**Report**

Info: Linux-5.10.146

**Program 1:**

First fork, go to child process and parent process if no error happens.

The child process executes the file, print out message related. Then, it returns status that contains information of child process.

Analyzing the status:

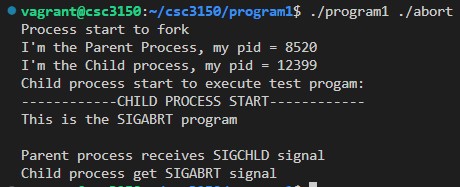
WIFEXITED for exited. If succeed, print related massage and exit status (using WEXITSTATUS).

WIFSIGNALED for situations that child process is terminated by signal.

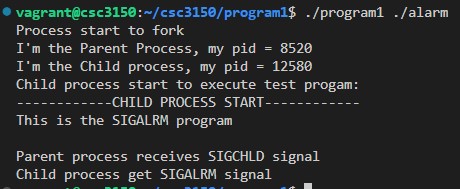
WTERMSIG for signal value.

WIFSTOPPED for checking whether child process is stopped or not.

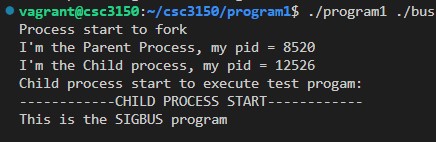
Below show the output graph:



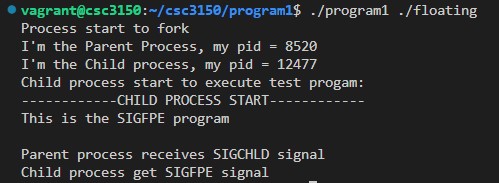
Abort



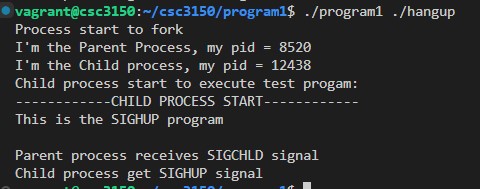
Alarm



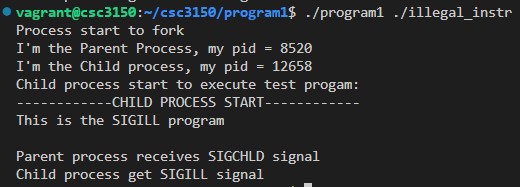
Bus



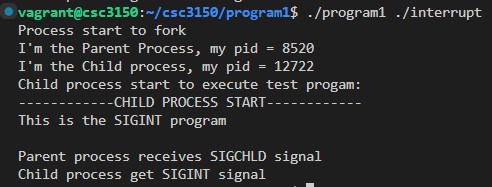
Floating



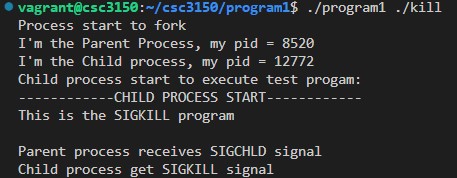
Hangup



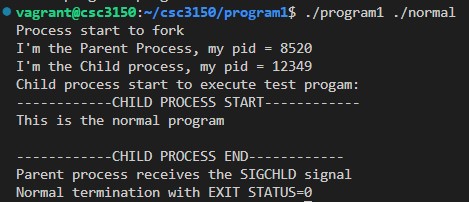
Illegal\_instr



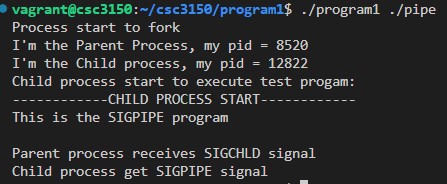
Interrupt



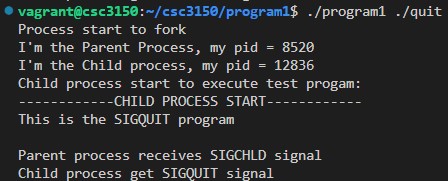
Kill



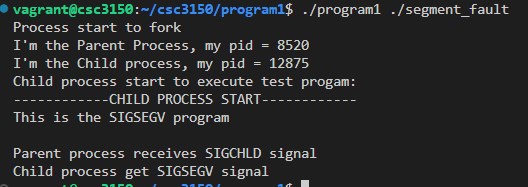
Normal



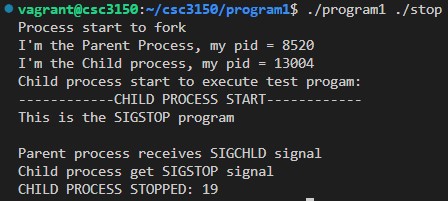
Pipe



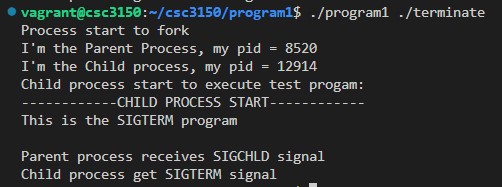
Quit



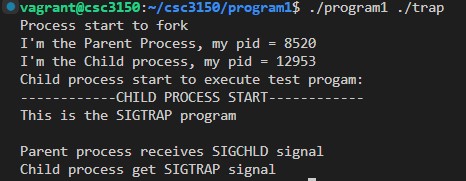
Segment\_fault



Stop



Terminate



Trap

**Program 2:**

Four export file: fork.c, exec.c, exit.c, namei.c.

Four funcs: \_do\_fork, do\_execve, do\_wait, getname\_kernel

Export method: use export symbol and then compile the kernel again

Program2\_init is invoked by module\_init().

Then the initialization function creates a new thread to run my\_fork.

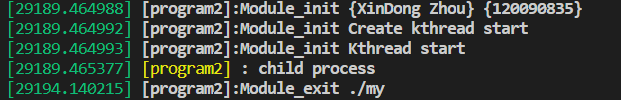
My\_fork(), print out child process pid and parent process pid.

Then, call my\_wait() to capture the return signal from child process.

Then, implement the child process in my\_exec, the do\_execve() executes the file in a given path. Return value can show wether it is succeed. If not, end child process by do\_exit() and return the result. If succeed, return 0.

Then, back to parent process and implement my\_wait and printk related information.

Below is the output graph:

****