CSC3150 Operating Systems Report

Name: Zhou Yuxiao	Student ID: 120090443

Assignment 1

1. **Development environment**

   (a) Environment requirement

   i. Linux Distribution: Ubuntu 20.04 (others are ok)

   ii. Linux Kernel Version: 5.10.x (Test Environment) (use uname -r to get it)

   iii. GCC Version: 4.9 above (use gcc -v to get it)

   (b) Set up

   i. My Linux Distribute is 14.04. Since the distributr is ok, we do not need to change anything.

   ii. As the original linux Kernel version is 4.4.0, we need to upgrade it to 5.10.x to satisfy the requirement.

   A. Download source code from the website http://www.kernel.org. I choose the stable version 5.19.14, and use instruction `wget` to download the source to the /home folder.

   B. Install Dependency and development tools
   sudo apt-get install libncurses-dev gawk flex bison openssl libssl-dev dkms libelf-devlibudev-dev libpci-dev libiberty-dev autoconf llvm dwarves

   C. Extract the source file
   sudo tar xvf KERNELFILE.tar.xz

   D. Copy config from /boot to /home
   cp CONFIGFILE /home/kernelfile/.config

   E. Login root account and go to kernel source directory

   F. Clean previous setting and start configuration
   make mrproper
   make clean
   make menuconfig
   save the config and exit

   G. Build kernel Image and modules
   make bzImage
   make modules
   make
   Use -j to define use how many cores. This procedure will take time.

   H. Install kernel modules and kernel
   make modules_install
   make install

   I. Reboot to load new kernel

   (c) Recompile kernel
   After modifying the kernel to use some API in other modules, we need to recompile the kernel. And just start from building procedure to save time.

2. **Task 1**

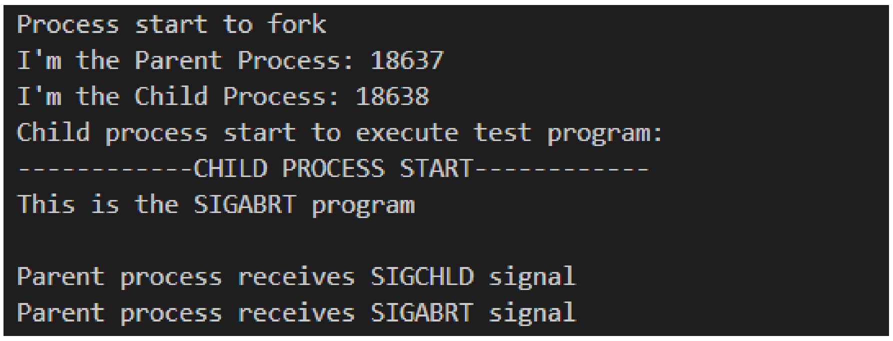
   (a) Program Design

   i. In user mode, fork a child process to execute the test program using `fork()` and `execve()` function. And we use `getpid()` the pid of the child process and the parent process.

   ii. When child process finish execution, the parent process will receive the SIGCHLD signal by `waitpid()` function.

iii. Print out the termination information of child process. If normal termination, print normal termination and exit status. If not, print out how did the child process terminates and what signal was raised in child process. We use
`int WIFEXITED (int status)`
`int WIFSIGNALED (int status)`
`int WIFSTOPPED (int status).`
to analyze the status referenced by the status argument.
And we use `int WEXITSTATUS (int status)`
`int WTERMSIG (int status)`
`int WSTOPSIG (int status).`
to evaluate child process's returned value of status argument(exact values).

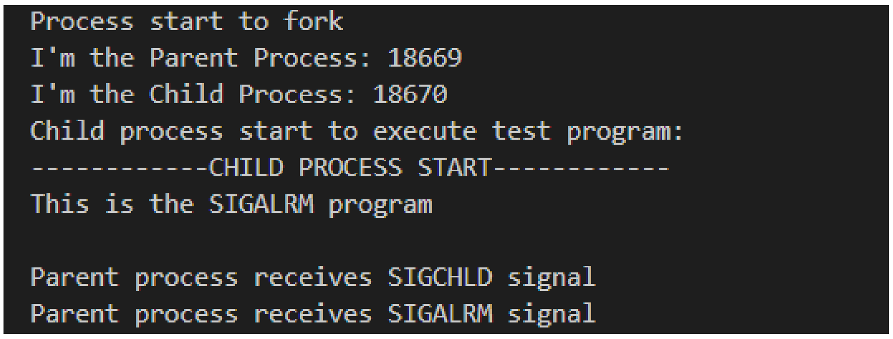
(b) The output display



```
Process start to fork
I'm the Parent Process: 18637
I'm the Child Process: 18638
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGABRT program

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

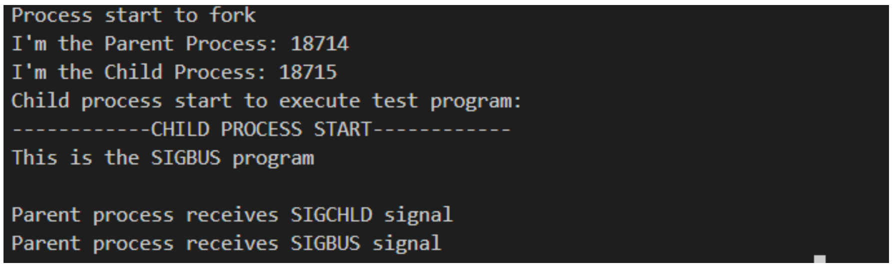
Figure 1: *A*bort Graph of Task 1



```
Process start to fork
I'm the Parent Process: 18669
I'm the Child Process: 18670
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGALRM program

Parent process receives SIGCHLD signal
Parent process receives SIGALRM signal
```

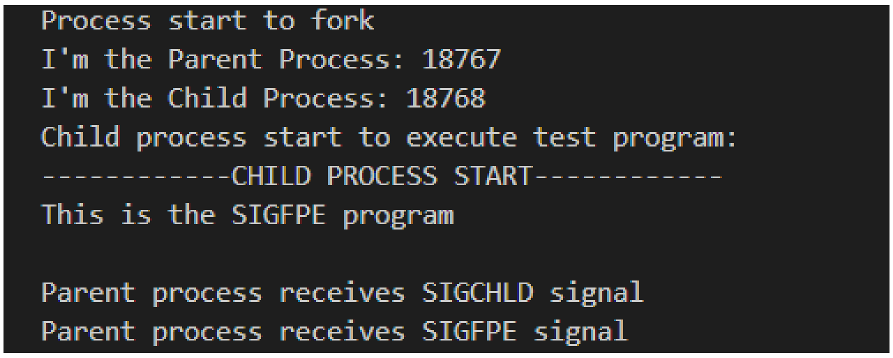
Figure 2: *A*larm Graph of Task 1



```
Process start to fork
I'm the Parent Process: 18714
I'm the Child Process: 18715
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGBUS program

Parent process receives SIGCHLD signal
Parent process receives SIGBUS signal
```

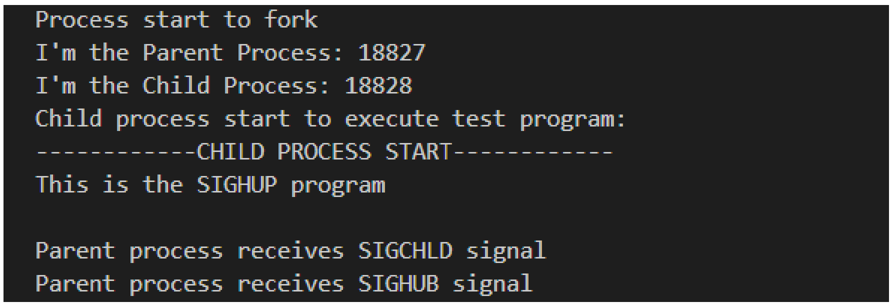
Figure 3: *B*us Graph of Task 1



```
Process start to fork
I'm the Parent Process: 18767
I'm the Child Process: 18768
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGFPE program

Parent process receives SIGCHLD signal
Parent process receives SIGFPE signal
```

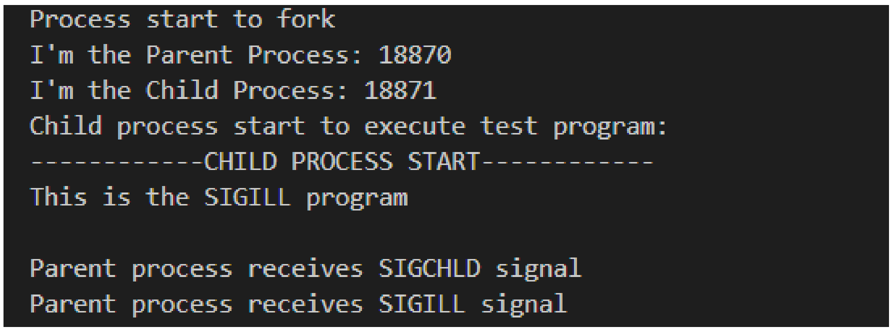
Figure 4: Floating Graph of Task 1



```
Process start to fork
I'm the Parent Process: 18827
I'm the Child Process: 18828
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGHUP program

Parent process receives SIGCHLD signal
Parent process receives SIGHUB signal
```

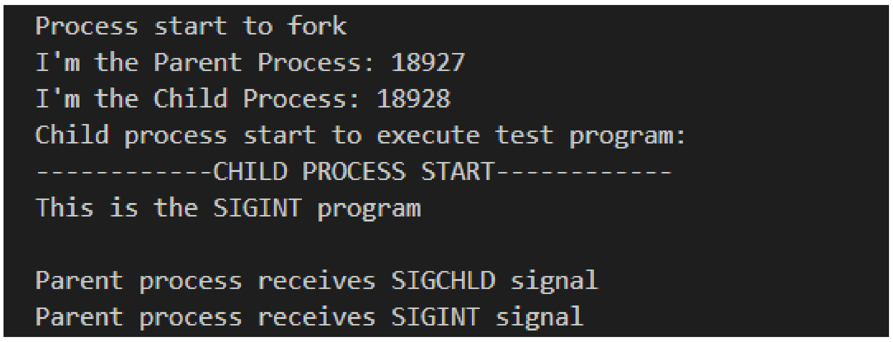
Figure 5: Hangup Graph of Task 1



```
Process start to fork
I'm the Parent Process: 18870
I'm the Child Process: 18871
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGILL program

Parent process receives SIGCHLD signal
Parent process receives SIGILL signal
```

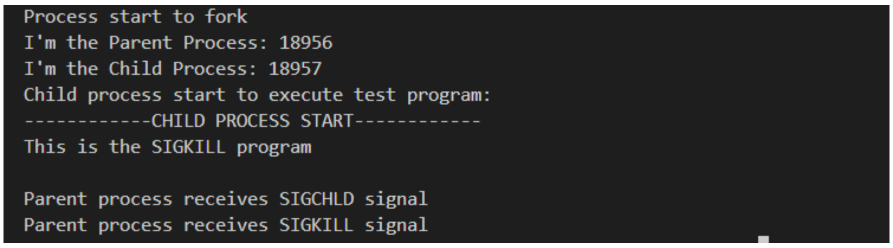
Figure 6: Illegal_instr Graph of Task 1



```
Process start to fork
I'm the Parent Process: 18927
I'm the Child Process: 18928
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGINT program

Parent process receives SIGCHLD signal
Parent process receives SIGINT signal
```

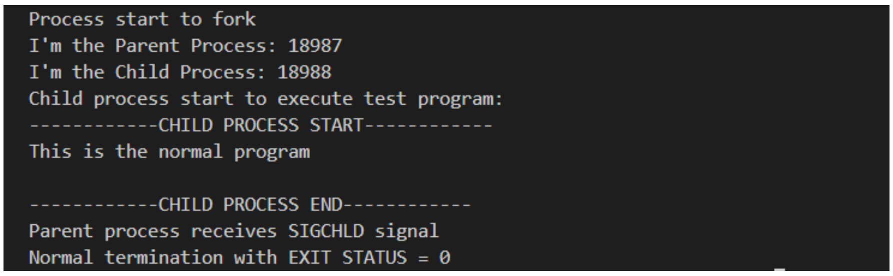
Figure 7: Interrupt Graph of Task 1



```
Process start to fork
I'm the Parent Process: 18956
I'm the Child Process: 18957
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGKILL program

Parent process receives SIGCHLD signal
Parent process receives SIGKILL signal
```

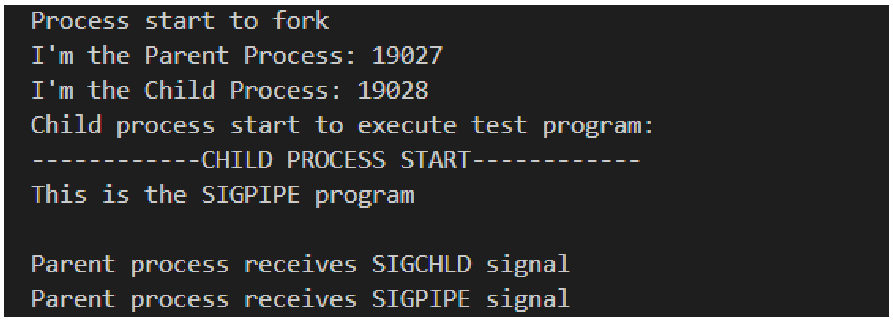
Figure 8: *K*ill Graph of Task 1



```
Process start to fork
I'm the Parent Process: 18987
I'm the Child Process: 18988
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
```

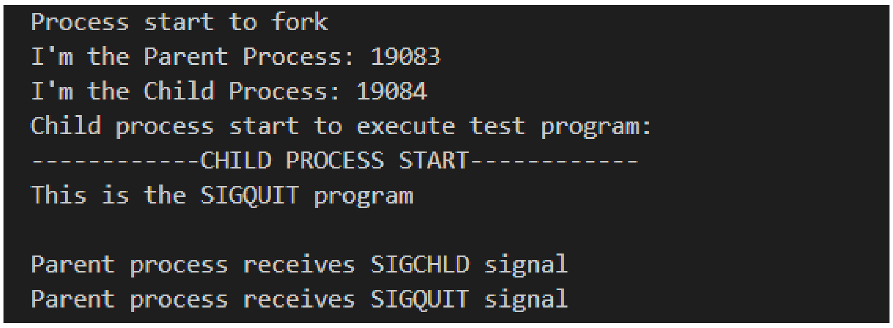
Figure 9: *N*ormal Graph of Task 1



```
Process start to fork
I'm the Parent Process: 19027
I'm the Child Process: 19028
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGPIPE program

Parent process receives SIGCHLD signal
Parent process receives SIGPIPE signal
```

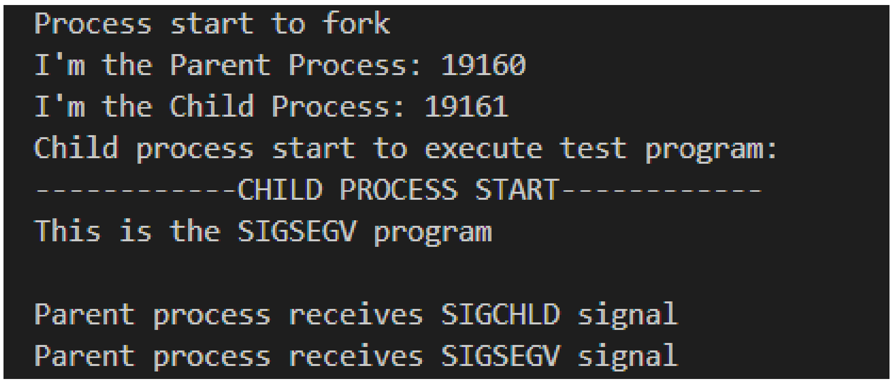
Figure 10: *P*ipe Graph of Task 1



```
Process start to fork
I'm the Parent Process: 19083
I'm the Child Process: 19084
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGQUIT program

Parent process receives SIGCHLD signal
Parent process receives SIGQUIT signal
```

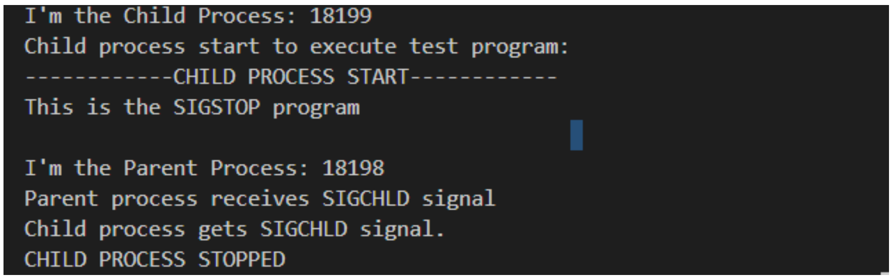
Figure 11: *Q*uit Graph of Task 1



```
Process start to fork
I'm the Parent Process: 19160
I'm the Child Process: 19161
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGSEGV program

Parent process receives SIGCHLD signal
Parent process receives SIGSEGV signal
```

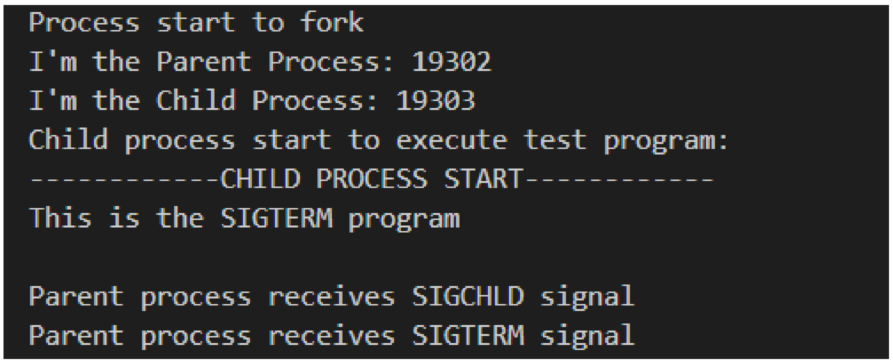
Figure 12: *S*egment_fault Graph of Task 1



```
I'm the Child Process: 18199
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGSTOP program

I'm the Parent Process: 18198
Parent process receives SIGCHLD signal
Child process gets SIGCHLD signal.
CHILD PROCESS STOPPED
```

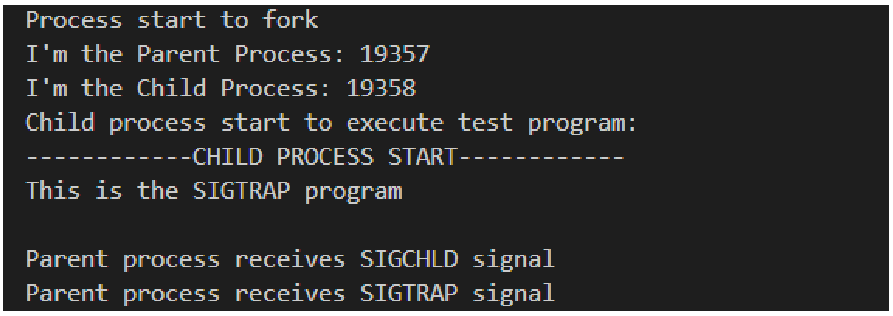
Figure 13: *S*top Graph of Task 1



```
Process start to fork
I'm the Parent Process: 19302
I'm the Child Process: 19303
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGTERM program

Parent process receives SIGCHLD signal
Parent process receives SIGTERM signal
```

Figure 14: *T*erminate Graph of Task 1



```
Process start to fork
I'm the Parent Process: 19357
I'm the Child Process: 19358
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGTRAP program

Parent process receives SIGCHLD signal
Parent process receives SIGTRAP signal
```

Figure 15: *T*rap Graph of Task 1

3. **Task 2**

   (a) Program Design

      i. Some functions we need to use are non-static, we should firstly use `EXPORT_SYMBOL()` after the functions definition so that it could be used in my own kernel module. After that, we should compile the kernel source code and install it.

      ii. Then we import the functions, using `extern FUNCTION`.

      iii. We use

```
extern int do_execve(struct filename *filename,
            const char __user *const __user *__argv,
            const char __user *const __user *__envp);
```

implement `int my_exec(void)` function to execute the test program. We use `getname_kernel()` to get the path. We set path as /tmp/test because of the requirement.

      iv. We use

```
extern long do_wait(struct wait_opts *wo);
```

implement `void my_wait(pid_t pid)` function to terminate to wait for the signal from the child process and print out the signal.

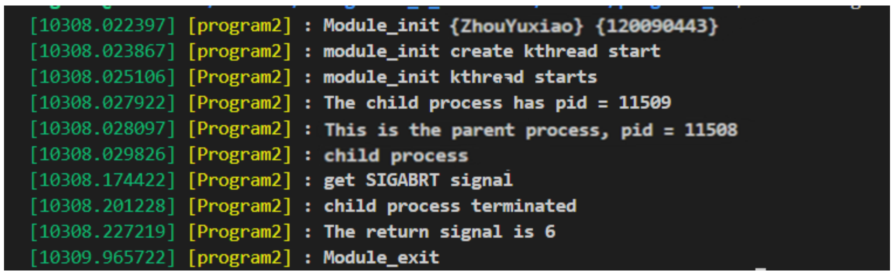
      v. Within `int my_fork(void *argc)`, fork a process to execute the test program. We use kernel_clone and set the arguments as follows.

```
extern pid_t kernel_clone(struct kernel_clone_args *args);

struct kernel_clone_args kargs = {
    .flags = SIGCHLD,
    .stack = (unsigned long)&my_exec,
    .stack_size = 0,
    .parent_tid = NULL,
    .child_tid = NULL,
    .tls = 0,
    .exit_signal = SIGCHLD };
```

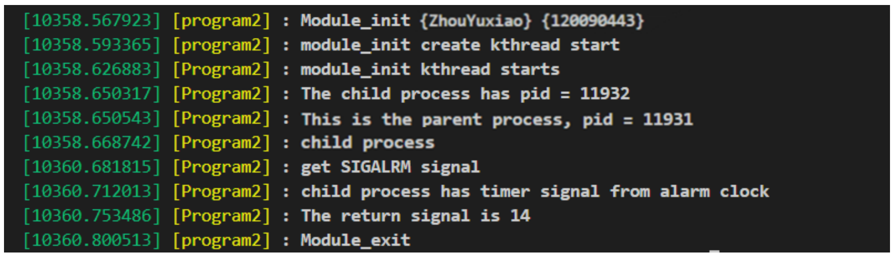
      vi. Then implement `static int __init program2_init(void)`. When program2.ko being initialized, create a kernel thread and run my_fork function. The parent process will wait until child process terminates and print out the process id for both parent and child process. Within this test program, it will raise signal. The signal could be caught and related message should be printed out in kernel log.

   (b) The output display



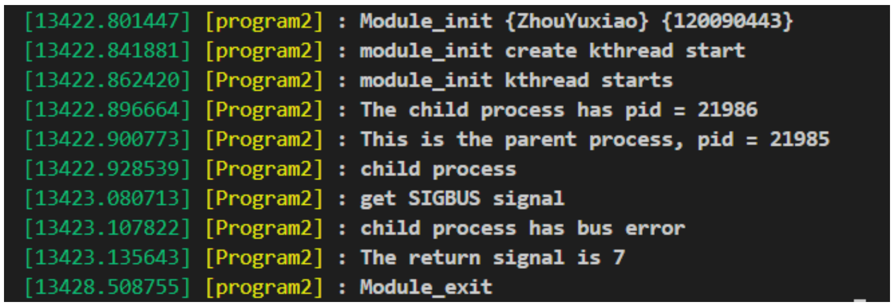
```
[10308.022397] [program2] : Module_init {ZhouYuxiao} {120090443}
[10308.023867] [program2] : module_init create kthread start
[10308.025106] [Program2] : module_init kthread starts
[10308.027922] [Program2] : The child process has pid = 11509
[10308.028097] [Program2] : This is the parent process, pid = 11508
[10308.029826] [Program2] : child process
[10308.174422] [Program2] : get SIGABRT signal
[10308.201228] [Program2] : child process terminated
[10308.227219] [Program2] : The return signal is 6
[10309.965722] [program2] : Module_exit
```

Figure 1: *Abort Graph of Task 2*



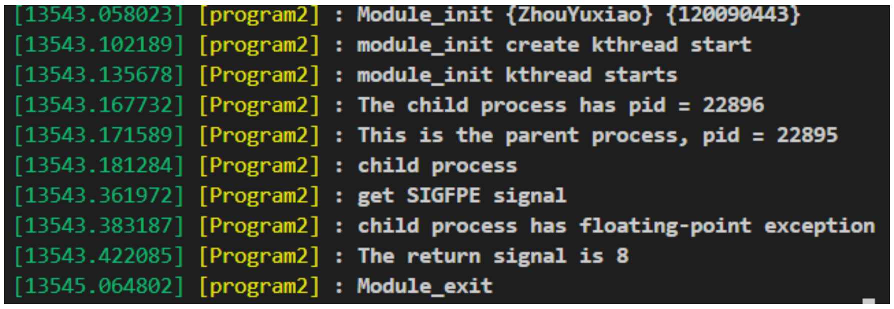
```
[10358.567923] [program2] : Module_init {ZhouYuxiao} {120090443}
[10358.593365] [program2] : module_init create kthread start
[10358.626883] [Program2] : module_init kthread starts
[10358.650317] [Program2] : The child process has pid = 11932
[10358.650543] [Program2] : This is the parent process, pid = 11931
[10358.668742] [Program2] : child process
[10360.681815] [Program2] : get SIGALRM signal
[10360.712013] [Program2] : child process has timer signal from alarm clock
[10360.753486] [Program2] : The return signal is 14
[10360.800513] [program2] : Module_exit
```

Figure 2: *A*larm Graph of Task 2



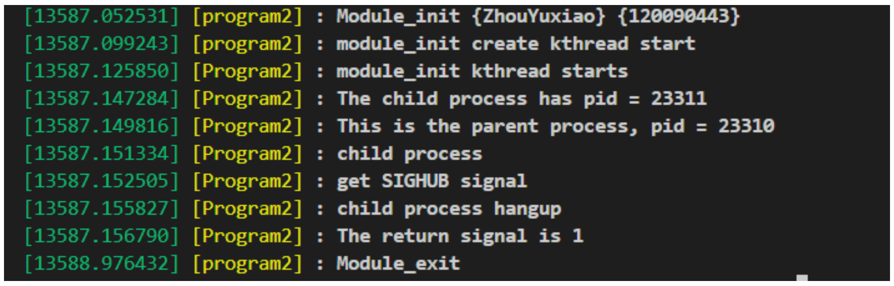
```
[13422.801447] [program2] : Module_init {ZhouYuxiao} {120090443}
[13422.841881] [program2] : module_init create kthread start
[13422.862420] [Program2] : module_init kthread starts
[13422.896664] [Program2] : The child process has pid = 21986
[13422.900773] [Program2] : This is the parent process, pid = 21985
[13422.928539] [Program2] : child process
[13423.080713] [Program2] : get SIGBUS signal
[13423.107822] [Program2] : child process has bus error
[13423.135643] [Program2] : The return signal is 7
[13428.508755] [program2] : Module_exit
```

Figure 3: *B*us Graph of Task 2



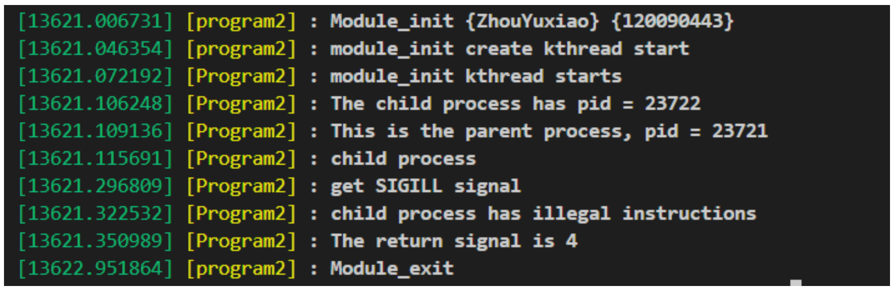
```
[13543.058023] [program2] : Module_init {ZhouYuxiao} {120090443}
[13543.102189] [program2] : module_init create kthread start
[13543.135678] [Program2] : module_init kthread starts
[13543.167732] [Program2] : The child process has pid = 22896
[13543.171589] [Program2] : This is the parent process, pid = 22895
[13543.181284] [Program2] : child process
[13543.361972] [Program2] : get SIGFPE signal
[13543.383187] [Program2] : child process has floating-point exception
[13543.422085] [Program2] : The return signal is 8
[13545.064802] [program2] : Module_exit
```

Figure 4: *F*loating Graph of Task 2



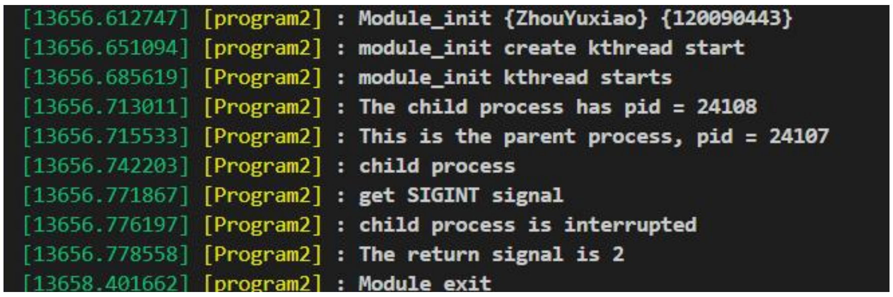
```
[13587.052531] [program2] : Module_init {ZhouYuxiao} {120090443}
[13587.099243] [program2] : module_init create kthread start
[13587.125850] [Program2] : module_init kthread starts
[13587.147284] [Program2] : The child process has pid = 23311
[13587.149816] [Program2] : This is the parent process, pid = 23310
[13587.151334] [Program2] : child process
[13587.152505] [Program2] : get SIGHUB signal
[13587.155827] [Program2] : child process hangup
[13587.156790] [Program2] : The return signal is 1
[13588.976432] [program2] : Module_exit
```

Figure 5: *H*angup Graph of Task 2



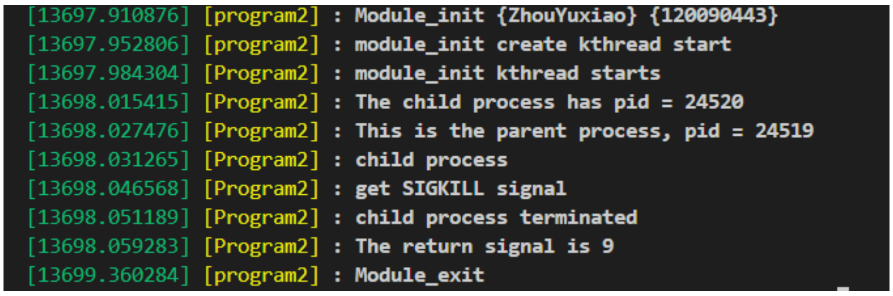
```
[13621.006731] [program2] : Module_init {ZhouYuxiao} {120090443}
[13621.046354] [program2] : module_init create kthread start
[13621.072192] [Program2] : module_init kthread starts
[13621.106248] [Program2] : The child process has pid = 23722
[13621.109136] [Program2] : This is the parent process, pid = 23721
[13621.115691] [Program2] : child process
[13621.296809] [Program2] : get SIGILL signal
[13621.322532] [Program2] : child process has illegal instructions
[13621.350989] [Program2] : The return signal is 4
[13622.951864] [program2] : Module_exit
```

Figure 6: *I*llegal_instr Graph of Task 2



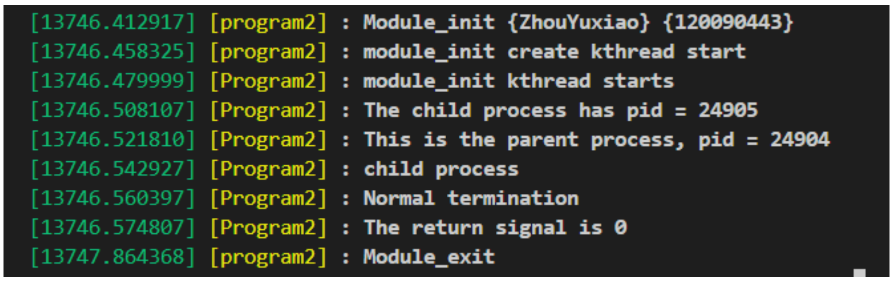
```
[13656.612747] [program2] : Module_init {ZhouYuxiao} {120090443}
[13656.651094] [program2] : module_init create kthread start
[13656.685619] [Program2] : module_init kthread starts
[13656.713011] [Program2] : The child process has pid = 24108
[13656.715533] [Program2] : This is the parent process, pid = 24107
[13656.742203] [Program2] : child process
[13656.771867] [Program2] : get SIGINT signal
[13656.776197] [Program2] : child process is interrupted
[13656.778558] [Program2] : The return signal is 2
[13658.401662] [program2] : Module exit
```

Figure 7: *I*nterrupt Graph of Task 2



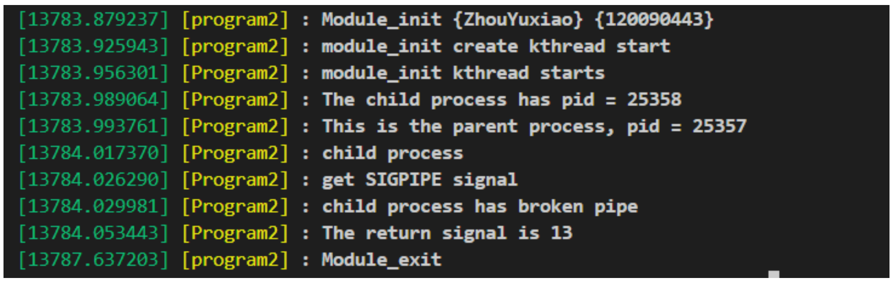
```
[13697.910876] [program2] : Module_init {ZhouYuxiao} {120090443}
[13697.952806] [program2] : module_init create kthread start
[13697.984304] [Program2] : module_init kthread starts
[13698.015415] [Program2] : The child process has pid = 24520
[13698.027476] [Program2] : This is the parent process, pid = 24519
[13698.031265] [Program2] : child process
[13698.046568] [Program2] : get SIGKILL signal
[13698.051189] [Program2] : child process terminated
[13698.059283] [Program2] : The return signal is 9
[13699.360284] [program2] : Module_exit
```

Figure 8: *K*ill Graph of Task 2



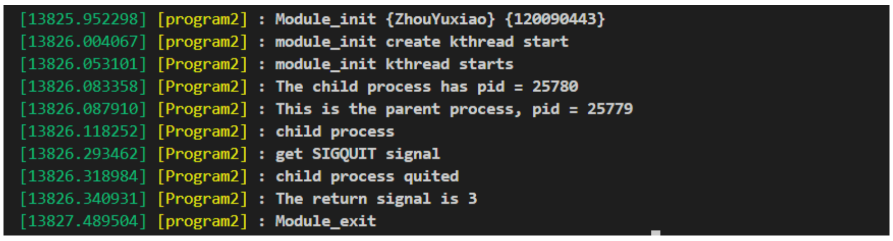
```
[13746.412917] [program2] : Module_init {ZhouYuxiao} {120090443}
[13746.458325] [program2] : module_init create kthread start
[13746.479999] [Program2] : module_init kthread starts
[13746.508107] [Program2] : The child process has pid = 24905
[13746.521810] [Program2] : This is the parent process, pid = 24904
[13746.542927] [Program2] : child process
[13746.560397] [Program2] : Normal termination
[13746.574807] [Program2] : The return signal is 0
[13747.864368] [program2] : Module_exit
```

Figure 9: *N*ormal Graph of Task 2



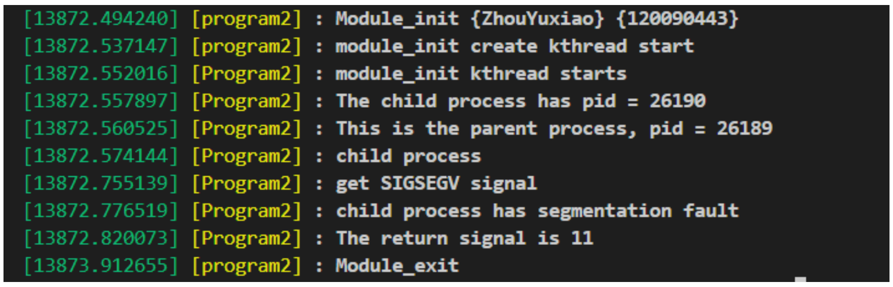
```
[13783.879237] [program2] : Module_init {ZhouYuxiao} {120090443}
[13783.925943] [program2] : module_init create kthread start
[13783.956301] [Program2] : module_init kthread starts
[13783.989064] [Program2] : The child process has pid = 25358
[13783.993761] [Program2] : This is the parent process, pid = 25357
[13784.017370] [Program2] : child process
[13784.026290] [Program2] : get SIGPIPE signal
[13784.029981] [Program2] : child process has broken pipe
[13784.053443] [Program2] : The return signal is 13
[13787.637203] [program2] : Module_exit
```

Figure 10: *P*ipe Graph of Task 2



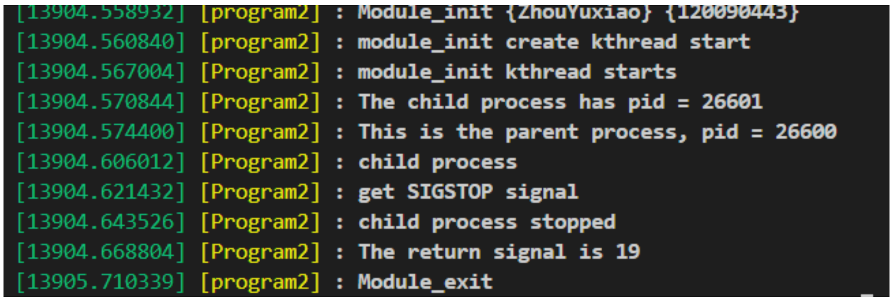
```
[13825.952298] [program2] : Module_init {ZhouYuxiao} {120090443}
[13826.004067] [program2] : module_init create kthread start
[13826.053101] [Program2] : module_init kthread starts
[13826.083358] [Program2] : The child process has pid = 25780
[13826.087910] [Program2] : This is the parent process, pid = 25779
[13826.118252] [Program2] : child process
[13826.293462] [Program2] : get SIGQUIT signal
[13826.318984] [Program2] : child process quited
[13826.340931] [Program2] : The return signal is 3
[13827.489504] [program2] : Module_exit
```

Figure 11: *Q*uit Graph of Task 2



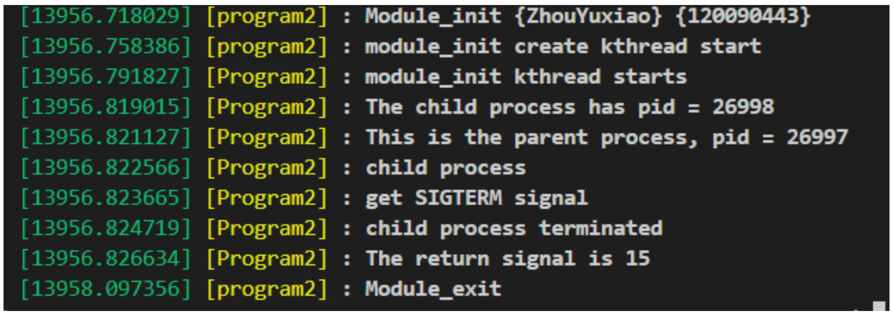
```
[13872.494240] [program2] : Module_init {ZhouYuxiao} {120090443}
[13872.537147] [program2] : module_init create kthread start
[13872.552016] [Program2] : module_init kthread starts
[13872.557897] [Program2] : The child process has pid = 26190
[13872.560525] [Program2] : This is the parent process, pid = 26189
[13872.574144] [Program2] : child process
[13872.755139] [Program2] : get SIGSEGV signal
[13872.776519] [Program2] : child process has segmentation fault
[13872.820073] [Program2] : The return signal is 11
[13873.912655] [program2] : Module_exit
```

Figure 12: Segment_fault Graph of Task 2



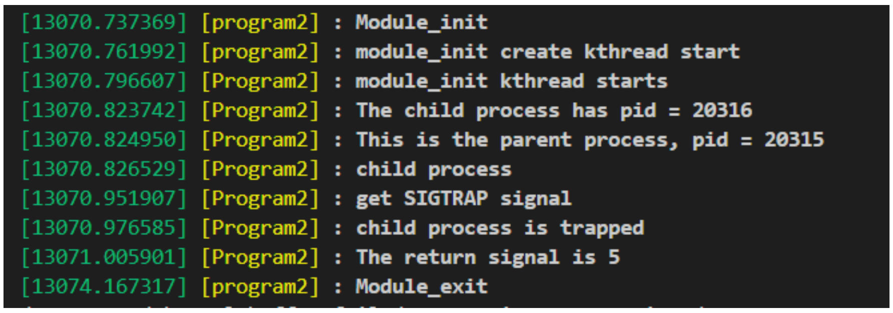
```
[13904.558932] [program2] : Module_init {ZhouYuxiao} {120090443}
[13904.560840] [program2] : module_init create kthread start
[13904.567004] [Program2] : module_init kthread starts
[13904.570844] [Program2] : The child process has pid = 26601
[13904.574400] [Program2] : This is the parent process, pid = 26600
[13904.606012] [Program2] : child process
[13904.621432] [Program2] : get SIGSTOP signal
[13904.643526] [Program2] : child process stopped
[13904.668804] [Program2] : The return signal is 19
[13905.710339] [program2] : Module_exit
```

Figure 13: Stop Graph of Task 2



```
[13956.718029] [program2] : Module_init {ZhouYuxiao} {120090443}
[13956.758386] [program2] : module_init create kthread start
[13956.791827] [Program2] : module_init kthread starts
[13956.819015] [Program2] : The child process has pid = 26998
[13956.821127] [Program2] : This is the parent process, pid = 26997
[13956.822566] [Program2] : child process
[13956.823665] [Program2] : get SIGTERM signal
[13956.824719] [Program2] : child process terminated
[13956.826634] [Program2] : The return signal is 15
[13958.097356] [program2] : Module_exit
```

Figure 14: Terminate Graph of Task 2

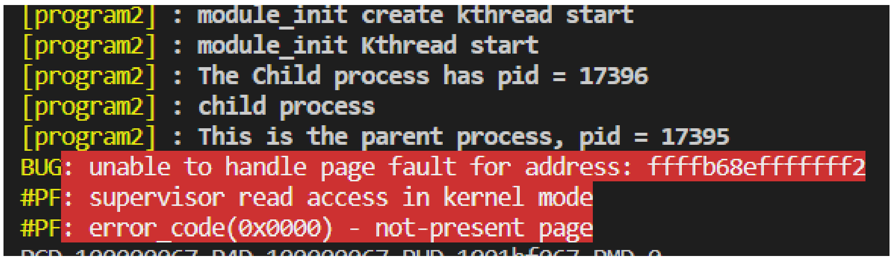


```
[13070.737369] [program2] : Module_init
[13070.761992] [program2] : module_init create kthread start
[13070.796607] [Program2] : module_init kthread starts
[13070.823742] [Program2] : The child process has pid = 20316
[13070.824950] [Program2] : This is the parent process, pid = 20315
[13070.826529] [Program2] : child process
[13070.951907] [Program2] : get SIGTRAP signal
[13070.976585] [Program2] : child process is trapped
[13071.005901] [Program2] : The return signal is 5
[13074.167317] [program2] : Module_exit
```

Figure 15: Trap Graph of Task 2

4. Problem summary (What did I learn from the tasks)

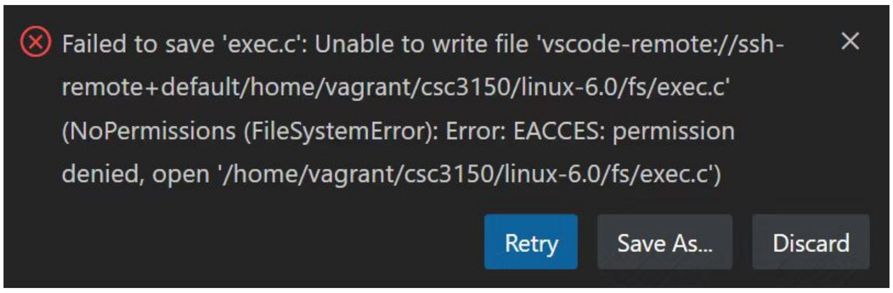
   (a) **Unable to handle page fault for address**



```
[program2] : module_init create kthread start
[program2] : module_init Kthread start
[program2] : The Child process has pid = 17396
[program2] : child process
[program2] : This is the parent process, pid = 17395
BUG: unable to handle page fault for address: ffffb68effffffff2
#PF: supervisor read access in kernel mode
#PF: error_code(0x0000) - not-present page
```

   In this task2, the main problem should be the **path** variable setting in the function, and the correct absolute path should be set.

   (b) **Unable write the file**



Failed to save 'exec.c': Unable to write file 'vscode-remote://ssh-remote+default/home/vagrant/csc3150/linux-6.0/fs/exec.c' (NoPermissions (FileSystemError): Error: EACCES: permission denied, open '/home/vagrant/csc3150/linux-6.0/fs/exec.c')

   Go to the corresponding folder, type the chmod a+w -filename, and open the write access.