CSC3150 Assignment1 Report

@October 7, 2023 @Yuzhe YANG 121090684

Environment

Basic Information

OS: Debian GNU/Linux 11

Kernel Version: 5.10.191

Architecture: aarch64

Compile Kernel

In order to set up my Linux environment, I need to compile the Linux kernel, here is how did I do.

1. Kernel Version: As I have already download the UTM virtual machine with kernel version 5.10.191, it is already satisfied the assignment's requirements.
2. Compile:

```
sudo su
cd /home/seed/work/linux-source-5.10

# Clean previous setting and start configuration
make mrproper
make clean
make menuconfign # save config and exit

# Build kernel Image and modules
make

# Install kernel modules
make modules_install

# Install kernel
make install

# Reboot to load new kernel
reboot
```

Task 1

Description

In task1, we need to call some basic Linux function to implement the main flow:

1. Create a Child Process: Use `fork()` to create a new process, which becomes the child process.
2. Raise a Signal in the Child Process: Use `raise()` to generate a signal within the child process.
3. Execute a Test Program in the Child Process: Use `execve()` to replace the child process's image with the specified test program.
4. Wait for Child Process to Terminate in the Parent Process: Use `waitpid()` in the parent process to wait for the child process to terminate.
5. Handle Signals in the Parent Process: Upon receiving a signal from the child process, use signal handlers or other mechanisms to determine the type of signal received.
6. Print Information Based on the Received Signal: Depending on the type of signal received, print relevant information in the parent process.

Implementation

In order to test all the file in the program1 folder, I write a shell file.

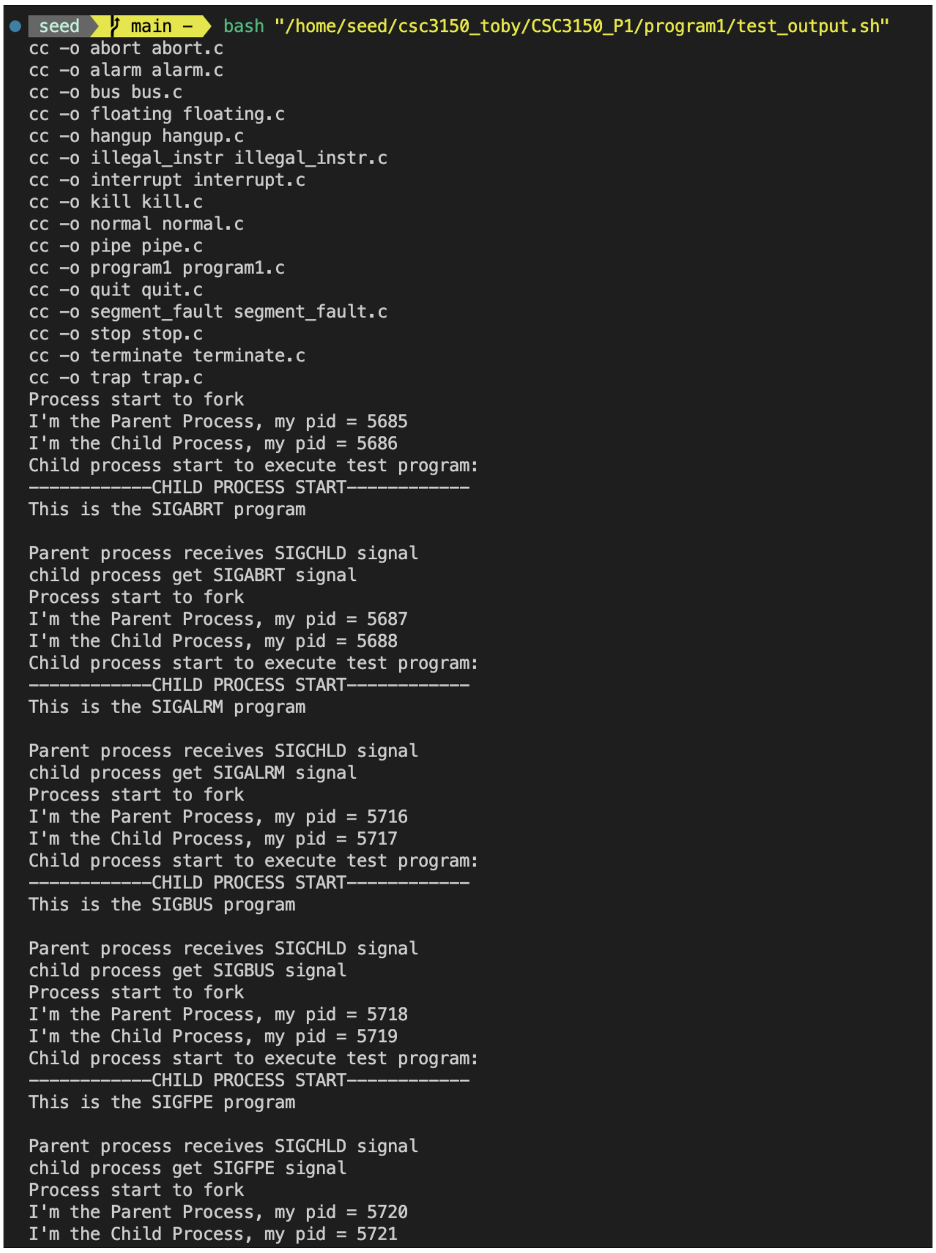
```
make all
cd /home/seed/csc3150_toby/CSC3150_P1/program1
./program1 ./abort
./program1 ./alarm
./program1 ./bus
./program1 ./floating
./program1 ./hangup
./program1 ./illegal_instr
./program1 ./interrupt
./program1 ./kill
./program1 ./normal
./program1 ./pipe
./program1 ./quit
./program1 ./segment_fault
./program1 ./stop
./program1 ./terminate
./program1 ./trap
```

How to run this:

1. Navigate into program1 folder
2. `./test_output.sh`

Output

This is the screenshot of a part of my output:



```
seed   main –  bash "/home/seed/csc3150_toby/CSC3150_P1/program1/test_output.sh"
  cc -o abort abort.c
  cc -o alarm alarm.c
  cc -o bus bus.c
  cc -o floating floating.c
  cc -o hangup hangup.c
  cc -o illegal_instr illegal_instr.c
  cc -o interrupt interrupt.c
  cc -o kill kill.c
  cc -o normal normal.c
  cc -o pipe pipe.c
  cc -o program1 program1.c
  cc -o quit quit.c
  cc -o segment_fault segment_fault.c
  cc -o stop stop.c
  cc -o terminate terminate.c
  cc -o trap trap.c
  Process start to fork
  I'm the Parent Process, my pid = 5685
  I'm the Child Process, my pid = 5686
  Child process start to execute test program:
  ------------CHILD PROCESS START------------
  This is the SIGABRT program

  Parent process receives SIGCHLD signal
  child process get SIGABRT signal
  Process start to fork
  I'm the Parent Process, my pid = 5687
  I'm the Child Process, my pid = 5688
  Child process start to execute test program:
  ------------CHILD PROCESS START------------
  This is the SIGALRM program

  Parent process receives SIGCHLD signal
  child process get SIGALRM signal
  Process start to fork
  I'm the Parent Process, my pid = 5716
  I'm the Child Process, my pid = 5717
  Child process start to execute test program:
  ------------CHILD PROCESS START------------
  This is the SIGBUS program

  Parent process receives SIGCHLD signal
  child process get SIGBUS signal
  Process start to fork
  I'm the Parent Process, my pid = 5718
  I'm the Child Process, my pid = 5719
  Child process start to execute test program:
  ------------CHILD PROCESS START------------
  This is the SIGFPE program

  Parent process receives SIGCHLD signal
  child process get SIGFPE signal
  Process start to fork
  I'm the Parent Process, my pid = 5720
  I'm the Child Process, my pid = 5721
```

And the full output:

```
seed   main -  bash "/home/seed/csc3150_toby/CSC3150_P1/program1/test_output.sh"
cc -o abort abort.c
cc -o alarm alarm.c
cc -o bus bus.c
cc -o floating floating.c
cc -o hangup hangup.c
cc -o illegal_instr illegal_instr.c
cc -o interrupt interrupt.c
cc -o kill kill.c
cc -o normal normal.c
cc -o pipe pipe.c
cc -o program1 program1.c
cc -o quit quit.c
cc -o segment_fault segment_fault.c
cc -o stop stop.c
cc -o terminate terminate.c
cc -o trap trap.c
Process start to fork
I'm the Parent Process, my pid = 5685
I'm the Child Process, my pid = 5686
Child process start to execute test program:
-----------CHILD PROCESS START-----------
This is the SIGABRT program
```

```
Parent process receives SIGCHLD signal
child process get SIGABRT signal
Process start to fork
I'm the Parent Process, my pid = 5687
I'm the Child Process, my pid = 5688
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGALRM program

Parent process receives SIGCHLD signal
child process get SIGALRM signal
Process start to fork
I'm the Parent Process, my pid = 5716
I'm the Child Process, my pid = 5717
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGBUS program

Parent process receives SIGCHLD signal
child process get SIGBUS signal
Process start to fork
I'm the Parent Process, my pid = 5718
I'm the Child Process, my pid = 5719
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGFPE program

Parent process receives SIGCHLD signal
child process get SIGFPE signal
Process start to fork
I'm the Parent Process, my pid = 5720
I'm the Child Process, my pid = 5721
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGHUP program

Parent process receives SIGCHLD signal
child process get SIGHUP signal
Process start to fork
I'm the Parent Process, my pid = 5722
I'm the Child Process, my pid = 5723
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGILL program

Parent process receives SIGCHLD signal
child process get SIGILL signal
Process start to fork
I'm the Parent Process, my pid = 5724
I'm the Child Process, my pid = 5725
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGINT program

Parent process receives SIGCHLD signal
child process get SIGINT signal
Process start to fork
I'm the Parent Process, my pid = 5726
I'm the Child Process, my pid = 5727
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGKILL program

Parent process receives SIGCHLD signal
child process get SIGKILL signal
Process start to fork
I'm the Parent Process, my pid = 5728
I'm the Child Process, my pid = 5729
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

Process start to fork
I'm the Parent Process, my pid = 5730
I'm the Child Process, my pid = 5731
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGPIPE program
```

```
Parent process receives SIGCHLD signal
child process get SIGPIPE signal
Process start to fork
I'm the Parent Process, my pid = 5732
I'm the Child Process, my pid = 5733
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGQUIT program

Parent process receives SIGCHLD signal
child process get SIGQUIT signal
Process start to fork
I'm the Parent Process, my pid = 5734
I'm the Child Process, my pid = 5735
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGSEGV program

Parent process receives SIGCHLD signal
child process get SIGSEGV signal
Process start to fork
I'm the Parent Process, my pid = 5736
I'm the Child Process, my pid = 5737
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGSTOP program

Parent process receives SIGCHLD signal
child process get SIGSTOP signal
Process start to fork
I'm the Parent Process, my pid = 5738
I'm the Child Process, my pid = 5739
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGTERM program

Parent process receives SIGCHLD signal
child process get SIGTERM signal
Process start to fork
I'm the Parent Process, my pid = 5740
I'm the Child Process, my pid = 5741
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGTRAP program

Parent process receives SIGCHLD signal
child process get SIGTRAP signal

 [~/csc3150_toby/CSC3150_P1/program1]
 seed ▯ ▯ main - ▯
```

Task 2

Description

The procedure of task 2 is similar like task 1, but all the work flow is under the kernel mode, which is different from task 1.

Task 2 involves creating a kernel module that operates in kernel mode and performs basic process management tasks such as forking, executing a test program, and waiting for the child process to terminate. The implementation includes creating a kernel thread that runs the `my_fork()` function, which is responsible for forking a child process `my_exec()` to execute a test program.

For parent process, I use `my_wait()` function to wait the end signal from child process. This function is implemented by `do_wait()` function, and the child process's status is stored in `*wo.wo_stat`.

At last, the program will output the information to kernel log according to the what signal the parent process gets. This procedure is as same as task 1.

Implementation

1. Modifying Kernel Files

   I have used four extern functions to implement task 2.

```
extern pid_t kernel_clone(struct kernel_clone_args *args);
extern struct filename *getname_kernel(const char *filename);
extern long do_wait(struct wait_opts *wo);
extern int do_execve(struct filename *filename, const char __user *const __user *__argv, const char __user *const __user *__envp);
```

Use `kernel_clone()` to fork a new process, and the corresponding kernel file path is `/kernel/fork.c`

Use `do_execve()` to execute the test program, and the corresponding kernel file path is `/fs/exec.c`

Use `getname_kernel()` to get filename, and the corresponding kernel file path is `/fs/namei.c`

Use `do_wait()` to wait for child process' termination status, and the correspinding kernel file pathe is `/kernel/exit.c`

I need to find all the function and its location in the corresponding kernel file, then I use `sudo vim <filepath>` to modify the original kernel file: add `EXPORT_SYMBOL()` at the end of the function. It allows me to call these funcitons properly in my program

After modify all the kernel file, we need to re-compile the kernel and reboot.

2. Normal signal and SIGSTOP signal

   In Task 1, I use `WIFEXITED()` and `WIFSTOPPED()` to judge the normal and stop signal. However these two function in located in the `signal.h` ,which is not supported in the kernel mode. In order to detect the signal properly, I wrote two function to implement `WIFEXITED()` and `WIFSTOPPED()` .

```
int my_WIFEXITED(int status)
{
  return (status & 0xff) == 0;
}

int my_WIFSTOPPED(int status)
{
  return ((status) & 0xff) == 0x7f;
}
```

3. Test program

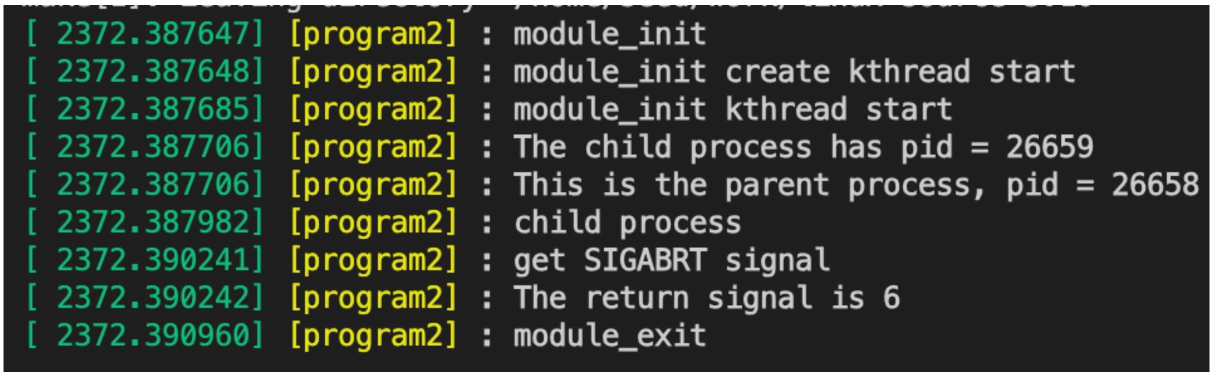
   I write a shell file to test the output:

```
# cd /home/seed/csc3150_toby/CSC3150_P1/program2
gcc test.c -o test
make clean
make
sudo insmod program2.ko
sudo rmmod program2.ko
sudo dmesg -c
```

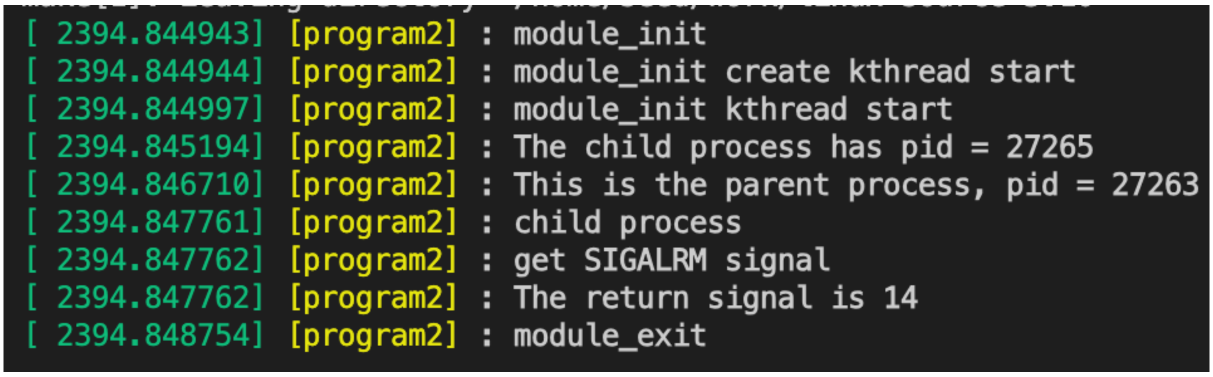
   How to run this:

   1. Navigate into program 2 folder
   2. run `./test_output.sh`

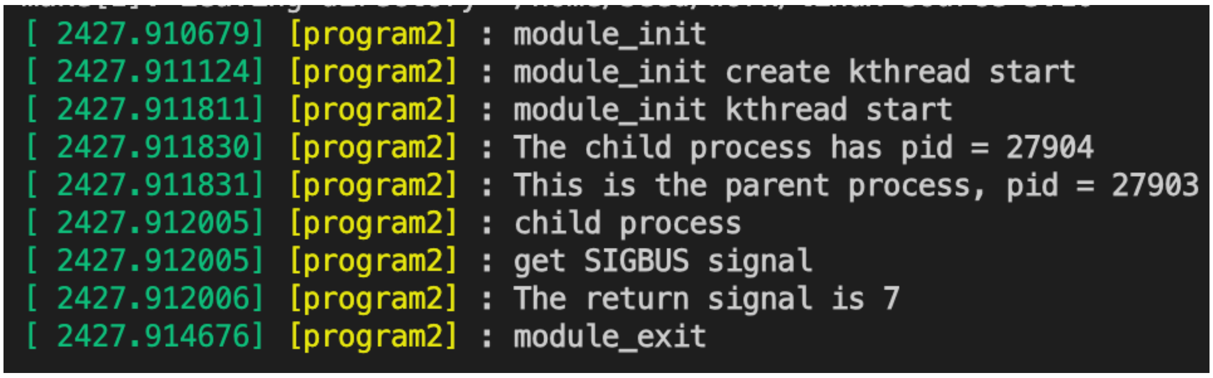
Output



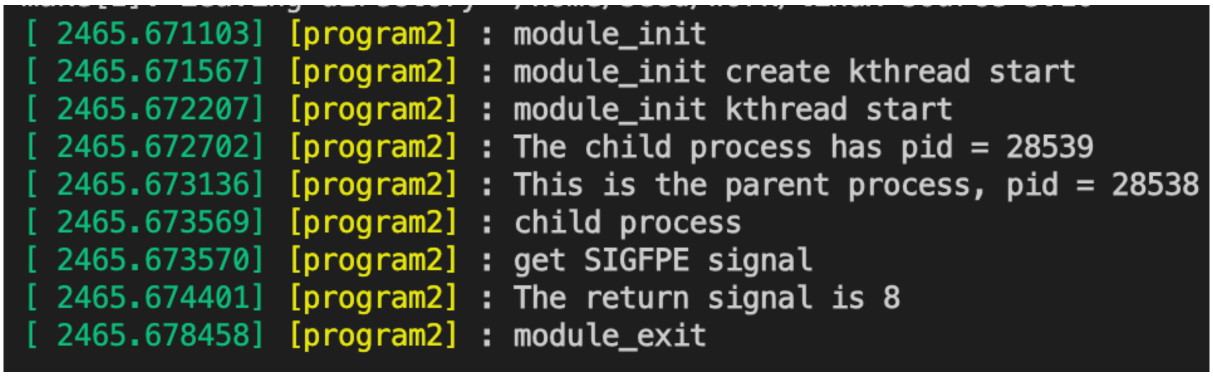
```
[ 2372.387647] [program2] : module_init
[ 2372.387648] [program2] : module_init create kthread start
[ 2372.387685] [program2] : module_init kthread start
[ 2372.387706] [program2] : The child process has pid = 26659
[ 2372.387706] [program2] : This is the parent process, pid = 26658
[ 2372.387982] [program2] : child process
[ 2372.390241] [program2] : get SIGABRT signal
[ 2372.390242] [program2] : The return signal is 6
[ 2372.390960] [program2] : module_exit
```



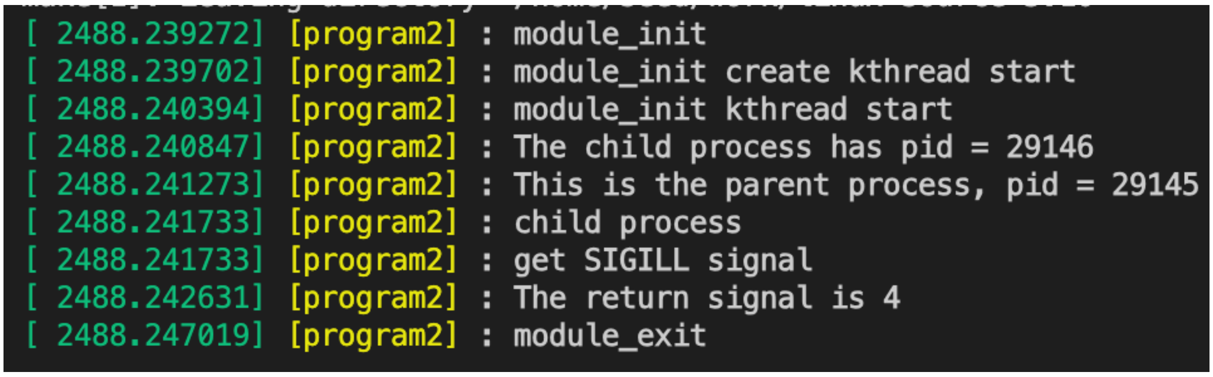
```
[ 2394.844943] [program2] : module_init
[ 2394.844944] [program2] : module_init create kthread start
[ 2394.844997] [program2] : module_init kthread start
[ 2394.845194] [program2] : The child process has pid = 27265
[ 2394.846710] [program2] : This is the parent process, pid = 27263
[ 2394.847761] [program2] : child process
[ 2394.847762] [program2] : get SIGALRM signal
[ 2394.847762] [program2] : The return signal is 14
[ 2394.848754] [program2] : module_exit
```



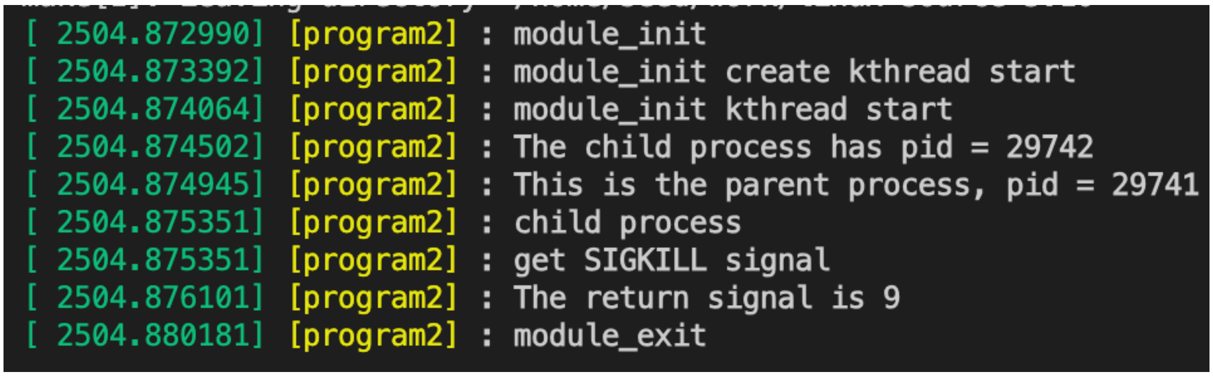
```
[ 2427.910679] [program2] : module_init
[ 2427.911124] [program2] : module_init create kthread start
[ 2427.911811] [program2] : module_init kthread start
[ 2427.911830] [program2] : The child process has pid = 27904
[ 2427.911831] [program2] : This is the parent process, pid = 27903
[ 2427.912005] [program2] : child process
[ 2427.912005] [program2] : get SIGBUS signal
[ 2427.912006] [program2] : The return signal is 7
[ 2427.914676] [program2] : module_exit
```



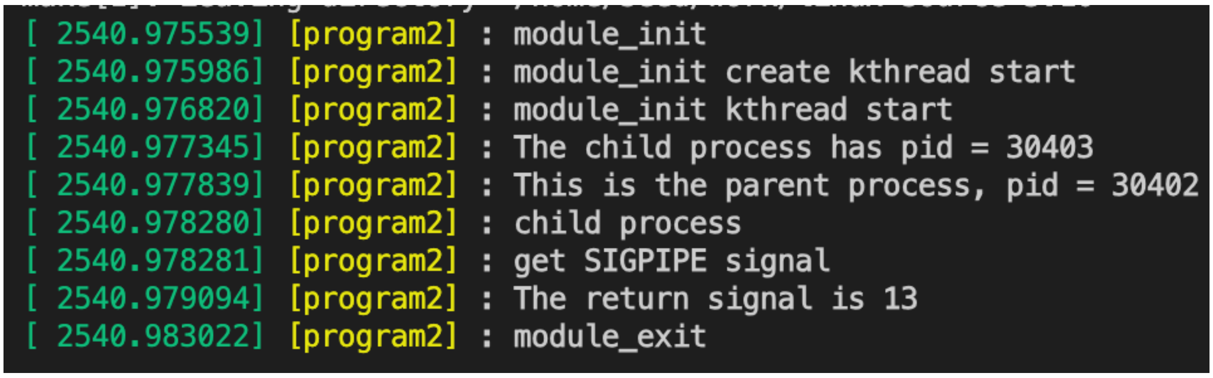
```
[ 2465.671103] [program2] : module_init
[ 2465.671567] [program2] : module_init create kthread start
[ 2465.672207] [program2] : module_init kthread start
[ 2465.672702] [program2] : The child process has pid = 28539
[ 2465.673136] [program2] : This is the parent process, pid = 28538
[ 2465.673569] [program2] : child process
[ 2465.673570] [program2] : get SIGFPE signal
[ 2465.674401] [program2] : The return signal is 8
[ 2465.678458] [program2] : module_exit
```



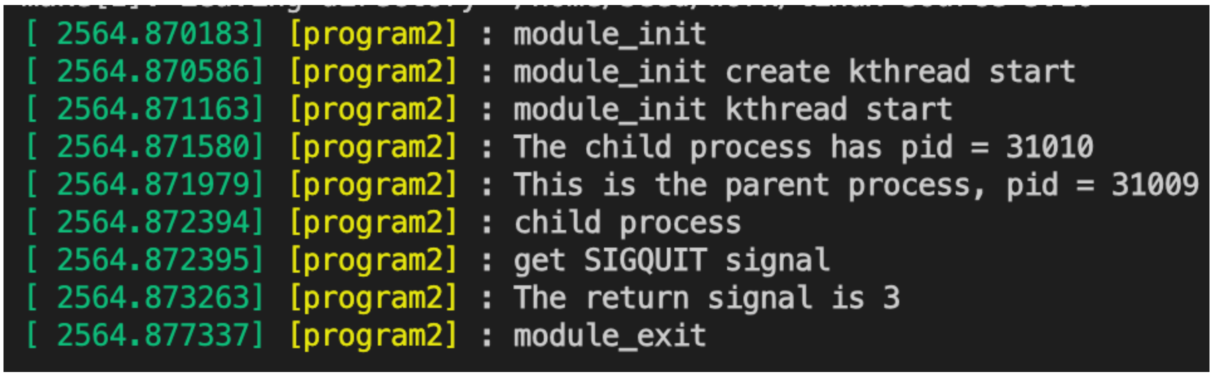
```
[ 2488.239272] [program2] : module_init
[ 2488.239702] [program2] : module_init create kthread start
[ 2488.240394] [program2] : module_init kthread start
[ 2488.240847] [program2] : The child process has pid = 29146
[ 2488.241273] [program2] : This is the parent process, pid = 29145
[ 2488.241733] [program2] : child process
[ 2488.241733] [program2] : get SIGILL signal
[ 2488.242631] [program2] : The return signal is 4
[ 2488.247019] [program2] : module_exit
```



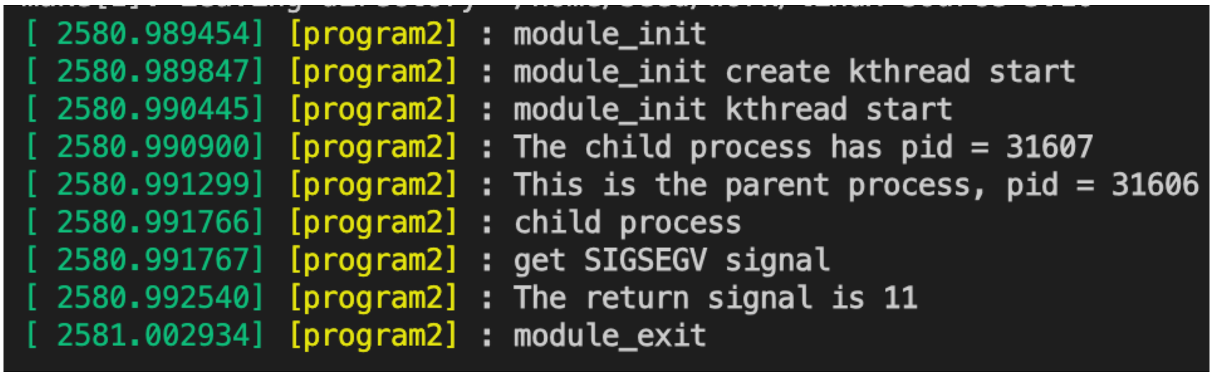
```
[ 2504.872990] [program2] : module_init
[ 2504.873392] [program2] : module_init create kthread start
[ 2504.874064] [program2] : module_init kthread start
[ 2504.874502] [program2] : The child process has pid = 29742
[ 2504.874945] [program2] : This is the parent process, pid = 29741
[ 2504.875351] [program2] : child process
[ 2504.875351] [program2] : get SIGKILL signal
[ 2504.876101] [program2] : The return signal is 9
[ 2504.880181] [program2] : module_exit
```

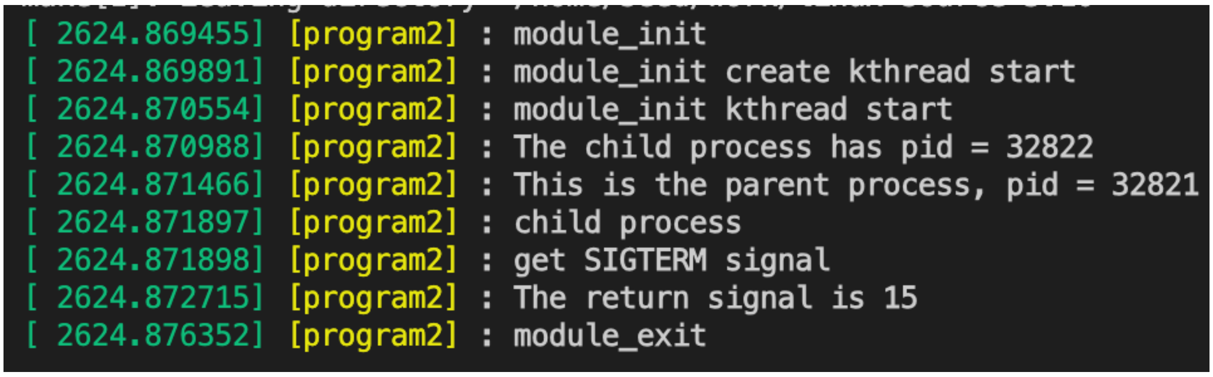


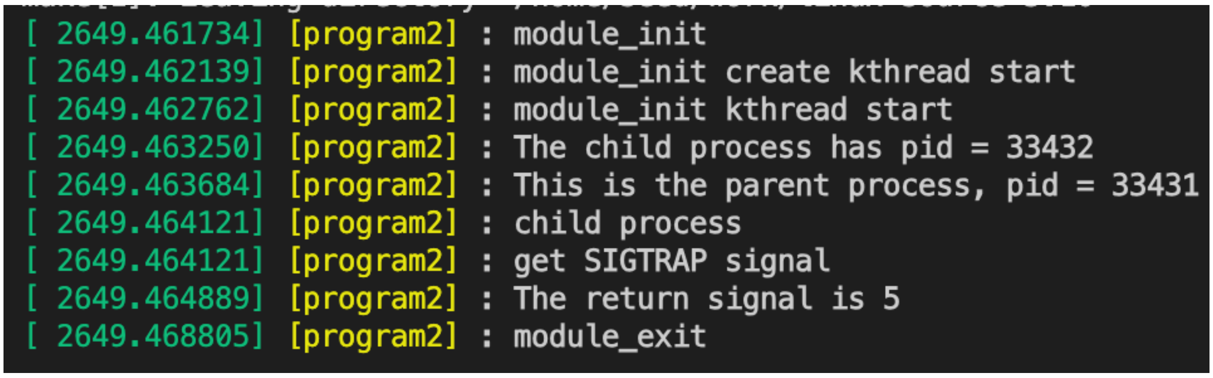
```
[ 2540.975539] [program2] : module_init
[ 2540.975986] [program2] : module_init create kthread start
[ 2540.976820] [program2] : module_init kthread start
[ 2540.977345] [program2] : The child process has pid = 30403
[ 2540.977839] [program2] : This is the parent process, pid = 30402
[ 2540.978280] [program2] : child process
[ 2540.978281] [program2] : get SIGPIPE signal
[ 2540.979094] [program2] : The return signal is 13
[ 2540.983022] [program2] : module_exit
```



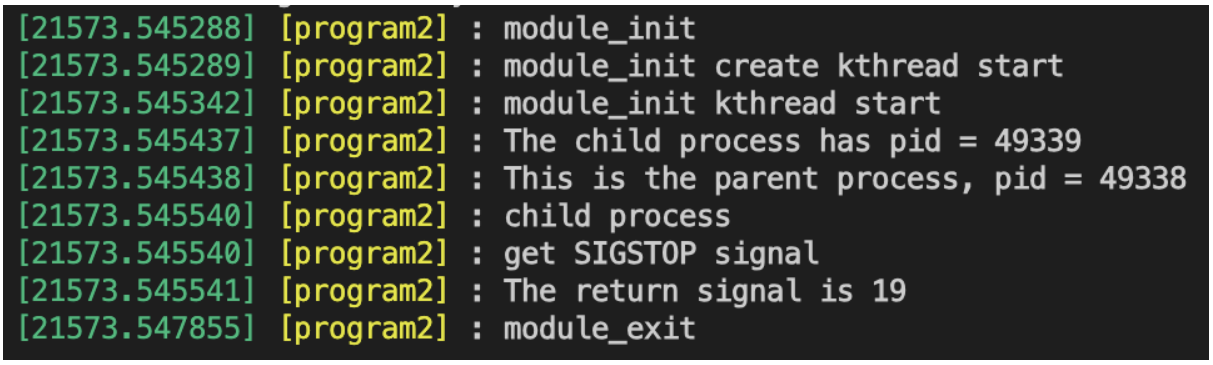
```
[ 2564.870183] [program2] : module_init
[ 2564.870586] [program2] : module_init create kthread start
[ 2564.871163] [program2] : module_init kthread start
[ 2564.871580] [program2] : The child process has pid = 31010
[ 2564.871979] [program2] : This is the parent process, pid = 31009
[ 2564.872394] [program2] : child process
[ 2564.872395] [program2] : get SIGQUIT signal
[ 2564.873263] [program2] : The return signal is 3
[ 2564.877337] [program2] : module_exit
```


```
[ 2580.989454] [program2] : module_init
[ 2580.989847] [program2] : module_init create kthread start
[ 2580.990445] [program2] : module_init kthread start
[ 2580.990900] [program2] : The child process has pid = 31607
[ 2580.991299] [program2] : This is the parent process, pid = 31606
[ 2580.991766] [program2] : child process
[ 2580.991767] [program2] : get SIGSEGV signal
[ 2580.992540] [program2] : The return signal is 11
[ 2581.002934] [program2] : module_exit
```

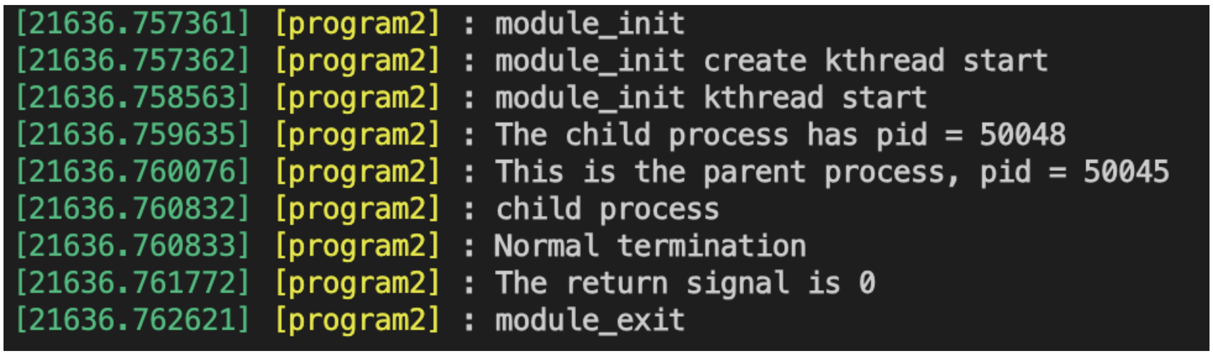

```
[ 2624.869455] [program2] : module_init
[ 2624.869891] [program2] : module_init create kthread start
[ 2624.870554] [program2] : module_init kthread start
[ 2624.870988] [program2] : The child process has pid = 32822
[ 2624.871466] [program2] : This is the parent process, pid = 32821
[ 2624.871897] [program2] : child process
[ 2624.871898] [program2] : get SIGTERM signal
[ 2624.872715] [program2] : The return signal is 15
[ 2624.876352] [program2] : module_exit
```


```
[ 2649.461734] [program2] : module_init
[ 2649.462139] [program2] : module_init create kthread start
[ 2649.462762] [program2] : module_init kthread start
[ 2649.463250] [program2] : The child process has pid = 33432
[ 2649.463684] [program2] : This is the parent process, pid = 33431
[ 2649.464121] [program2] : child process
[ 2649.464121] [program2] : get SIGTRAP signal
[ 2649.464889] [program2] : The return signal is 5
[ 2649.468805] [program2] : module_exit
```

SIGSTOP signal


```
[21573.545288] [program2] : module_init
[21573.545289] [program2] : module_init create kthread start
[21573.545342] [program2] : module_init kthread start
[21573.545437] [program2] : The child process has pid = 49339
[21573.545438] [program2] : This is the parent process, pid = 49338
[21573.545540] [program2] : child process
[21573.545540] [program2] : get SIGSTOP signal
[21573.545541] [program2] : The return signal is 19
[21573.547855] [program2] : module_exit
```

Normal termination



```
[21636.757361] [program2] : module_init
[21636.757362] [program2] : module_init create kthread start
[21636.758563] [program2] : module_init kthread start
[21636.759635] [program2] : The child process has pid = 50048
[21636.760076] [program2] : This is the parent process, pid = 50045
[21636.760832] [program2] : child process
[21636.760833] [program2] : Normal termination
[21636.761772] [program2] : The return signal is 0
[21636.762621] [program2] : module_exit
```

Bonus

Description

The process are stored in `/proc` directory. In order to output all the process information in the tree structure, we need to traverse the information from this directory.

In my `pstree.c` program, I was concentrated on these three pstree:

```
pstree -c
pstree -p
pstree -T
```

If we input `man pstree`, we will find more detailed description about different pstree types. And here is some detail about these three pstree:

```
OPTIONS

       -c     Disable compaction of identical subtrees.  By default, subtrees are compacted whenever possible.

       -p     Show PIDs.  PIDs are shown as decimal numbers in parentheses after each process name.  -p implicitly disables compaction

       -T     Hide threads and only show processes.
```

Implementation

Firstly I create a node structure to store the information of each thread or process. It includes pid, name of the process or thread and a bool type variable to record if the process have a child process.

```
typedef struct
{
    pid_t pid;
    char name[MAX_NAME_LEN];
    bool have_child;
} Node;
```

Then I design a DFS recursive algorithm:

```
/**
 * Recursively prints the process tree in depth-first order.
 * @param pid The process ID to start the tree from.
 * @param depth The current depth of the tree.
 * @param is_last_child Whether this process is the last child of its parent.
 * @param sibling_map Array of flags indicating whether the previous siblings at each depth were last children.
 */
void printProcessTreeDFS(pid_t pid, int depth, int is_last_child, int *sibling_map)
```

By recursively calling this function, we can get ever process stored in `"/proc/%ld/status"` . And for each current father-process, we traverse all the child-process by similar way.

After we got all the child-process informations, I want to print by the alphabet ordering. Hence I use `qsort()` function to sort child-process's name by alphabet ordering.

```
qsort(children, child_count, sizeof(Node), compareChildProcess);
```

As the thread information may also needed, I write about another function to traverse through the thread information under each process.

```
/**
 * Reads all threads for a given process ID and stores them in the threads array.
 * @param pid Tqhe process ID for which to read threads.
 * @param threads Array to store thread information.
 * @param thread_count Pointer to an integer to store the number of threads.
 */
void readThreadsInProc(pid_t pid, Node *threads, int *thread_count)
```

Finally, the main function parses command-line arguments. If no arguments are provided, or if an invalid number of arguments is given, it prints an error message.

- If the argument is `c` , it prints the process tree with only process names.
- If the argument is `p` , it prints the process tree with both process names and IDs.
- If the argument is `T` , it prints the process tree with only process names and no IDs.

And I write a Makefile file to compile this program.

```
pstree: pstree.c

clean:
  $(RM) pstree
```

How to run `pstree.c` :

1. Navigate into `/CSC3150_P1/bonus` folder
2. Compile the program first
3. Print pstree you wanted:

   ```
   ./pstree -c
   ./pstree -p
   ./pstree -T
   ```

Output

1. `./pstree -c`

   ```
   systemd
   ├─agetty
   ├─cpptools-srv
   │   ├─{cpptools-srv}
   │   ├─{cpptools-srv}
   │   ├─{cpptools-srv}
   │   ├─{cpptools-srv}
   │   ├─{cpptools-srv}
   │   ├─{cpptools-srv}
   │   ├─{cpptools-srv}
   │   ├─{cpptools-srv}
   │   ├─{cpptools-srv}
   ```

```
│   ├─{cpptools-srv}
│   └─{cpptools-srv}
├─cpptools-srv
│   ├─{cpptools-srv}
│   ├─{cpptools-srv}
│   ├─{cpptools-srv}
│   ├─{cpptools-srv}
│   ├─{cpptools-srv}
│   ├─{cpptools-srv}
│   ├─{cpptools-srv}
│   ├─{cpptools-srv}
│   ├─{cpptools-srv}
│   ├─{cpptools-srv}
│   └─{cpptools-srv}
├─cpptools-srv
│   ├─{cpptools-srv}
│   ├─{cpptools-srv}
│   ├─{cpptools-srv}
│   ├─{cpptools-srv}
│   ├─{cpptools-srv}
│   ├─{cpptools-srv}
│   ├─{cpptools-srv}
│   ├─{cpptools-srv}
│   ├─{cpptools-srv}
│   ├─{cpptools-srv}
│   └─{cpptools-srv}
├─dbus-daemon
├─dhclient
│   ├─{isc-worker0000}
│   ├─{isc-socket}
│   └─{isc-timer}
├─imhere
│   └─sleep
├─sshd
│   └─sshd
│       └─sshd
│           └─bash
│               ├─code-insiders-8
│               │   ├─{tokio-runtime-w}
│               │   ├─{tokio-runtime-w}
│               │   ├─{tokio-runtime-w}
│               │   └─{tokio-runtime-w}
│               │   └─sh
│               │       └─node
│               │           ├─{node}
│               │           ├─{node}
│               │           ├─{node}
│               │           ├─{node}
│               │           ├─{node}
│               │           ├─{node}
│               │           ├─{node}
│               │           ├─{node}
│               │           ├─{node}
│               │           └─{node}
│               │           ├─node
│               │           │   ├─{node}
│               │           │   ├─{node}
│               │           │   ├─{node}
│               │           │   ├─{node}
│               │           │   ├─{node}
│               │           │   ├─{node}
│               │           │   ├─{node}
│               │           │   ├─{node}
│               │           │   ├─{node}
│               │           │   ├─{node}
│               │           │   ├─{node}
│               │           │   ├─{node}
│               │           │   ├─{node}
│               │           │   └─{node}
│               │           │   ├─cpptools
│               │           │   │   ├─{cpptools}
│               │           │   │   ├─{cpptools}
│               │           │   │   ├─{cpptools}
│               │           │   │   ├─{cpptools}
│               │           │   │   ├─{cpptools}
│               │           │   │   ├─{cpptools}
│               │           │   │   ├─{cpptools}
│               │           │   │   ├─{cpptools}
│               │           │   │   ├─{cpptools}
│               │           │   │   ├─{cpptools}
│               │           │   │   ├─{cpptools}
│               │           │   │   ├─{cpptools}
│               │           │   │   └─{cpptools}
│               │           │   ├─node
│               │           │   │   ├─{node}
│               │           │   │   ├─{node}
│               │           │   │   ├─{node}
│               │           │   │   ├─{node}
│               │           │   │   ├─{node}
│               │           │   │   └─{node}
│               │           │   └─node
│               │           │       ├─{node}
│               │           │       ├─{node}
│               │           │       ├─{node}
│               │           │       ├─{node}
│               │           │       ├─{node}
│               │           │       ├─{node}
│               │           │       ├─{node}
│               │           │       ├─{node}
│               │           │       ├─{node}
│               │           │       └─{node}
│               │           ├─node
│               │           │   ├─{node}
│               │           │   ├─{node}
│               │           │   ├─{node}
│               │           │   ├─{node}
│               │           │   ├─{node}
│               │           │   ├─{node}
│               │           │   ├─{node}
│               │           │   ├─{node}
│               │           │   ├─{node}
│               │           │   ├─{node}
│               │           │   └─{node}
│               │           └─node
│               │               ├─{node}
│               │               ├─{node}
│               │               ├─{node}
│               │               ├─{node}
│               │               ├─{node}
│               │               ├─{node}
│               │               ├─{node}
│               │               ├─{node}
│               │               ├─{node}
│               │               ├─{node}
│               │               ├─{node}
│               │               └─{node}
│               │               ├─sh
│               │               │   └─cpuUsage.sh
│               │               │       └─sleep
│               │               ├─zsh
│               │               └─zsh
│               │                   └─pstree
│               └─sleep
├─systemd
│   └─(sd-pam)
├─systemd-journal
├─systemd-logind
├─systemd-timesyn
│   └─{sd-resolve}
└─systemd-udevd
```

2. `./pstree -p`

```
systemd(1)
├─agetty(556)
├─cpptools-srv(1066)
│   ├─{cpptools-srv}(1067)
│   ├─{cpptools-srv}(1068)
│   ├─{cpptools-srv}(1069)
│   ├─{cpptools-srv}(1070)
│   ├─{cpptools-srv}(1071)
│   ├─{cpptools-srv}(1072)
│   ├─{cpptools-srv}(1073)
│   ├─{cpptools-srv}(1074)
│   ├─{cpptools-srv}(1075)
│   ├─{cpptools-srv}(1076)
│   └─{cpptools-srv}(1084)
├─cpptools-srv(5397)
│   ├─{cpptools-srv}(5398)
│   ├─{cpptools-srv}(5399)
│   ├─{cpptools-srv}(5400)
│   ├─{cpptools-srv}(5401)
│   ├─{cpptools-srv}(5402)
│   ├─{cpptools-srv}(5403)
│   ├─{cpptools-srv}(5404)
│   ├─{cpptools-srv}(5405)
│   ├─{cpptools-srv}(5406)
│   ├─{cpptools-srv}(5407)
│   └─{cpptools-srv}(14654)
├─cpptools-srv(30908)
│   ├─{cpptools-srv}(30909)
│   ├─{cpptools-srv}(30910)
│   ├─{cpptools-srv}(30911)
│   ├─{cpptools-srv}(30912)
│   ├─{cpptools-srv}(30913)
│   ├─{cpptools-srv}(30914)
│   ├─{cpptools-srv}(30915)
│   ├─{cpptools-srv}(30916)
│   ├─{cpptools-srv}(30917)
│   ├─{cpptools-srv}(30918)
│   └─{cpptools-srv}(36014)
├─dbus-daemon(531)
├─dhclient(506)
│   ├─{isc-worker0000}(511)
│   ├─{isc-socket}(512)
│   └─{isc-timer}(513)
├─imhere(534)
│   └─sleep(44951)
├─sshd(557)
│   └─sshd(581)
│       └─sshd(594)
│           └─bash(595)
│               ├─code-insiders-8(614)
│               │   ├─{tokio-runtime-w}(621)
│               │   ├─{tokio-runtime-w}(622)
│               │   ├─{tokio-runtime-w}(623)
│               │   └─{tokio-runtime-w}(624)
│               │   └─sh(638)
│               │       └─node(642)
│               │           ├─{node}(643)
│               │           ├─{node}(644)
│               │           ├─{node}(645)
│               │           ├─{node}(646)
│               │           ├─{node}(647)
│               │           ├─{node}(648)
│               │           ├─{node}(649)
│               │           ├─{node}(650)
│               │           ├─{node}(651)
│               │           └─{node}(652)
│               │           ├─node(682)
│               │           │   ├─{node}(683)
│               │           │   ├─{node}(684)
│               │           │   ├─{node}(685)
│               │           │   ├─{node}(686)
│               │           │   ├─{node}(687)
│               │           │   ├─{node}(688)
│               │           │   ├─{node}(689)
│               │           │   ├─{node}(690)
│               │           │   ├─{node}(691)
│               │           │   ├─{node}(692)
│               │           │   ├─{node}(706)
│               │           │   ├─{node}(1013)
│               │           │   ├─{node}(1014)
│               │           │   └─{node}(1015)
│               │           │   ├─cpptools(905)
│               │           │   │   ├─{cpptools}(906)
│               │           │   │   ├─{cpptools}(907)
│               │           │   │   ├─{cpptools}(908)
│               │           │   │   ├─{cpptools}(909)
│               │           │   │   ├─{cpptools}(910)
│               │           │   │   ├─{cpptools}(911)
│               │           │   │   ├─{cpptools}(912)
│               │           │   │   ├─{cpptools}(913)
│               │           │   │   ├─{cpptools}(1080)
│               │           │   │   ├─{cpptools}(1081)
│               │           │   │   ├─{cpptools}(1082)
│               │           │   │   ├─{cpptools}(1083)
│               │           │   │   └─{cpptools}(4960)
│               │           │   ├─node(811)
│               │           │   │   ├─{node}(812)
│               │           │   │   ├─{node}(813)
│               │           │   │   ├─{node}(814)
│               │           │   │   ├─{node}(815)
│               │           │   │   ├─{node}(816)
│               │           │   │   └─{node}(817)
│               │           │   └─node(980)
│               │           │       ├─{node}(982)
│               │           │       ├─{node}(983)
│               │           │       ├─{node}(984)
│               │           │       ├─{node}(985)
│               │           │       ├─{node}(986)
│               │           │       ├─{node}(987)
│               │           │       ├─{node}(994)
│               │           │       ├─{node}(995)
│               │           │       ├─{node}(996)
│               │           │       └─{node}(997)
│               │           ├─node(693)
│               │           │   ├─{node}(694)
│               │           │   ├─{node}(695)
│               │           │   ├─{node}(696)
│               │           │   ├─{node}(697)
│               │           │   ├─{node}(698)
│               │           │   ├─{node}(699)
│               │           │   ├─{node}(700)
│               │           │   ├─{node}(701)
│               │           │   ├─{node}(702)
│               │           │   ├─{node}(703)
│               │           │   └─{node}(704)
│               │           └─node(710)
│               │               ├─{node}(711)
│               │               ├─{node}(712)
│               │               ├─{node}(713)
│               │               ├─{node}(714)
│               │               ├─{node}(715)
│               │               ├─{node}(716)
│               │               ├─{node}(717)
│               │               ├─{node}(718)
│               │               ├─{node}(719)
│               │               ├─{node}(720)
│               │               ├─{node}(1167)
│               │               └─{node}(5200)
│               │               ├─zsh(1166)
│               │               └─zsh(5199)
│               │                   └─pstree(44954)
│               └─sleep(44914)
├─systemd(584)
│   └─(sd-pam)(585)
├─systemd-journal(275)
├─systemd-logind(536)
├─systemd-timesyn(520)
│   └─{sd-resolve}(522)
└─systemd-udevd(291)
```

3. `./pstree -T`

```
systemd
├─agetty
├─cpptools-srv
├─cpptools-srv
├─cpptools-srv
├─dbus-daemon
├─dhclient
├─imhere
│   └─sleep
├─sshd
│   └─sshd
│       └─sshd
│           └─bash
│               ├─code-insiders-8
│               │   └─sh
│               │       └─node
│               │           ├─node
│               │           │   ├─cpptools
│               │           │   ├─node
│               │           │   └─node
│               │           ├─node
│               │           └─node
│               │               ├─zsh
│               │               └─zsh
│               │                   └─pstree
│               └─sleep
├─systemd
│   └─(sd-pam)
├─systemd-journal
├─systemd-logind
├─systemd-timesyn
└─systemd-udevd
```

What do I learn?

Task 1:

Task 1 allowed me to navigate the intricacies of kernel compilation and the execution of various Linux system calls. This included creating child processes, generating signals, executing test programs, and handling relevant information in the parent process. It was a hands-on experience that solidified my understanding and proficiency in Linux system calls and process management.

Task 2:

Moving into Task 2, I transitioned into kernel mode. Here, I crafted a kernel module implementing fundamental process management functionalities such as fork, execve, and waitpid. This task showcased my grasp of the Linux kernel and my ability to modify and extend its capabilities.

Bonus Task:

The Bonus Task was a deeper exploration into process tree representation. By implementing functionality akin to the `pstree` command, I demonstrated a profound understanding of process hierarchy within the Linux system. Traversing and sorting processes alphabetically added an extra layer of complexity, revealing a thorough understanding of the system's internals.

Overall Learning:

1. **Kernel Compilation and Modification:** I learned how to compile and modify the Linux kernel, providing insights into the underlying workings of the system.
2. **System Calls:** Through Tasks 1 and 2, I gained a deeper understanding of Linux system calls, particularly those related to process management.
3. **Kernel Module Writing:** Task 2 introduced me to the creation of kernel modules, offering an opportunity to engage in lower-level system-level programming.
4. **Process Hierarchy Understanding:** The Bonus Task delved into the hierarchical structure of processes, showcasing a deeper understanding of how processes are organized within the system.