Assignment_1_Report

120090377 Yuyang LIN

Overview

In this assignment, we are required to accomplish some tasks related with create thread in the OS. Task 1 is about using fork() function in the user mode to create a thread and do some specific tasks. Taks 2 is basically about doing the same thing but in kernel mode. And **I have also accomplished the bonus part to some extent.**

Design-Program-1

In this task, I have write a program called program1.c, which can fork a child process to execute the test program and get the corrected termination information. The overall process can be break down into following parts; fork a child process, make the child process to execute the provided program, wait for the child process to stop/terminate because of a signal/exit normally, catch the information in the parent process and print the formatted information into the terminal. This is basically how I break down the logic and implement the task1 with the help from tutorial 2.

To fork a child process, a function called fork() is used in the user mode, and it will basically copy the current program and start the child process elsewhere. In order to differentiate from parent and child, we got the return value of fork(), which is of type pid_t. if the program is the parent program,

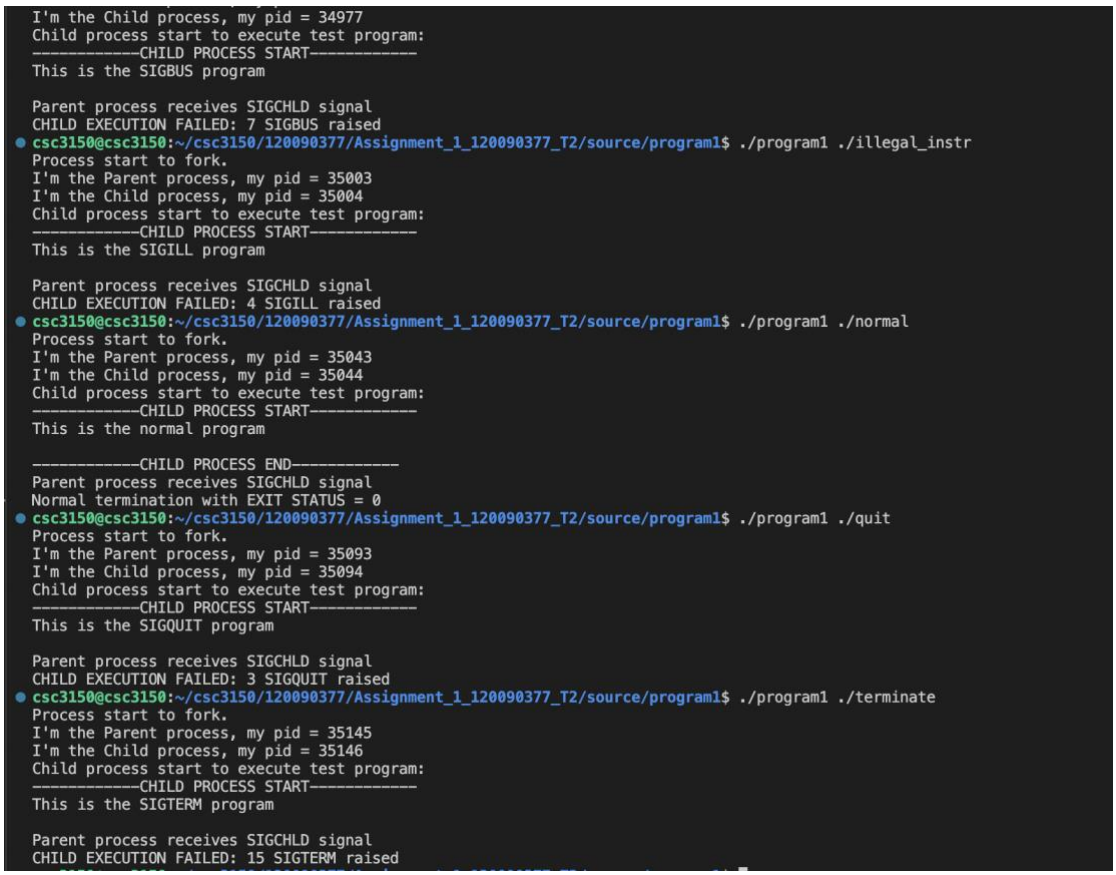
it will return 0 and if it is the child process, we will receive the pid of the child process. I then use a if-else statement do different tasks in different process. if it is the child process, I will pass the path of the executable file into a function called execve() to execute the file in the child process. else in the parent process, I will called a function called waitpid() and pass the pid of the child process. Then, this function will wait the child process to stop/exit/terminate and return the state of child process. After I got the return status of child process, I will use function like WIFEXITED(), WIFSIGNALED(), WIFSTOPPED() to detect the status and WEXITSTATUS(), WTERMSIG(), WSTOPSIG() to evaluate the status and return the correct signal it represents. All above functions in defined in C to handle signals return from operating system in linux.

Program-Output-Screenshot-1

The followings are the screen shot of the result of program 1, after typing make && ./program1 ./another_file in the terminal, you shall see the same result.

```
csc3150@csc3150:~/csc3150/120090377/Assignment_1_120090377_T2/source/program1$ ./program1 ./abort
Process start to fork.
I'm the Parent process, my pid = 34549
I'm the Child process, my pid = 34550
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGABRT program

Parent process receives SIGCHLD signal
CHILD EXECUTION FAILED: 6 SIGABRT raised
csc3150@csc3150:~/csc3150/120090377/Assignment_1_120090377_T2/source/program1$ ./program1 ./floating
Process start to fork.
I'm the Parent process, my pid = 34588
I'm the Child process, my pid = 34589
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGFPE program

Parent process receives SIGCHLD signal
CHILD EXECUTION FAILED: 8 SIGFPE raised
csc3150@csc3150:~/csc3150/120090377/Assignment_1_120090377_T2/source/program1$ ./program1 ./interrupt
Process start to fork.
I'm the Parent process, my pid = 34627
I'm the Child process, my pid = 34628
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGINT program

Parent process receives SIGCHLD signal
CHILD EXECUTION FAILED: 2 SIGINT raised
csc3150@csc3150:~/csc3150/120090377/Assignment_1_120090377_T2/source/program1$ ./program1 ./pipe
Process start to fork.
I'm the Parent process, my pid = 34704
I'm the Child process, my pid = 34705
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGPIPE program

Parent process receives SIGCHLD signal
CHILD EXECUTION FAILED: 13 SIGPIPE raised
csc3150@csc3150:~/csc3150/120090377/Assignment_1_120090377_T2/source/program1$ ./program1 ./segment_fault
Process start to fork.
I'm the Parent process, my pid = 34754
I'm the Child process, my pid = 34755
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGSEGV program

Parent process receives SIGCHLD signal
CHILD EXECUTION FAILED: 11 SIGSEGV raised
csc3150@csc3150:~/csc3150/120090377/Assignment_1_120090377_T2/source/program1$ ./program1 ./trap
Process start to fork.
I'm the Parent process, my pid = 34793
I'm the Child process, my pid = 34794
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGTRAP program
```

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

Parent process receives SIGCHLD signal
CHILD EXECUTION FAILED: 5 SIGTRAP raised
csc3150@csc3150:~/csc3150/120090377/Assignment_1_120090377_T2/source/program1$ ./program1 ./alarm
Process start to fork.
I'm the Parent process, my pid = 34832
I'm the Child process, my pid = 34833
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGALRM program

Parent process receives SIGCHLD signal
CHILD EXECUTION FAILED: 14 SIGALRM raised
csc3150@csc3150:~/csc3150/120090377/Assignment_1_120090377_T2/source/program1$ ./program1 ./hangup
Process start to fork.
I'm the Parent process, my pid = 34862
I'm the Child process, my pid = 34863
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGHUP program

Parent process receives SIGCHLD signal
CHILD EXECUTION FAILED: 1 SIGHUP raised
csc3150@csc3150:~/csc3150/120090377/Assignment_1_120090377_T2/source/program1$ ./program1 ./kill
Process start to fork.
I'm the Parent process, my pid = 34900
I'm the Child process, my pid = 34901
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGKILL program

Parent process receives SIGCHLD signal
CHILD EXECUTION FAILED: 9 SIGKILL raised
csc3150@csc3150:~/csc3150/120090377/Assignment_1_120090377_T2/source/program1$ ./program1 ./stop
Process start to fork.
I'm the Parent process, my pid = 34938
I'm the Child process, my pid = 34939
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGSTOP program

Parent process receives SIGCHLD signal
CHILD PROCESS STOPPED
STOP SIG = 19
csc3150@csc3150:~/csc3150/120090377/Assignment_1_120090377_T2/source/program1$ ./program1 ./bus
Process start to fork.
I'm the Parent process, my pid = 34976
I'm the Child process, my pid = 34977
```



Design-Program-2

The logic of this program is actually quite similar to program 1 instead we have to do all the thing in kernel mode, which is the main difficulty of this task.

The first thing I encountered is that some function can not be directly used in the c program, which are getname_kernel(), do_execve, do_wait and kernel_clone. The process of finding those function and export them is in the set up dev env part in this report together with how I complile the kernel after launching the virtual machine.

In the program2_init(), aftert the program2.ko being initialized, I called a function called kthread_create() to create a kernel thread and run my_fork function. This is done by calling kthread_create(&my_fork, NULL,

“MyThread”);. This will create a thread for this program. The return value of this function is used to detect whether the thread is creared or not. And then the thread will run my_fork() function.

In the my_fork() function, the template of the source code have finish the process of setting the default sigaction for the current process, so I will skip this part.

Then, I fork a child process just as what I did in the program 1. But for the kernel mode, I have to called a function called kernel_clone(). This function is also what the fork() in the user mode will call, so I basically just follow this process. To call kernel_clone, I have to pass an argument called kernel_clone_args, which is a struct defined in the kernel source code. I set the flags to SIGCHLD, .stack to point to &my_child_stack, which is the starting point of the function I want the child process execute and set the stack_size to 0, child_tid and parent_tid to NULL as what the fork() will do. By this way, I successfully created a child process and makes it to execute the function I want, which is my_child_stack().

In the my_child_stack(), the only thing I do is to make it run the test program in a given directory. To do this in the kernel mode, I have to called a function called getnam_kernel to convert a path into a struct called filename, which is used in the kernel to indicate the location of a file. And then, I have to called a function called do_execve(), which is also what execve() in the user mode will call. At first, I thought I have to pass three

parameters as path, argv, envp to call this function according to the source code. But I found out this will not make do_execve() to function properly. After long time of googling, debugging, I found out that if I make the argv and envp to NULL, all goes well. Then , up to now, the child process can successfully execute a file using do_execve(). The only task left here is to wait for the child process to terminate/stop/exit and got its return signal.

This part is implemented in my_wait() function. This function is basically to call the do_wait() and evaluat the status of child process it returns. To invoke the do_wait() from kernel, I have to set up a structure called wait_opts to initialize the passing arguments. Some hit here is that the .wo_flags should be set to WEXITED | WUNTRACED to deal with the STOP signal and all the other signals altogether. Also, the ,wo_stat should be of type int instead of a pointer points to int, which is a new feature in 5.10.x ‘s kernel. The .wo_pid should be the pid of child process since we are here to wait for the child process. And then I invoke the do_wait(&wo). If the function executed successfully, it will output the status of child process in .wo_stat, so I use a int STATUS to receive the signal. For evaluation the signal, I follow the way how OS interpret them, I search for the document and found the implementation of WIFEXITED(), WIFSIGNALED(), WIFSTOPPED, WEXITSTATUS, WTERMSIG, WSTOPSIG in the os kernel and use my way to implement them in this part to do the job of evaluating the signal. The logic is quite similar, first

we have to detect the type of signal, and based on that, we found a way to interpreted them. For example, if WIFEXITED(STATUS) returns true, then we should take the low-order 8 bits of the status, which is the correct output we should have.

Program-Output-Screenshot-2

The followings are the screen shot of the result of program 2. after typing make, sudo su insmode program2.ko, dmesg, you should see the same output.

```
[190881.385978] [program2] : Module_init YuyangLIN 120090377
[190881.385979] [program2] : Module_init create kthread start
[190881.386289] [program2] : Module_init kthread start
[190881.386400] [program2] : The child process has pid = 32381
[190881.386401] [program2] : This is the parent process, pid = 32380
[190881.386455] [program2] : child process
[190881.472287] [program2] : get SIGABRT signal
[190881.472288] [program2] : Child process terminated
[190881.472288] [program2] : The return signal is 6
```

```
[190964.704548] [program2] : Module_init YuyangLIN 120090377
[190964.704549] [program2] : Module_init create kthread start
[190964.704797] [program2] : Module_init kthread start
[190964.704927] [program2] : The child process has pid = 318
[190964.704928] [program2] : This is the parent process, pid = 317
[190964.705154] [program2] : child process
[190966.722003] [program2] : get SIGALRM signal
[190966.722004] [program2] : Child process terminated
[190966.722005] [program2] : The return signal is 14
```

```
[191005.440376] [program2] : Module_init YuyangLIN 120090377
[191005.440377] [program2] : Module_init create kthread start
[191005.440610] [program2] : Module_init kthread start
[191005.441426] [program2] : The child process has pid = 781
[191005.441427] [program2] : This is the parent process, pid = 780
[191005.441878] [program2] : child process
[191005.536576] [program2] : get SIGBUS signal
[191005.536578] [program2] : Child process terminated
[191005.536578] [program2] : The return signal is 7
```

```
[191045.241836] [program2] : Module_init YuyangLIN 120090377
[191045.241838] [program2] : Module_init create kthread start
[191045.242176] [program2] : Module_init kthread start
[191045.242322] [program2] : The child process has pid = 1584
[191045.242324] [program2] : This is the parent process, pid = 1583
[191045.243684] [program2] : child process
[191045.324603] [program2] : get SIGFPE signal
[191045.324605] [program2] : Child process terminated
[191045.324605] [program2] : The return signal is 8
```

```
[191096.811519] [program2] : Module_init YuyangLIN 120090377
[191096.811520] [program2] : Module_init create kthread start
[191096.811733] [program2] : Module_init kthread start
[191096.812084] [program2] : The child process has pid = 2046
[191096.812085] [program2] : This is the parent process, pid = 2044
[191096.812142] [program2] : child process
[191096.819984] [program2] : get SIGHUP signal
[191096.819986] [program2] : Child process terminated
[191096.819986] [program2] : The return signal is 1
```

```
[191184.035280] [program2] : Module_init YuyangLIN 120090377
[191184.035281] [program2] : Module_init create kthread start
[191184.035533] [program2] : Module_init kthread start
[191184.035919] [program2] : The child process has pid = 2428
[191184.035920] [program2] : This is the parent process, pid = 2426
[191184.036164] [program2] : child process
[191184.122271] [program2] : get SIGILL signal
[191184.122272] [program2] : Child process terminated
[191184.122273] [program2] : The return signal is 4
```

```
[191247.463484] [program2] : Module_init YuyangLIN 120090377
[191247.463486] [program2] : Module_init create kthread start
[191247.463750] [program2] : Module_init kthread start
[191247.463900] [program2] : The child process has pid = 2830
[191247.463901] [program2] : This is the parent process, pid = 2829
[191247.465109] [program2] : child process
[191247.468719] [program2] : get SIGINT signal
[191247.468720] [program2] : Child process terminated
[191247.468721] [program2] : The return signal is 2
```

```
[191293.491305] [program2] : Module_init YuyangLIN 120090377
[191293.491307] [program2] : Module_init create kthread start
[191293.491539] [program2] : Module_init kthread start
[191293.495949] [program2] : The child process has pid = 3232
[191293.495950] [program2] : This is the parent process, pid = 3230
[191293.496071] [program2] : child process
[191293.514302] [program2] : get SIGKILL signal
[191293.514304] [program2] : Child process terminated
[191293.514305] [program2] : The return signal is 9
```

```
[191328.273888] [program2] : Module_init YuyangLIN 120090377
[191328.273889] [program2] : Module_init create kthread start
[191328.274241] [program2] : Module_init kthread start
[191328.275732] [program2] : The child process has pid = 3632
[191328.275734] [program2] : This is the parent process, pid = 3630
[191328.275832] [program2] : child process
[191328.282579] [program2] : Normal termination with EXIT STATUS = 0
[191328.282580] [program2] : Child process terminated
[191328.282581] [program2] : The return signal is 0
```

```
[191359.608576] [program2] : Module_init YuyangLIN 120090377
[191359.608577] [program2] : Module_init create kthread start
[191359.608884] [program2] : Module_init kthread start
[191359.608994] [program2] : The child process has pid = 4036
[191359.608995] [program2] : This is the parent process, pid = 4035
[191359.609087] [program2] : child process
[191359.615368] [program2] : get SIGPIPE signal
[191359.615371] [program2] : Child process terminated
[191359.615372] [program2] : The return signal is 13
```

```
[191399.210304] [program2] : Module_init YuyangLIN 120090377
[191399.210306] [program2] : Module_init create kthread start
[191399.210607] [program2] : Module_init kthread start
[191399.210741] [program2] : The child process has pid = 4441
[191399.210742] [program2] : This is the parent process, pid = 4440
[191399.210786] [program2] : child process
[191399.288559] [program2] : get SIGQUIT signal
[191399.288560] [program2] : Child process terminated
[191399.288560] [program2] : The return signal is 3
[191482.856185] [program2] : Module_init YuyangLIN 120090377
[191482.856186] [program2] : Module_init create kthread start
[191482.856406] [program2] : Module_init kthread start
[191482.856523] [program2] : The child process has pid = 5141
[191482.856524] [program2] : This is the parent process, pid = 5140
[191482.856583] [program2] : child process
[191482.933203] [program2] : get SIGSEGV signal
[191482.933204] [program2] : Child process terminated
[191482.933205] [program2] : The return signal is 11
```

```
[191820.890147] [program2] : Module_init YuyangLIN 120090377
[191820.890149] [program2] : Module_init create kthread start
[191820.890535] [program2] : Module_init kthread start
[191820.890683] [program2] : The child process has pid = 5566
[191820.890721] [program2] : This is the parent process, pid = 5565
[191820.890794] [program2] : child process
[191820.891240] [program2] : child process STOPPED
[191820.891240] [program2] : get SIGSTOP signal
[191820.891241] [program2] : Child process terminated
[191820.891241] [program2] : The return signal is 19
```

```
[192086.177141] [program2] : Module_init YuyangLIN 120090377
[192086.177145] [program2] : Module_init create kthread start
[192086.177678] [program2] : Module_init kthread start
[192086.177895] [program2] : The child process has pid = 5970
[192086.177896] [program2] : This is the parent process, pid = 5969
[192086.179199] [program2] : child process
[192086.185749] [program2] : get SIGTERM signal
[192086.185752] [program2] : Child process terminated
[192086.185753] [program2] : The return signal is 15
```

```
[192137.992745] [program2] : Module_init YuyangLIN 120090377
[192137.992746] [program2] : Module_init create kthread start
[192137.993036] [program2] : Module_init kthread start
[192137.993162] [program2] : The child process has pid = 6392
[192137.993163] [program2] : This is the parent process, pid = 6391
[192137.993535] [program2] : child process
[192138.087145] [program2] : get SIGTRAP signal
[192138.087147] [program2] : Child process terminated
[192138.087147] [program2] : The return signal is 5
```

Development-Env-Setup-program-2

The operating system used is Ubuntu 16.04.7 LTS, the linux kernel is 5.10.5, which is correspond to what the tutorial is offering. The process about booting up VM and compiling kernel is the same as I chose to followed the tutorial. To compile kernel, we have to first copy a .config file from the /boot into the kernel file and name it .config. after that, we can type make menuconfig to add that newly created file into it. and then we use make -j$(nproc) to do the compilling process. After that, we type in make modules -j$(nproc) to check everything. And then is the make install. After that, we do the reboot process and it all finishes.

Before compiling the linuxe kernel, we have to export some symbol in the kernel source code since we are going to use them in our code. This is a feature in C language since those are non-static. So I first go into the kernel source code to find those function and add EXPORT_SYMBOL(func_name) to do this. And then, I compile the kernel.

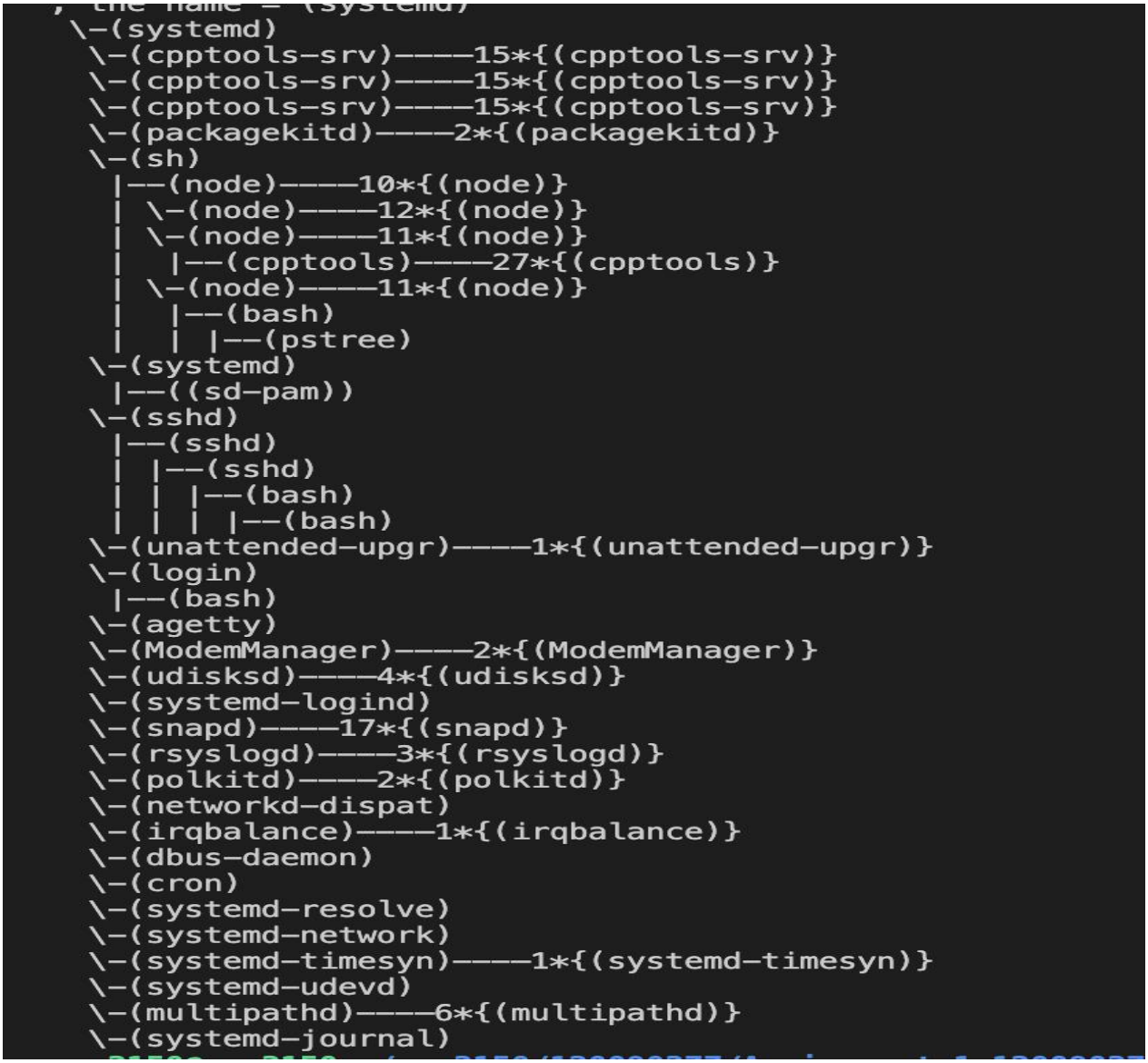
Summary-What-did-I-learn

Overall, I learned the knowledge about how to set up a child process using fork() and how it will behave differently due to the return value of

fork(). And then I learned the bottom realization of this process by looking into the kernel source code and using them, this is really helpful. Also, I learned the usage of execve(), wait() and its implementation in the kernel source code.

Bonus

I implemented everything except that the printed tree structure is not so good, I don't have time to format it and I think this can still give me some points. The sample output is like that:



The overall idea is quite simple. All the information about one process is stored in ./proc/pid. The information needed is organized in ./proc/pid/stat. So, the first step is to go to those file to fetch relative information and store them. After I just used those information to create node. The node is then used to create file tree structure. And then the problem is about dump tree, which I failed to accomplish.s