Report

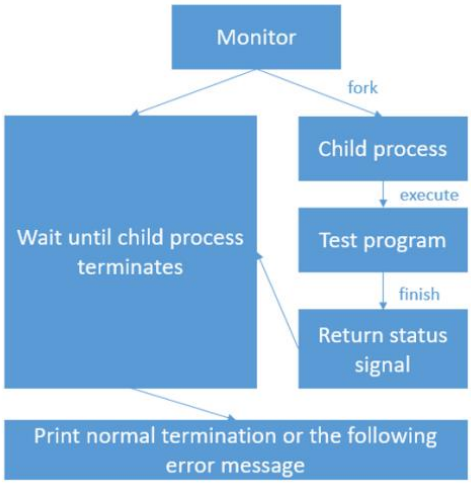
Yifu TIAN
121090517

Program 1

Task Retell

In program 1 we need to fork a child process to execute different test programs(include 1 normal termination and 14 exception cases). The parent process will receive the SIGCHLD signal by wait() function after the finish of child process execution. All the termination information of child process should be printed out. For normal termination, print normal termination and exit status. Else print out the way child process terminates and the signal which was raised in child process.

Here is the main flow chart of Task 1:



Design and Implementation

The main step of program 1 is to fork a child process and execute test program.

Firstly, use fork() function to create a child process.

```
pid_t pid = fork();
```

Then check the value of pid and use “if ... else ...” sentence to do the corresponding operations. If pid is equal to 0, which means the process is exactly the child process, execute the test program.

```
printf("Child process start to execute test program:\n");
/* execute test program */
execve(arg[0], arg, NULL);
```

If pid is equal to -1, which means an error occurs, use exit(1) to tell user the error. If pid is not equal to 0 or -1, which means the process is the parent process and it will wait for the terminated signal from child process. Use waitpid() function to receive signal from the child process.

```
else if(pid>0){
    // pid>0
    /* wait for child process terminates */
    // waitpid(pid, &status, 0);
    pid_t rpid = waitpid(-1,&status,WUNTRACED);

    printf("Parent process receives SIGCHLD signal\n");
```

Finally print ou the detailed information of the signal.

Development Environment Set-up

Version of Operating System: Ubuntu 16.04.7 LTS

```
vagrant@csc3150:~/csc3150/Assignment_1_121090517/source/program1$ cat /etc/os-release
NAME="Ubuntu"
VERSION="16.04.7 LTS (Xenial Xerus)"
```

Version of Kernel: 5.10.196

```
vagrant@csc3150:~/csc3150/Assignment_1_121090517/source/program1$ uname -a
Linux csc3150 5.10.196 #4 SMP Mon Oct 2 07:54:34 UTC 2023 x86_64 x86_64 x86_64 GNU/Linux
```

Execution

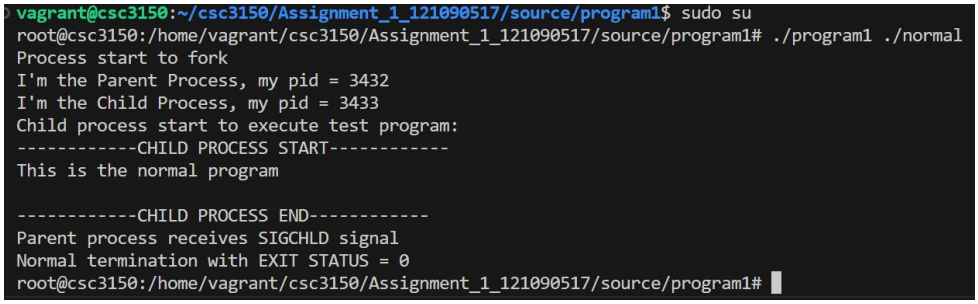
I. Open terminal under the folder of program 1

II. Enter command “sudo make” to compile program

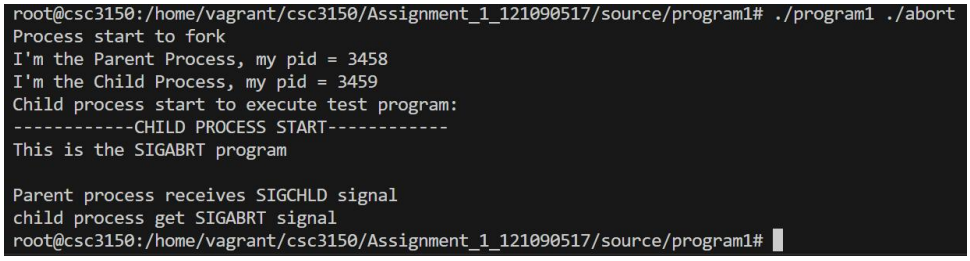
III. Enter command “./program1 ./[TESTPROGRAM]”. Remember to replace the [TESTPROGRAM] to the test file you want. e.g. “./program1 ./abort”

Screenshot of Output

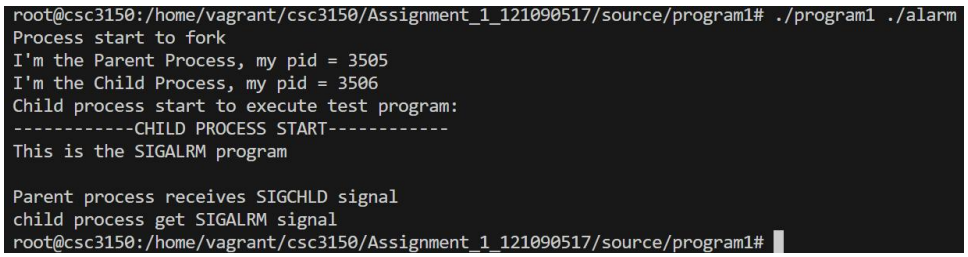
1. Normal



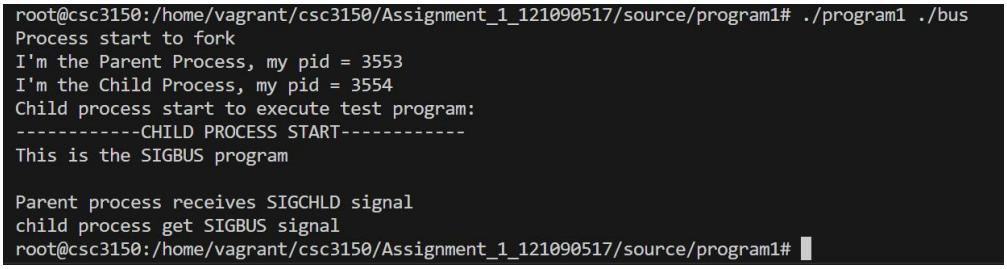
2. Abort



3. Alarm



4. Bus



5. Floating

```
root@csc3150:/home/vagrant/csc3150/Assignment_1_121090517/source/program1# ./program1 ./floating
Process start to fork
I'm the Parent Process, my pid = 3598
I'm the Child Process, my pid = 3599
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
root@csc3150:/home/vagrant/csc3150/Assignment_1_121090517/source/program1#
```

6. Hangup

```
root@csc3150:/home/vagrant/csc3150/Assignment_1_121090517/source/program1# ./program1 ./hangup
Process start to fork
I'm the Parent Process, my pid = 3643
I'm the Child Process, my pid = 3644
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
root@csc3150:/home/vagrant/csc3150/Assignment_1_121090517/source/program1#
```

7. Illegal_instr

```
root@csc3150:/home/vagrant/csc3150/Assignment_1_121090517/source/program1# ./program1 ./illegal_instr
Process start to fork
I'm the Parent Process, my pid = 3687
I'm the Child Process, my pid = 3688
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
root@csc3150:/home/vagrant/csc3150/Assignment_1_121090517/source/program1#
```

8. Interrupt

```
root@csc3150:/home/vagrant/csc3150/Assignment_1_121090517/source/program1# ./program1 ./interrupt
Process start to fork
I'm the Parent Process, my pid = 3732
I'm the Child Process, my pid = 3733
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
root@csc3150:/home/vagrant/csc3150/Assignment_1_121090517/source/program1#
```

9. Kill

```
root@csc3150:/home/vagrant/csc3150/Assignment_1_121090517/source/program1# ./program1 ./kill
Process start to fork
I'm the Parent Process, my pid = 3855
I'm the Child Process, my pid = 3856
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
root@csc3150:/home/vagrant/csc3150/Assignment_1_121090517/source/program1#
```

10. Pipe

```
root@csc3150:/home/vagrant/csc3150/Assignment_1_121090517/source/program1# ./program1 ./pipe
Process start to fork
I'm the Parent Process, my pid = 3908
I'm the Child Process, my pid = 3909
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
root@csc3150:/home/vagrant/csc3150/Assignment_1_121090517/source/program1#
```

11. Quit

```
root@csc3150:/home/vagrant/csc3150/Assignment_1_121090517/source/program1# ./program1 ./quit
Process start to fork
I'm the Parent Process, my pid = 3952
I'm the Child Process, my pid = 3953
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
root@csc3150:/home/vagrant/csc3150/Assignment_1_121090517/source/program1#
```

12. Stop

```
root@csc3150:/home/vagrant/csc3150/Assignment_1_121090517/source/program1# ./program1 ./stop
Process start to fork
I'm the Parent Process, my pid = 4018
I'm the Child Process, my pid = 4019
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
root@csc3150:/home/vagrant/csc3150/Assignment_1_121090517/source/program1#
```

13. Terminate

```
root@csc3150:/home/vagrant/csc3150/Assignment_1_121090517/source/program1# ./program1 ./terminate
Process start to fork
I'm the Parent Process, my pid = 4083
I'm the Child Process, my pid = 4084
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
root@csc3150:/home/vagrant/csc3150/Assignment_1_121090517/source/program1#
```

14. Trap

```
root@csc3150:/home/vagrant/csc3150/Assignment_1_121090517/source/program1# ./program1 ./trap
Process start to fork
I'm the Parent Process, my pid = 4127
I'm the Child Process, my pid = 4128
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
root@csc3150:/home/vagrant/csc3150/Assignment_1_121090517/source/program1#
```

15. Segment_fault

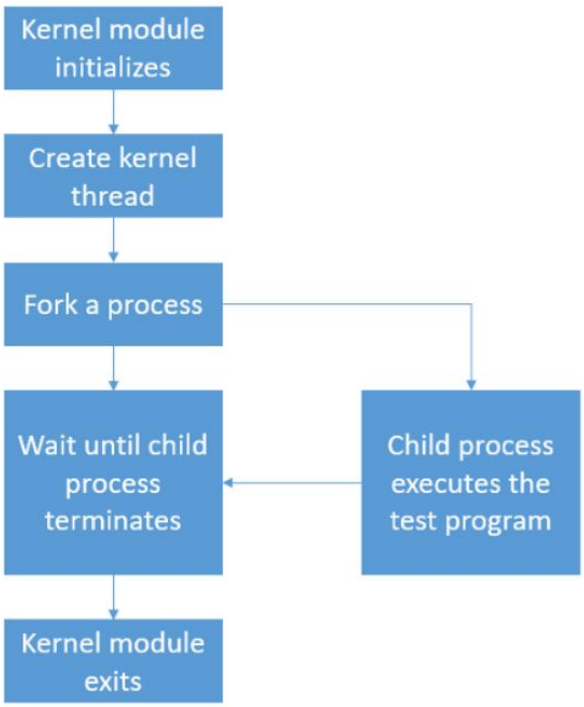
```
root@csc3150:/home/vagrant/csc3150/Assignment_1_121090517/source/program1# ./program1 ./segment_fault
Process start to fork
I'm the Parent Process, my pid = 4214
I'm the Child Process, my pid = 4215
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGSEGV program

Parent process receives SIGCHLD signal
child process get SIGSEGV signal
root@csc3150:/home/vagrant/csc3150/Assignment_1_121090517/source/program1#
```

Program 2

Task Retell

In program 2, create a kernel thread and run my_fork function. In my_fork function, fork a process to execute the test program. The parent process will wait until child process terminates and print out the process id of parent and child process. Also, the raised signal from the test program could be caught and should be printed out in kernel log.

Here is the main flow chart of Task 1:



Design and Implementation

Create a kernel thread and fork a child process to make it to execute another program in the thread. The parent process will wait for the signal from the child process and print out corresponding information.

Firstly, create a kernel thread using kthread_create() to run my_fork() function. Then fork a child process to execute another program.

```
static int __init program2_init(void){

    printk("[program2] : module_init\n");
    printk("[program2] : module_init create kthread start\n");
    /* create a kernel thread to run my_fork */

    // create thread
    fork_task = kthread_create(&my_fork, NULL, "my_fork");
```

Fig1: Create a kernel thread in program2_init()

```
    struct kernel_clone_args args = {
        .flags = SIGCHLD,
        .stack = (unsigned long)&my_exec,
        .stack_size = 0,
        .child_tid = NULL,
        .parent_tid = NULL,
        .exit_signal = SIGCHLD,
        .tls = 0
    };
    /* fork a process using kernel_clone or kernel_thread */
    pid_t pid = kernel_clone(&args);
```

Fig2: Fork a child process to execute another program in my_fork()

In my_fork() function, I implemented several functions to make it all right.

1. my_exec(void)

```c
int my_exec(void){
    int result;
    const char path[] = "/tmp/test";
    struct filename * my_filename = getname_kernel(path);
    if (IS_ERR(my_filename)) {
        printk(KERN_ERR "getname failed with error: %ld\n", PTR_ERR(my_filename));
        return -1;
    }
    result = do_execve(my_filename, NULL, NULL);
    if(!result){
        return 0;
    }
    do_exit(result);
}
```

Fig3: The detailed code of my_exec()

First get the information of the test program by inputting path. (Use getname_kernel() function) Then use do_execve() to execute the test file.

2. printSignal(status)

```c
void printSignal(stat){

    if (stat == 0){
        printk("[program2] : child process exit normally"
               "[program2] : The return signal is %d\n", stat);
    }

    else if (stat == 1){
        printk("[program2] : get SIGHUP signal\n"
               "[program2] : child process is hung up\n"
               "[program2] : The return signal is %d\n", stat);
    }

    else if (stat == 2){
        printk("[program2] : get SIGINT signal\n"
               "[program2] : terminal interrupt\n"
               "[program2] : The return signal is %d\n", stat);
    }

    else if (stat == 131){
        printk("[program2] : get SIGQUIT signal\n"
               "[program2] : terminal quit\n"
               "[program2] : The return signal is %d\n", stat);
    }
```

Fig 4: Part of code in the printSignal()

Input the status of child process, this function will output corresponding message, which is decided upon the test file.

Also, before forking a child process, we need to define a structure “kernle_clone_args” to act as the input parameter of kernel_clone() function. To get the status of child process, use find_get_pid() and do_wait() function, which is shown as below.

```
169     struct wait_opts wo;
170     struct pid *wo_pid = NULL;
171     enum pid_type type;
172     type = PIDTYPE_PID;
173     wo_pid = find_get_pid(pid);
174
175     wo.wo_type = type;
176     wo.wo_pid = wo_pid;
177     wo.wo_flags = WEXITED|WUNTRACED;
178     wo.wo_info = NULL;
179     wo.wo_stat = (int __user*)&status;
180     wo.wo_rusage = NULL;
181     int a = do_wait(&wo);
182     status = wo.wo_stat;
183     put_pid(wo_pid);
```

Fig 5: Get the status of child process

After creating the kernel thread, wake up the program if kthread_create() function creates a thread successfully.

Development Environment Set-up

A. Upgrade the kernel version to 5.10.X

1. Download source code from http://www.kernel.org

2. 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”

3. cd /home/seed/wor and extract the source file using “sudo tar xvf KERNEL_FILE.tar.xz”

4. Clean previous setting and start configuration “$make mrproper” “$make clean” “$make menuconfig” “save the config and exit”

5. Build kernel Image and modules “$make bzImage -j$(nproc)”“$make modules -j$(nproc)”

6. Install kernel modules “$make modules_install” and install kernel “$make install”

7. Reboot to load new kernel “$reboot”

B. Export the functions in kernel files using EXPORT_SYMBOL(kernel_clone in “/kernel/fork.c”, do_exec in “/fs/exec.c”, getname_kernel in “/fs/namei.c”, and do_wait in “kernel/exit.c”).

C. After modifications, recompile the kernel from “make bzImage”, the following steps are the same as A.

Execution

I. Enter “sudo su” to change to root

II. Enter “gcc -o test test.c” to compile the test file

III. Enter “make” under the folder of program 2 to compile program

IV. Enter “insmod program2.ko” to insert the module

V. Enter “rmmod program.ko” to remove the module

VI. Enter “dmesg” in the terminal to display the corresponding message

Screenshot of Output

1. Test.c

```
[10086.618759] [program2] : module_init
[10086.620073] [program2] : module_init create kthread start
[10086.622386] [program2] : module_init kthread start
[10086.752849] [program2] : The child process has pid = 2105
[10086.754339] [program2] : This is the parent process, pid = 2104
[10086.755642] [program2] : child process
[10086.756451] [program2] : get SIGBUS signal
               [program2] : child process has bus error
               [program2] : The return signal is 7
[10090.615692] [program2] : module_exit
root@csc3150:/home/vagrant/csc3150/Assignment_1_121090517/source/program2#
```

2. Normal(normal in program 1)

```
[10556.634270] [program2] : module_init
[10556.634271] [program2] : module_init create kthread start
[10556.634420] [program2] : module_init kthread start
[10556.634912] [program2] : The child process has pid = 6789
[10556.634913] [program2] : This is the parent process, pid = 6787
[10556.634914] [program2] : child process
[10556.634914] [program2] : child process exit normally
               [program2] : The return signal is 0
[10558.777249] [program2] : module_exit
root@csc3150:/home/vagrant/csc3150/Assignment_1_121090517/source/program2#
```

3. Stop(stop in program 1)

```
[ 9830.787113] [program2] : module_init
[ 9830.788307] [program2] : module_init create kthread start
[ 9830.790114] [program2] : module_init kthread start
[ 9830.792025] [program2] : The child process has pid = 1623
[ 9830.793039] [program2] : This is the parent process, pid = 1622
[ 9830.794175] [program2] : child process
[ 9830.794972] [program2] : CHILD PROCESS STOPPED
               [program2] : child process get SIGSTOP signal
               [program2] : The return signal is 19
[ 9834.571363] [program2] : module_exit
root@csc3150:/home/vagrant/csc3150/Assignment_1_121090517/source/program2#
```

Learn from Project 1

In this project program 1, I learned how to fork a child process and moreover, execute a program in this child process using execve() function. Also, I learned that the child process will raise different signal and parent process will receive them. In program 2, I learned how to upgrade/modify kernel and compile the kernel. I have some basic knowledge on the kernel module and learned how to insert/remove the self-defined module. It's quit interesting to realize these processes by hands, which is beneficial to deepen my understanding towards kernel-mode multi-process programming.