Report

Xiang Xinli

120090484

Program1:

Design:

In main program, use fork() to create a child process. After error checking, go child process, use getpid()to print its pid and use execve() to execute the file that we input in parameter, which will raise a signal. Then in parent process, use getpid() to print its pid, then use waitpid() to wait for the status signal that return from child process, use WUNTRACED as a parameter: if the child process is stopped ,then return immediately. To progress the status signal, I use WIFEXIT to check if the child process exit normally, if so, use WEXITSTATUS to get the exit status; use WIFSTOP to check if the child process is stopped, if so, use WSTOPSIG to get the stop signal value; use WIFSIGNALED to check if the child process is terminated by signals, if so, use WTERMSIG to get the terminate signal value, and use if statement to print the corresponding signal.

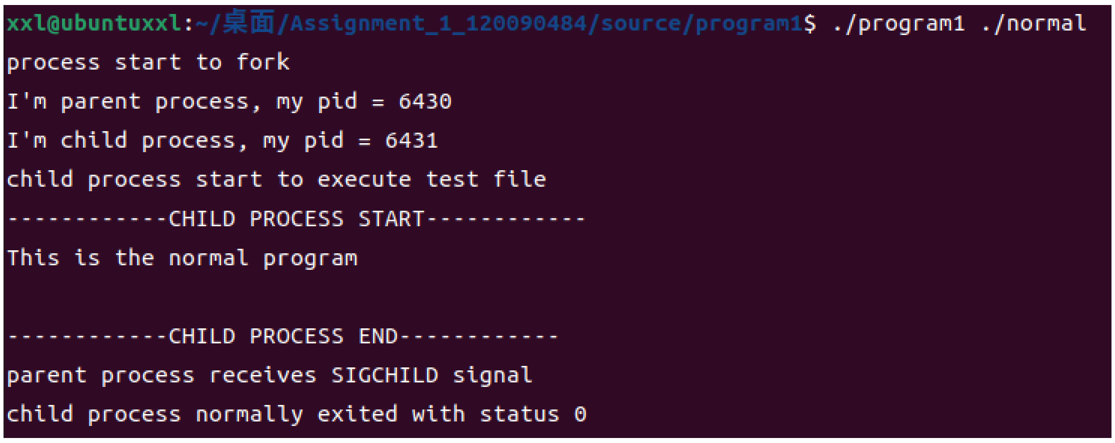
Environment:

Ubuntu22.04, kernel version 5.10.27, set up following by tutoral2.

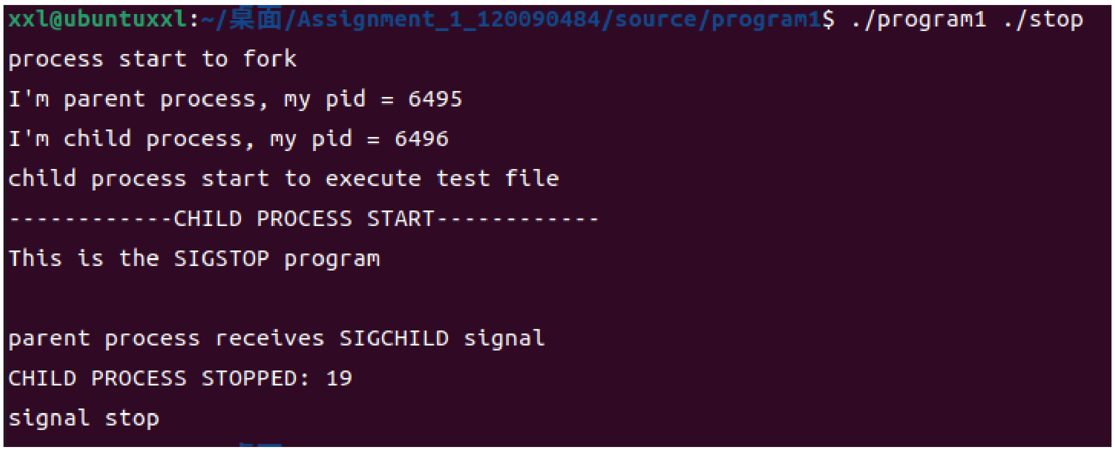
Sample output:

Type “make” to compile .c files

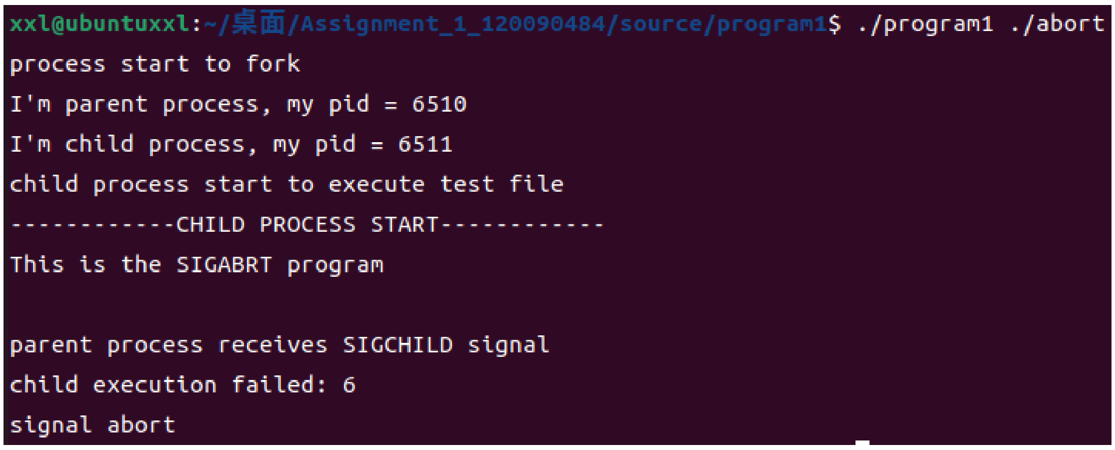
Type ./program1 ./<file name you want to test>



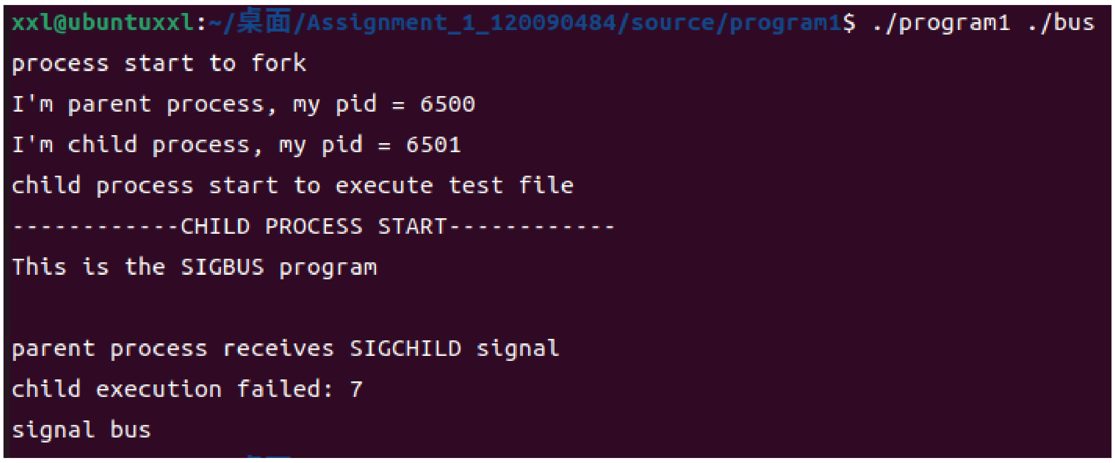
```
xxl@ubuntuxxl:~/桌面/Assignment_1_120090484/source/program1$ ./program1 ./normal
process start to fork
I'm parent process, my pid = 6430
I'm child process, my pid = 6431
child process start to execute test file
------------CHILD PROCESS START------------
This is the normal program

------------CHILD PROCESS END------------
parent process receives SIGCHILD signal
child process normally exited with status 0
```



```
xxl@ubuntuxxl:~/桌面/Assignment_1_120090484/source/program1$ ./program1 ./stop
process start to fork
I'm parent process, my pid = 6495
I'm child process, my pid = 6496
child process start to execute test file
------------CHILD PROCESS START------------
This is the SIGSTOP program

parent process receives SIGCHILD signal
CHILD PROCESS STOPPED: 19
signal stop
```



```
xxl@ubuntuxxl:~/桌面/Assignment_1_120090484/source/program1$ ./program1 ./abort
process start to fork
I'm parent process, my pid = 6510
I'm child process, my pid = 6511
child process start to execute test file
------------CHILD PROCESS START------------
This is the SIGABRT program

parent process receives SIGCHILD signal
child execution failed: 6
signal abort
```



```
xxl@ubuntuxxl:~/桌面/Assignment_1_120090484/source/program1$ ./program1 ./bus
process start to fork
I'm parent process, my pid = 6500
I'm child process, my pid = 6501
child process start to execute test file
------------CHILD PROCESS START------------
This is the SIGBUS program

parent process receives SIGCHILD signal
child execution failed: 7
signal bus
```

Learn:

How to fork a child process and how to process status signal that return from child process.

Program2:

Design:

program2_init() is invoked by module_init(). Then program2_init() call kthread_create() and pass my_fork as a parameter, it will create a new thread to run my_fork(). After wake_up_process(), run my_fork().

In my_fork(), call kernel_clone() and pass my_exec in its parameter to fork a child process to run my_exec(). Then print the pid of child and parent thread. Finally, it will call my_wait() and pass the child thread pid as parameter to process the status signal.

In my_exec(), use do_execve() to execute the test file and use getname_kernel to get filename as the parameter. If the test file is executed successfully, do_execve will return result = 0.

In signal processing part, I feel quite confused. I tried the similar method in program1, but run into bug. So, I searched on the internet and tried many method, finally, I use macros to deal with it. Redefine the macros in the front of the code, and do the similar things with program1.

Reference: http://cs341.cs.illinois.edu/coursebook/Processes

Environment:

Ubuntu22.04, kernel version 5.10.27, set up following by tutoral2.

Use EXPORT_SYMBOL() in /kernel/fork.c, /fs/exec.c, /kernel/exit.c,

/fs/namei.c, then compile kernel again, and extern getname_kernel(),

kernel_clone(), do_execve(), do_wait() to enable program2.c to use these

functions.

Sample output:

Type “gcc -o test test.c”

Type “make”

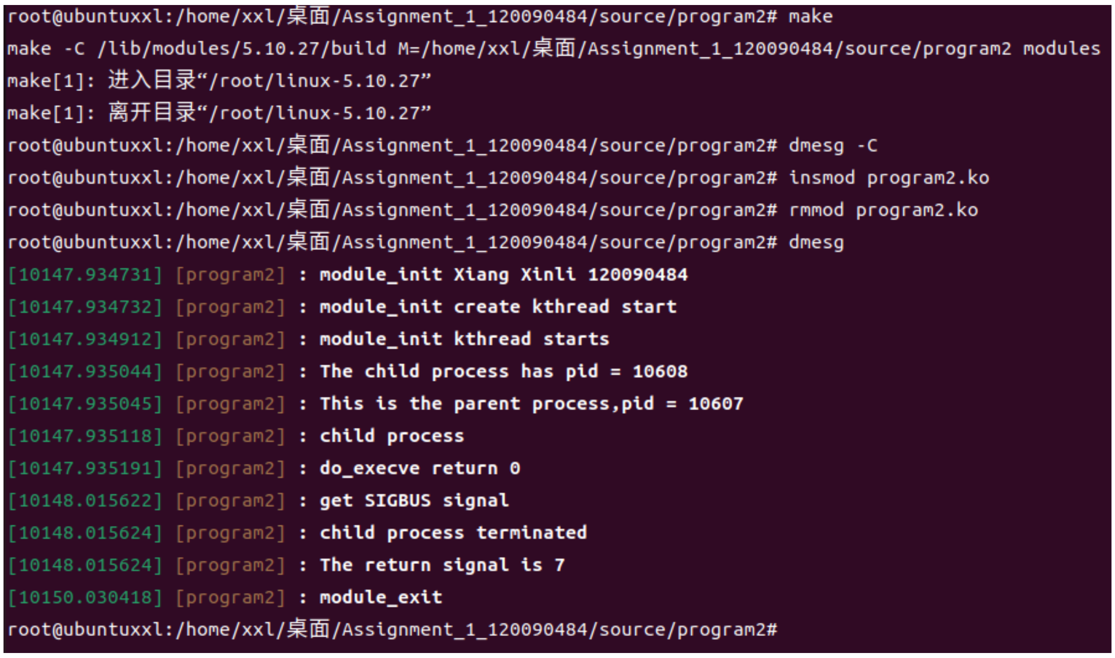
Type “dmesg -C”

Type “insmod program2.ko”

Type “rmmod program2.ko”

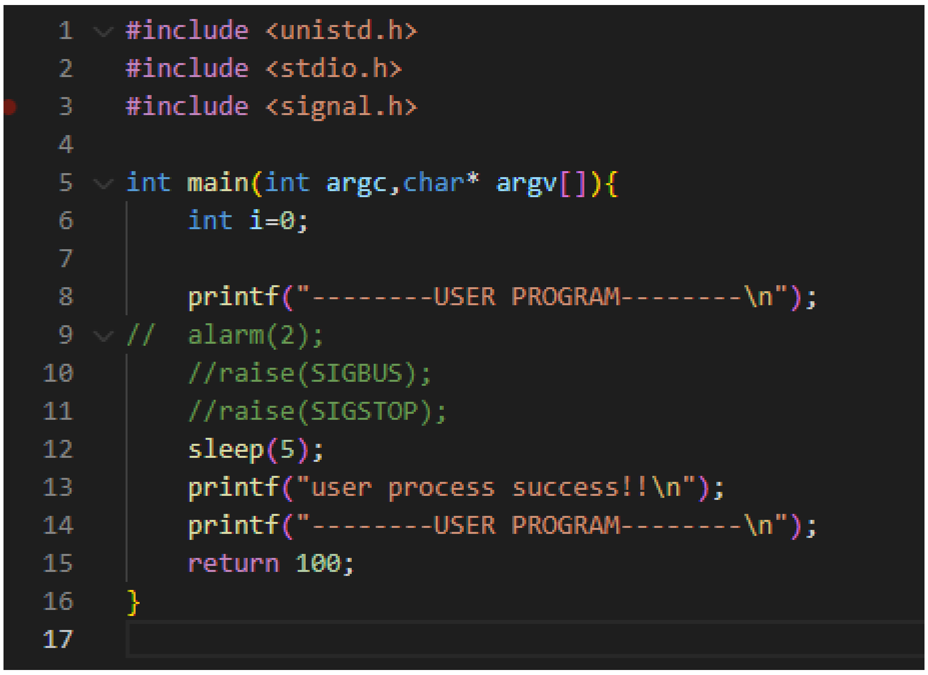
Type “dmesg”

Default test.c, SIGBUS:

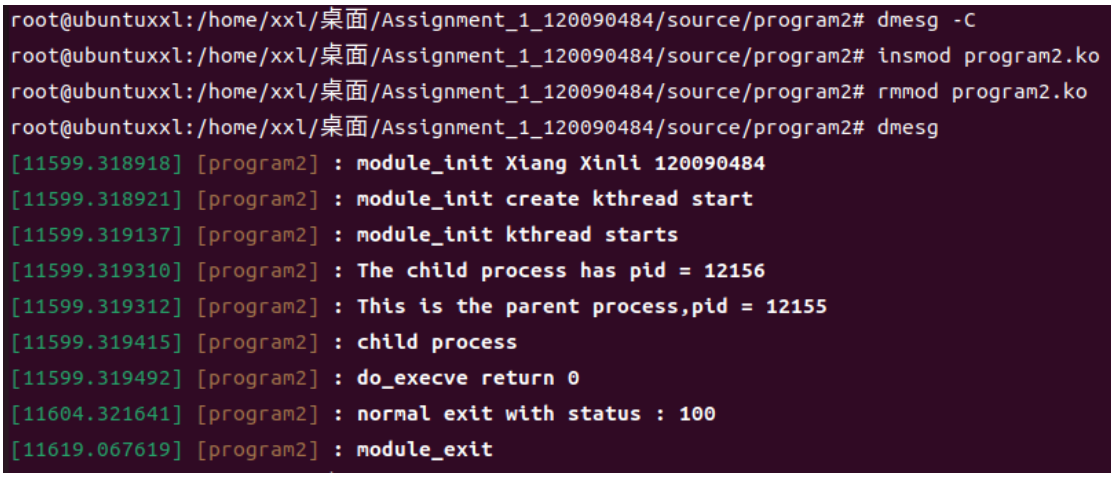


```
root@ubuntuxxl:/home/xxl/桌面/Assignment_1_120090484/source/program2# make
make -C /lib/modules/5.10.27/build M=/home/xxl/桌面/Assignment_1_120090484/source/program2 modules
make[1]: 进入目录“/root/linux-5.10.27”
make[1]: 离开目录“/root/linux-5.10.27”
root@ubuntuxxl:/home/xxl/桌面/Assignment_1_120090484/source/program2# dmesg -C
root@ubuntuxxl:/home/xxl/桌面/Assignment_1_120090484/source/program2# insmod program2.ko
root@ubuntuxxl:/home/xxl/桌面/Assignment_1_120090484/source/program2# rmmod program2.ko
root@ubuntuxxl:/home/xxl/桌面/Assignment_1_120090484/source/program2# dmesg
[10147.934731] [program2] : module_init Xiang Xinli 120090484
[10147.934732] [program2] : module_init create kthread start
[10147.934912] [program2] : module_init kthread starts
[10147.935044] [program2] : The child process has pid = 10608
[10147.935045] [program2] : This is the parent process,pid = 10607
[10147.935118] [program2] : child process
[10147.935191] [program2] : do_execve return 0
[10148.015622] [program2] : get SIGBUS signal
[10148.015624] [program2] : child process terminated
[10148.015624] [program2] : The return signal is 7
[10150.030418] [program2] : module_exit
root@ubuntuxxl:/home/xxl/桌面/Assignment_1_120090484/source/program2#
```

Normal exit:

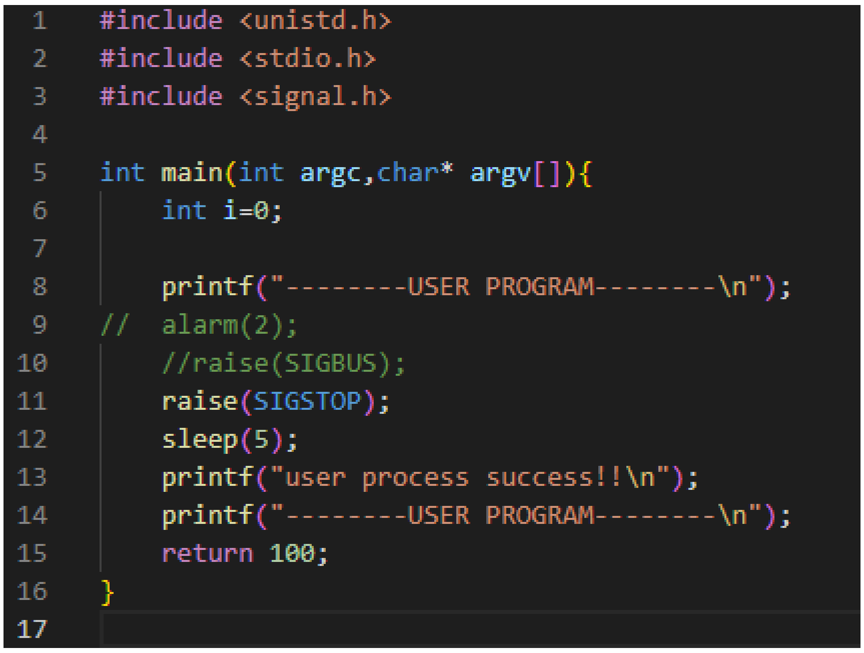


```c
#include <unistd.h>
#include <stdio.h>
#include <signal.h>

int main(int argc,char* argv[]){
    int i=0;

    printf("--------USER PROGRAM--------\n");
//  alarm(2);
    //raise(SIGBUS);
    //raise(SIGSTOP);
    sleep(5);
    printf("user process success!!\n");
    printf("--------USER PROGRAM--------\n");
    return 100;
}

```

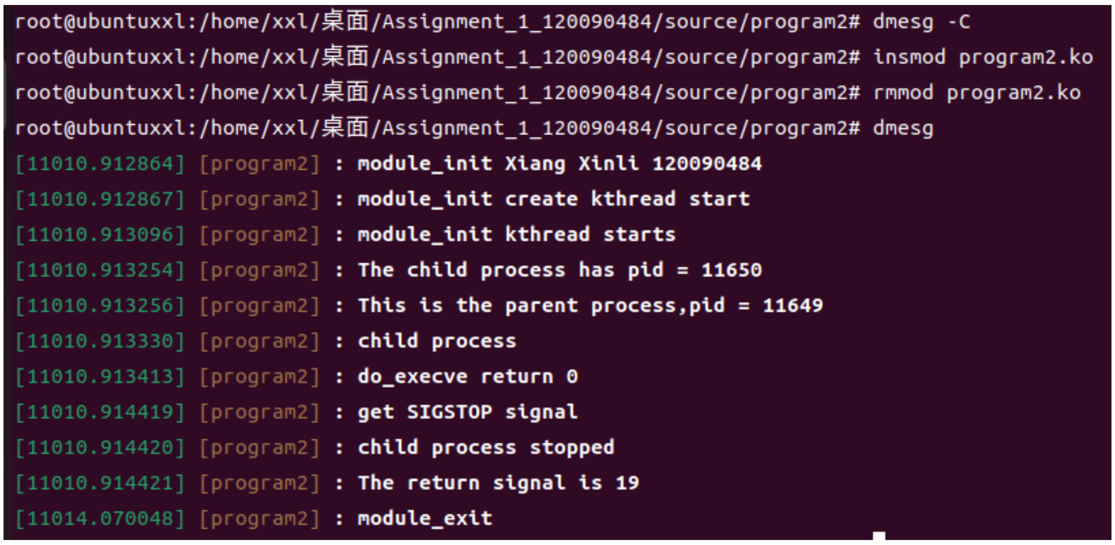


```
root@ubuntuxxl:/home/xxl/桌面/Assignment_1_120090484/source/program2# dmesg -C
root@ubuntuxxl:/home/xxl/桌面/Assignment_1_120090484/source/program2# insmod program2.ko
root@ubuntuxxl:/home/xxl/桌面/Assignment_1_120090484/source/program2# rmmod program2.ko
root@ubuntuxxl:/home/xxl/桌面/Assignment_1_120090484/source/program2# dmesg
[11599.318918] [program2] : module_init Xiang Xinli 120090484
[11599.318921] [program2] : module_init create kthread start
[11599.319137] [program2] : module_init kthread starts
[11599.319310] [program2] : The child process has pid = 12156
[11599.319312] [program2] : This is the parent process,pid = 12155
[11599.319415] [program2] : child process
[11599.319492] [program2] : do_execve return 0
[11604.321641] [program2] : normal exit with status : 100
[11619.067619] [program2] : module_exit
```

SIGSTOP:



```
#include <unistd.h>
#include <stdio.h>
#include <signal.h>

int main(int argc,char* argv[]){
    int i=0;

    printf("--------USER PROGRAM--------\n");
//  alarm(2);
    //raise(SIGBUS);
    raise(SIGSTOP);
    sleep(5);
    printf("user process success!!\n");
    printf("--------USER PROGRAM--------\n");
    return 100;
}

```



```
root@ubuntuxxl:/home/xxl/桌面/Assignment_1_120090484/source/program2# dmesg -C
root@ubuntuxxl:/home/xxl/桌面/Assignment_1_120090484/source/program2# insmod program2.ko
root@ubuntuxxl:/home/xxl/桌面/Assignment_1_120090484/source/program2# rmmod program2.ko
root@ubuntuxxl:/home/xxl/桌面/Assignment_1_120090484/source/program2# dmesg
[11010.912864] [program2] : module_init Xiang Xinli 120090484
[11010.912867] [program2] : module_init create kthread start
[11010.913096] [program2] : module_init kthread starts
[11010.913254] [program2] : The child process has pid = 11650
[11010.913256] [program2] : This is the parent process,pid = 11649
[11010.913330] [program2] : child process
[11010.913413] [program2] : do_execve return 0
[11010.914419] [program2] : get SIGSTOP signal
[11010.914420] [program2] : child process stopped
[11010.914421] [program2] : The return signal is 19
[11014.070048] [program2] : module_exit
```

Learn:

How to read and understand source code. How to create kernel thread and process the status signal that return from child process. How to use macros. How to make kernel module and initialize it.