CSC3150 Operating System

Assignment Report #1

Name: Lyu Minen

Student ID: 120090828

Date: 2022/10/07

The Chinese University of Hong Kong, Shenzhen

1. Program1

1.1 program design

In this program, we are asked to fork a child process and let the parent process wait until the child process is finished. The corresponding message and signals should be printed. Since the program is in user mode, I used the system call functions to help.

In the beginning, I need to fork the process. The function fork ( ) is used and it returns the PID. If the process is successfully created, the child process can execute the test program then. When the test program finishes, the child process would return the signal to the parent process. The parent process would wait until it receives the signal. For the waiting process. I used the function waitpid ( ) instead of wait ( ) since I hope to pass the argument “WUNTRACED” to report the stop of the child process. Otherwise, the parent process would keep waiting in SIGSTOP case. Then, I used an if statement to check the termination state of the child process. The three functions we used to get the status is WIFEXITED, WIFSTOPPED and WIFSIGNALED. For the third case WIFSIGNALED, its situations can be further divided by WTERMSIG. The things we need to print at the end of the program are related to those return values. At last, I exit the program normally by the exit function.

The structure of the whole program is similar to this:

```
pid = fork();

if (pid == -1){//cannot fork successfully
}

else {
    //child process
    if (pid == 0){
        //deal with the argc, argv of child process
        //print related things

    /* execute test program */
    execve(arg[0], arg,NULL);
    perror("execve");//avoid error
    exit(EXIT_FAILURE);
    }
    else {//parent process
        printf("I'm Parent Process, my pid = %d\n", getpid());

    /* wait for child process terminates */
        waitpid(-1, &status, WUNTRACED);
        printf("Parent process receives SIGCHLD signal \n");

    /* check child process'  termination status */
        if (WIFEXITED(status)){//normal
        }
        else if (WIFSTOPPED(status)) {//stop
        }
        else if (WIFSIGNALED(status)){//exit because of a signal
            if (WTERMSIG(status) == 13) {
               //print different things according to the signal received
               }
        }
        else {//avoid there is other case
        }
        exit (0);//exit normally
```

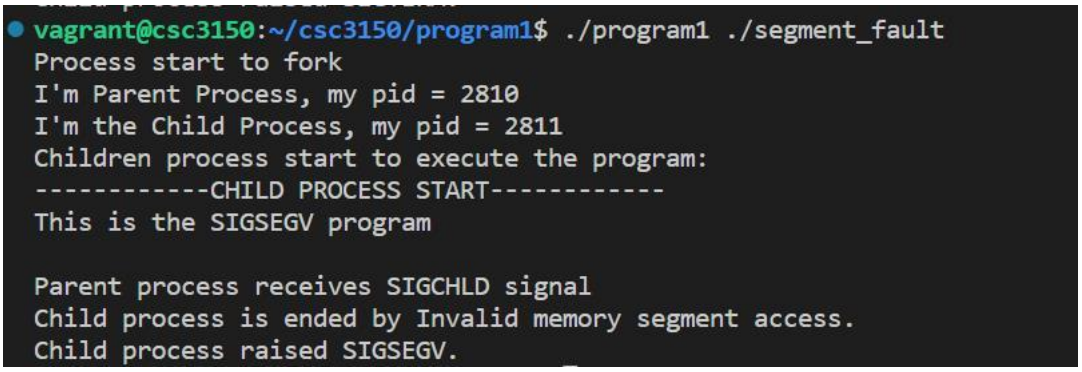
1.2 environment development

Since the program is designed in user mode, I do not need to change the environment specifically. The only things I need to make sure is the version of GCC and Linux.

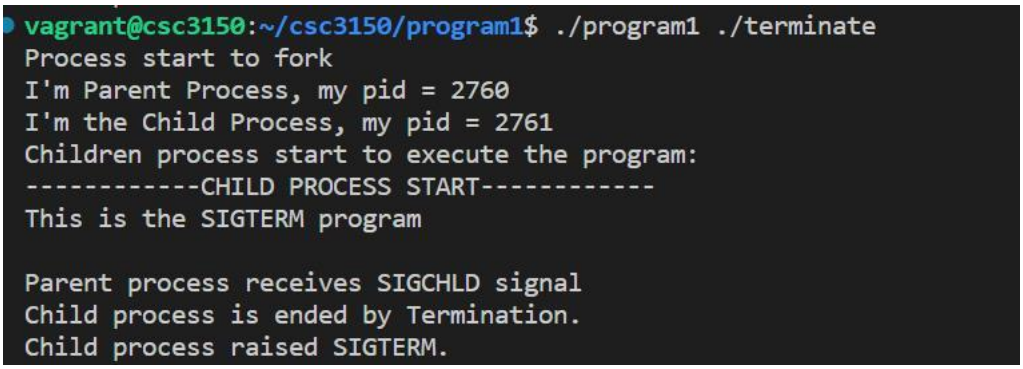
1.3 outputs of the program

The results of the program are listed here:

Segment fault:



Terminate:



Quit:

```
vagrant@csc3150:~/csc3150/program1$ ./program1 ./quit
Process start to fork
I'm Parent Process, my pid = 2850
I'm the Child Process, my pid = 2851
Children process start to execute the program:
------------CHILD PROCESS START------------
This is the SIGQUIT program

Parent process receives SIGCHLD signal
Child process is ended by Terminal quit.
Child process raised SIGQUIT.
```

Pipe:

```
vagrant@csc3150:~/csc3150/program1$ ./program1 ./pipe
Process start to fork
I'm Parent Process, my pid = 2914
I'm the Child Process, my pid = 2915
Children process start to execute the program:
------------CHILD PROCESS START------------
This is the SIGPIPE program

Parent process receives SIGCHLD signal
Child process is ended by pipe error.
Child process raised SIGPIPE.
```

Normal:

```
vagrant@csc3150:~/csc3150/program1$ ./program1 ./normal
Process start to fork
I'm Parent Process, my pid = 2940
I'm the Child Process, my pid = 2941
Children process start to execute the program:
------------CHILD PROCESS START------------
This is the normal program

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

Kill:

```
vagrant@csc3150:~/csc3150/program1$ ./program1 ./kill
Process start to fork
I'm Parent Process, my pid = 2966
I'm the Child Process, my pid = 2967
Children process start to execute the program:
------------CHILD PROCESS START------------
This is the SIGKILL program

Parent process receives SIGCHLD signal
Child process is killed.
Child process raised SIGKILL.
```

Interrupt:

```
vagrant@csc3150:~/csc3150/program1$ ./program1 ./interrupt
Process start to fork
I'm Parent Process, my pid = 3008
I'm the Child Process, my pid = 3009
Children process start to execute the program:
------------CHILD PROCESS START------------
This is the SIGINT program

Parent process receives SIGCHLD signal
Child process is ended by Terminal interrupt.
Child process raised SIGINT.
```

Hang up:

```
vagrant@csc3150:~/csc3150/program1$ ./program1 ./hangup
Process start to fork
I'm Parent Process, my pid = 3046
I'm the Child Process, my pid = 3047
Children process start to execute the program:
------------CHILD PROCESS START------------
This is the SIGHUP program

Parent process receives SIGCHLD signal
Child process is ended by Hangup signal.
Child process raised SIGHUP.
```

Floating:

```
Child process raised SIGHUP.
vagrant@csc3150:~/csc3150/program1$ ./program1 ./floating
Process start to fork
I'm Parent Process, my pid = 3085
I'm the Child Process, my pid = 3086
Children process start to execute the program:
------------CHILD PROCESS START------------
This is the SIGFPE program

Parent process receives SIGCHLD signal
Child process is ended by Floating point exception.
Child process raised SIGFPE.
```

Bus:

```
vagrant@csc3150:~/csc3150/program1$ ./program1 ./bus
Process start to fork
I'm Parent Process, my pid = 3124
I'm the Child Process, my pid = 3125
Children process start to execute the program:
------------CHILD PROCESS START------------
This is the SIGBUS program

Parent process receives SIGCHLD signal
Child process is ended by Bus error.
Child process raised SIGBUS.
```

Alarm:

```
vagrant@csc3150:~/csc3150/program1$ ./program1 ./alarm
Process start to fork
I'm Parent Process, my pid = 3151
I'm the Child Process, my pid = 3152
Children process start to execute the program:
------------CHILD PROCESS START------------
This is the SIGALRM program

Parent process receives SIGCHLD signal
Child process is ended by alarm clock.
Child process raised SIGALRM.
```

Abort:

```
vagrant@csc3150:~/csc3150/program1$ ./program1 ./abort
Process start to fork
I'm Parent Process, my pid = 3432
I'm the Child Process, my pid = 3433
Children process start to execute the program:
------------CHILD PROCESS START------------
This is the SIGABRT program

Parent process receives SIGCHLD signal
Child process is aborted.
Child process raised SIGABRT.
```

Stop:

```
vagrant@csc3150:~/csc3150/program1$ ./program1 ./stop
Process start to fork
I'm Parent Process, my pid = 2215
I'm the Child Process, my pid = 2216
Children process start to execute the program:
------------CHILD PROCESS START------------
This is the SIGSTOP program

Parent process receives SIGCHLD signal
CHILD EXECUTION STOPPED
```

Trap:

```
vagrant@csc3150:~/csc3150/program1$ ./program1 ./trap
Process start to fork
I'm Parent Process, my pid = 2280
I'm the Child Process, my pid = 2281
Children process start to execute the program:
------------CHILD PROCESS START------------
This is the SIGTRAP program

Parent process receives SIGCHLD signal
Child process is ended by Trace trap.
Child process raised SIGTRAP.
```

Illegal instruction:

```
vagrant@csc3150:~/csc3150/program1$ ./program1 ./illegal_instr
Process start to fork
I'm Parent Process, my pid = 2365
I'm the Child Process, my pid = 2366
Children process start to execute the program:
------------CHILD PROCESS START------------
This is the SIGILL program

Parent process receives SIGCHLD signal
Child process is ended by Illegal instruction.
Child process raised SIGILL.
```

According to the newest announcement, I modify the code. The new output version is like this:

```
vagrant@csc3150:~/csc3150/program1$ ./program1 ./kill
Process start to fork
I'm Parent Process, my pid = 2746
I'm the Child Process, my pid = 2747
Children process start to execute the program:
------------CHILD PROCESS START------------
This is the SIGKILL program

Parent process receives SIGCHLD signal
Child process get SIGKILL signal.
```

2. Program2

2.1 program design

In program2, we are asked to create the process by ourselves. The process of creating procedures for this program is similar to the first one. However, we need

to write related functions by ourselves. Thus, I have my_wait ( ), my_exec ( ), my_fork ( ) and related initialize and exit functions.

In my_wait ( ), I use the function do_wait ( ) in /fs/exit.c. Since the signal received can be calculated by wo.wo_stat & 0x7f, I set the printing result according to the signal I figured out. Then, I print the corresponding signals and lines. In the end, I free the memory and decrease the count by put_pid ( ).

In my_exec ( ), I use the function do_execve ( ) inside. I first get the pointer of my filename through getname_kernel. Then, I pass the argument into the function do_exec ( ). If there is an error, my function would exit directly.

In my_fork ( ), I use the function kernel_thread ( ) to form a process. Then, I call my wait function. The parent process would wait until the child process finishes.

2.2 environment development

In this program, we need to compile the kernel and modify the c files to change the environment. First, I update the kernel to version 5.10.100.

Then, I modify the c files to let some functions export. These functions are do_wait, do_exec, kernel_thread and getname_kernel.

```
2518
2519        return kernel_clone(&args);
2520    }
2521
2522    EXPORT_SYMBOL(kernel_thread);
2523
```

Then I compile the kernel again. Otherwise, there would be the following errors:

```
  MODPOST /home/vagrant/csc3150/assignment1/src/program2/Module.symvers
ERROR: modpost: "do_wait" [/home/vagrant/csc3150/assignment1/src/program2/program2.ko] undefined!
ERROR: modpost: "do_execve" [/home/vagrant/csc3150/assignment1/src/program2/program2.ko] undefined!
ERROR: modpost: "getname_kernel" [/home/vagrant/csc3150/assignment1/src/program2/program2.ko] undefined!
scripts/Makefile.modpost:124: recipe for target '/home/vagrant/csc3150/assignment1/src/program2/Module.symvers' failed
```

After compiling the kernel, program2 can be compiled. Then, I insert the program2.ko and then remove it. The message inside the buffer can be listed through the command “dmesg”. The error I met here is “Unknown symbol in module”. I reinstall the module and reboot it. The problem is solved.

```
vagrant@csc3150:~/csc3150/assignment1/src/program2$ sudo su
root@csc3150:/home/vagrant/csc3150/assignment1/src/program2# insmod program2.ko
insmod: ERROR: could not insert module program2.ko: Unknown symbol in module
root@csc3150:/home/vagrant/csc3150/assignment1/src/program2#
```

2.3 outputs of the program

The example output is listed as followed:

SIGHUP:

```
[ 9320.956193] [program2] : Module_init {Lyu Minen} {120090828}
[ 9320.956197] [program2] : Module_init create kthread start
[ 9320.956355] [program2] : Module_init kthread start
[ 9320.956699] [program2] : The child process has pid = 13880
[ 9320.956702] [program2] : The parent process has pid = 13879
[ 9320.956708] [program2] : child process
[ 9320.957870] [program2] : get SIGHUP signal
[ 9320.957873] [program2] : child process hung up
[ 9320.957875] [program2] : The return signal is 1
[ 9324.738058] [program2] : Module_exit
```

Interrupt:

```
[ 9414.065237] [program2] : Module_init {Lyu Minen} {120090828}
[ 9414.065240] [program2] : Module_init create kthread start
[ 9414.065416] [program2] : Module_init kthread start
[ 9414.066754] [program2] : The child process has pid = 14864
[ 9414.066756] [program2] : The parent process has pid = 14862
[ 9414.066760] [program2] : child process
[ 9414.067481] [program2] : get SIGINT signal
[ 9414.067490] [program2] : child process terminal interrupt
[ 9414.067491] [program2] : The return signal is 2
[ 9417.178300] [program2] : Module_exit
```

Terminal:

```
[ 9452.288277] [program2] : Module_init {Lyu Minen} {120090828}
[ 9452.288281] [program2] : Module_init create kthread start
[ 9452.309926] [program2] : Module_init kthread start
[ 9452.310076] [program2] : The child process has pid = 15632
[ 9452.310079] [program2] : The parent process has pid = 15631
[ 9452.313886] [program2] : child process
[ 9452.557932] [program2] : get SIGQUIT signal
[ 9452.557935] [program2] : child process terminal quit
[ 9452.557937] [program2] : The return signal is 3
[ 9455.271052] [program2] : Module_exit
```

Instruction error:

```
[ 9485.525344] [program2] : Module_init {Lyu Minen} {120090828}
[ 9485.525347] [program2] : Module_init create kthread start
[ 9485.525466] [program2] : Module_init kthread start
[ 9485.527154] [program2] : The child process has pid = 16025
[ 9485.527157] [program2] : The parent process has pid = 16023
[ 9485.527162] [program2] : child process
[ 9485.768430] [program2] : get SIGILL signal
[ 9485.768433] [program2] : child process illegal instruction error
[ 9485.768434] [program2] : The return signal is 4
[ 9488.915850] [program2] : Module_exit
```

Trap error:

```
[ 9527.027785] [program2] : Module_init {Lyu Minen} {120090828}
[ 9527.027789] [program2] : Module_init create kthread start
[ 9527.027902] [program2] : Module_init kthread start
[ 9527.029557] [program2] : The child process has pid = 16795
[ 9527.029560] [program2] : The parent process has pid = 16793
[ 9527.029565] [program2] : child process
[ 9527.256837] [program2] : get SIGTRAP signal
[ 9527.256840] [program2] : child process trap error
[ 9527.256842] [program2] : The return signal is 5
[ 9529.751224] [program2] : Module_exit
```

Abort error:

```
[ 9554.602951] [program2] : Module_init {Lyu Minen} {120090828}
[ 9554.602955] [program2] : Module_init create kthread start
[ 9554.603009] [program2] : Module_init kthread start
[ 9554.604579] [program2] : The child process has pid = 17061
[ 9554.604582] [program2] : The parent process has pid = 17059
[ 9554.604588] [program2] : child process
[ 9554.841539] [program2] : get SIGABRT signal
[ 9554.841541] [program2] : child process abort error
[ 9554.841542] [program2] : The return signal is 6
[ 9557.481505] [program2] : Module_exit
```

Bus error:

```
[14040.486818] [program2] : Module_init {Lyu Minen} {120090828}
[14040.489552] [program2] : Module_init create kthread start
[14040.492696] [program2] : Module_init kthread start
[14040.496251] [program2] : The child process has pid = 4006
[14040.498844] [program2] : The parent process has pid = 4004
[14040.504416] [program2] : child process
[14040.744992] [program2] : get SIGBUS signal
[14040.750542] [program2] : child process bus error
[14040.753060] [program2] : The return signal is 7
[14043.489676] [program2] : Module_exit
```

Float error:

```
[ 9590.357627] [program2] : Module_init {Lyu Minen} {120090828}
[ 9590.357630] [program2] : Module_init create kthread start
[ 9590.357666] [program2] : Module_init kthread start
[ 9590.358614] [program2] : The child process has pid = 17367
[ 9590.358616] [program2] : The parent process has pid = 17365
[ 9590.358619] [program2] : child process
[ 9590.597803] [program2] : get SIGFPE signal
[ 9590.597807] [program2] : child process float error
[ 9590.597809] [program2] : The return signal is 8
[ 9592.490334] [program2] : Module_exit
```

Killed:

```
[ 9627.596190] [program2] : Module_init {Lyu Minen} {120090828}
[ 9627.596193] [program2] : Module_init create kthread start
[ 9627.596333] [program2] : Module_init kthread start
[ 9627.598037] [program2] : The child process has pid = 17720
[ 9627.598075] [program2] : The parent process has pid = 17718
[ 9627.598077] [program2] : child process
[ 9627.599501] [program2] : get SIGKILL signal
[ 9627.599503] [program2] : child process killed
[ 9627.599505] [program2] : The return signal is 9
[ 9630.033647] [program2] : Module_exit
```

Fault error:

```
[ 9662.178172] [program2] : Module_init {Lyu Minen} {120090828}
[ 9662.178176] [program2] : Module_init create kthread start
[ 9662.178331] [program2] : Module_init kthread start
[ 9662.179965] [program2] : The child process has pid = 18236
[ 9662.179968] [program2] : The parent process has pid = 18234
[ 9662.179975] [program2] : child process
[ 9662.426342] [program2] : get SIGSEGV signal
[ 9662.426345] [program2] : child process segmentation fault error
[ 9662.426346] [program2] : The return signal is 11
[ 9664.970491] [program2] : Module_exit
```

Pipe error:

```
[ 9690.211178] [program2] : Module_init {Lyu Minen} {120090828}
[ 9690.211182] [program2] : Module_init create kthread start
[ 9690.211370] [program2] : Module_init kthread start
[ 9690.212384] [program2] : The child process has pid = 18628
[ 9690.212386] [program2] : The parent process has pid = 18626
[ 9690.212390] [program2] : child process
[ 9690.213078] [program2] : get SIGPIPE signal
[ 9690.213080] [program2] : child process pipe error
[ 9690.213082] [program2] : The return signal is 13
[ 9692.596754] [program2] : Module_exit
```

Alarm error:

```
[ 9761.043598] [program2] : Module_init {Lyu Minen} {120090828}
[ 9761.043601] [program2] : Module_init create kthread start
[ 9761.043711] [program2] : Module_init kthread start
[ 9761.043779] [program2] : The child process has pid = 19402
[ 9761.043781] [program2] : The parent process has pid = 19401
[ 9761.043787] [program2] : child process
[ 9763.046116] [program2] : get SIGALARM signal
[ 9763.046121] [program2] : child process alarm error
[ 9763.046125] [program2] : The return signal is 14
[ 9763.865385] [program2] : Module_exit
```

Terminated:

```
[ 9800.842284] [program2] : Module_init {Lyu Minen} {120090828}
[ 9800.842288] [program2] : Module_init create kthread start
[ 9800.842576] [program2] : Module_init kthread start
[ 9800.842746] [program2] : The child process has pid = 19794
[ 9800.842773] [program2] : The parent process has pid = 19793
[ 9800.843062] [program2] : child process
[ 9800.846149] [program2] : get SIGTERM signal
[ 9800.846153] [program2] : child process terminated
[ 9800.846155] [program2] : The return signal is 15
[ 9803.182138] [program2] : Module_exit
```

Stop:

```
[23635.279188] [program2] : Module_init {Lyu Minen} {120090828}
[23635.279191] [program2] : Module_init create kthread start
[23635.279227] [program2] : Module_init kthread start
[23635.279886] [program2] : The child process has pid = 6605
[23635.279887] [program2] : The parent process has pid = 6603
[23635.279889] [program2] : child process
[23635.280177] [program2] : get SIGSTOP signal
[23635.280178] [program2] : child process stopped
[23638.767112] [program2] : Module_exit
```

Normal:

```
[13926.408928] [program2] : Module_init {Lyu Minen} {120090828}
[13926.412427] [program2] : Module_init create kthread start
[13926.416270] [program2] : Module_init kthread start
[13926.420409] [program2] : The child process has pid = 3380
[13926.423421] [program2] : The parent process has pid = 3379
[13926.429283] [program2] : child process
[13926.430922] [program2] : child process exit normally
[13926.437093] [program2] : The return signal is 0
[13929.644798] [program2] : Module_exit
```

3. Bonus

3.1 program design

In bonus, we are asked to build a pstree by ourself. The reference we can have is the file inside /proc. Since the status file is the most readable, I choose to read PID and PPID from "/proc/PID/status". The name, PID, PPID are clear there.

```
vagrant@csc3150:~/csc3150/assignment1/bonus$ cat /proc/978/status
Name:   acpid
Umask:  0022
State:  S (sleeping)
Tgid:   978
Ngid:   0
Pid:    978
PPid:   1
```

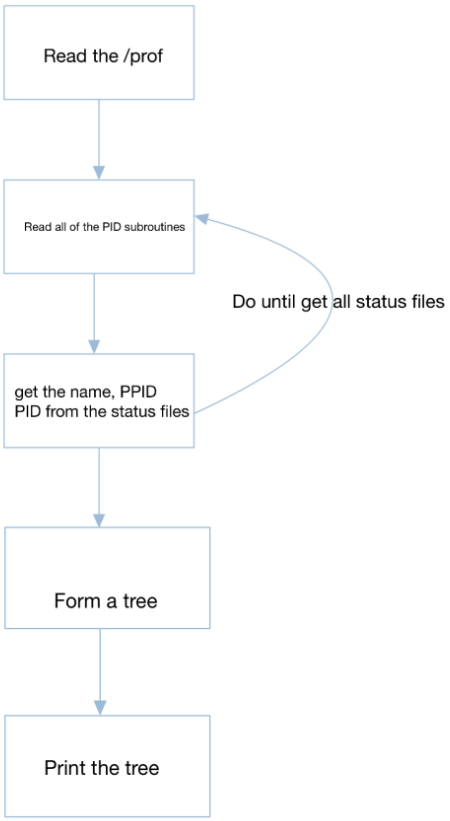
In this program, I can only form a tree after I have all information of PID, PPID and their name. Thus, the node structure I form here is not only used to build a tree, but used to build a linked list. The "next" is to form a linked list. The "parent" and "children" is used to form a tree.

Thus, for the whole program, the first thing I need to do is open the directory and read all of the related files. opendir ( ) and readdir ( ) is used to read through the "/proc". For the return value of readdir ( ), I write another function read_files_inside ( ) to read through the files inside its subroutines. This function would open the PID number folder to see the "status" file of it. The PID, PPID and name can be got by picking things before and after ":". Then, I store the pid information in the node I created. After reading all directories, the structure I have is only a linked list.

Then, I start to build a tree. Since I already know the PPID of each node, I can link the parent node with the children node directly. The function find_node ( ) is used to find the parent node here.

Next, I start to print the tree. I find the node with PID equals to 1 first. Then, I print the information for each node by recursion function.

Then, the whole process is finished. The structure of my program is as below:



3.2 environment development

In this program, I write the pstree in user mode. Thus, I do not change the environment. The Makefile I write help me to compile the pstree.

3.3 outputs of the program

The example output is listed as followed: I failed to print as a real pstree, which means father and the first son are in the same line. But the PID, PPID and name I get is right. I printed their information as a normal tree before (this is stored in test.c. You can try by yourself.) After “gcc” and “./test”, it looks like this:

```
vagrant@csc3150:~/csc3150/assignment1/bonus$ ./test
name:   kthreadd pid: 2, ppid: 0
name:   kworker/0 pid: 3152, ppid: 2
name:   kworker/u4 pid: 3011, ppid: 2
name:   rdma_cm pid: 448, ppid: 2
name:   ib_nl_sa_wq pid: 439, ppid: 2
name:   ib_mcast pid: 438, ppid: 2
name:   ib-comp-unb-wq pid: 437, ppid: 2
name:   ib-comp-wq pid: 435, ppid: 2
name:   iscsi_destroy pid: 399, ppid: 2
name:   iscsi_eh pid: 398, ppid: 2
name:   ext4-rsv-conver pid: 302, ppid: 2
name:   jbd2/sda1-8 pid: 301, ppid: 2
name:   raid5wq pid: 247, ppid: 2
name:   scsi_tmf_2 pid: 169, ppid: 2
name:   scsi_eh_2 pid: 168, ppid: 2
name:   kworker/1 pid: 150, ppid: 2
name:   cryptd pid: 139, ppid: 2
name:   mpt/0 pid: 138, ppid: 2
name:   mpt_poll_0 pid: 137, ppid: 2
name:   kworker/0 pid: 136, ppid: 2
name:   kworker/u5 pid: 85, ppid: 2
name:   charger_manager pid: 84, ppid: 2
name:   zswap-shrink pid: 79, ppid: 2
name:   kworker/1 pid: 76, ppid: 2
name:   ipv6_addrconf pid: 75, ppid: 2
name:   scsi_tmf_1 pid: 72, ppid: 2
name:   scsi_eh_1 pid: 71, ppid: 2
name:   scsi_tmf_0 pid: 70, ppid: 2
name:   scsi_eh_0 pid: 69, ppid: 2
name:   acpi_thermal_pm pid: 68, ppid: 2
name:   kthrotld pid: 67, ppid: 2
name:   ecryptfs-kthrea pid: 66, ppid: 2
name:   kswapd0 pid: 65, ppid: 2
```

However, the tree I printed is like this. (The source code is in pstree.c in my bonus folder. For the process whose parent is killed or removed, I would not print them.

```
vagrant@csc3150:~/csc3150/assignment1/bonus$ ./pstree
systemd
├──  systemd-journal
├──  lvmetad
├──  systemd-udevd
├──  systemd-timesyn
├──  dhclient
├──  iscsid
├──  iscsid
├──  atd
├──  systemd-logind
├──  lxcfs
├──  rsyslogd
├──  accounts-daemon
├──  acpid
├──  cron
├──  dbus-daemon
├──  sshd
└──  sshd
  └──  sshd
    └──  bash
      ├──  sh
      └──  node
        ├──  node
        └──  bash
          └──  pstree
├──  node
└──  node
└──  sleep
```

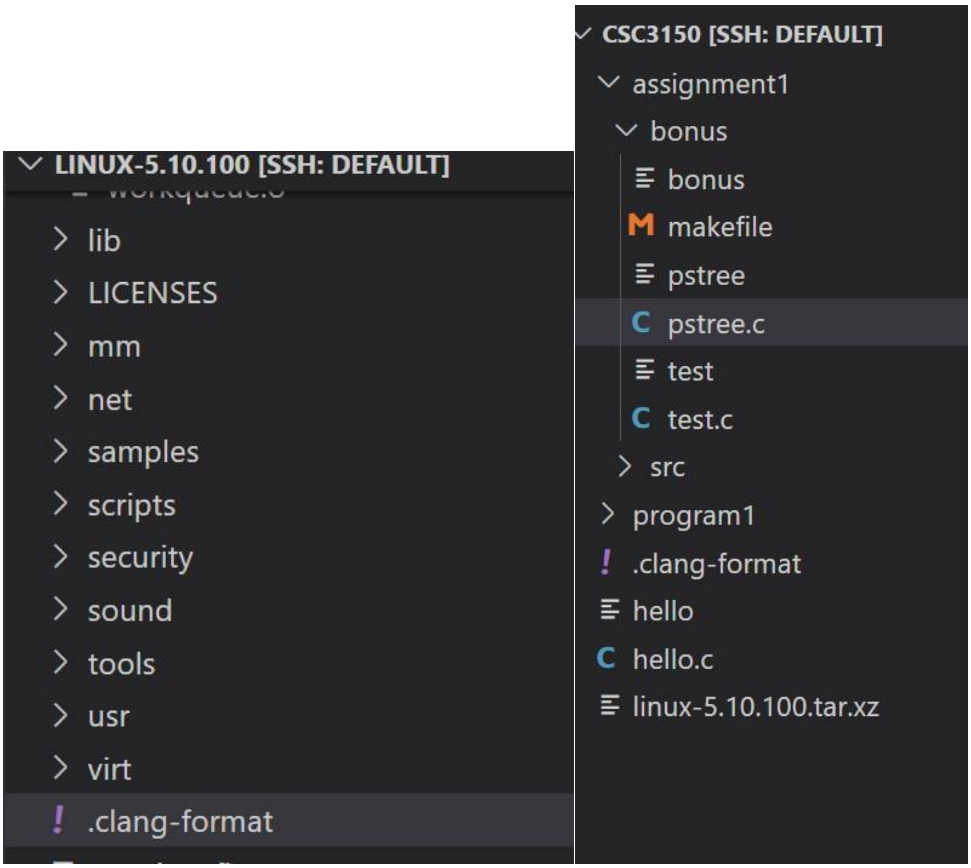
The screenshot is not fully displayed. You can see it in the terminal.

4. Notice

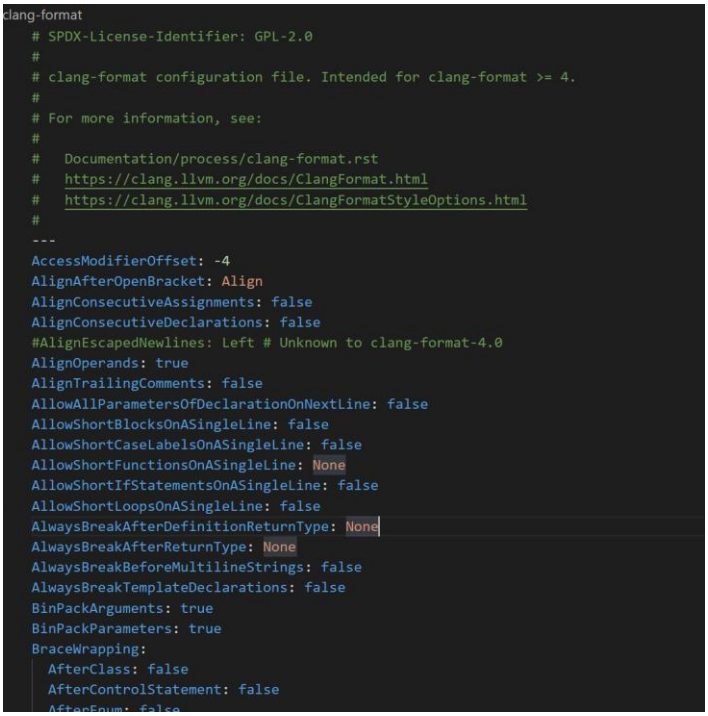
4.1 clang format

For the clang-format, I have already change the .clang-format file into the kernel version one. However, the coding style is not the same as the kernel files in my VS Code. There are some of my setting screenshots:

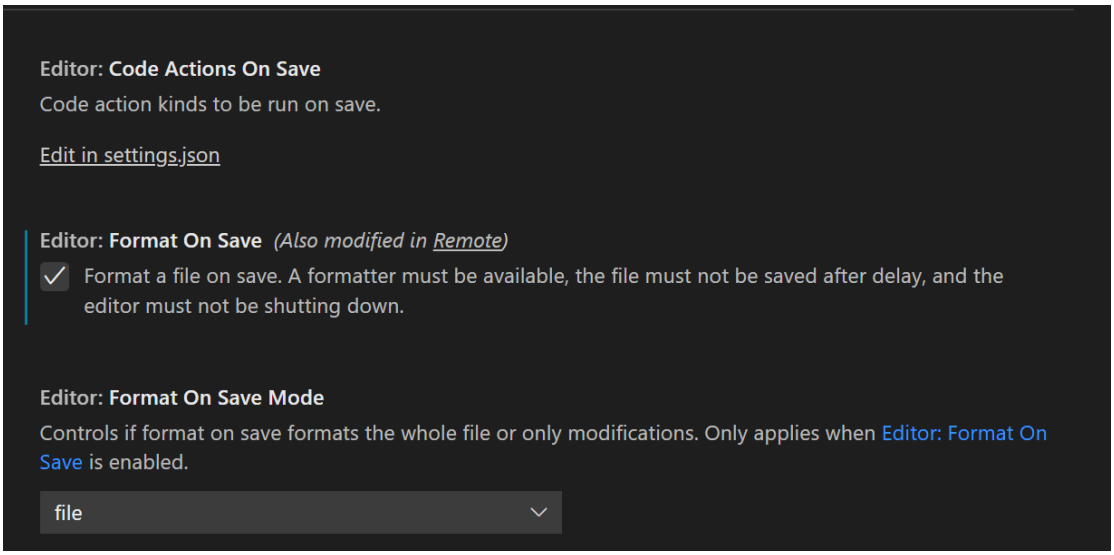
I do have .clang-format:



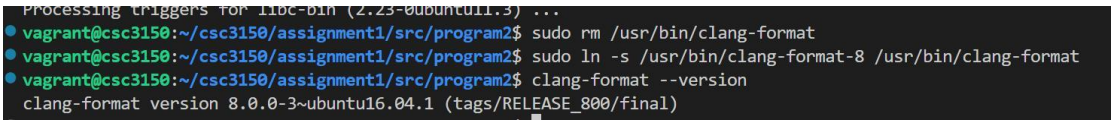
The things inside the .clang-format (just a small part):



I have changed the setting of VS Code to format on save and already use the command:

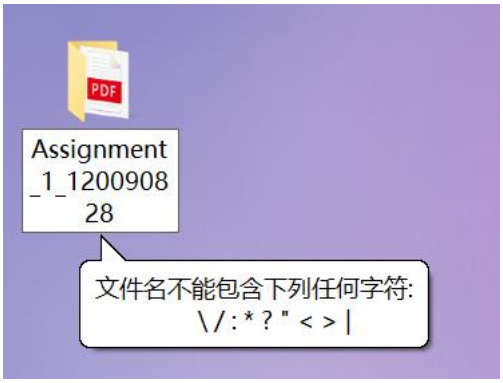


The version of clang format is new enough:



4.2 the name of my .zip file:

The naming of the .zip file cannot include “<” and “>”:




```
vagrant@csc3150:~/csc3150/assignment1/bonus$ zip -q -r Assignment_1_<120090828>.zip
bash: syntax error near unexpected token `120090828'
vagrant@csc3150:~/csc3150/assignment1/bonus$
```

Thus, my file name can only be Assignment_1_120090828.

5. Conclusion

In this program, I have clearer ideas of how to create processes from user mode and kernel mode, which is mainly the content from tutorial 2 and lectures. In the user mode (program 1), we cannot access the kernel directly. Thus, we use functions like execve( ), waitpid( ) and exit( ). These functions help us to execute system calls in the kernel and then return to user mode. In the kernel mode, things are different. We need to modify the codes in the kernel directly to export the function we want and modify the environment. Then, we fork the new process by calling the function directly instead of doing system calls. After building the kernel object, we need to insert and remove the module. The command “dmesg” would help us to read the message inside the ring buffer. The error I met while doing these programs help me to understand the process creating better.

In the bonus, I know more about how the OS manipulate the process. I used the command “pstree” and have a look at the files in /proc. I have a better understanding of “Everything is a file” and can build a process tree by myself.

In all, the things I learned while doing assignment 1 would help me better understand how the operating system creates and manages processes.