CSC3150_A1_report

Yixin Jin_120090509

10/10/2022

1. General Environment set up

   a) Using Ubuntu 5.4.0-6ubuntu1~16.04.12

      Run on the Windows 10

   b) How to compile the kernel?

      First download the source file from http://www.kernel.org

      Then use "sudo apt-get install libncurses-dev gawk flex bison openssl libssl-dev dkms libelf-dev libudev-dev libpci-dev libiberty-dev autoconf llvm dwarves" to install Dependency and development tools.

      Then use mkdir to create work/seed dictionary, i**f encounter the problem of "permission denied" we can fix it with "chmod 777"**. And then put the source file to the dictionary we created and use "$sudo tar xvf KERNEL_FILE.tar.xz" to extract. Then copy .config to the file we just extracted.

      Then use "sudo su" to login in the root account and use cd to get into the kernel source file.

      Then use "make mrpropr" "make clean" "make menuconfig" to set and start configuration, then in the window which pops out, choose save and exit.

      Then use make -j$(nproc) to build kernel and modules. After finishing building, use make modules_install and make install to install kernel modules and kernel respectfully.

      Finally use "reboot" to load the kernel.

2. Program 1

   a) Basic idea of the program:

      The program is to run a process under user mode. The main process will fork a child process to execute a test program and wait for its returning signal. After receiving the terminated signal, the parent process will print out the related information of the signal.

   b) How I design my program:

For This problem, we need to fork the process in user mode. we first use fork() function to fork a identical children process but with different logical memory. after that, we should use execve() function to execute the test program. At the same time, parent process should wait the process until it terminate and send some siganls back, we use wait_pid() here, also can use wait() function, which have no difference here. we also handle some different cases for different send back signal. In summary, 1. Fork a child process to execute test programs (15 of them) 2. Use wait() to let the parent process receives the SIGCHLD signal 3. Print out the termination information of child process (normal or abnormal)

c) Output:

Abort

```
vagrant@csc3150:~/csc3150/source/program1$ ./program1 ./abort
Process start to fork
I'm the Parent Process, my pid = 5755
I'm the Child Process, my pid = 5756
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGABRT program

Parent process receives SIGCHLD signal
```

Alarm

```
vagrant@csc3150:~/csc3150/source/program1$ ./program1 ./alarm
Process start to fork
I'm the Parent Process, my pid = 5806
I'm the Child Process, my pid = 5807
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGALRM program

Parent process receives SIGCHLD signal
```

Bus

```
vagrant@csc3150:~/csc3150/source/program1$ ./program1 ./bus
Process start to fork
I'm the Parent Process, my pid = 5874
I'm the Child Process, my pid = 5875
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGBUS program

Parent process receives SIGCHLD signal
```

Floating

```
vagrant@csc3150:~/csc3150/source/program1$ ./program1 ./floating
Process start to fork
I'm the Parent Process, my pid = 5948
I'm the Child Process, my pid = 5949
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGFPE program

Parent process receives SIGCHLD signal
```

Hangup

```
vagrant@csc3150:~/csc3150/source/program1$ ./program1 ./hangup
Process start to fork
I'm the Parent Process, my pid = 6007
I'm the Child Process, my pid = 6008
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGHUP program

Parent process receives SIGCHLD signal
```

Illegal_instr

```
vagrant@csc3150:~/csc3150/source/program1$ ./program1 ./illegal_instr
Process start to fork
I'm the Parent Process, my pid = 6129
I'm the Child Process, my pid = 6130
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGILL program

Parent process receives SIGCHLD signal
```

Interrupt

```
vagrant@csc3150:~/csc3150/source/program1$ ./program1 ./interrupt
Process start to fork
I'm the Parent Process, my pid = 6192
I'm the Child Process, my pid = 6193
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGINT program

Parent process receives SIGCHLD signal
```

Kill

```
vagrant@csc3150:~/csc3150/source/program1$ ./program1 ./kill
Process start to fork
I'm the Parent Process, my pid = 6233
I'm the Child Process, my pid = 6234
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGKILL program

Parent process receives SIGCHLD signal
```

Normal

```
vagrant@csc3150:~/csc3150/source/program1$ ./program1 ./normal
Process start to fork
I'm the Parent Process, my pid = 6271
I'm the Child Process, my pid = 6272
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
```

Pipe

```
vagrant@csc3150:~/csc3150/source/program1$ ./program1 ./pipe
Process start to fork
I'm the Parent Process, my pid = 6309
I'm the Child Process, my pid = 6310
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGPIPE program

Parent process receives SIGCHLD signal
```

Quit

```
vagrant@csc3150:~/csc3150/source/program1$ ./program1 ./quit
Process start to fork
I'm the Parent Process, my pid = 6347
I'm the Child Process, my pid = 6348
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGQUIT program

Parent process receives SIGCHLD signal
```

Segment_fault

```
vagrant@csc3150:~/csc3150/source/program1$ ./program1 ./segment_fault
Process start to fork
I'm the Parent Process, my pid = 6410
I'm the Child Process, my pid = 6411
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGSEGV program

Parent process receives SIGCHLD signal
```

Stop

```
vagrant@csc3150:~/csc3150/source/program1$ ./program1 ./stop
Process start to fork
I'm the Parent Process, my pid = 6439
I'm the Child Process, my pid = 6440
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGSTOP program

Parent process receives SIGCHLD signal
CHILD PROCESS STOPPED
```

Terminate

```
vagrant@csc3150:~/csc3150/source/program1$ ./program1 ./terminate
Process start to fork
I'm the Parent Process, my pid = 6486
I'm the Child Process, my pid = 6487
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGTERM program

Parent process receives SIGCHLD signal
```

d) What I learned:

First, I learnt how to create child processes and how to use them to do some task. I also learnt how to execute other programs in the process using execve( ) function. I also learned how signal is transported between father process and child process. Then, I also got some feelings about the terminated signals and know some basic knowledge of them.

3. Program 2:

a) Basic ideas:

This program will create a kernel thread. In the thread, the program will fork a child process and make it to execute another program. The parent process will wait for the child's terminated signal and print out related information.

b) How I design my program:

1) Firstly, since we need to use these 4 functions: “_do_fork” (/kernel/fork.c) , “do_execve” (/fs/exec.c), “getname_kernel” (/fs/namei.c) and “do_wait”(/kernel/exit.c), we need to find their corresponding files in kernel and export them using EXPORT_SYMBOL. After modifications, we need to recompile the kernel. Then we just need to use extern to import them in program2.c.

2) Then to start the module, I choose to create a kernel thread using kernel_clone( ) to run my_fork( ) function, where the program will fork a child process to execute another program.

3) But we need to write a few functions in advance

   my_wait(): this function will get the terminated signal of the child process using do_wait( ).

   my_exec(): This function will first get the information (using getname_kernel( )) of the test program using the address set inside it. After that, it will use do_execve( ) to pass in the arguments and execute the file.

c) Output: