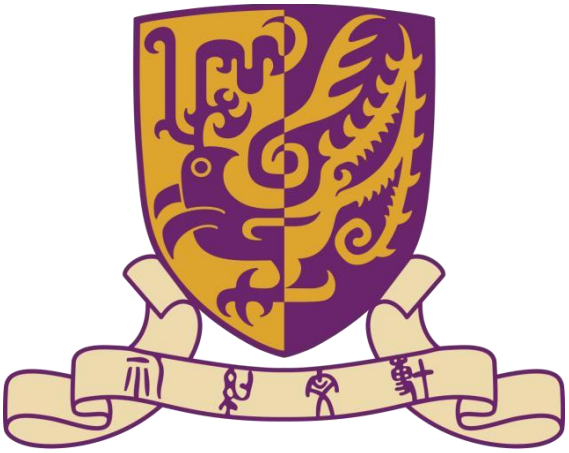

THE CNINESE UNIVERSITY OF HONG KONG, SHENZHEN

CSC3150

Operating Systems

Report for Assignment 1

Author:
Li Jingyu (李璟瑜)

Student Number:
118010141

Oct 11, 2020

Contents

1 Design

Assignment 1 requires us to do some programming about process in user mode and kernel mode. We need to set up the environment for the virtual machine, write the code to deal with process operations, compile the kernel and modify its source code.

Task 1:

Task 1 aims to fork a process, execute the test program and print out the signal information. Since it is based on user mode, we can use the API like fork(), execve() and waitpid() conveniently. Following the guide of tutorial 1, I fork a process first, then for the child process, it will print out the information, get the name of file to be executed, and execute the file; for the parent process, it will wait until the termination of child process. Receiving the status returned by child process, it will judge which signal it is (normal/SIG/stop). Here I use switch-case to do the classification. The whole procedure is clear and easy for task 1.

Task 2:

Task 2 is similar with task 1 in procedure. The difference is that it is written in kernel mode, which means the implementation of fork, execute and wait is more underlying/basic. First a kernel thread is created, running the myfork() function. Myfork function is used to fork a process (my_exec()) to execute the test program. In the my_exec() function, the executable file name is specified, and the kernel API

function do_execve() is responsible for running the .exe file. This is what child process will do. For the parent process, it will wait for the end of its child due to my_wait(). In my_wait() function, the actual waiting action is implemented by the do_wait() function. The information of the child process's status is stored in *wo.wo_stat variable. Just like task 1, I classify the signal according to its value, and then print out the corresponding information to kernel log. Finally, exit the module. (Code part)

As for the part beyond the code, we need to export the symbol of do_action() and getname() functions to make use of them since they belong to the kernel API. After the modification, recompilation of the kernel is needed. The updated modules need to be installed. The difficulty of task 2 distributes a little to the environment setup part.

Bonus

Bonus question is based on user mode. It is the extension of program1 which requires the program to execute several test programs in order. Besides, the process tree (inheritance) and signal information should also be printed out, which is the most difficult part in this question.

Since fork() will create two identical processes with different address spaces, it's hard to record and combine the information. The key to print out the messages is to handle a shared memory which record the pid and signal information of the process. So here I use two text files, one is to store the pid of each process, the other is to store the

signal information. For the execution part, I use recursion to make them execute in order.

First, I clear the two text files for information storage. Then start to execute the program according to the program lists. For each execution, first fork a child process to do the recursion (continue to invoke execute() function), since the last program on list should run the first. When it comes to the last program, execute it using execve() and record its pid information. As for the parent process, waitpid() is used to let it wait for the termination of the child process. Here pid information of the parent will be recorded and its child's pid will be extracted from the text file. The two pid as well as the status will be sent as the parameters to the info2txt() function, which generates the signal information (The classification is similar with task 1 and 2, but no stop signal). Finally it will come to myfork process itself, just print out the information that it executes normally. After execution, extract the information from the two text files and print out the process tree and signals.

Some details:

1. The flag of wo (struct wait_pots) should be set to WEXITED | WUNTRACED to deal with the stop signal.

2. The classification of stop signal in task 2 requires wo_stat to right shift 8 bits (>>8) or its return value is 4991(because of the macro definition and bottom-line design). The exit status of normal should also shift right 8 bits.

2 Environment

OS version: Linux Ubuntu 16.04

Kernel version: 4.10.14

```
[10/11/20]seed@VM:~$ uname -r
4.10.14
```

3 Execution

This part shows the execution steps of the program.

Task 1:

1. Cd to the directory of program 1 (containing all the source codes and makefile)
2. Type “make” (make clean is used to clean and rebuild)
3. Type “./program1 TEST”. TEST is the file to be executed. For example, “./program1 abort”

Task 2:

Before the execution, certain operations to kernel need to be done.

1. Update the kernel source code (add EXPORT_SYMBOL() for 4 functions invoked)
2. Compile the kernel: sudo su; cd to kernel file; make mrproper; make clean; make menuconfig (need to install a tool here); (Recompile start from here) make bzImage; make modules; make modules_install; make install;
3. Reboot
4. (Start execution) Cd to the directory of program 2 (containing all the source codes and makefile).

5. Compile the test program(s): “gcc –o test test.c”. (The .exe files from task 1 can be moved here)
6. Type “make” (“make clean” can clean the rebuild files)
7. Type “sudo insmod program2.ko” to install the module.
8. Type “sudo rmmod program2” to remove the module.
9. Type “dmesg” to check the kernel log.

Bonus:

1. Cd to the directory of bonus (containing all the source codes and makefile)
2. Type “make” (“make clean” is used to clean and rebuild)
3. Type “./myfork test1 test2 …” to execute the programs. For example: “./myfork abort bus normal3 normal9 quit”

4 Output

This part shows some outputs of the programs.

Task 1

1. Normal execution

```
[10/08/20]seed@VM:~/.../program1$ ./program1 ./normal
Process start to fork
I'm the parent process, my pid = 4234
I'm the child process, my pid = 4235
Child process start to execute the program
------------CHILD PROCESS START------------
This is the normal program

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

2. Stop signal

```
[10/08/20]seed@VM:~/.../program1$ ./program1 ./stop
Process start to fork
I'm the parent process, my pid = 4310
I'm the child process, my pid = 4311
Child process start to execute the program
------------CHILD PROCESS START------------
This is the SIGSTOP program

Parent process receiving the SIGCHLD signal
child process get SIGSTOP signal
child process stopped
CHILD PROCESS STOPPED
```

3. Alarm signal

```
[10/08/20]seed@VM:~/.../program1$ ./program1 ./alarm
Process start to fork
I'm the parent process, my pid = 4336
I'm the child process, my pid = 4337
Child process start to execute the program
------------CHILD PROCESS START------------
This is the SIGALRM program

Parent process receiving the SIGCHLD signal
child process get SIGALRM signal
child process releases alarm signal
CHILD EXECUTION FAILED!!
```

4. Illegal instruction signal

```
[10/08/20]seed@VM:~/.../program1$ ./program1 ./illegal_instr
Process start to fork
I'm the parent process, my pid = 4354
I'm the child process, my pid = 4355
Child process start to execute the program
------------CHILD PROCESS START------------
This is the SIGILL program

Parent process receiving the SIGCHLD signal
child process get SIGILL signal
child process has illegal instruction
CHILD EXECUTION FAILED!!
```

5. Terminate signal

```
[10/08/20]seed@VM:~/.../program1$ ./program1 ./terminate
Process start to fork
I'm the parent process, my pid = 4415
I'm the child process, my pid = 4416
Child process start to execute the program
------------CHILD PROCESS START------------
This is the SIGTERM program

Parent process receiving the SIGCHLD signal
child process get SIGTERM signal
child process terminates
CHILD EXECUTION FAILED!!
```

Task 2:

1. Execute the test program

```
[  488.768443] [program2] : Module_init
[  488.768443] [program2] : Module_init create kthread start
[  488.768587] [program2] : Module_init kthread start
[  488.768719] [program2] : The child process has pid = 3850
[  488.768720] [program2] : This is the parent process, pid = 3849
[  488.769312] [program2] : child process get SIGBUS signal
[  488.769313] [program2] : child process has bus error
[  488.769314] [program2] : The return signal is 7
[  498.603415] [program2] : Module_exit
```

2. Execute terminate program

```
 950.519477] [program2] : Module_init
 950.519478] [program2] : Module_init create kthread start
 950.519659] [program2] : Module_init kthread start
 950.519708] [program2] : The child process has pid = 6090
 950.519709] [program2] : This is the parent process, pid = 6089
 950.520671] [program2] : child process get SIGTERM signal
 950.520671] [program2] : child process terminates
 950.520672] [program2] : The return signal is 15
 950.528908] [program2] : Module_exit
```

3. Execute hangup program

```
[ 1021.915564] [program2] : Module_init
[ 1021.915566] [program2] : Module_init create kthread start
[ 1021.915823] [program2] : Module_init kthread start
[ 1021.915939] [program2] : The child process has pid = 6781
[ 1021.915940] [program2] : This is the parent process, pid = 6780
[ 1021.920896] [program2] : child process get SIGHUP signal
[ 1021.920897] [program2] : child process is hung up
[ 1021.920898] [program2] : The return signal is 1
[ 1021.925644] [program2] : Module_exit
```

4. Execute stop program

```
[12918.430184] [program2] : Module_init
[12918.430185] [program2] : Module_init create kthread start
[12918.430710] [program2] : Module_init kthread start
[12918.430800] [program2] : The child process has pid = 29802
[12918.430801] [program2] : This is the parent process, pid = 29800
[12918.432526] [program2] : child process get SIGSTOP signal
[12918.432527] [program2] : child process stopped
[12918.432528] [program2] : The return signal is 19
[12918.437495] [program2] : Module_exit
```

5. Execute normal program

```
[12233.423787] [program2] : Module_init
[12233.423788] [program2] : Module_init create kthread start
[12233.425881] [program2] : Module_init kthread start
[12233.426022] [program2] : The child process has pid = 25656
[12233.426023] [program2] : This is the parent process, pid = 25655
[12233.426805] [program2] : Normal termination
[12233.426806] [program2] : The return signal is 0
[12233.432787] [program2] : Module_exit
```

6. Execute normal program with return value 100

```
[14187.881361] [program2] : Module_init
[14187.881362] [program2] : Module_init create kthread start
[14187.881494] [program2] : Module_init kthread start
[14187.881555] [program2] : The child process has pid = 32668
[14187.881556] [program2] : This is the parent process, pid = 32667
[14187.882125] [program2] : Normal termination
[14187.882126] [program2] : The return signal is 100
[14187.889517] [program2] : Module_exit
```

Task 3:

1. Execute hangup + normal8 + trap

```
[10/11/20]seed@VM:~/.../bonus$ ./myfork hangup normal8 trap
------------CHILD PROCESS START------------
This is the SIGTRAP program

This is normal8 program
------------CHILD PROCESS START------------
This is the SIGHUP program

the process tree : 4796->4797->4799->4800
The child process(pid=4800) of parent process(pid=4799) is terminated by signal
Its signal number = 5
child process get SIGTRAP signal
child process reach a breakpoint

The child process(pid=4799) of parent process(pid=4797) has normal execution
Its exit status = 0

The child process(pid=4797) of parent process(pid=4796) is terminated by signal
Its signal number = 1
child process get SIGHUP signal
child process is hung up

Myfork process(pid=4796) execute normally
```

2. Execute terminate + trap + quit + normal3

```
[10/10/20]seed@VM:~/.../bonus$ ./myfork terminate trap quit normal3
This is normal3 program
------------CHILD PROCESS START------------
This is the SIGQUIT program

------------CHILD PROCESS START------------
This is the SIGTRAP program

------------CHILD PROCESS START------------
This is the SIGTERM program

the process tree : 5214->5215->5216->5217->5218
The child process(pid=5218) of parent process(pid=5217) has normal execution
Its exit status = 0

The child process(pid=5217) of parent process(pid=5216) is terminated by signal
Its signal number = 3
child process get SIGQUIT signal
child process has terminal quit

The child process(pid=5216) of parent process(pid=5215) is terminated by signal
Its signal number = 5
child process get SIGTRAP signal
child process reach a breakpoint

The child process(pid=5215) of parent process(pid=5214) is terminated by signal
Its signal number = 15
child process get SIGTERM signal
child process terminates

Myfork process(pid=5214) execute normally
```

3. Execute many programs …

```
[10/11/20]seed@VM:~/.../bonus$ ./myfork alarm bus hangup interrupt normal10 normal3 s
egment_fault normal9 trap
------------CHILD PROCESS START------------
This is the SIGTRAP program

This is normal9 program
------------CHILD PROCESS START------------
This is the SIGSEGV program

This is normal3 program
This is normal10 program
------------CHILD PROCESS START------------
This is the SIGINT program

------------CHILD PROCESS START------------
This is the SIGHUP program

------------CHILD PROCESS START------------
This is the SIGBUS program

------------CHILD PROCESS START------------
This is the SIGALRM program

the process tree : 22836->22837->22838->22840->22841->22842->22843->22844->22845->228
46
```

```
The child process(pid=22846) of parent process(pid=22845) is terminated by signal
Its signal number = 5
child process get SIGTRAP signal
child process reach a breakpoint

The child process(pid=22845) of parent process(pid=22844) has normal execution
Its exit status = 0

The child process(pid=22844) of parent process(pid=22843) is terminated by signal
Its signal number = 11
child process get SIGSEGV signal
child process has invalid memory segment access

The child process(pid=22843) of parent process(pid=22842) has normal execution
Its exit status = 0

The child process(pid=22842) of parent process(pid=22841) has normal execution
Its exit status = 0

The child process(pid=22841) of parent process(pid=22840) is terminated by signal
Its signal number = 2
child process get SIGINT signal
child process has teminal interrupt

The child process(pid=22840) of parent process(pid=22838) is terminated by signal
Its signal number = 1
child process get SIGHUP signal
child process is hung up
```

```
The child process(pid=22838) of parent process(pid=22837) is terminated by signal
Its signal number = 7
child process get SIGBUS signal
43 GB Volume cess has bus error

The child process(pid=22837) of parent process(pid=22836) is terminated by signal
Its signal number = 14
child process get SIGALRM signal
child process releases alarm signal

Myfork process(pid=22836) execute normally
```

5 Feeling

CSC3150 is another hard course taught by professor Zhong. I think we must prepare well for the challenges.

Assignment 1 itself is not too difficult but troublesome. For the knowledge and coding part, we just need to understand some definitions about the process & kernel and imitate the codes from tutorials. However, we need to set up the environment by ourselves during which there will be mountains of errors and bugs from the system. For me, I am not so familiar with the Linux system and need to search for the solutions of bugs from wechat groups and bb discussion. However, after doing this project I become more skillful in Linux system and virtual machine. I have learned some operations about process, how to compile and modify kernel, how to execute a program in the codes, and so on. It does enhance my computer ability.

Discussion is quite important in handling the large project like this. We encounter different kinds of problems when finishing the tasks, and others may also meet the same trouble. Once we communicate with each other, we can share our experience and help walk out of the difficulty. It does reduce many troubles after discussing with some excellent peers. As for the discussion platform, I think bb forum is more effective than wechat group, since for the latter we may miss some information. (It took me much time to go through the message history this time!)

Tutorial is important. It is totally different from the lectures because it teaches some practical knowledge highly related to the projects. I think I need to treat tutorials seriously.

I can feel that TAs are hard-working and laborious for this course. They have large workloads and will be disturbed by mountains of questions raised by students. Good luck to you!

That’s all.