report.md

CSC3150 Assignment1 Report

Part 0: Student Information

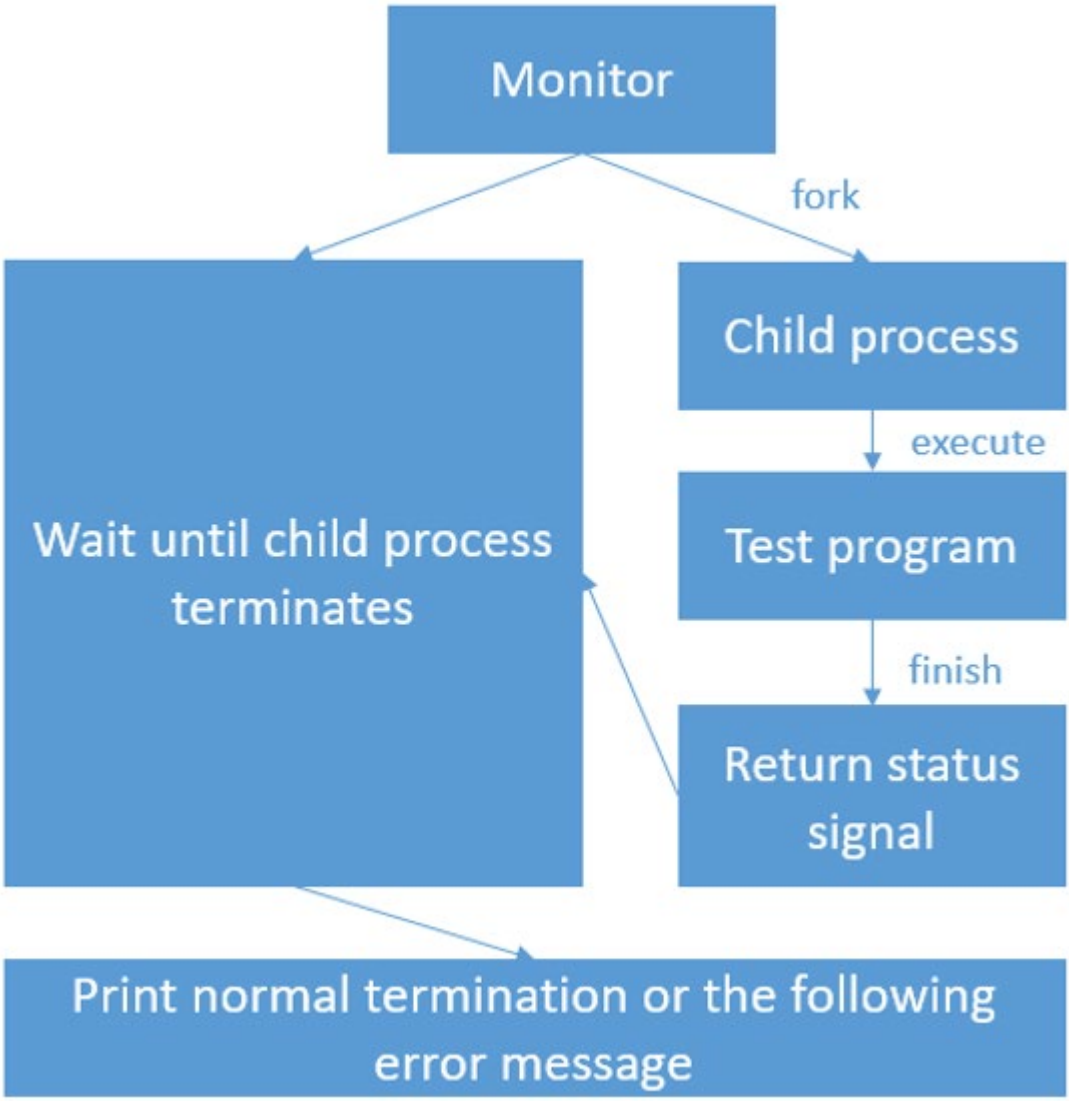
Student ID: 120090575
Name: 杜五洲 DuWuzhou

Part 1: Project Overview

The assignment includes two tasks.
The task 1 mainly focuses on manipulating multiple processes in user-mode of operating system with C POSIX library functions. It mainly includes:

1. Fork a child process to execute 15 different test programs, each with a different signal raised.
2. Let the parent process wait the child process until it exits or stops, according to the raised signal.
3. After receiving the signal of child process, print the normal or abnormal termination information of child process.

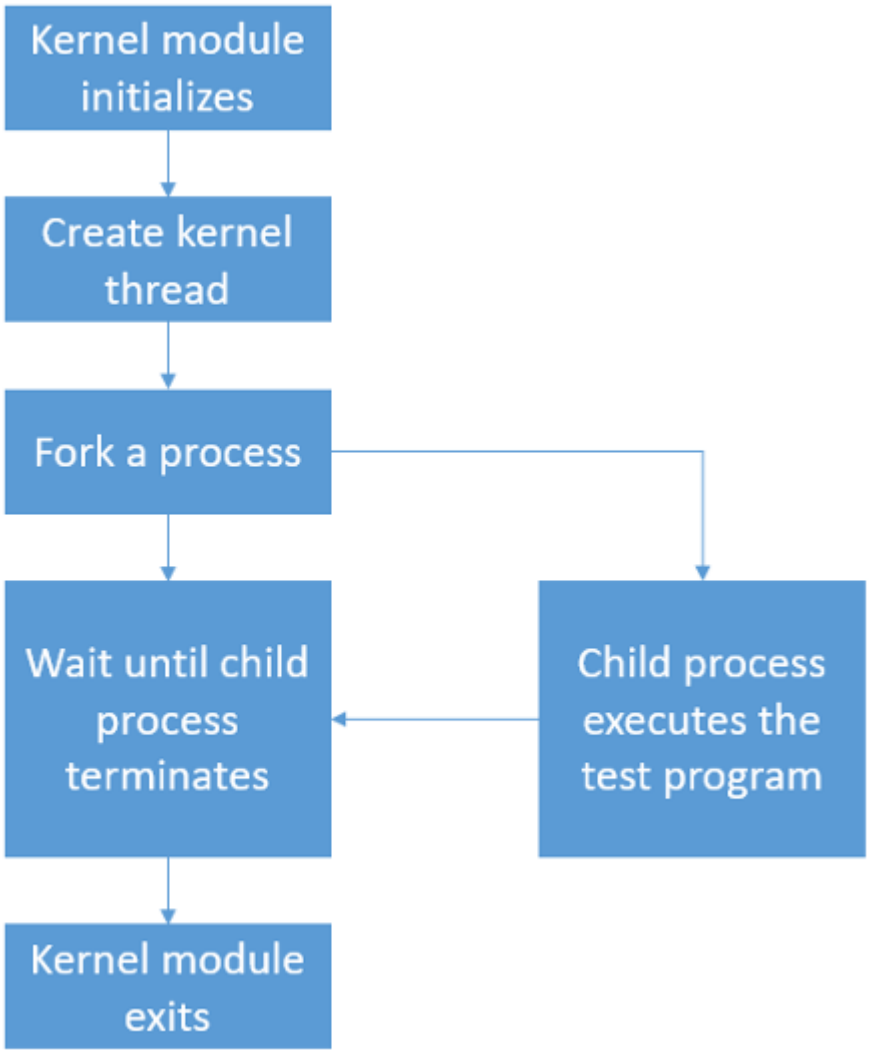
The main flow chart of task 1 is:



The tasks 2 is similar to task 1, both cares about manipulating multiple processes. But task 2 instead focuses on the kernel-mode of operating system rather than in user-mode. What's more, it requires tracing the Linux kernel source code, since in kernel-mode the only feasible APIs or functions are from there. This task mainly includes:

1. Create a kernel thread and run a self-implemented fork function in this thread to fork a child process. If the child process is successfully forked, print the parent process id and child process id.
2. After forking a child process, the parent process should wait the child process until it exits or stops with a self-implemented wait function. In the child process, it executes the test program and receives signal in the execution. This signal will be received by parent process.
3. After receiving the signal, print it according to the received value.

The main flow chart of task 2 is:



Part 2: Code Detail Explanation

program1.c:

```c
int main(int argc, char *argv[])
{
    /* fork a child process */
    int status;
    pid_t pid;

    printf("Process start to fork\n");
    pid = fork();

    if (pid == -1) {
        perror("fork");
        exit(1);
    }
```

Since program1 is in user-mode, so the

code directly begins execution in main function.

- As showed above, the variable `int status` is used to store the return status of function `waitpid`, which will be displayed in the following.
- The variable `pid_t pid` is the process identifier which is used to determine whether the execution is in the parent process or the child process. This variable is the return value of function `fork`.
- The function `fork` forks a child process to execute the program from the next line of code. It will return -1 if forking process fails, else it will return the process identifier of the child process in the parent process's following execution, or it will return 0 in the child process's following execution.

Now we go deep into the `if else` statement blocks. So, as stated above, if `pid` value is -1, it means that the `fork` function fails to fork a child process, the program will raise an error into the terminal.

```c
else {
    // Child Process
    if (pid == 0) {
        int i;
        char *arg[argc];
        for (i = 0; i < argc - 1; i++) {
            arg[i] = argv[i + 1];
        }
        arg[argc - 1] = NULL;

        printf("I'm the Child Process, my pid = %d\n",
               getpid());
        printf("Child process start to execute test program:\n");

        /* execute test program */
        execve(arg[0], arg, NULL);

        printf("Child Process continued!!!!\n");
        perror("execve");
        exit(EXIT_FAILURE);
    }

    // Parent Procee
    else {
        printf("I'm the Parent Process, my pid = %d\n",
               getpid());

        /* wait for child process terminates */

        waitpid(pid, &status, WUNTRACED);
        printf("Parent process receives SIGCHLD signal\n");
```

If `pid` value is not -1, it means that the program successfully forks a child process and enters the following code blocks in both two processes. So, to help program run smoothly in both processes, it is drafted in the style of separating two processes' execution with `if` condition. Now, `pid` is the identifier to distinguish the parent process and child process.

If `pid` value is 0, it means that execution now is the child process. Then the child process created an argument variable `arg` to store the `argv` variable except the first argument as the following executed file.

`execve` function executes the file whose path is `arg[0]` and giving argument to that executable file as `arg` , environment argument is just `NULL` .

If `pid` value is not 0, it means that execution goes on the parent process. So parent process needs to wait the child process through function `waitpid` . The function receives 3 arguments, first is the pid of child process, second is an integer pointer to store the execution status of the child process,

third is the flag when the `waitpid` goes on. This program sets the flag to be `WUNTRACED` since the test program may raise `SIGSTOP` which stops the child process and does not raise exit signal. `WUNTRACED` means return to the parent process until child process stops or exits normally.

Now let's see what happens after receiving signal and `status`.

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

else if (WIFSIGNALED(status)) {
    switch (WTERMSIG(status)) {
    case SIGHUP:
        printf("child process get SIGHUP signal\n");
        break;
    case SIGINT:
        printf("child process get SIGINT signal\n");
        break;
    case SIGQUIT:
        printf("child process get SIGQUIT signal\n");
        break;
    case SIGILL:
        printf("child process get SIGILL signal\n");
        break;
    case SIGTRAP:
        printf("child process get SIGTRAP signal\n");
        break;
    case SIGABRT:
        printf("child process get SIGABRT signal\n");
        break;
    case SIGBUS:
        printf("child process get SIGBUS signal\n");
        break;
    case SIGFPE:
        printf("child process get SIGFPE signal\n");
        break;
    case SIGKILL:
        printf("child process get SIGKILL signal\n");
        break;
    case SIGSEGV:
        printf("child process get SIGSEGV signal\n");
        break;
    case SIGPIPE:
        printf("child process get SIGPIPE signal\n");
        break;
    case SIGALRM:
        printf("child process get SIGALRM signal\n");
        break;
    case SIGTERM:
        printf("child process get SIGTERM signal\n");
        break;
    default:
        printf("Undefined Signal!!!!!");
    }
}

else if (WIFSTOPPED(status)) {
    printf("child process get SIGSTOP signal\n");
}
exit(0);
```

`WIFEXITED` macro queries the child termination status provided by the and `waitpid` function, and determines whether the child process ended normally. If child process ends normally, print the return status value.

`WIFSIGNALED` macro indicates whether the child process exits because of raising a signal. If true, consider the cases of all test signals and print respective information.

`WIFSTOPPED` macro indicates whether the child process stops. If true, print respective information.

program2.c:

```c
static int __init program2_init(void)
{
    printk("[program2] : Module_init %s %d\n", "DuWuzhou", 120090575);

    /* write your code here */

    /* create a kernel thread to run my_fork */
    printk("[program2] : Module_init create kthread start\n");
    task = kthread_create(&my_fork, NULL, "forkProcess");

    if (!IS_ERR(task)) {
        // printk("[program2] : kernel tnread successfully created!!!!\n");
        wake_up_process(task);
    }

    return 0;
}
```

`insmod program2.ko` will enter `program2_init` function. Firstly, create a kernel thread to begin execution of `my_fork` . `kthread_create` function receives three arguments which respectively represent the function will be executed in this thread, the data passed to the function, the process name. If no error detected, wake up this thread.

```c
int my_fork(void *argc)
{
    pid_t pid;
    struct kernel_clone_args args = { .flags = SIGCHLD,
                      .stack = (unsigned long)&my_execve,
                      .stack_size = 0,
                      .parent_tid = NULL,
                      .child_tid = NULL,
                      .tls = 0,
                      .exit_signal = SIGCHLD };

    // 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");
    /* fork a process using kernel_clone or kernel_thread */
    pid = kernel_clone(&args);

    printk("[program2] : The child process has pid = %d\n", pid);
    printk("[program2] : This is the parent process, pid = %d\n", (int)current->pid);

    /* execute a test program in child process */
    my_wait(pid);

    return 0;
}
```

In `my_fork` function, the most significant part is about forking a child process in function `kernel_clone`. This function receive a unique arguments designed for this funcion, which is `struct kernel_clone_args`. `args.flags` is the set of flags of this newly process. `args.stack` is the beginning stack of user space, its size `args.size` is often 0. `args.parent_tid` and `args.child_tid` are both `NULL` since these two are the pids in the user space, which has nothing to do with this program in kernel mode. `args.exit_signal` is just the sent signal when exiting this process.

After calling `kernel_clone`, program gets the returned child process pid. This child pid is passed to execute `my_wait`.

```c
void my_wait(pid_t pid)
{
    // printk("begin my_wait\n");
    int a;
    int status;
    struct wait_opts wo;
    struct pid *wo_pid = NULL;
    enum pid_type type;
    type = PIDTYPE_PID;
    wo_pid = find_get_pid(pid);
    status = 0;

    wo.wo_type = type;
    wo.wo_pid = wo_pid;
    wo.wo_flags = WEXITED | WUNTRACED;
    wo.wo_info = NULL;
    wo.wo_stat = (int __user)status;
    wo.wo_rusage = NULL;

    a = do_wait(&wo);
    status = wo.wo_stat;
```

`my_wait` function just receives one pid argument and passes the pid address to the `wait_opts` `wo_pid` attribute, indicating that the parent process should wait the process with this pid. `wo_flags` indicates what signals the parent receives will trigger the following code execution after `do_wait`. `do_wait` function stops the current process and let it wait until it receives the signal from the process with pid parameter. If successfully execution, it will return value 0 and wo_stat will be the signal raised in the child process.

Let's go deeper in the `status` handling.

```c
if (status < 0b11111111) {
    status &= 0x7f;
    switch (status) {
    case 0:
        printk("[program2] : get SIGCHLD signal\n");
        // printk("[program2] : child process normal termination\n");
        break;

    case 1:
        printk("[program2] : get SIGHUP signal\n");
        // printk("[program2] : child process hung up\n");
        break;

    case 2:
        printk("[program2] : get SIGINT siganl\n");
        // printk("[program2] : child process interrupted\n");
        break;

    case 3:
        printk("[program2] : get SIGQUIT signal\n");
        // printk("[program2] : child process quit\n");
        break;

    case 4:
        printk("[program2] : get SIGILL signal\n");
        // printk("[program2] : child process illegal instruction\n");
        break;

    case 5:
        printk("[program2] : get SIGTRAP signal\n");
        // printk("[program2] : child process trapped\n");
        break;
```

If `status` is in the lower eight bits, it means that the waited process signal exits with signal raised. The lower eighth bit is to show whether core dump. So we can only consider the lower seven bits to get the signal-> `status &= 0x7f`

```
} else {
    status = status >> 8;
    if (status == 19) {
        printk("[program2] : get SIGSTOP signal\n");
        // printk("[program2] : child process stopped\n");
    } else
        printk("[program2] : unkown signal\n");
}

printk("[program2] : child process terminated\n");

printk("[program2] : The return signal is %d\n", status);

put_pid(wo_pid);
// printk("my_wait end\n");

return;
```

If `status` is larger than 0xff, it means the signal is `SIGSTOP` or `SIGTTIN`. The lower eight bits of `status` are 0x7f, the second lower eight bits are the return signal. So doing `status >>= 8` returns the signal value.

Now let's go to child process function `my_execve`.

```
int my_execve(void)
{
    int status;
    const char path[] = "/temp/test";
    struct filename *my_file = getname_kernel(path);

    // printk("[program2] : child process: do my_execve\n");
    printk("[program2] : child process\n");
    status = do_execve(my_file, NULL, NULL);

    // printk("status = %d\n", status);
    if (!status) {
        // printk("test program success!\n");
        return 0;
    }

    printk("not succeed in child process executing do_execve!\n");
    do_exit(status);
}
```

This function is relatively easier. The address of executed file is in `path`. Pass `NULL` for both environment arguments and function arguments. The return status is stored in `status`.

Part 3: How to Run My Code

1. in program1 folder:

```
vagrant@csc3150:~/csc3150/Assignment_1_120090575/source/program1$ make
vagrant@csc3150:~/csc3150/Assignment_1_120090575/source/program1$ ./program1 test
```

2. in program2 folder

```
vagrant@csc3150:~/csc3150/Assignment_1_120090575/source/program2$ make
vagrant@csc3150:~/csc3150/Assignment_1_120090575/source/program2$ sudo insmod
program2.ko
vagrant@csc3150:~/csc3150/Assignment_1_120090575/source/program2$ sudo rmmod
program2
vagrant@csc3150:~/csc3150/Assignment_1_120090575/source/program2$ dmesg -c
```

Part 4: Environment and Compile Kernel

Environment

Distributor ID: Ubuntu Description: Ubuntu 16.04.7 LTS Release: 16.04 Codename: xenial

kernel version: 5.10.146

Compile Kernel

1. using `EXPORT_SYMBOL` macro to expose the functions in kernel source code, like `namei.c` so program can use out side the source code through `extern`. for example: `EXPORT_SYMBOL(kernel_clone);` after the implementation of `kernel_clone` function in `fork.c`, then `extern pid_t kernel_clone(struct kernel_clone_args *args);` in `program2.c`

2. using

```
$make -j$(nproc)
$make modules_install
$make install
$reboot
```

to install the recompile the kernel.

Part 5: What I've learned

1. all the kernel source code functions like `kernel_clone` `do_wait` `do_execve` and `kthread_created` etc. Tracing the source code is such a challenging experience, but I've learnt a lot about how the low-level systems work.

2. The resources on Google are so abundant and I've got a lot hints from others' blog or answers to help me trace code.

Part 6: Output Demo

program1:

```
vagrant@csc3150:~/csc3150/Assignment_1_120090575/source/program1$ ./program1 normal
Process start to fork
I'm the Parent Process, my pid = 4627
I'm the Child Process, my pid = 4628
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/Assignment_1_120090575/source/program1$ ./program1 abort
Process start to fork
I'm the Parent Process, my pid = 4657
I'm the Child Process, my pid = 4658
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_120090575/source/program1$ ./program1 alarm
Process start to fork
I'm the Parent Process, my pid = 4687
I'm the Child Process, my pid = 4688
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_120090575/source/program1$ ./program1 bus
Process start to fork
I'm the Parent Process, my pid = 4744
I'm the Child Process, my pid = 4745
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_120090575/source/program1$ ./program1 floating
Process start to fork
I'm the Parent Process, my pid = 4774
I'm the Child Process, my pid = 4775
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_120090575/source/program1$ ./program1 hangup
Process start to fork
I'm the Parent Process, my pid = 4816
I'm the Child Process, my pid = 4817
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_120090575/source/program1$ ./program1 illegal_instr
Process start to fork
I'm the Parent Process, my pid = 4845
I'm the Child Process, my pid = 4846
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_120090575/source/program1$ ./program1 interrupt
Process start to fork
I'm the Parent Process, my pid = 4863
I'm the Child Process, my pid = 4864
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_120090575/source/program1$ ./program1 kill
Process start to fork
I'm the Parent Process, my pid = 4892
I'm the Child Process, my pid = 4893
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_120090575/source/program1$ ./program1 pipe
Process start to fork
I'm the Parent Process, my pid = 4909
I'm the Child Process, my pid = 4910
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_120090575/source/program1$ ./program1 quit
Process start to fork
I'm the Parent Process, my pid = 4939
I'm the Child Process, my pid = 4940
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGQUIT program

Parent process receives SIGCHLD signal
child process get SIGQUIT signal
vagrant@csc3150:~/csc3150/Assignment_1_120090575/source/program1$ ./program1 segment_fault
Process start to fork
I'm the Parent Process, my pid = 4981
I'm the Child Process, my pid = 4982
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_120090575/source/program1$ ./program1 stop
Process start to fork
I'm the Parent Process, my pid = 5011
I'm the Child Process, my pid = 5012
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_120090575/source/program1$ ./program1 trap
Process start to fork
I'm the Parent Process, my pid = 5043
I'm the Child Process, my pid = 5044
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_120090575/source/program1$ ./program1 terminate
Process start to fork
I'm the Parent Process, my pid = 5073
I'm the Child Process, my pid = 5074
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
```

program2:

```
[17534.258910] [program2] : Module_init DuWuzhou 120090575
[17534.258912] [program2] : Module_init create kthread start
[17534.259104] [program2] : module_init kthread start
[17534.259135] [program2] : The child process has pid = 7791
[17534.259135] [program2] : This is the parent process, pid = 7790
[17534.259166] [program2] : child process
[17534.385162] [program2] : get SIGABRT signal
[17534.385164] [program2] : child process terminated
[17534.385165] [program2] : The return signal is 6
[17537.114837] [program2] : Module_exit ./my
```

```
[17639.194132] [program2] : Module_init DuWuzhou 120090575
[17639.214722] [program2] : Module_init create kthread start
[17639.235687] [program2] : module_init kthread start
[17639.237242] [program2] : The child process has pid = 8310
[17639.237303] [program2] : child process
[17639.239094] [program2] : This is the parent process, pid = 8308
[17641.324504] [program2] : get SIGALRM signal
[17641.368401] [program2] : child process terminated
[17641.408615] [program2] : The return signal is 14
[17646.803877] [program2] : Module_exit ./my
```

```
[17686.494555] [program2] : Module_init DuWuzhou 120090575
[17686.527166] [program2] : Module_init create kthread start
[17686.550780] [program2] : module_init kthread start
[17686.579284] [program2] : The child process has pid = 8744
[17686.579549] [program2] : child process
[17686.581394] [program2] : This is the parent process, pid = 8742
[17686.735507] [program2] : get SIGFPE signal
[17686.766914] [program2] : child process terminated
[17686.789374] [program2] : The return signal is 8
[17687.821657] [program2] : Module_exit ./my
```

```
[17713.930302] [program2] : Module_init DuWuzhou 120090575
[17713.967038] [program2] : Module_init create kthread start
[17713.997662] [program2] : module_init kthread start
[17714.003187] [program2] : The child process has pid = 9133
[17714.003369] [program2] : child process
[17714.010120] [program2] : This is the parent process, pid = 9131
[17714.048276] [program2] : get SIGHUP signal
[17714.079330] [program2] : child process terminated
[17714.100890] [program2] : The return signal is 1
[17715.192610] [program2] : Module_exit ./my
```

```
[17743.464574] [program2] : Module_init DuWuzhou 120090575
[17743.501402] [program2] : Module_init create kthread start
[17743.528163] [program2] : module_init kthread start
[17743.531124] [program2] : The child process has pid = 9533
[17743.531150] [program2] : child process
[17743.560106] [program2] : This is the parent process, pid = 9531
[17743.728707] [program2] : get SIGILL signal
[17743.755219] [program2] : child process terminated
[17743.773581] [program2] : The return signal is 4
[17744.791208] [program2] : Module_exit ./my
```

```
[17769.615078] [program2] : Module_init DuWuzhou 120090575
[17769.644372] [program2] : Module_init create kthread start
[17769.671892] [program2] : module_init kthread start
[17769.673485] [program2] : The child process has pid = 9947
[17769.673514] [program2] : child process
[17769.675398] [program2] : This is the parent process, pid = 9945
[17769.724939] [program2] : get SIGINT siganl
[17769.726608] [program2] : child process terminated
[17769.741158] [program2] : The return signal is 2
[17771.059927] [program2] : Module_exit ./my
```

```
[17790.972072] [program2] : Module_init DuWuzhou 120090575
[17791.009642] [program2] : Module_init create kthread start
[17791.038040] [program2] : module_init kthread start
[17791.039196] [program2] : The child process has pid = 10348
[17791.039221] [program2] : child process
[17791.040199] [program2] : This is the parent process, pid = 10346
[17791.042864] [program2] : get SIGKILL signal
[17791.044461] [program2] : child process terminated
[17791.045270] [program2] : The return signal is 9
[17792.407076] [program2] : Module_exit ./my
```

```
[17817.974318] [program2] : Module_init DuWuzhou 120090575
[17818.001320] [program2] : Module_init create kthread start
[17818.037805] [program2] : module_init kthread start
[17818.052572] [program2] : The child process has pid = 10751
[17818.052597] [program2] : child process
[17818.067128] [program2] : This is the parent process, pid = 10749
[17818.088913] [program2] : get SIGCHLD signal
[17818.112782] [program2] : child process terminated
[17818.135314] [program2] : The return signal is 0
[17819.720302] [program2] : Module_exit ./my
```

```
[17841.217713] [program2] : Module_init DuWuzhou 120090575
[17841.256113] [program2] : Module_init create kthread start
[17841.285897] [program2] : module_init kthread start
[17841.301392] [program2] : The child process has pid = 11149
[17841.301539] [program2] : child process
[17841.303612] [program2] : This is the parent process, pid = 11146
[17841.319266] [program2] : get SIGPIPE signal
[17841.328425] [program2] : child process terminated
[17841.360653] [program2] : The return signal is 13
[17843.394386] [program2] : Module_exit ./my
```

```
[17886.590791] [program2] : Module_init DuWuzhou 120090575
[17886.614469] [program2] : Module_init create kthread start
[17886.629244] [program2] : module_init kthread start
[17886.633927] [program2] : The child process has pid = 11547
[17886.634001] [program2] : child process
[17886.634861] [program2] : This is the parent process, pid = 11545
[17886.759537] [program2] : get SIGQUIT signal
[17886.789898] [program2] : child process terminated
[17886.827000] [program2] : The return signal is 3
[17888.290025] [program2] : Module_exit ./my
```

```
[17916.508863] [program2] : Module_init DuWuzhou 120090575
[17916.539680] [program2] : Module_init create kthread start
[17916.575002] [program2] : module_init kthread start
[17916.591541] [program2] : The child process has pid = 11950
[17916.591807] [program2] : child process
[17916.596325] [program2] : This is the parent process, pid = 11948
[17916.731500] [program2] : get SIGSEGV signal
[17916.756913] [program2] : child process terminated
[17916.786105] [program2] : The return signal is 11
[17917.861016] [program2] : Module_exit ./my
```

```
[17937.459053] [program2] : Module_init DuWuzhou 120090575
[17937.488069] [program2] : Module_init create kthread start
[17937.509504] [program2] : module_init kthread start
[17937.511153] [program2] : The child process has pid = 12349
[17937.511205] [program2] : child process
[17937.514089] [program2] : This is the parent process, pid = 12347
[17937.542521] [program2] : get SIGSTOP signal
[17937.558983] [program2] : child process terminated
[17937.589293] [program2] : The return signal is 19
[17938.968907] [program2] : Module_exit ./my
```

```
[17960.745123] [program2] : Module_init DuWuzhou 120090575
[17960.798156] [program2] : Module_init create kthread start
[17960.821876] [program2] : module_init kthread start
[17960.823492] [program2] : The child process has pid = 12747
[17960.823544] [program2] : child process
[17960.825755] [program2] : This is the parent process, pid = 12745
[17960.872185] [program2] : get SIGTERM signal
[17960.896768] [program2] : child process terminated
[17960.929130] [program2] : The return signal is 15
[17961.940326] [program2] : Module_exit ./my
```

```
[17984.566313] [program2] : Module_init DuWuzhou 120090575
[17984.595620] [program2] : Module_init create kthread start
[17984.633804] [program2] : module_init kthread start
[17984.656122] [program2] : The child process has pid = 13144
[17984.656150] [program2] : child process
[17984.657009] [program2] : This is the parent process, pid = 13142
[17984.818364] [program2] : get SIGTRAP signal
[17984.849550] [program2] : child process terminated
[17984.886677] [program2] : The return signal is 5
[17985.996066] [program2] : Module_exit ./my
```

```
[18148.989912] [program2] : Module_init DuWuzhou 120090575
[18148.991495] [program2] : Module_init create kthread start
[18148.992661] [program2] : module_init kthread start
[18149.029633] [program2] : The child process has pid = 13545
[18149.029688] [program2] : child process
[18149.031574] [program2] : This is the parent process, pid = 13543
[18149.201099] [program2] : get SIGBUS signal
[18149.228326] [program2] : child process terminated
[18149.230444] [program2] : The return signal is 7
[18150.301747] [program2] : Module_exit ./my
```