Assignment 1 report
Tan Jinzhen 120090543
Problem1

Program Design

In this program, we need to create a child process to execute the test program, then receive the signal raised by the child process.

In the parent process, we use int getpid() function to get the parent pid and print it out. Then, use int fork() function to create the child process and assign the return value, which is the child pid, to the variable int pid.

In the parent process, we call the function pid_t waitpid(), then the parent process will wait until the child process was terminated.

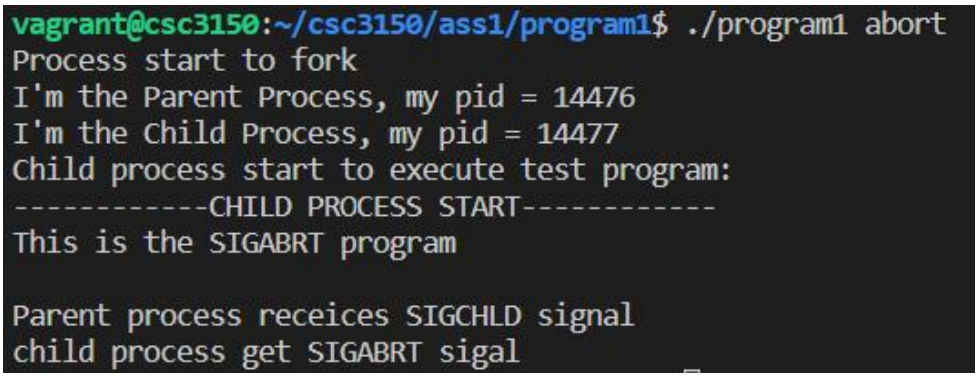
In the child process, the program will first use int getpid() function to get the child pid and print it out, then it will call int execl() function to execute the test program. In the test program, the child process will be terminated and sent the signal to the parent process.

The signal will be caught by the function waitpid() and its second argument int status will record the signal from the child process. Then we use function WIFSTOPPED(status) to determine whether the child process raised SIGSTOP signal, if not, we use function WIFEXITED(status) to determine whether the child process was terminated normally with exit status = 0. Else, we use WIFSINGALED(status) to determine whether the child process raised other signals, and use WTERMSIG(status) to get the integer that represents the signal. All the signal was stored in a string array signallist, we can find the signal according to the return value from WTERMSIG(status).

Finally, we use void exit(0) to terminate the parent process.

Program Output

1. abort



2. alarm

```
vagrant@csc3150:~/csc3150/ass1/program1$ ./program1 alarm
Process start to fork
I'm the Parent Process, my pid = 14579
I'm the Child Process, my pid = 14580
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGALRM program

Parent process receices SIGCHLD signal
child process get SIGALRM sigal
```

3. bus

```
Process start to fork
I'm the Parent Process, my pid = 14633
I'm the Child Process, my pid = 14634
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGBUS program

Parent process receices SIGCHLD signal
child process get SIGBUS sigal
```

4. floating

```
Process start to fork
I'm the Parent Process, my pid = 14679
I'm the Child Process, my pid = 14680
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGFPE program

Parent process receices SIGCHLD signal
child process get SIGFPE sigal
```

5. hangup

```
Process start to fork
I'm the Parent Process, my pid = 14788
I'm the Child Process, my pid = 14789
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGHUP program

Parent process receices SIGCHLD signal
child process get SIGHUP sigal
```

6. illegal

```
Process start to fork
I'm the Parent Process, my pid = 14842
I'm the Child Process, my pid = 14843
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGILL program

Parent process receices SIGCHLD signal
child process get SIGILL sigal
```

7. Interrupt

```
Process start to fork
I'm the Parent Process, my pid = 14885
I'm the Child Process, my pid = 14886
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGINT program

Parent process receices SIGCHLD signal
child process get SIGINT sigal
```

8. Kill

```
Process start to fork
I'm the Parent Process, my pid = 14918
I'm the Child Process, my pid = 14919
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGKILL program

Parent process receices SIGCHLD signal
child process get SIGKILL sigal
```

9. Normal

```
Process start to fork
I'm the Parent Process, my pid = 14980
I'm the Child Process, my pid = 14981
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the normal program

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

10. Pipe

```
Process start to fork
I'm the Parent Process, my pid = 15033
I'm the Child Process, my pid = 15034
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGPIPE program

Parent process receices SIGCHLD signal
child process get SIGPIPE sigal
```

11. Quit

```
Process start to fork
I'm the Parent Process, my pid = 15080
I'm the Child Process, my pid = 15081
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGQUIT program

Parent process receices SIGCHLD signal
child process get SIGQUIT sigal
```

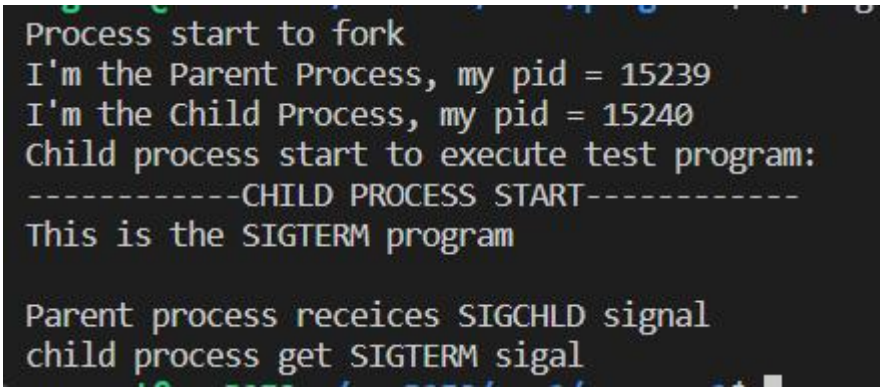
12. Segment_fault

```
Process start to fork
I'm the Parent Process, my pid = 15126
I'm the Child Process, my pid = 15127
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGSEGV program

Parent process receices SIGCHLD signal
child process get SIGSEGV sigal
```

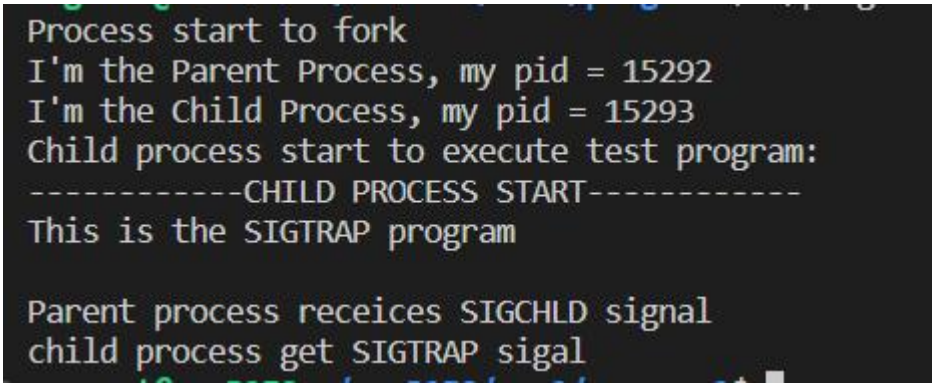
13. Stop

```
Process start to fork
I'm the Parent Process, my pid = 15186
I'm the Child Process, my pid = 15187
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGSTOP program

Parent process receices SIGCHLD signal
child process get SIGSTOP sigal
```

14. Terminated



15. Trap



Problem2

Program Design

In this task, we need to create a child process in the kernel mode and execute the test program in the child process.

After initialization with the program2_init(), we use kthread_create(&my_fork, NULL, ”MThread”) and function wake_up_process() to create a parent process to execute my_fork function.

In my_fork function, we use kernel_clone(&my_args) to create a child process, and the return value, which is the child pid, will be assign to the variable pid. Then parent process will call the function my_wait(pid) to wait until the child process terminates. At the same time, the child process will execute the test program.

In my_wait(pid_t) function, we call do_wait(&wo) function. After child process terminates, the status of child process will be written to the structure wo, which is also the argument of do_wait() function. Among all the attributes, wo.wo_stat will record the status and the signal from child process, so that the parent process can get the signal number.

Output

```
[ 10068.625950] [program2] : Module_init {Tan Jinzhen} {120090543}
[ 10068.625951] [program2] : Module_init create kthread start
[ 10068.626008] [program2] : Module_init kthread start
[ 10068.626048] [program2] : The child process has pid = 3122
[ 10068.626049] [program2] : This is the parent process, pid = 3121
[ 10068.626052] [program2] : Child process
[ 10068.627125] [program2] : get SIGBUS signal
[ 10068.627126] [program2] : The return signal was 7
[ 10068.627127] [program2] : Child process has bus error
[ 10068.627567] [program2] : Module_exit
```

Environment and Kernel Compiling

We implement this program in virtual machine with Linux kernel. First we set up the VM with Virtualbox and vagrant, so that we can control the remote virtual machine in local computer with VScode. Second, we download the Linux kernel to the VM and compile it. To compile the kernel, we enter the administrator mode, open the kernel file after uncompressing. Then we use $make mrproper and $make clean to clean previous setting, and configurate the kernel using $make menuconfig. After that, we use $make -j(nproc) to build kernel image and modules. Then we install the kernel modules and the kernel.

Harvest

I have a deeper understanding of what is a process. I can create a process and create more child process after it. Also, I know how signals are raised in child process and received by parent process. Most importantly, I know how important the environment is, at least, no less than writing program itself.