1. Program1

a) basic ideas 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) implementation

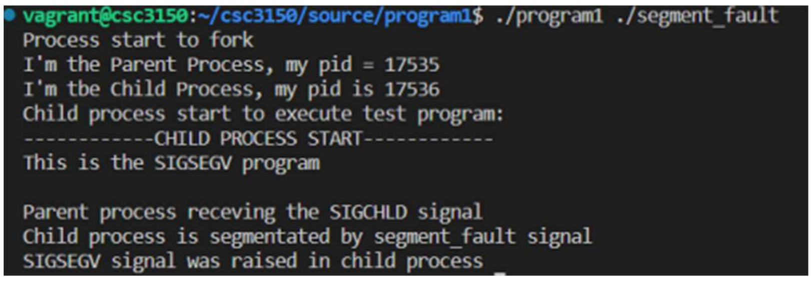
Firstly, the program will use fork( ) function to fork a child process at the beginning. Then, the program will check the pid of the process and do the corresponding operations. If pid equals to 0, which means the process is the child process, then the program will let it execute the test program. If pid not equals to 0 or -1, then it is the parent process and it will wait for the terminated signal of the child process (using waitpid( )) and print out the information of the signal

c) Set up environment:

follow the tutorial recording to compile the kernel.

In the ‘program1’ directory, type ‘make’ and enter, type ‘make clean’ and enter.

Then type ‘./program1 $TEST_CASE $ARG1 $ARG2…’



```
vagrant@csc3150:~/csc3150/source/program1$ ./program1 ./segment_fault
Process start to fork
I'm the Parent Process, my pid = 17535
I'm tbe Child Process, my pid is 17536
Child process start to execute test program:
------------CHILD PROCESS START-----------
This is the SIGSEGV program

Parent process receving the SIGCHLD signal
Child process is segmentated by segment_fault signal
SIGSEGV signal was raised in child process _
```

d) what I learnt from the project:

In this project, t, 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.

2. Program2

a) basic ideas of the program

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) implementation

i. Firstly, since we need to use these 4 functions: “_do_fork” (/kernel/fork.c) , “do_execve” (/fs/exec.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.

ii. Then, when initiating the module, we need to create a kernel thread using kthread_create( ) to run my_fork( ) function, where the program will fork a child process to execute another program.

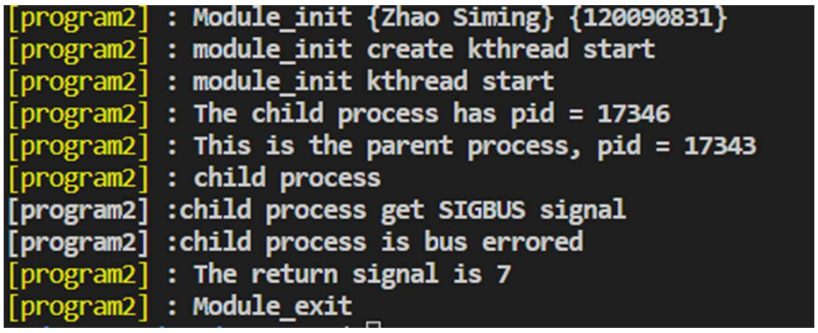
iii. To implement my_fork( ), several functions need to be implemented in advance.

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

b) b) my_wait( ): this function will get the terminated signal of the child process using do_wait( ). With these 2 functions, my_fork( ) is implemented in the following: First, my_fork( ) will use _do_fork to generate a child process to run my_exec( ) function. Then it will use my_wait( ) to get the terminated signal of the child process and print out corresponding information.

iv. After creating the kernel thread, the program will wake up the program if there kthread_create( ) create a thread successfully

c) How to compile:
   1. go to the folder of program2 and type "sudo make" in the terminal.
   2. use "gcc test.c -o test" to compile the test file
   3. then type "sudo insmod program2.ko" and "sudo rmmod program2.ko" to insert and remove the module
   4. using "dmesg" in the terminal to display the corresponding message.



```
[program2] : Module_init {Zhao Siming} {120090831}
[program2] : module_init create kthread start
[program2] : module_init kthread start
[program2] : The child process has pid = 17346
[program2] : This is the parent process, pid = 17343
[program2] : child process
[program2] :child process get SIGBUS signal
[program2] :child process is bus errored
[program2] : The return signal is 7
[program2] : Module_exit
```

d) What I learnt from the project:

I learnt how to modify the kernel files and recompile the kernel to use it. I also learnt how to insert and remove modules to kernels