

Report for CSC3150 assignment 1

Xiao Zitong 120090766

October 10, 2022

Contents

1	Program design	3
1.1	Task 1: Design in user mode	3
1.2	Task 2: Design in kernel mode	3
1.3	Bonus: Pstree	3
2	Environment set up	4
3	Screenshot of output	4
3.1	Task 1	4
3.2	Task2	6
3.3	Bonus	9
4	Gain from learning	11

1 Program design

1.1 Task 1: Design in user mode

The whole program is mainly constructed by 3 functions in C: `fork()`, `exeve()`, and `waitpid()`.

`fork()` is used to clone a child process from the parent. After the child process is forked, `exeve()` is called to make child process run the test program, while an if-else structure is used to differ the child and parent process. When the child process is executing the test program, parent process will wait until the child process terminates.

1.2 Task 2: Design in kernel mode

Design the program in kernel mode, which means the functions in `libc` are forbid. Instead, `kernel_clone()`, `do_exeve()`, `kthread_create()`, and `do_wait()` are used. `kernel_clone()` is encapsulated in `my_fork()`, `do_exeve()` is encapsulated in `my_exec()`, and `do_wait()` is encapsulated in `my_wait()`.

Generally, the program works like this: when the program2 is initialized, `kthread_create()` is called to run `my_fork()`. In `my_fork()`, `kernel_clone()` is used to run `my_exec()`. When `my_exec()` is called, child process will start to run the test file, and at the same time parent process is controlled by `my_wait()` to wait for the child process to terminate. After that the parent process will receive the signal returned by the child process.

1.3 Bonus: Pstree

Basically, the program needs to do two things: get all the processes and list them into the pstree.

The program is written in C++. To get the current process, `scandir()` is called to scan the `/proc/` folder.

To list the process into the pstree, a structure called `process` is defined, which includes: `pid`, `ppid`, `name` and `children`. And the pstree is listed in a recursive way by the function `printTree()`. In the `printTree()`, the oldest parent process will be firstly printed, and then lead to several recursive condition: if the oldest parent only has one children, "—" structure will be printed, and `printTree` will call itself again; else if the number of children is bigger than one and the next child is the first child,

"T" structure will be printed, if it is the middle child, "|-" structure will be printed, else it must be the last child, and "|_" structure will be printed, after that printTree will call itself again; else, the parent does not have any children and the will print a line break and return.

2 Environment set up

There are mainly two things need to be done. Firstly, the kernel source code need to be downloaded and compiled. And because some functions, for example: kernel_clone(), are not open to the user, therefore, you need to add the export symbol to such functions in the source code to make it available to the user. After that, you need to recompile the kernel source code to make the change in the source code work.

3 Screenshot of output

For reading convenience, instead of posting every output, only typical outputs are posted in task1 and task2.

3.1 Task 1

The results shown below include: normal.c, abort.c, stop.c, which are the three most typical situations.

```
root@csc3150:/home/vagrant/csc3150/hw1/source/program1# ./program1 normal
Process start to fork
I'm the Parent Process, my pid = 3849
I'm the Child Process, my pid = 3850
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
```

Figure 1: Output of normal exit

```
root@csc3150:/home/vagrant/csc3150/hw1/source/program1# ./program1 abort
Process start to fork
I'm the Parent Process, my pid = 3987
I'm the Child Process, my pid = 3988
Child process start to execute test program:
-----CHILD PROCESS START-----
This is the SIGABRT program

Parent process receives SIGCHLD signal
Parent process get SIGABRT signal
```

Figure 2: Output of abort

```
root@csc3150:/home/vagrant/csc3150/hw1/source/program1# ./program1 stop
Process start to fork
I'm the Parent Process, my pid = 4074
I'm the Child Process, my pid = 4075
Child process start to execute test program:
-----CHILD PROCESS START-----
This is the SIGSTOP program

Parent process receives SIGCHLD signal
CHILD PROCESS STOPPED
```

Figure 3: Output of stop

3.2 Task2

The results shown below include: normal.c, bus.c, abort.c, alarm.c, pipe.c, hangup.c, illegal_instr.c, stop.c.

```
[ 1787.735336] [program2] : Module_init {Xiao Zitong} {120090766}
[ 1787.735507] [program2] : module_init create kthread start
[ 1787.735508] [program2] : module_init kthread start
[ 1787.735645] [program2] : The child process has pid = 2543
[ 1787.736200] [program2] : This is the parent process, pid = 2542
[ 1787.736201] [program2] : child process
[ 1787.736202] [program2] : child process normal exit
[ 1787.736202] [program2] : child process terminated
[ 1787.736203] [program2] : The return signal is 0
[ 1793.990019] [program2] : module_exit
```

Figure 4: Output of normal exit

```
[ 114.378392] [program2] : Module_init {Xiao Zitong} {120090766}
[ 114.378444] [program2] : module_init create kthread start
[ 114.378444] [program2] : module_init kthread start
[ 114.378529] [program2] : The child process has pid = 2057
[ 114.483238] [program2] : This is the parent process, pid = 2056
[ 114.483239] [program2] : child process
[ 114.483240] [program2] : get SIGBUS signal
[ 114.483241] [program2] : child process terminated
[ 114.483241] [program2] : The return signal is 7
[ 125.247423] [program2] : module_exit
```

Figure 5: Output of bus

```

[ 566.249329] [program2] : Module_init {Xiao Zitong} {120090766}
[ 566.249629] [program2] : module_init create kthread start
[ 566.249630] [program2] : module_init kthread start
[ 566.249766] [program2] : The child process has pid = 3733
[ 566.390088] [program2] : This is the parent process, pid = 3732
[ 566.390089] [program2] : child process
[ 566.390090] [program2] : get SIGABRT signal
[ 566.390091] [program2] : child process terminated
[ 566.390091] [program2] : The return signal is 6
[ 573.160889] [program2] : module_exit

```

Figure 6: Output of abort

```

[ 776.337508] [program2] : Module_init {Xiao Zitong} {120090766}
[ 776.337689] [program2] : module_init create kthread start
[ 776.337690] [program2] : module_init kthread start
[ 776.337827] [program2] : The child process has pid = 3923
[ 778.344888] [program2] : This is the parent process, pid = 3922
[ 778.344890] [program2] : child process
[ 778.344891] [program2] : get SIGALRM signal
[ 778.344892] [program2] : child process terminated
[ 778.344892] [program2] : The return signal is 14
[ 784.825153] [program2] : module_exit

```

Figure 7: Output of alarm

```

[ 910.369029] [program2] : Module_init {Xiao Zitong} {120090766}
[ 910.369326] [program2] : module_init create kthread start
[ 910.369328] [program2] : module_init kthread start
[ 910.369426] [program2] : The child process has pid = 4148
[ 910.487469] [program2] : This is the parent process, pid = 4147
[ 910.487470] [program2] : child process
[ 910.487471] [program2] : get SIGFPE signal
[ 910.487472] [program2] : child process terminated
[ 910.487472] [program2] : The return signal is 8
[ 914.925459] [program2] : module_exit

```

Figure 8: Output of pipe

```
[ 1044.565478] [program2] : Module_init {Xiao Zitong} {120090766}
[ 1044.565834] [program2] : module_init create kthread start
[ 1044.565835] [program2] : module_init kthread start
[ 1044.565947] [program2] : The child process has pid = 4231
[ 1044.566780] [program2] : This is the parent process, pid = 4230
[ 1044.566782] [program2] : child process
[ 1044.566783] [program2] : get SIGHUP signal
[ 1044.566784] [program2] : child process terminated
[ 1044.566785] [program2] : The return signal is 1
[ 1052.757459] [program2] : module_exit
```

Figure 9: Output of SIGHUP

```
[ 1151.386480] [program2] : Module_init {Xiao Zitong} {120090766}
[ 1151.386619] [program2] : module_init create kthread start
[ 1151.386620] [program2] : module_init kthread start
[ 1151.386765] [program2] : The child process has pid = 4481
[ 1151.521725] [program2] : This is the parent process, pid = 4480
[ 1151.521727] [program2] : child process
[ 1151.521728] [program2] : get SIGILL signal
[ 1151.521742] [program2] : child process terminated
[ 1151.521743] [program2] : The return signal is 4
[ 1159.107245] [program2] : module_exit
```

Figure 10: Output of SIGILL

```
[ 1675.020937] [program2] : Module_init {Xiao Zitong} {120090766}
[ 1675.021059] [program2] : module_init create kthread start
[ 1675.021060] [program2] : module_init kthread start
[ 1675.021237] [program2] : The child process has pid = 2326
[ 1675.021904] [program2] : This is the parent process, pid = 2325
[ 1675.021904] [program2] : child process
[ 1675.021905] [program2] : get SIGSTOP signal
[ 1675.021905] [program2] : child process terminated
[ 1675.021906] [program2] : The return signal is 19
[ 1681.402859] [program2] : module_exit
```

Figure 11: Output of stop

3.3 Bonus



Figure 12: Output of pstree

```

vagrant@csc3150:~/csc3150/hw1/source/bonus$ ./pstree p
systemd(1)─rsyslogd(1034)
          └─iscsid(1035)
             └─iscsid(1036)
                └─accounts-daemon(1038)
                   └─dbus-daemon(1040)
                      └─systemd-logind(1049)
                         └─lxcfs(1055)
                            └─atd(1057)
                               └─acpid(1062)
                                  └─cron(1067)
                                     └─sshd(1072)
                                        └─sshd(1490)─sshd(1528)─bash(1529)─sh(1574)─node(1584)
                                           └─node(1640)─bash(1728)─sudo(1848)─su(1849)─bash(1850)
                                              └─bash(3552)─sudo(3618)─su(3619)─bash(3620)
                                                 └─bash(4397)─pstree(4553)
                                                    └─node(1673)─cpptools(1739)
                                                       └─node(1684)
                                                          └─node(3466)─cpptools(3521)
                                                             └─node(3477)
                                                                └─node(4298)─cpptools(4354)
                                                                   └─node(4309)
                                                                      └─sleep(4210)
                                                                         └─sshd(3379)─sshd(3413)─bash(3414)─sleep(4118)
                                                                            └─sshd(4211)─sshd(4245)─bash(4246)─sleep(4297)
                                                                               └─unattended-upgr(1081)
                                                                                  └─polkitd(1084)
                                                                                     └─mdadm(1091)
                                                                                        └─agetty(1137)
                                                                                           └─agetty(1142)
                                                                                              └─irqbalance(1159)
                                                                                                 └─systemd(1492)─(sd-pam)(1493)
                                                                                                    └─cpptools-srv(2018)
                                                                                                       └─cpptools-srv(2181)
                                                                                                          └─test(2326)
                                                                                                             └─cpptools-srv(3757)
                                                                                                                └─systemd-journal(413)
                                                                                                                   └─lvm2metad(439)
                                                                                                                      └─cpptools-srv(4497)
                                                                                                                         └─systemd-udev(459)
                                                                                                                            └─dhcpcd(892)

```

Figure 13: Output of pstree p

4 Gain from learning

From task 1, I learn the basic program structure in C and get familiar with the C program compiling. Besides, I also gain a rough understanding on the relationship between the parent and child process.

From task 2, I learn some basic operations on linux kernel source code. Besides, I also have a better understanding on the basic functions to create process in kernel mode. What's more, I get to know what is really happening in the kernel when you call the `fork()` in C.

From task 3, firstly I learn how to gain the current processes. Then, I learn some basic techniques to construct a `pstree`, which is mainly about data structure.