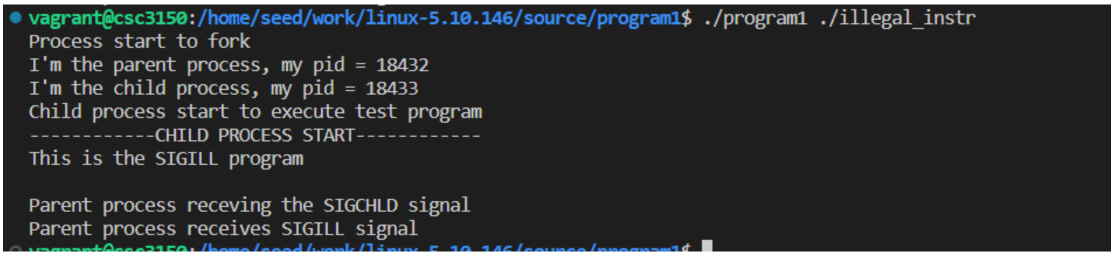
```
vagrant@csc3150:/home/seed/work/linux-5.10.146/source/program1$ ./program1 ./floating
Process start to fork
I'm the parent process, my pid = 18304
I'm the child process, my pid = 18305
Child process start to execute test program
------------CHILD PROCESS START------------
This is the SIGFPE program

Parent process receving the SIGCHLD signal
Parent process receives SIGFPE signal
```



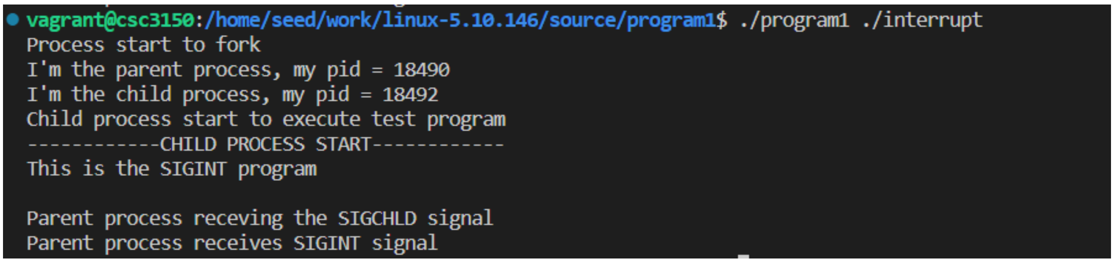
```
vagrant@csc3150:/home/seed/work/linux-5.10.146/source/program1$ ./program1 ./hangup
Process start to fork
I'm the parent process, my pid = 18369
I'm the child process, my pid = 18370
Child process start to execute test program
------------CHILD PROCESS START------------
This is the SIGHUP program

Parent process receving the SIGCHLD signal
Parent process receives SIGHUP signal
```



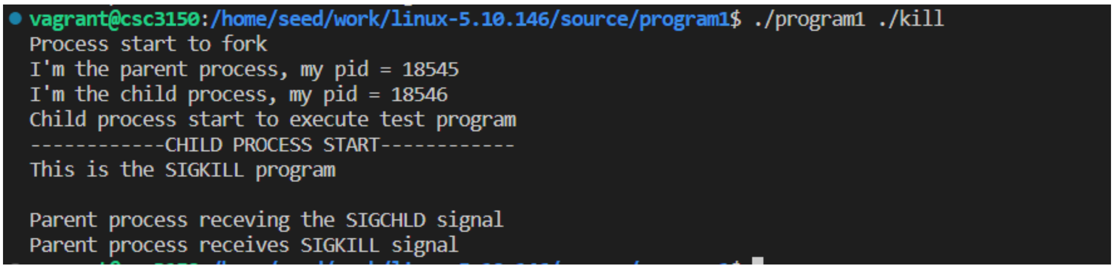
```
vagrant@csc3150:/home/seed/work/linux-5.10.146/source/program1$ ./program1 ./illegal_instr
Process start to fork
I'm the parent process, my pid = 18432
I'm the child process, my pid = 18433
Child process start to execute test program
------------CHILD PROCESS START------------
This is the SIGILL program

Parent process receving the SIGCHLD signal
Parent process receives SIGILL signal
```



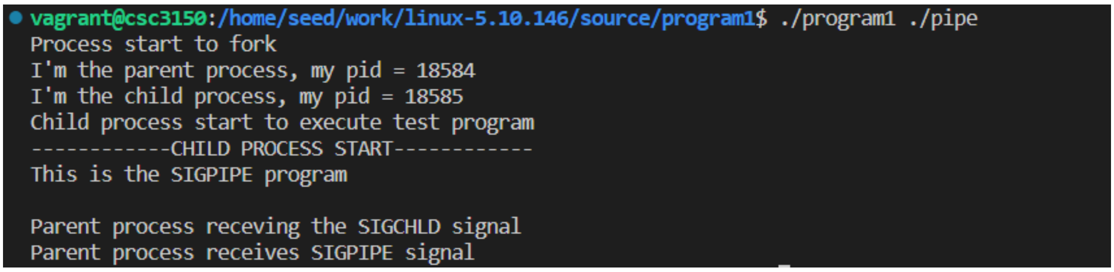
```
vagrant@csc3150:/home/seed/work/linux-5.10.146/source/program1$ ./program1 ./interrupt
Process start to fork
I'm the parent process, my pid = 18490
I'm the child process, my pid = 18492
Child process start to execute test program
------------CHILD PROCESS START------------
This is the SIGINT program

Parent process receving the SIGCHLD signal
Parent process receives SIGINT signal
```



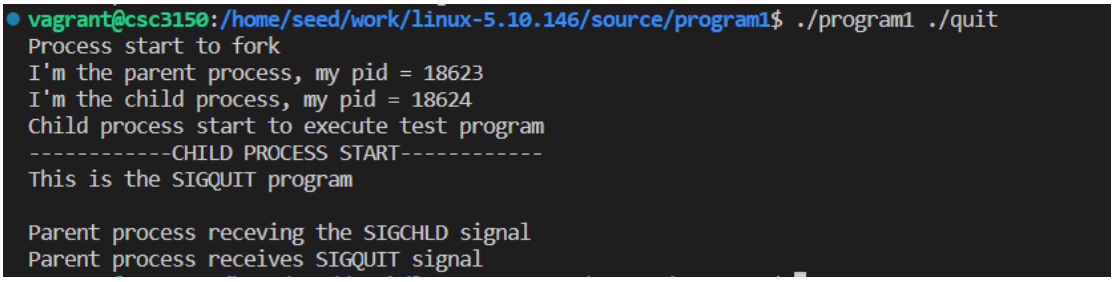
```
vagrant@csc3150:/home/seed/work/linux-5.10.146/source/program1$ ./program1 ./kill
Process start to fork
I'm the parent process, my pid = 18545
I'm the child process, my pid = 18546
Child process start to execute test program
------------CHILD PROCESS START------------
This is the SIGKILL program

Parent process receving the SIGCHLD signal
Parent process receives SIGKILL signal
```



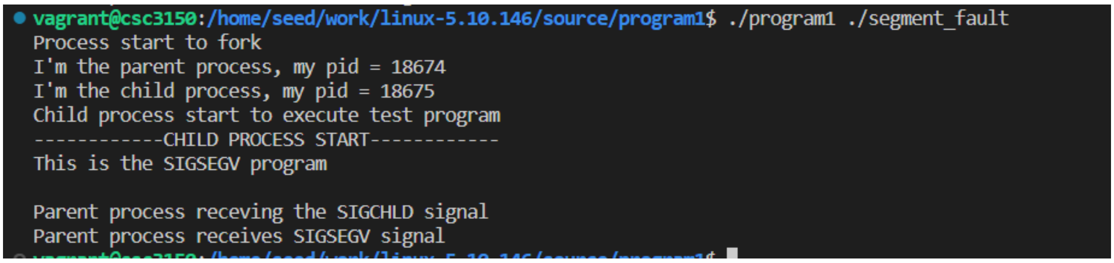
```
vagrant@csc3150:/home/seed/work/linux-5.10.146/source/program1$ ./program1 ./pipe
Process start to fork
I'm the parent process, my pid = 18584
I'm the child process, my pid = 18585
Child process start to execute test program
------------CHILD PROCESS START------------
This is the SIGPIPE program

Parent process receving the SIGCHLD signal
Parent process receives SIGPIPE signal
```



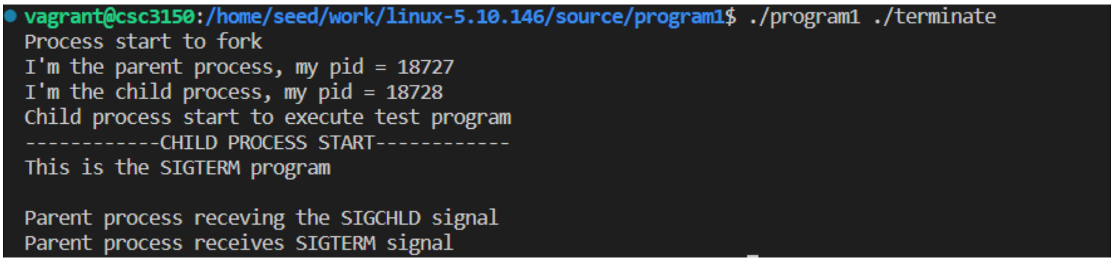
```
● vagrant@csc3150:/home/seed/work/linux-5.10.146/source/program1$ ./program1 ./quit
  Process start to fork
  I'm the parent process, my pid = 18623
  I'm the child process, my pid = 18624
  Child process start to execute test program
  ------------CHILD PROCESS START------------
  This is the SIGQUIT program

  Parent process receving the SIGCHLD signal
  Parent process receives SIGQUIT signal
```



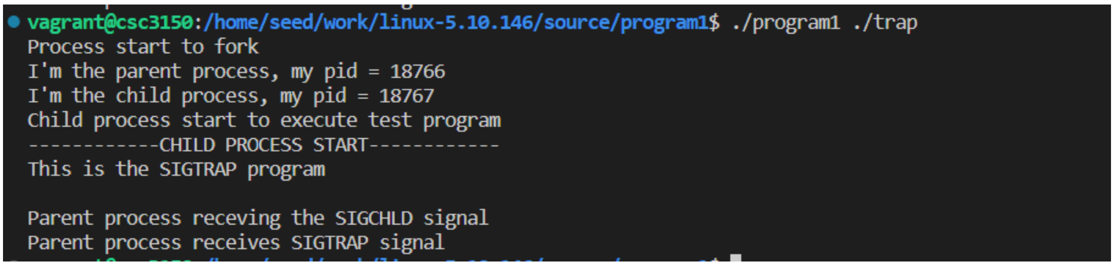
```
● vagrant@csc3150:/home/seed/work/linux-5.10.146/source/program1$ ./program1 ./segment_fault
  Process start to fork
  I'm the parent process, my pid = 18674
  I'm the child process, my pid = 18675
  Child process start to execute test program
  ------------CHILD PROCESS START------------
  This is the SIGSEGV program

  Parent process receving the SIGCHLD signal
  Parent process receives SIGSEGV signal
```



```
● vagrant@csc3150:/home/seed/work/linux-5.10.146/source/program1$ ./program1 ./terminate
  Process start to fork
  I'm the parent process, my pid = 18727
  I'm the child process, my pid = 18728
  Child process start to execute test program
  ------------CHILD PROCESS START------------
  This is the SIGTERM program

  Parent process receving the SIGCHLD signal
  Parent process receives SIGTERM signal
```



```
● vagrant@csc3150:/home/seed/work/linux-5.10.146/source/program1$ ./program1 ./trap
  Process start to fork
  I'm the parent process, my pid = 18766
  I'm the child process, my pid = 18767
  Child process start to execute test program
  ------------CHILD PROCESS START------------
  This is the SIGTRAP program

  Parent process receving the SIGCHLD signal
  Parent process receives SIGTRAP signal
```

Task 2:

1. Design Logic

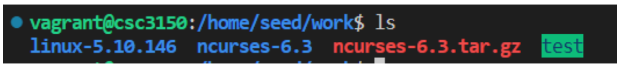
This program aims to create a kernel thread in the kernel space. In this task, we will create a loadable kernel module (LKM) which used to expand the kernel function. It uses part of the kernel functions by adding an EXPORT_SYMBOL in the Linux source code and being exported in the LKM. In the LKM of this task, a new child kernel thread will be created and execute certain programs, while the parent kernel thread will wait for the signals raised by the child thread. After receiving the signals, the output message can be checked in the kernel log. This task is an "inner" version task 1 and reveals more detailed information about process management.

2. Development Environment set up

In this part I will explain how to set up the environment to finish the task 2. Since we need to export some functions from Linux kernel, EXPORT_SYMBOL needed to be added directly to the source code. It means we need to recompile the source code. From

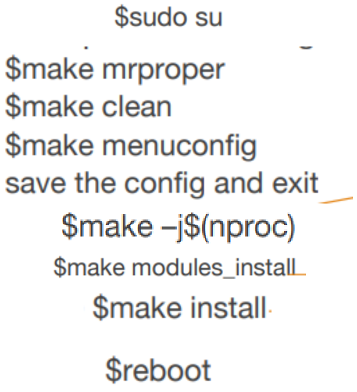
http://www.kernel.org

we can download the linux source code (5.10.146). Next, copy the zip in to the /home/seed/work directory and unzip the file. Then we get



```
vagrant@csc3150:/home/seed/work$ ls
  linux-5.10.146  ncurses-6.3  ncurses-6.3.tar.gz  test
```

Then we copy the ./config from /boot to the linux-5.10.146.
Use following commands to compile the kernel.

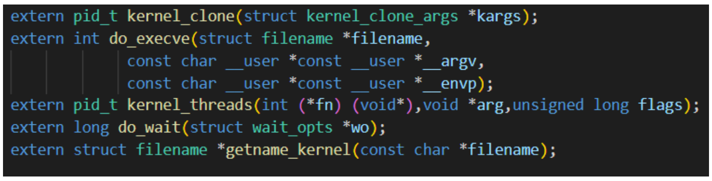


```
$sudo su
$make mrproper
$make clean
$make menuconfig
save the config and exit
$make –j$(nproc)
$make modules_install
$make install
$reboot
```

sudo su: change from vagrant to root.
make mrproper: restore the downloaded file to initial state
make clean: remove all the previous excutable files.
make menuconfig: read .config and gets configurations to be used later
make -j$(nproc) compile with (nproc) kernel threads
make module_install: copy the already compiled module to the /lib/modules
make install: install the already compiled program
reboot: reboot the VM.

3. Functions and variables spec

Before further explanation, here I list the export functions here.

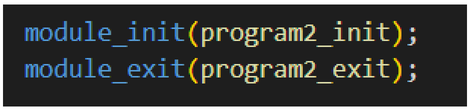


```
extern pid_t kernel_clone(struct kernel_clone_args *kargs);
extern int do_execve(struct filename *filename,
			const char __user *const __user *__argv,
			const char __user *const __user *__envp);
extern pid_t kernel_threads(int (*fn) (void*),void *arg,unsigned long flags);
extern long do_wait(struct wait_opts *wo);
extern struct filename *getname_kernel(const char *filename);
```

I will explain this part by following the executing sequence of how the program flows.

Module_init and module_exit:

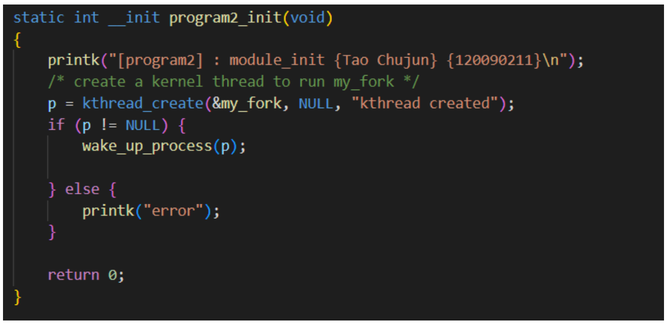
Module_init and module_exit functions are entry point of the LKM. The usage of module_init is to dynamically load a driver program into the kernel, and the usage of module_exit is to dynamically remove the module from the kernel. As we insert a LKM with “insmod” command, a function (the argument of module_init function) will be executed automatically. When we remove a LKM with “remmod” command, a function (the argument of module_exit function) will be called automatically as well. These two functions are macros written in /linux/include/init/h.



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

Program2_init:

This is the implementation of the above program2_init function.

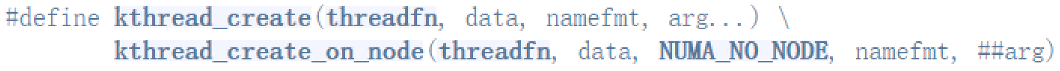


```
static int __init program2_init(void)
{
    printk("[program2] : module_init {Tao Chujun} {120090211}\n");
    /* create a kernel thread to run my_fork */
    p = kthread_create(&my_fork, NULL, "kthread created");
    if (p != NULL) {
        wake_up_process(p);

    } else {
        printk("error");
    }

    return 0;
}
```

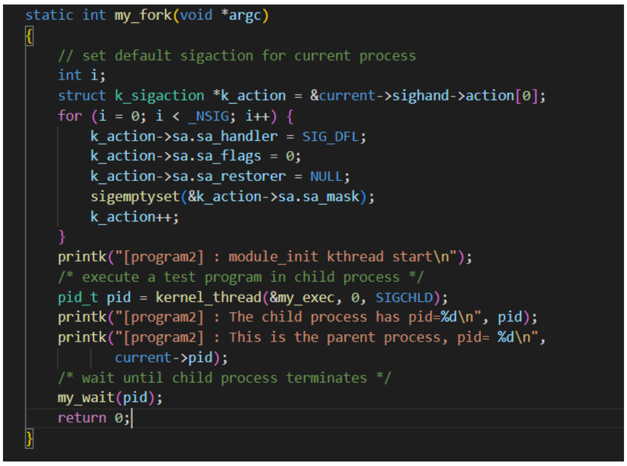
printk will print the message in the kernel log.



```
#define kthread_create(threadfn, data, namefmt, arg...) \
        kthread_create_on_node(threadfn, data, NUMA_NO_NODE, namefmt, ##arg)
```

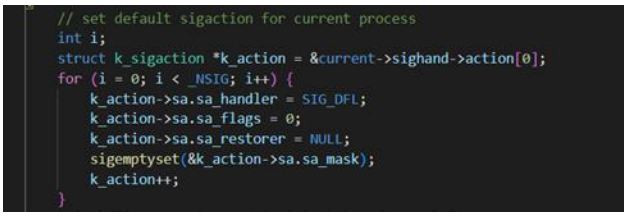
Kthread_create is a function recorded in linux/kernel/kthread.c. This function aims to create a kernel thread. The first argument is a pointer to a function to run after the creation of the process whose argument is the second argument of the kthread_create function. The third argument is the printf-style name for the newly created thread. The return result is a task struct struct task_struct *p;. Task struct is a data structure (PCB) to describe a process, containing information like pid, the state of process, some registers' value etc. If the return pointer is not a NULL pointer, it means a process is successfully created and the wake_up_process will be called to start the child process. In this program, the function to be run is my_fork.

My_fork:



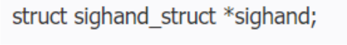
```
static int my_fork(void *argc)
{
    // set default sigaction for current process
    int i;
    struct k_sigaction *k_action = &current->sighand->action[0];
    for (i = 0; i < _NSIG; i++) {
        k_action->sa.sa_handler = SIG_DFL;
        k_action->sa.sa_flags = 0;
        k_action->sa.sa_restorer = NULL;
        sigemptyset(&k_action->sa.sa_mask);
        k_action++;
    }
    printk("[program2] : module_init kthread start\n");
    /* execute a test program in child process */
    pid_t pid = kernel_thread(&my_exec, 0, SIGCHLD);
    printk("[program2] : The child process has pid=%d\n", pid);
    printk("[program2] : This is the parent process, pid= %d\n",
            current->pid);
    /* wait until child process terminates */
    my_wait(pid);
    return 0;
}
```

As is mentioned in the comment, the my_fork function can be divided into three parts. The first part is setting signal handlers.

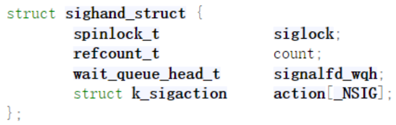


```c
// set default sigaction for current process
int i;
struct k_sigaction *k_action = &current->sighand->action[0];
for (i = 0; i < _NSIG; i++) {
    k_action->sa.sa_handler = SIG_DFL;
    k_action->sa.sa_flags = 0;
    k_action->sa.sa_restorer = NULL;
    sigemptyset(&k_action->sa.sa_mask);
    k_action++;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

It will initiate how it deals with certain signals. 64 (_NSIG) kinds of signals are set. Current is a task_struct pointer of current process. Sighand

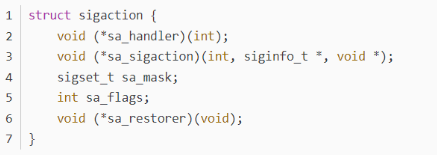


```c
struct sighand_struct *sighand;
```



```c
struct sighand_struct {
        spinlock_t              siglock;
        refcount_t              count;
        wait_queue_head_t       signalfd_wqh;
        struct k_sigaction      action[_NSIG];
};
```

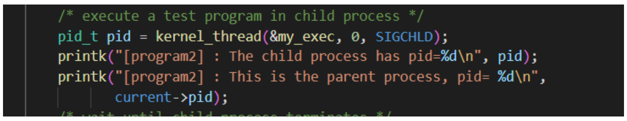
has a sigaction list called action containing a series of signal handler functions.



```c
struct sigaction {
    void (*sa_handler)(int);
    void (*sa_sigaction)(int, siginfo_t *, void *);
    sigset_t sa_mask;
    int sa_flags;
    void (*sa_restorer)(void);
}
```

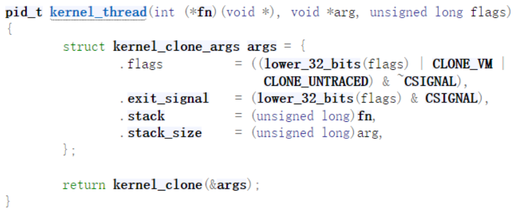
SIG_DEL means it uses default signal handlers. Sa_flags defines the relationship between signals and their handlers. Sigemptyset will initiate a signal set as an empty set. In this case, it will make sure all signal handlers have no blocked signals.

In the second part, a child process will be created and run a testing program.



```c
/* execute a test program in child process */
pid_t pid = kernel_thread(&my_exec, 0, SIGCHLD);
printk("[program2] : The child process has pid=%d\n", pid);
printk("[program2] : This is the parent process, pid= %d\n",
        current->pid);
/* wait until child process terminates */
```

The child process is created by kernel_threads function.



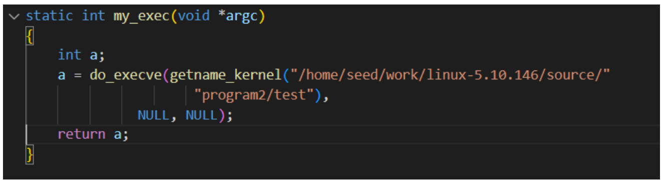
```c
pid_t kernel_thread(int (*fn)(void *), void *arg, unsigned long flags)
{
        struct kernel_clone_args args = {
                .flags          = ((lower_32_bits(flags) | CLONE_VM |
                                    CLONE_UNTRACED) & ~CSIGNAL),
                .exit_signal    = (lower_32_bits(flags) & CSIGNAL),
                .stack          = (unsigned long)fn,
                .stack_size     = (unsigned long)arg,
        };

        return kernel_clone(&args);
}
```

Kernel_threads function will initiate a kernel_clone_args struct and execute kernel_clone functions.

Inside the kernel_clone_args, flags determine how to clone the child process. When executing fork() system call, it is set as SIGCHLD. CLONE_VM means VM shared between processes. CLONE_UNTRACED is set if the tracing process cannot force CLONE_PTRACE on this clone. CSIGNAL means signal mask to be sent at exit. Stack specifies the location of the stack used by the child process. In this case, it is set by the function pointer in the passed arguments. Stack_size is set as 0 since the process is unused yet.

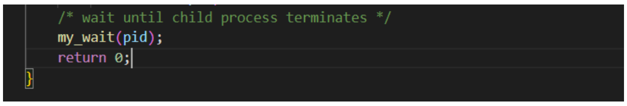
The kernel_clone function assign a available pid to the new stack, create new kernel stack, thread_info structure, and task_struct for the new process and copy parent process information. Then it will add new task to the running queue and wake up the new task. The new task in this program will execute the my_exec function.



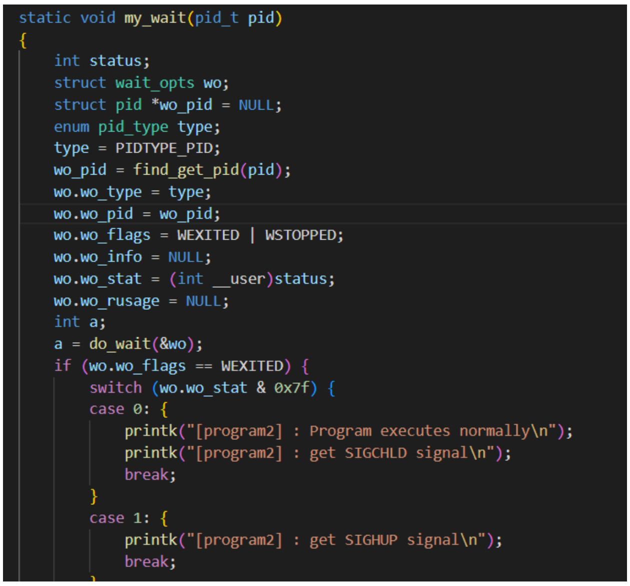
```
static int my_exec(void *argc)
{
    int a;
    a = do_execve(getname_kernel("/home/seed/work/linux-5.10.146/source/"
                                 "program2/test"),
                  NULL, NULL);
    return a;
}
```

Inside my_exec function, getname_kernel will return a filename struct with the given cstring of an executable file ("/home/seed/work/linux-5.10.146/source/program2/test") do_execve will execute the executable file with some given arguments and environment parameters (NULL, NULL). In this program, the testing code will raise some signals.

The third part of the my_fork function is my_wait function. Inside my_wait, struct wait_opts is created for the do_wait function.

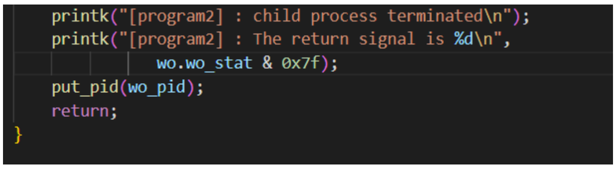


```
    /* wait until child process terminates */
    my_wait(pid);
    return 0;
}
```



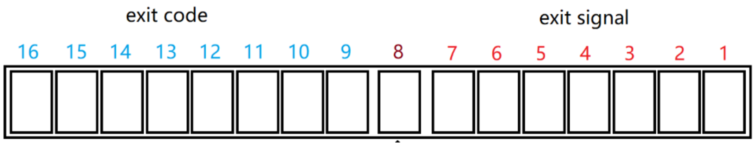
```
static void my_wait(pid_t pid)
{
    int status;
    struct wait_opts wo;
    struct pid *wo_pid = NULL;
    enum pid_type type;
    type = PIDTYPE_PID;
    wo_pid = find_get_pid(pid);
    wo.wo_type = type;
    wo.wo_pid = wo_pid;
    wo.wo_flags = WEXITED | WSTOPPED;
    wo.wo_info = NULL;
    wo.wo_stat = (int __user)status;
    wo.wo_rusage = NULL;
    int a;
    a = do_wait(&wo);
    if (wo.wo_flags == WEXITED) {
        switch (wo.wo_stat & 0x7f) {
        case 0: {
            printk("[program2] : Program executes normally\n");
            printk("[program2] : get SIGCHLD signal\n");
            break;
        }
        case 1: {
            printk("[program2] : get SIGHUP signal\n");
            break;
```

……



```
    printk("[program2] : child process terminated\n");
    printk("[program2] : The return signal is %d\n",
           wo.wo_stat & 0x7f);
    put_pid(wo_pid);
    return;
}
```

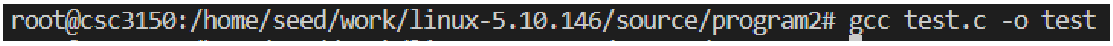
Wait_opts defines how do_wait function behaves. wo_type defines as PIDTYPE_PID, indicating a process ID, find_get_pid gets corresponding struct pid with given pid_t value. Wo_flags set as WEXITED means it waits for processes that have exited. WIFSTOPPED means it will waits for processes that is stopped. The returning signal will be stored into the status. __user here indicates user address space. The returning signal as shown below, is the last 7 bit of the status.



Therefore, we use bit calculation to get the exit signal. With different exit signal, different messages will be output in the kernel log.

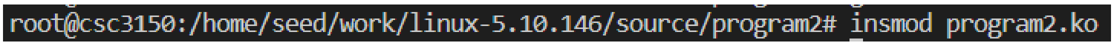
4. Sample Outputs

Compile the test program



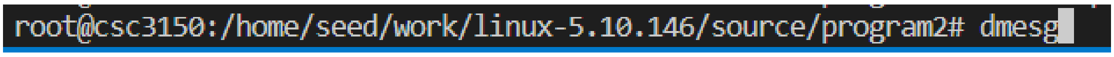
```
root@csc3150:/home/seed/work/linux-5.10.146/source/program2# gcc test.c -o test
```

Insert program2 module



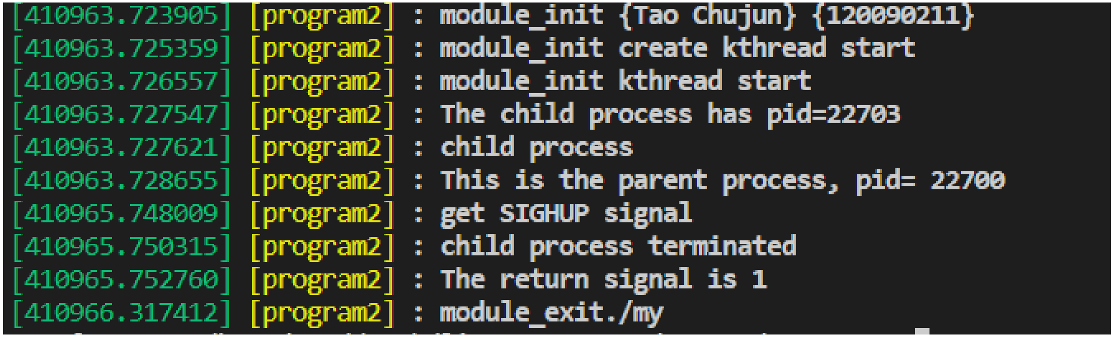
```
root@csc3150:/home/seed/work/linux-5.10.146/source/program2# insmod program2.ko
```

Check the log:



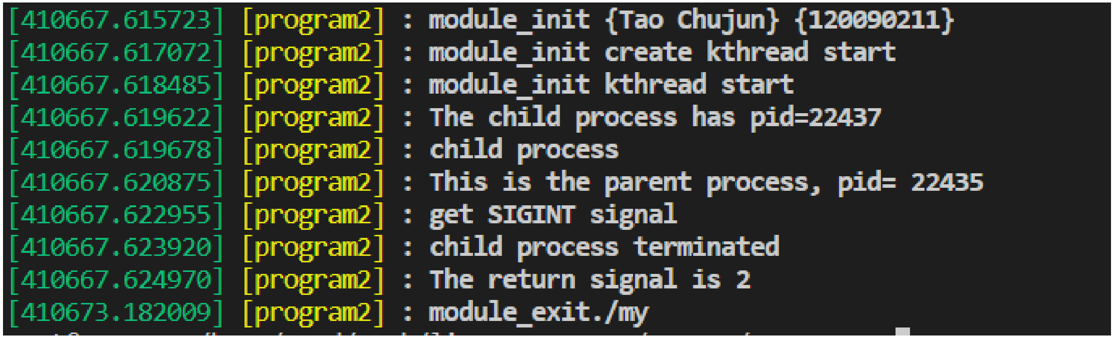
```
root@csc3150:/home/seed/work/linux-5.10.146/source/program2# dmesg
```

- SIGHUP signal



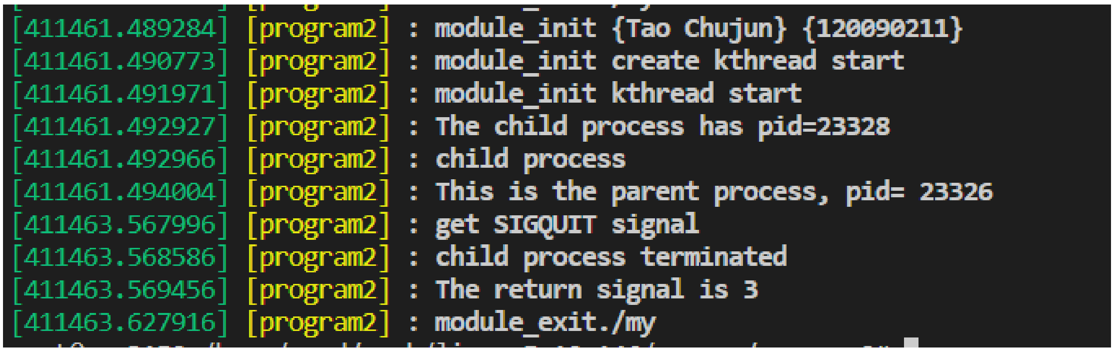
```
[410963.723905] [program2] : module_init {Tao Chujun} {120090211}
[410963.725359] [program2] : module_init create kthread start
[410963.726557] [program2] : module_init kthread start
[410963.727547] [program2] : The child process has pid=22703
[410963.727621] [program2] : child process
[410963.728655] [program2] : This is the parent process, pid= 22700
[410965.748009] [program2] : get SIGHUP signal
[410965.750315] [program2] : child process terminated
[410965.752760] [program2] : The return signal is 1
[410966.317412] [program2] : module_exit./my
```

- SIGINT signal



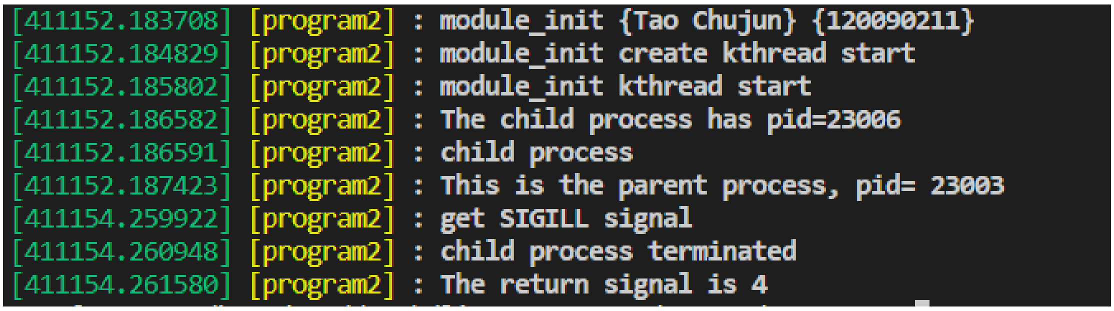
```
[410667.615723] [program2] : module_init {Tao Chujun} {120090211}
[410667.617072] [program2] : module_init create kthread start
[410667.618485] [program2] : module_init kthread start
[410667.619622] [program2] : The child process has pid=22437
[410667.619678] [program2] : child process
[410667.620875] [program2] : This is the parent process, pid= 22435
[410667.622955] [program2] : get SIGINT signal
[410667.623920] [program2] : child process terminated
[410667.624970] [program2] : The return signal is 2
[410673.182009] [program2] : module_exit./my
```

- SIGQUIT signal



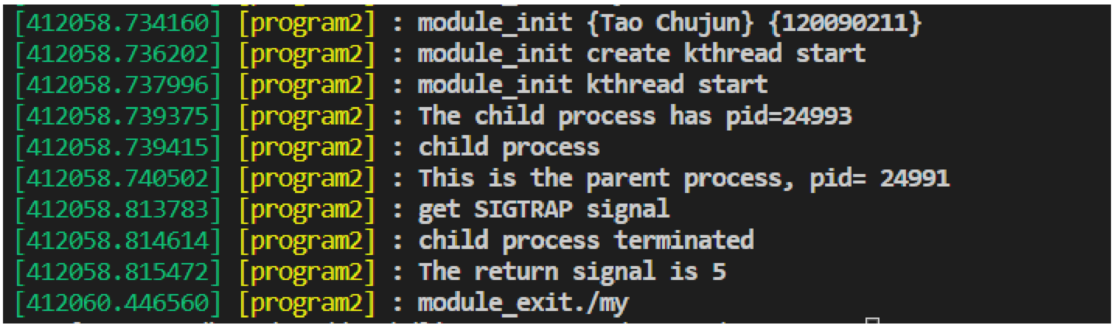
```
[411461.489284] [program2] : module_init {Tao Chujun} {120090211}
[411461.490773] [program2] : module_init create kthread start
[411461.491971] [program2] : module_init kthread start
[411461.492927] [program2] : The child process has pid=23328
[411461.492966] [program2] : child process
[411461.494004] [program2] : This is the parent process, pid= 23326
[411463.567996] [program2] : get SIGQUIT signal
[411463.568586] [program2] : child process terminated
[411463.569456] [program2] : The return signal is 3
[411463.627916] [program2] : module_exit./my
```

- SIGILL signal



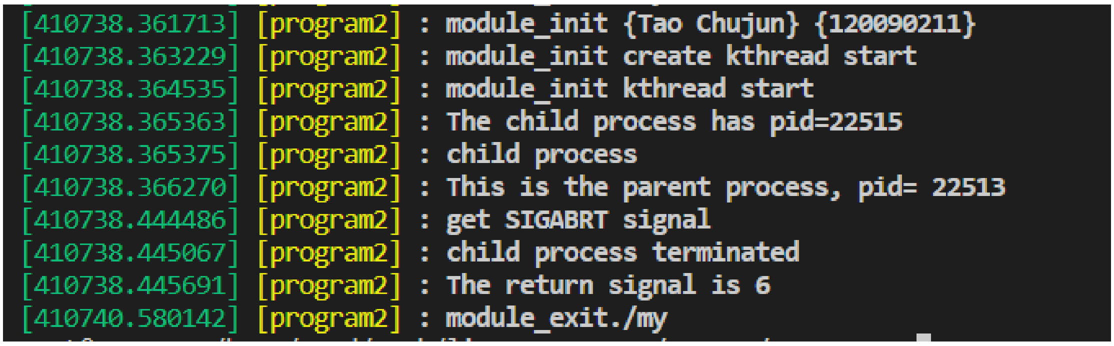
```
[411152.183708] [program2] : module_init {Tao Chujun} {120090211}
[411152.184829] [program2] : module_init create kthread start
[411152.185802] [program2] : module_init kthread start
[411152.186582] [program2] : The child process has pid=23006
[411152.186591] [program2] : child process
[411152.187423] [program2] : This is the parent process, pid= 23003
[411154.259922] [program2] : get SIGILL signal
[411154.260948] [program2] : child process terminated
[411154.261580] [program2] : The return signal is 4
```

- SIGTRAP signal



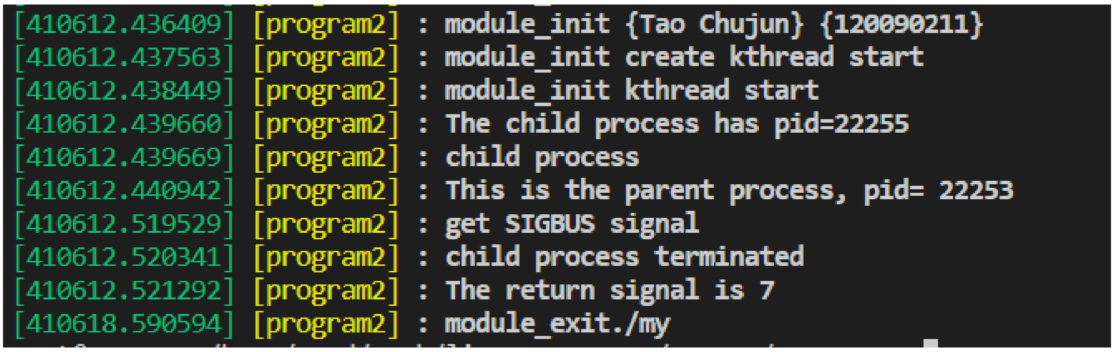
```
[412058.734160] [program2] : module_init {Tao Chujun} {120090211}
[412058.736202] [program2] : module_init create kthread start
[412058.737996] [program2] : module_init kthread start
[412058.739375] [program2] : The child process has pid=24993
[412058.739415] [program2] : child process
[412058.740502] [program2] : This is the parent process, pid= 24991
[412058.813783] [program2] : get SIGTRAP signal
[412058.814614] [program2] : child process terminated
[412058.815472] [program2] : The return signal is 5
[412060.446560] [program2] : module_exit./my
```

- SIGABRT signal



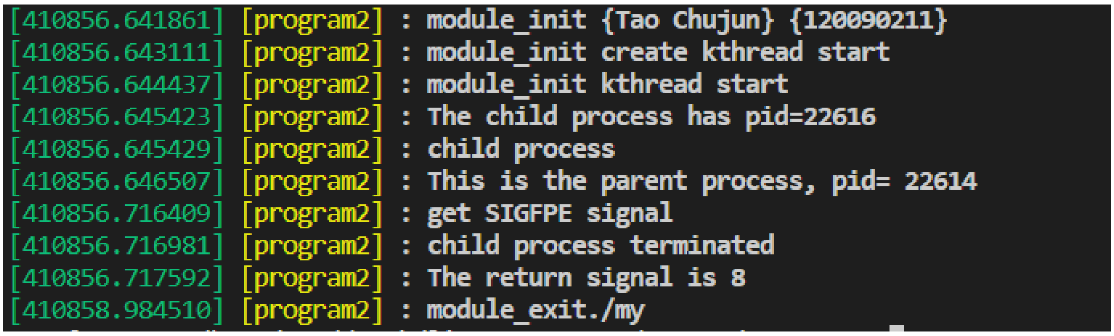
```
[410738.361713] [program2] : module_init {Tao Chujun} {120090211}
[410738.363229] [program2] : module_init create kthread start
[410738.364535] [program2] : module_init kthread start
[410738.365363] [program2] : The child process has pid=22515
[410738.365375] [program2] : child process
[410738.366270] [program2] : This is the parent process, pid= 22513
[410738.444486] [program2] : get SIGABRT signal
[410738.445067] [program2] : child process terminated
[410738.445691] [program2] : The return signal is 6
[410740.580142] [program2] : module_exit./my
```

- SIGBUS signal



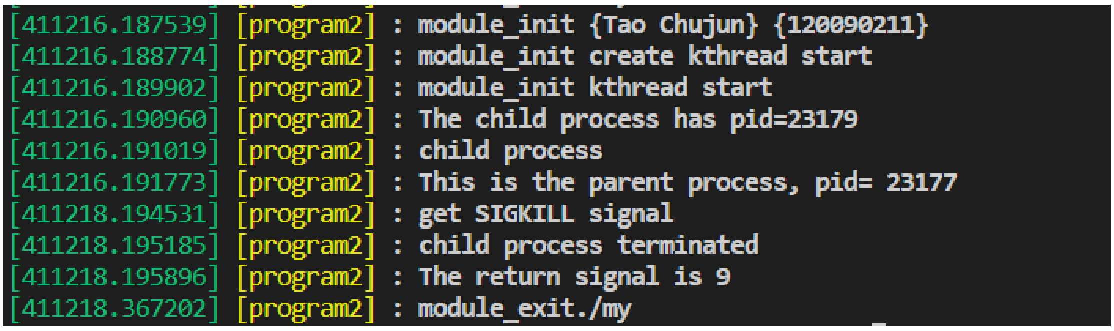
```
[410612.436409] [program2] : module_init {Tao Chujun} {120090211}
[410612.437563] [program2] : module_init create kthread start
[410612.438449] [program2] : module_init kthread start
[410612.439660] [program2] : The child process has pid=22255
[410612.439669] [program2] : child process
[410612.440942] [program2] : This is the parent process, pid= 22253
[410612.519529] [program2] : get SIGBUS signal
[410612.520341] [program2] : child process terminated
[410612.521292] [program2] : The return signal is 7
[410618.590594] [program2] : module_exit./my
```

- SIGFPE signal



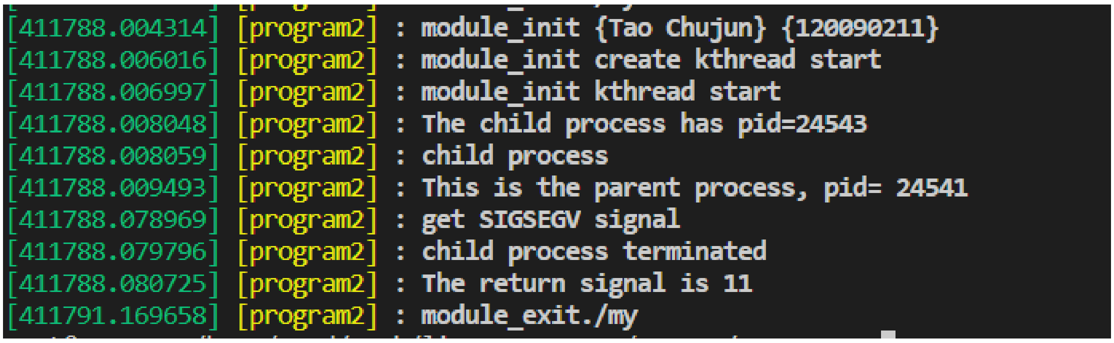
```
[410856.641861] [program2] : module_init {Tao Chujun} {120090211}
[410856.643111] [program2] : module_init create kthread start
[410856.644437] [program2] : module_init kthread start
[410856.645423] [program2] : The child process has pid=22616
[410856.645429] [program2] : child process
[410856.646507] [program2] : This is the parent process, pid= 22614
[410856.716409] [program2] : get SIGFPE signal
[410856.716981] [program2] : child process terminated
[410856.717592] [program2] : The return signal is 8
[410858.984510] [program2] : module_exit./my
```

- SIGKILL signal



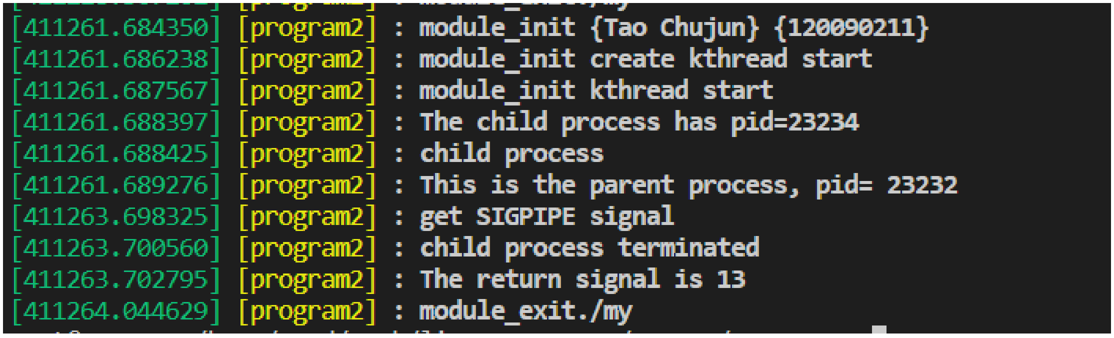
```
[411216.187539] [program2] : module_init {Tao Chujun} {120090211}
[411216.188774] [program2] : module_init create kthread start
[411216.189902] [program2] : module_init kthread start
[411216.190960] [program2] : The child process has pid=23179
[411216.191019] [program2] : child process
[411216.191773] [program2] : This is the parent process, pid= 23177
[411218.194531] [program2] : get SIGKILL signal
[411218.195185] [program2] : child process terminated
[411218.195896] [program2] : The return signal is 9
[411218.367202] [program2] : module_exit./my
```

- SIGSEGV signal



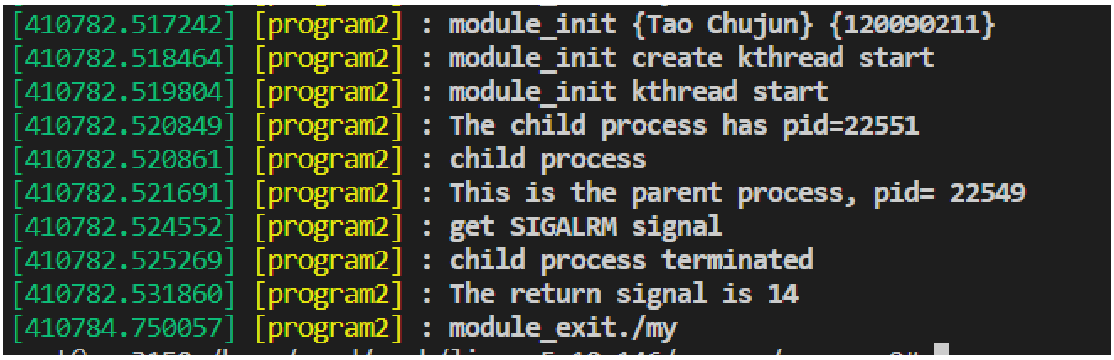
```
[411788.004314] [program2] : module_init {Tao Chujun} {120090211}
[411788.006016] [program2] : module_init create kthread start
[411788.006997] [program2] : module_init kthread start
[411788.008048] [program2] : The child process has pid=24543
[411788.008059] [program2] : child process
[411788.009493] [program2] : This is the parent process, pid= 24541
[411788.078969] [program2] : get SIGSEGV signal
[411788.079796] [program2] : child process terminated
[411788.080725] [program2] : The return signal is 11
[411791.169658] [program2] : module_exit./my
```

- SIGPIPE signal



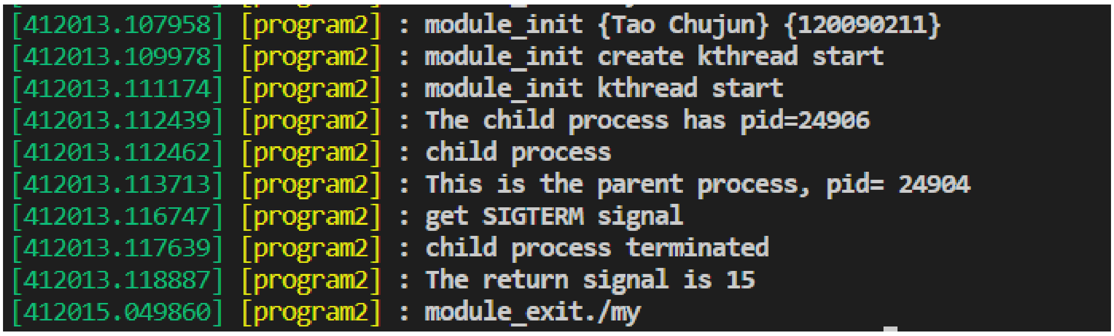
```
[411261.684350] [program2] : module_init {Tao Chujun} {120090211}
[411261.686238] [program2] : module_init create kthread start
[411261.687567] [program2] : module_init kthread start
[411261.688397] [program2] : The child process has pid=23234
[411261.688425] [program2] : child process
[411261.689276] [program2] : This is the parent process, pid= 23232
[411263.698325] [program2] : get SIGPIPE signal
[411263.700560] [program2] : child process terminated
[411263.702795] [program2] : The return signal is 13
[411264.044629] [program2] : module_exit./my
```

- SIGALRM signal



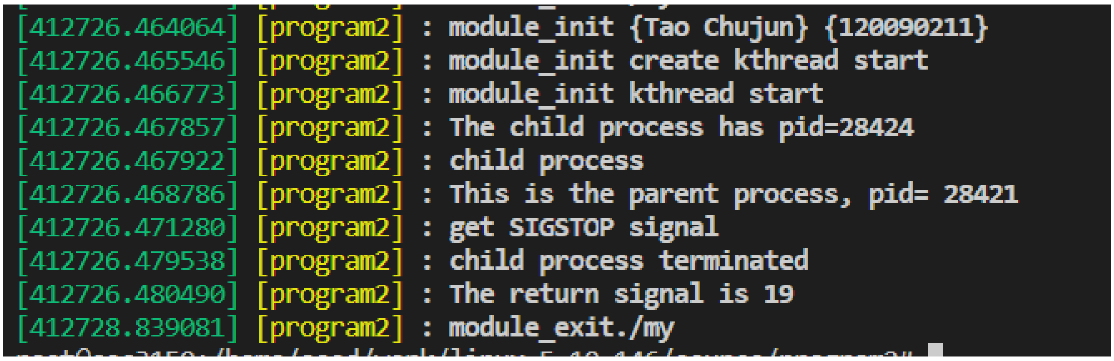
```
[410782.517242] [program2] : module_init {Tao Chujun} {120090211}
[410782.518464] [program2] : module_init create kthread start
[410782.519804] [program2] : module_init kthread start
[410782.520849] [program2] : The child process has pid=22551
[410782.520861] [program2] : child process
[410782.521691] [program2] : This is the parent process, pid= 22549
[410782.524552] [program2] : get SIGALRM signal
[410782.525269] [program2] : child process terminated
[410782.531860] [program2] : The return signal is 14
[410784.750057] [program2] : module_exit./my
```

- SIGTERM signal



```
[412013.107958] [program2] : module_init {Tao Chujun} {120090211}
[412013.109978] [program2] : module_init create kthread start
[412013.111174] [program2] : module_init kthread start
[412013.112439] [program2] : The child process has pid=24906
[412013.112462] [program2] : child process
[412013.113713] [program2] : This is the parent process, pid= 24904
[412013.116747] [program2] : get SIGTERM signal
[412013.117639] [program2] : child process terminated
[412013.118887] [program2] : The return signal is 15
[412015.049860] [program2] : module_exit./my
```

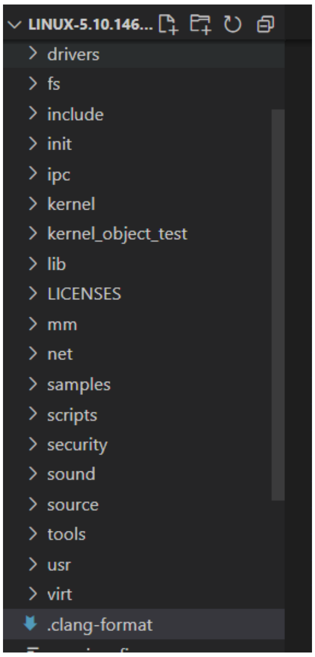
- SIGSTOP signal



```
[412726.464064] [program2] : module_init {Tao Chujun} {120090211}
[412726.465546] [program2] : module_init create kthread start
[412726.466773] [program2] : module_init kthread start
[412726.467857] [program2] : The child process has pid=28424
[412726.467922] [program2] : child process
[412726.468786] [program2] : This is the parent process, pid= 28421
[412726.471280] [program2] : get SIGSTOP signal
[412726.479538] [program2] : child process terminated
[412726.480490] [program2] : The return signal is 19
[412728.839081] [program2] : module_exit./my
```

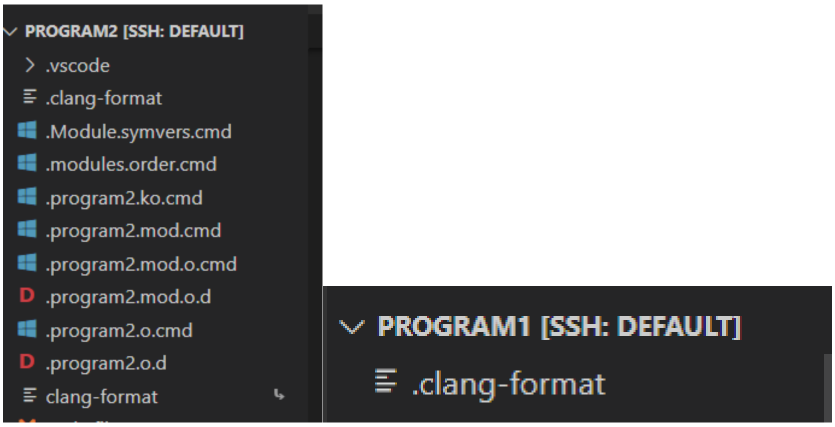
About Formatting:

Use clang format in the kernel source code to format the code.
The clang format is in



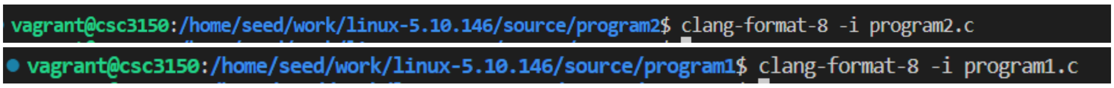
```
LINUX-5.10.146...
drivers
fs
include
init
ipc
kernel
kernel_object_test
lib
LICENSES
mm
net
samples
scripts
security
sound
source
tools
usr
virt
.clang-format
```

Copy the .clang-format to the program1/2 folder. Use following commands to format the code.



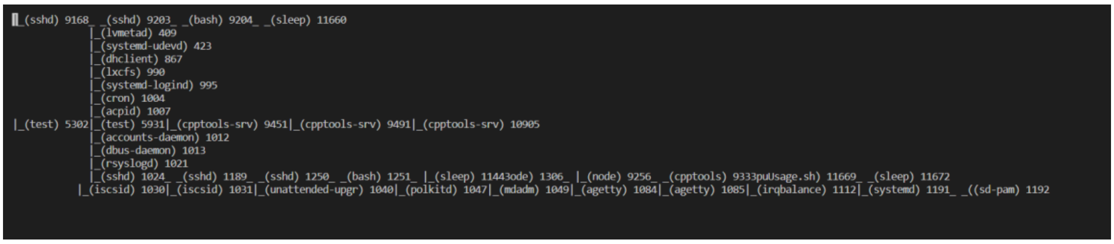
```
PROGRAM2 [SSH: DEFAULT]
.vscode
.clang-format
.Module.symvers.cmd
.modules.order.cmd
.program2.ko.cmd
.program2.mod.cmd
.program2.mod.o.cmd
.program2.mod.o.d
.program2.o.cmd
.program2.o.d
clang-format
```

```
PROGRAM1 [SSH: DEFAULT]
.clang-format
```



```
vagrant@csc3150:/home/seed/work/linux-5.10.146/source/program2$ clang-format-8 -i program2.c
vagrant@csc3150:/home/seed/work/linux-5.10.146/source/program1$ clang-format-8 -i program1.c
```

5. Bonus



```
|_(sshd) 9168_ _(sshd) 9203_ _(bash) 9204_ _(sleep) 11660
             |_(lvmetad) 409
             |_(systemd-udevd) 423
             |_(dhclient) 867
             |_(lxcfs) 990
             |_(systemd-logind) 995
             |_(cron) 1004
             |_(acpid) 1007
|_(test) 5302|_(test) 5931|_(cpptools-srv) 9451|_(cpptools-srv) 9491|_(cpptools-srv) 10905
             |_(accounts-daemon) 1012
             |_(dbus-daemon) 1013
             |_(rsyslogd) 1021
             |_(sshd) 1024_ _(sshd) 1189_ _(sshd) 1250_ _(bash) 1251_ |_(sleep) 11443ode) 1306_ |_(node) 9256_ _(cpptools) 9333puUsage.sh) 11669_ _(sleep) 11672
           |_(iscsid) 1030|_(iscsid) 1031|_(unattended-upgr) 1040|_(polkitd) 1047|_(mdadm) 1049|_(agetty) 1084|_(agetty) 1085|_(irqbalance) 1112|_(systemd) 1191_ _((sd-pam) 1192
```

(not very successful)