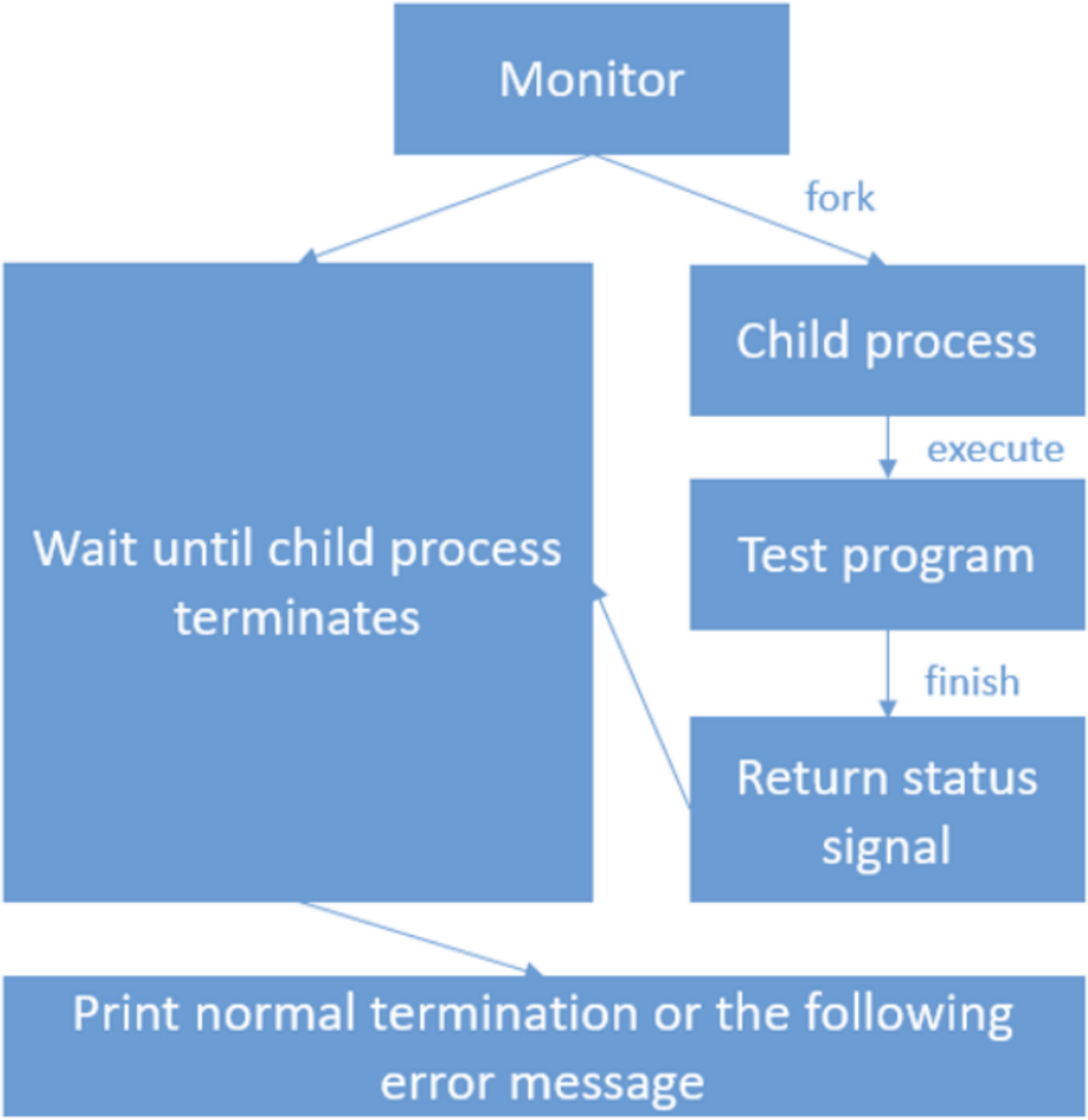
Report for Assignment 1

Name: XIAO Weizhao ID:120090588

1. Design of my program

1.1 Task 1

The flow chart of task1:



- Use **fork()** to create the child process.

```
pid_t pid = fork();
```

In order to make sure that "I'm the Parent Process" is printed out before "I'm the Child Process" as the demo output demanded, I use **sleep(1)** to let child process sleep for a second.

```
//child process:
else if (pid == 0) {
    sleep(1);
    int index;
    char *arg[argc];
    for (index = 0; index < argc - 1; index++) {
        arg[index] = argv[index + 1];
    }
    arg[argc - 1] = NULL;
    printf("I'm the Child Process, my pid = %d\n", getpid());
    printf("Child process start to execute test program:\n");
    execve(arg[0], arg, NULL);
    perror("execve");
    exit(EXIT_FAILURE);
}
```

The **pid of the child process** will be printed. Use **arg** to store the filename of the test files. Use **execve()** to run test program in the child process. If the execution fails, the program will exit with signal "EXIT_FAILURE". Otherwise, the codes after execve() will not be used.

- Use **waitpid(pid, &status, WUNTRACED)** to receive the SIGCHLD signal. Wait() is not feasible in this case. It needs "WUNTRACED" to receive SIGSTOP without stopping the program.

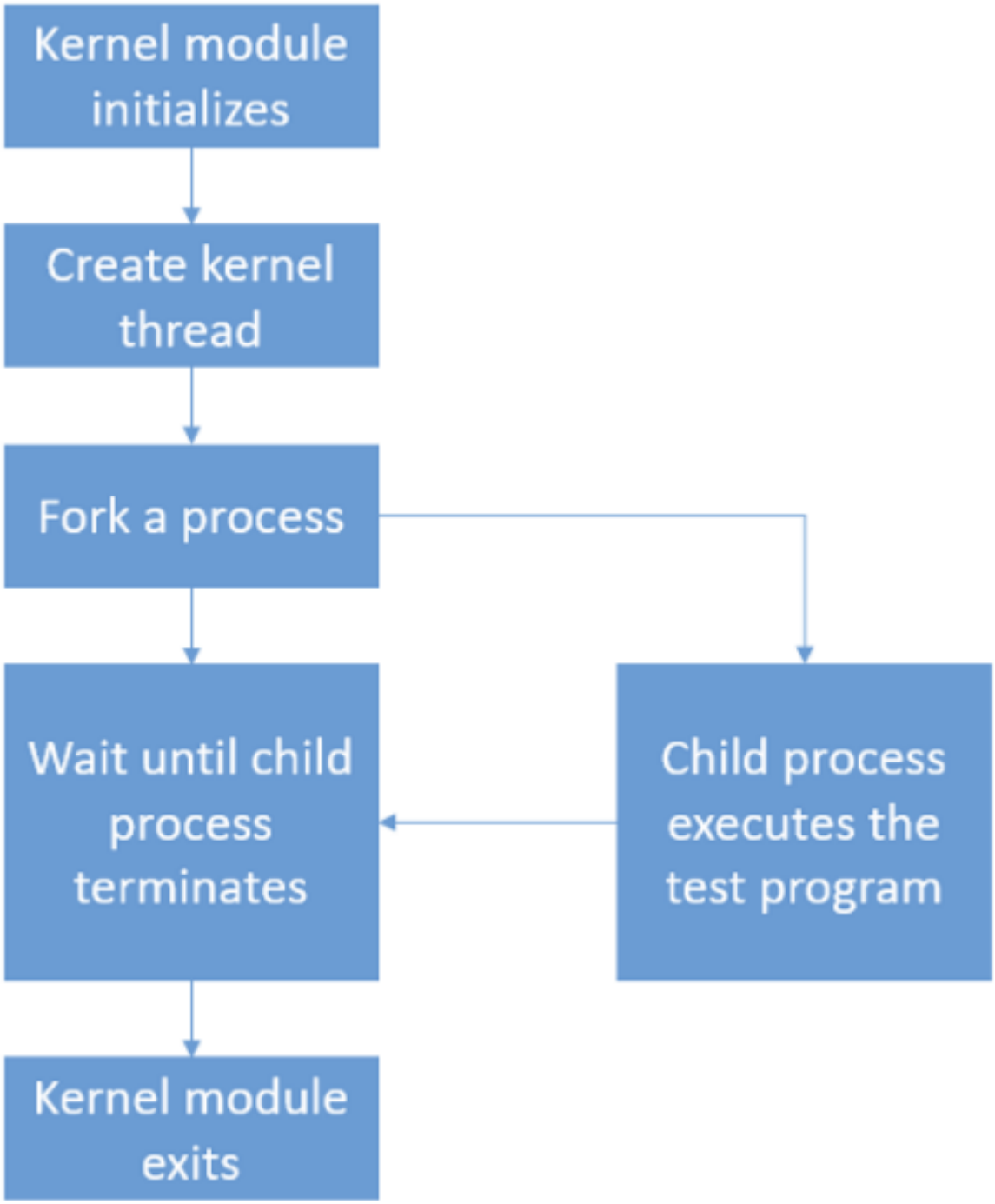
```
//parent process:
else {
    printf("I'm the Parent Process, my pid = %d\n", getpid());
    waitpid(pid, &status, WUNTRACED);
    printf("Parent process receives SIGCHLD signal\n");
    if (WIFEXITED(status)) {
        printf("Normal termination with EXIT STATUS = 0\n");
    } else if (WIFSIGNALED(status)) {
        int signal = WTERMSIG(status);
        if (signal == SIGABRT) {
            printf("Child process get SIGABRT signal\n");
        } else if (signal == SIGALRM) {
            printf("Child process get SIGALRM signal\n");
        } else if (signal == SIGBUS) {
            printf("Child process get SIGBUS signal\n");
        } else if (signal == SIGFPE) {
            printf("Child process get SIGFPE signal\n");
        } else if (signal == SIGHUP) {
            printf("Child process get SIGHUP signal\n");
        } else if (signal == SIGILL) {
            printf("Child process get SIGILL signal\n");
        } else if (signal == SIGINT) {
            printf("Child process get SIGINT signal\n");
        } else if (signal == SIGPIPE) {
            printf("Child process get SIGPIPE signal\n");
```

```
        } else if (signal == SIGQUIT) {
            printf("Child process get SIGQUIT signal\n");
        } else if (signal == SIGSEGV) {
            printf("Child process get SIGSEGV signal\n");
        } else if (signal == SIGTERM) {
            printf("Child process get SIGTERM signal\n");
        } else if (signal == SIGTRAP) {
            printf("Child process get SIGTRAP signal\n");
        } else if (signal = SIGKILL) {
            printf("Child process get SIGKILL signal\n");
        } else {
            printf("The signal is not correct\n");
        }
    } else if (WIFSTOPPED(status)) {
        printf("Child process get SIGSTOP signal\n");
    } else {
        printf("CHILD PROCESS CONTINUED\n");
    }
    exit(0);
}
```

After receivng the SIGCHLD signal, the parent process will print out the corresponding information of the received signal(**WIFEXITED(status)**, **IFSIGNALED(status)**, **WIFSTOPPED(status)** will be used to judge the received signal).

1.2 Task 2

The flow chart of task2:



- Use **kthread_create(&my_fork, NULL, "MyThread")** to create a kernel thread and run **my_fork()**. At this moment, the task formed by the thread is sleeping. If there is no error, use **wake_up_process(task)** to wake it up.

```
task = kthread_create(&my_fork, NULL, "MyThread");
if (!IS_ERR(task)) {
    printk("[program2] : Module_init kthread starts\n");
    wake_up_process(task);
}
```

- Before defining my_fork(), which is used to fork a child process, define **my_exec()** and **my_wait()**.

```
int my_exec(void)
{
    const char path[] ="/tmp/test";
    struct filename *file1 = getname_kernel(path);
    int i = do_execve(file1, NULL, NULL);
    if (i == 0) {
        return 0;
    } else {
        do_exit(i);
    }
}
```

Path is the address of the test file. Use **getname_kernel(path)** to get the filename and use **do_execve(file1, NULL, NULL)** to execute the test file. In my_wait(pid_t pid), **do_wait()** is the core function used to wait for child process("pid" in the funciton is the pid of the child process). Define struct wait_opts for do_wait():

```
struct wait_opts {
  enum pid_type wo_type;
  int wo_flags;
  struct pid *wo_pid;
  struct waitid_info *wo_info;
  int wo_stat;
  struct rusage *wo_rusage;
  wait_queue_entry_t child_wait;
  int notask_error;
};
```

Configure the contents in wait_opts:

```
long a;
struct wait_opts wo;
struct pid *wo_pid = NULL;
enum pid_type type;
type = PIDTYPE_PID;
wo_pid = find_get_pid(pid);
wo.wo_type = PIDTYPE_PID;
wo.wo_pid = wo_pid;
wo.wo_flags = WEXITED | WSTOPPED;
wo.wo_info = NULL;
wo.wo_rusage = NULL;
a = do_wait(&wo);
```

- wo.wo_stat&127 is the signal number taken from the child process. Use it to judge the signal.

```
if ((wo.wo_stat & 127) == SIGABRT) {
    printk("[program2] : get SIGABRT signal\n");
    printk("[program2] : child process was aborted\n");
    printk("[program2] : The return signal is %d\n",
           (wo.wo_stat & 127));
  } else if ((wo.wo_stat & 127) == SIGALRM) {
    printk("[program2] : get SIGALRM signal\n");
    printk("[program2] : child process was alarmed\n");
    printk("[program2] : The return signal is %d\n",
           (wo.wo_stat & 127));
  } else if ((wo.wo_stat & 127) == SIGBUS) {
    printk("[program2] : get SIGBUS signal\n");
    printk("[program2] : child process had bus error\n");
    printk("[program2] : The return signal is %d\n",
           (wo.wo_stat & 127));
  } else if ((wo.wo_stat & 127) == SIGFPE) {
    printk("[program2] : get SIGFPE signal\n");
    printk("[program2] : Child process had floating point exception\n");
    printk("[program2] : The return signal is %d\n",
           (wo.wo_stat & 127));
  } else if ((wo.wo_stat & 127) == SIGHUP) {
    printk("[program2] : get SIGHUP signal\n");
    printk("[program2] : Child process was hung up\n");
    printk("[program2] : The return signal is %d\n",
           (wo.wo_stat & 127));
  } else if ((wo.wo_stat & 127) == SIGILL) {
    printk("[program2] : get SIGILL signal\n");
    printk("[program2] : Child process had illegal instruction\n");
    printk("[program2] : The return signal is %d\n",
           (wo.wo_stat & 127));
  } else if ((wo.wo_stat & 127) == SIGINT) {
    printk("[program2] : get SIGINT signal\n");
    printk("[program2] : Child process was interrupted by teminal\n");
    printk("[program2] : The return signal is %d\n",
           (wo.wo_stat & 127));
  } else if ((wo.wo_stat & 127) == SIGPIPE) {
    printk("[program2] : get SIGPIPE signal\n");
    printk("[program2] : Child process was interrupted by broken pipe\n");
    printk("[program2] : The return signal is %d\n",
           (wo.wo_stat & 127));
  } else if ((wo.wo_stat & 127) == SIGQUIT) {
    printk("[program2] : get SIGQUIT signal\n");
    printk("[program2] : Child process had terminal quit\n");
    printk("[program2] : The return signal is %d\n",
           (wo.wo_stat & 127));
  } else if ((wo.wo_stat & 127) == SIGSEGV) {
    printk("[program2] : get SIGSEGV signal\n");
    printk("[program2] : Child process had problems in memeory segment
access\n");
    printk("[program2] : The return signal is %d\n",
           (wo.wo_stat & 127));
  } else if ((wo.wo_stat & 127) == SIGTERM) {
    printk("[program2] : get SIGTERM signal\n");
```

```
    printk("[program2] : Child process terminated\n");
    printk("[program2] : The return signal is %d\n",
           (wo.wo_stat & 127));
  } else if ((wo.wo_stat & 127) == SIGTRAP) {
    printk("[program2] : get SIGTRAP signal\n");
    printk("[program2] : Child process reached a trap\n");
    printk("[program2] : The return signal is %d\n",
           (wo.wo_stat & 127));
  } else if ((wo.wo_stat & 127) == SIGKILL) {
    printk("[program2] : get SIGKILL signal\n");
    printk("[program2] : Child process was killed\n");
    printk("[program2] : The return signal is %d\n",
           (wo.wo_stat & 127));
  } else if ((wo.wo_stat & 127) == 0) {
    printk("[program2] : Normal termination\n");
    printk("[program2] : The return signal is %d\n",
           (wo.wo_stat & 127));
  } else {
    printk("[program2] : get SIGSTOP signal\n");
    printk("[program2] : Child process was stopped\n");
    printk("[program2] : The return signal is 19");
}
```

The information of signal received will be printed out as above.

- In my_fork(), use **kernel_clone(&kernel_clone_args_0)** to create a new process. Print out the pid of child process and parent process. Then proform my_wait((pid_t)pid) to wait for the child process.

```
struct kernel_clone_args kernel_clone_args_0 = {
    .flags = SIGCHLD,
    .stack = (ul)&my_exec,
    .stack_size = 0,
    .parent_tid = NULL,
    .child_tid = NULL,
    .tls = 0,
    .pidfd = NULL,
    .exit_signal = (SIGCHLD & CSIGNAL)
};
long pid;
struct k_sigaction *k_action = &current->sighand->action[0];
int i;
for (i = 0; i < _NSIG; i++) {
    k_action->sa.sa_handler = SIG_DFL;
    k_action->sa.sa_flags = 0;
    k_action->sa.sa_restorer = NULL;
    sigemptyset(&k_action->sa.sa_mask);
    k_action++;
}
pid = kernel_clone(&kernel_clone_args_0);
printk("[program2] : The child process has pid = %ld\n", pid);
printk("[program2] : This is the parent process, pid = %d\n",(int)current-
```

```
>pid);
my_wait((pid_t)pid);
```

- Finally, perform **program2_exit()** to exit.

```
static void __exit program2_exit(void)
{
    printk("[program2] : Module_exit\n");
}
module_init(program2_init);
module_exit(program2_exit);
```

1.3 Bonus

- Create struct pstree, children and process.

```
struct pstree {
  pid_t pid;
  char name[50];
  struct children *child;
};

struct children {
    struct children *nextChild;
    struct pstree *pstree;
};

typedef struct process { //every process is a "Process" in this program
    char name[60];
    __pid_t pid;
    __pid_t ppid;
} Process;
```

  The pointer of the **first child** of a pstree root is **pstree.child**. Its other children can be found by **children.nextchild**.
- Correctly extract information from the commend input by programmer.

```
for (int i = 0; i < argc; i++) {
    assert(argv[i]);
}
char *option;
option = argv[1];
```

  Use **strcmp(option, str)** to judge the case.

- Define **getProcessInfomation(__pid_t pid, char processName[])** to get the **name** and **ppid** of a process according to its **pid**(Through the filenames, which are numbers in "/proc").

```
char *buffer = (char *)malloc(sizeof(char) * 30);
sprintf(buffer, "%d", pid);
char informationPath[30] = "/proc/";
strcat(informationPath, buffer);
strcat(informationPath, "/stat");
FILE *bufferFile = fopen(informationPath, "r");
__pid_t PID, PPID;
char bufferChar;
char bufferStr[40];
fscanf(bufferFile, "%d %s %c %d", &PID, bufferStr, &bufferChar, &PPID);
bufferStr[strlen(bufferStr) - 1] = '\0';
strcpy(processName, bufferStr);
```

- Use **opendir("/proc")**, and **readdir(bufferDir)** to read the pid of every process. Use **getProcessInfomation(__pid_t pid, char processName[])** to get their ppids and names. Then put the information in an **array** called **processList**.

```
int bufferInt = 0;
struct dirent *direntExample;
DIR *bufferDir = opendir("/proc");
while ((direntExample = readdir(bufferDir)) != NULL) {
    bufferInt = atoi(direntExample->d_name);
    if (bufferInt == 0)
        continue;
    else {
        processList[processCount].pid = bufferInt;
        processList[processCount].ppid = getProcessInfomation(
            bufferInt, processList[processCount].name);
        processCount++;
    }
}
```

- Define **makeTree(struct pstree root, int length)* to form the pstree according to the array(**processList**) above. Use an array **childrenBuffer[600]** to represent the children list of the root. Int **intBuffer** is used to control the spaces in the pstree. The base case is when **childrenBuffer[0] == 0**. In this case, the root is a leaf.

```
    if (childrenBuffer[0] == 0) {
    printf("%s(%d)", root->name, root->pid);
    return;
  }
```

In other cases, the root choosen has one or more children.

```
if (childrenBuffer[1] != 0)
    printf("┬─");
else
    intBuffer = 0;
```

After dealing with the root as above, respectively handle each child of the root. In each case, set the corresponding child as the new root for **makeTree()**.

```
for (int index = 0; index < 600 && childrenBuffer[index] != 0; index++) {
  child->pstree =(struct pstree *)malloc(sizeof(struct pstree));
  child->pstree->pid=processList[childrenBuffer[index]].pid;
  strcpy(child->pstree->name,
  processList[childrenBuffer[index]].name);
  ...
```

Then recursively sort the children.

```
makeTree(child->pstree,strlen(bufferString)+length+intBuffer);
```

Finally, modify the structure with several "\n", " ", and " ├─"(special string used to make the pstree smooth).

```
if (index + 1 < 600 && childrenBuffer[index + 1] != 0) {
  child->nextChild = (struct children *)malloc(sizeof(struct children));
  child = child->nextChild;
  printf("\n");
  for (size_t i = 0;
      i < strlen(bufferString) + length; i++) {
      printf(" ");
  }
  printf("├─");
}
```

At first, I know nothing about how to realize the makeTree() function using the processList. Therefore, I tried to search online about other people's realization of pstree. I was strongly inspired by many of relevant works and imitated some ways to finish the makeTree() function. I did learn a lot of knowledge about recursive methods and finished the task.

2. Set up the development environment

1. Install the virtualbox and vagrant according to the instructions from "https://csc3150.cyzhu.dev/vm-configuration/windows-amd-intel-x86". Set up VM and do corresponding operations to configure and set up Remote SSH plugin in VS Code. Find SSH Target "default" and connect to the VM.

2. Install dependency and development tools.

```
sudo apt-get install libncurses-dev gawk flex bison openssl libssl-dev dkms
libelf-dev libudev-dev libpci-dev libiberty-dev autoconf llvm dwarves
```

3. Download linux kernel source code from "https://mirror.tuna.tsinghua.edu.cn/kernel/v5.x/"("linux-5.10.10.tar.xz"). Extract the source file to a newly created folder(/home/vagrant/5.10.10) in the VM and decompress it. Then I get the kernel source file "linux-5.10.10".

4. Copy the "config-4.4.0-210-generic" from "/boot" to the kernel source file, in order to get the configuration of the previous kernel.

5. Login root account and go to kernel source directory.

```
sudo su
cd /home/vagrant/5.10.10/linux-5.10.10
```

6. Clean the previous settings, start configuration, save the configuration and exit.

```
make mrproper
make clean
make menuconfig
```

7. Build kernel image and modules.

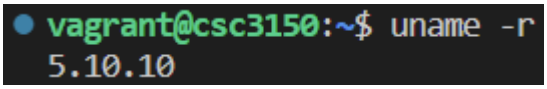
```
make -j$(nproc)
```

8. Install kernel modules, kernel.

```
make modules_install
make install
```

9. Reboot and check exiting kernel version.

```
reboot
uname -r
```



10. Search for "kernel_clone", "do_execve", "do_wait", "getname_kernel" in the kernel source file. Use "vim" to delete "static" of the functions and export the symbols.

```
EXPORT_SYMBOL(kernel_clone)
EXPORT_SYMBOL(do_execve)
EXPORT_SYMBOL(getname_kernel)
EXPORT_SYMBOL(do_wait)
```

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

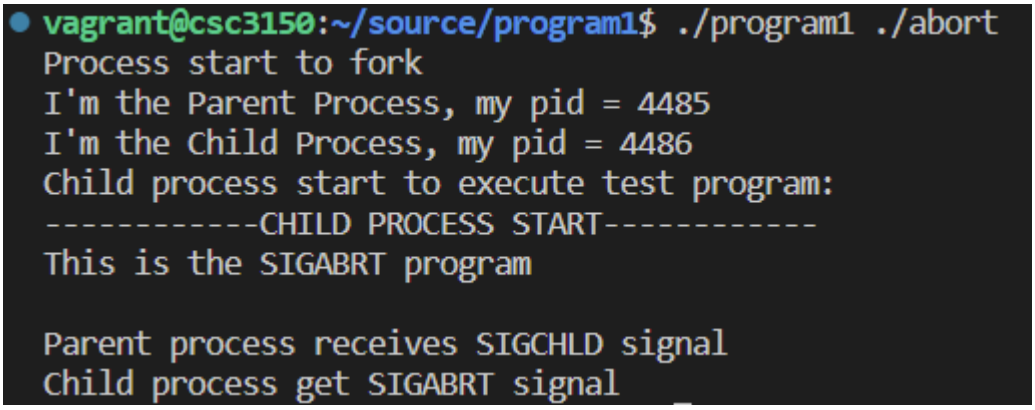
11. Recompile the kernel.

```
make bzImage
make modules
make modules_install
make install
reboot
```

3. Screenshot of program output

3.1 Task 1

abort



alarm

```
vagrant@csc3150:~/source/program1$ ./program1 ./alarm
Process start to fork
I'm the Parent Process, my pid = 4535
I'm the Child Process, my pid = 4536
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
```

bus

```
vagrant@csc3150:~/source/program1$ ./program1 ./bus
Process start to fork
I'm the Parent Process, my pid = 4580
I'm the Child Process, my pid = 4581
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
```

floating

```
vagrant@csc3150:~/source/program1$ ./program1 ./floating
Process start to fork
I'm the Parent Process, my pid = 4627
I'm the Child Process, my pid = 4628
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
```

hangup

```
vagrant@csc3150:~/source/program1$ ./program1 ./hangup
Process start to fork
I'm the Parent Process, my pid = 4670
I'm the Child Process, my pid = 4671
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
```

illegal_instr

```
vagrant@csc3150:~/source/program1$ ./program1 ./illegal_instr
Process start to fork
I'm the Parent Process, my pid = 4742
I'm the Child Process, my pid = 4743
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
```

interrupt

```
vagrant@csc3150:~/source/program1$ ./program1 ./interrupt
Process start to fork
I'm the Parent Process, my pid = 4803
I'm the Child Process, my pid = 4804
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
```

kill

```
vagrant@csc3150:~/source/program1$ ./program1 ./kill
Process start to fork
I'm the Parent Process, my pid = 4841
I'm the Child Process, my pid = 4842
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
```

normal

```
vagrant@csc3150:~/source/program1$ ./program1 ./normal
Process start to fork
I'm the Parent Process, my pid = 4891
I'm the Child Process, my pid = 4892
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
```

pipe

```
vagrant@csc3150:~/source/program1$ ./program1 ./pipe
Process start to fork
I'm the Parent Process, my pid = 4929
I'm the Child Process, my pid = 4930
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
```

quit

```
vagrant@csc3150:~/source/program1$ ./program1 ./quit
Process start to fork
I'm the Parent Process, my pid = 5003
I'm the Child Process, my pid = 5004
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
```

segment_fault

```
vagrant@csc3150:~/source/program1$ ./program1 ./segment_fault
Process start to fork
I'm the Parent Process, my pid = 5074
I'm the Child Process, my pid = 5075
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
```

stop

```
vagrant@csc3150:~/source/program1$ ./program1 ./stop
Process start to fork
I'm the Parent Process, my pid = 5134
I'm the Child Process, my pid = 5135
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
```

terminate

```
vagrant@csc3150:~/source/program1$ ./program1 ./terminate
Process start to fork
I'm the Parent Process, my pid = 5182
I'm the Child Process, my pid = 5183
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
```

trap

```
vagrant@csc3150:~/source/program1$ ./program1 ./trap
Process start to fork
I'm the Parent Process, my pid = 5209
I'm the Child Process, my pid = 5210
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
```

3.2 Task 2

test

```
[  615.584056] [program2] : Module_init XIAO Weizhao 120090588
[  615.584056] [program2] : Module_init create kthread start
[  615.584164] [program2] : Module_init kthread start
[  615.584462] [program2] : The child process has pid = 5569
[  615.584463] [program2] : This is the parent process, pid = 5567
[  615.679464] [program2] : Child process
[  615.679465] [program2] : get SIGBUS signal
[  615.679466] [program2] : child process had bus error
[  615.679466] [program2] : The return signal is 7
[  615.679467] [program2] : Child process terminated
[  617.671748] [program2] : Module_exit./my
```

abort

```
[  759.727888] [program2] : Module_init XIAO Weizhao 120090588
[  759.727889] [program2] : Module_init create kthread start
[  759.728000] [program2] : Module_init kthread start
[  759.728057] [program2] : The child process has pid = 5993
[  759.728059] [program2] : This is the parent process, pid = 5992
[  759.822387] [program2] : Child process
[  759.822388] [program2] : get SIGABRT signal
[  759.822388] [program2] : child process was aborted
[  759.822389] [program2] : The return signal is 6
[  759.822389] [program2] : Child process terminated
[  765.086109] [program2] : Module_exit./my
```

alarm

```
[  819.821675] [program2] : Module_init XIAO Weizhao 120090588
[  819.823344] [program2] : Module_init create kthread start
[  819.825183] [program2] : Module_init kthread start
[  819.827000] [program2] : The child process has pid = 6490
[  819.828761] [program2] : This is the parent process, pid = 6488
[  821.853713] [program2] : Child process
[  821.854918] [program2] : get SIGALRM signal
[  821.856195] [program2] : child process was alarmed
[  821.858371] [program2] : The return signal is 14
[  821.859699] [program2] : Child process terminated
[  822.171618] [program2] : Module_exit./my
```

floating

```
[  880.645924] [program2] : Module_init XIAO Weizhao 120090588
[  880.647152] [program2] : Module_init create kthread start
[  880.648471] [program2] : Module_init kthread start
[  880.649607] [program2] : The child process has pid = 6890
[  880.651387] [program2] : This is the parent process, pid = 6889
[  880.737715] [program2] : Child process
[  880.738851] [program2] : get SIGFPE signal
[  880.740028] [program2] : Child process had floating point exception
[  880.741766] [program2] : The return signal is 8
[  880.743056] [program2] : Child process terminated
[  882.228362] [program2] : Module_exit./my
```

hangup

```
[  972.909714] [program2] : Module_init XIAO Weizhao 120090588
[  972.911373] [program2] : Module_init create kthread start
[  972.913015] [program2] : Module_init kthread start
[  972.914439] [program2] : The child process has pid = 7340
[  972.916033] [program2] : This is the parent process, pid = 7339
[  972.917739] [program2] : Child process
[  972.918905] [program2] : get SIGHUP signal
[  972.920185] [program2] : Child process was hung up
[  972.921527] [program2] : The return signal is 1
[  972.922813] [program2] : Child process terminated
[  974.659388] [program2] : Module_exit./my
```

illegal_instr

```
[ 1029.647631] [program2] : Module_init XIAO Weizhao 120090588
[ 1029.649203] [program2] : Module_init create kthread start
[ 1029.650402] [program2] : Module_init kthread start
[ 1029.651465] [program2] : The child process has pid = 7741
[ 1029.653236] [program2] : This is the parent process, pid = 7740
[ 1029.737686] [program2] : Child process
[ 1029.738713] [program2] : get SIGILL signal
[ 1029.739799] [program2] : Child process had illegal instruction
[ 1029.742264] [program2] : The return signal is 4
[ 1029.743446] [program2] : Child process terminated
[ 1031.190693] [program2] : Module_exit./my
```

interrupt

```
[ 1075.622085] [program2] : Module_init XIAO Weizhao 120090588
[ 1075.623757] [program2] : Module_init create kthread start
[ 1075.625396] [program2] : Module_init kthread start
[ 1075.626756] [program2] : The child process has pid = 8154
[ 1075.627965] [program2] : This is the parent process, pid = 8153
[ 1075.629368] [program2] : Child process
[ 1075.630361] [program2] : get SIGINT signal
[ 1075.631417] [program2] : Child process was interrupted by teminal
[ 1075.632877] [program2] : The return signal is 2
[ 1075.633830] [program2] : Child process terminated
[ 1077.206622] [program2] : Module_exit./my
```

kill

```
[ 1121.131197] [program2] : Module_init XIAO Weizhao 120090588
[ 1121.132328] [program2] : Module_init create kthread start
[ 1121.133468] [program2] : Module_init kthread start
[ 1121.134481] [program2] : The child process has pid = 8574
[ 1121.136034] [program2] : This is the parent process, pid = 8573
[ 1121.137892] [program2] : Child process
[ 1121.138970] [program2] : get SIGKILL signal
[ 1121.145411] [program2] : Child process was killed
[ 1121.146717] [program2] : The return signal is 9
[ 1121.148414] [program2] : Child process terminated
[ 1123.152662] [program2] : Module_exit./my
```

normal

```
[ 1194.440562] [program2] : Module_init XIAO Weizhao 120090588
[ 1194.441633] [program2] : Module_init create kthread start
[ 1194.442734] [program2] : Module_init kthread start
[ 1194.443702] [program2] : The child process has pid = 9006
[ 1194.445266] [program2] : This is the parent process, pid = 9005
[ 1194.447005] [program2] : Child process
[ 1194.448132] [program2] : Normal termination
[ 1194.457374] [program2] : The return signal is 0
[ 1194.458589] [program2] : Child process terminated
[ 1196.089389] [program2] : Module_exit./my
```

pipe

```
[ 1366.494093] [program2] : Module_init XIAO Weizhao 120090588
[ 1366.495633] [program2] : Module_init create kthread start
[ 1366.497189] [program2] : Module_init kthread start
[ 1366.498533] [program2] : The child process has pid = 10135
[ 1366.500452] [program2] : This is the parent process, pid = 10134
[ 1366.502245] [program2] : Child process
[ 1366.503356] [program2] : get SIGPIPE signal
[ 1366.504508] [program2] : Child process was interrupted by broken pipe
[ 1366.506158] [program2] : The return signal is 13
[ 1366.507404] [program2] : Child process terminated
[ 1368.000223] [program2] : Module_exit./my
```

quit

```
[ 1425.680456] [program2] : Module_init XIAO Weizhao 120090588
[ 1425.681861] [program2] : Module_init create kthread start
[ 1425.683453] [program2] : Module_init kthread start
[ 1425.684861] [program2] : The child process has pid = 10641
[ 1425.686365] [program2] : This is the parent process, pid = 10640
[ 1425.767288] [program2] : Child process
[ 1425.768536] [program2] : get SIGQUIT signal
[ 1425.769740] [program2] : Child process had terminal quit
[ 1425.771195] [program2] : The return signal is 3
[ 1425.772428] [program2] : Child process terminated
[ 1428.895865] [program2] : Module_exit./my
```

segment_fault

```
[ 1492.174545] [program2] : Module_init XIAO Weizhao 120090588
[ 1492.175990] [program2] : Module_init create kthread start
[ 1492.177600] [program2] : Module_init kthread start
[ 1492.178950] [program2] : The child process has pid = 11070
[ 1492.180389] [program2] : This is the parent process, pid = 11069
[ 1492.269084] [program2] : Child process
[ 1492.270190] [program2] : get SIGSEGV signal
[ 1492.271460] [program2] : Child process had problems in memeory segment access
[ 1492.273820] [program2] : The return signal is 11
[ 1492.275108] [program2] : Child process terminated
[ 1493.924252] [program2] : Module_exit./my
```

stop

```
[ 1606.587576] [program2] : Module_init XIAO Weizhao 120090588
[ 1606.589109] [program2] : Module_init create kthread start
[ 1606.590755] [program2] : Module_init kthread start
[ 1606.592240] [program2] : The child process has pid = 11474
[ 1606.593850] [program2] : This is the parent process, pid = 11473
[ 1606.595394] [program2] : Child process
[ 1606.597417] [program2] : get SIGSTOP signal
[ 1606.598533] [program2] : Child process was stopped
[ 1606.599866] [program2] : The return signal is 19
[ 1606.599866] [program2] : Child process terminated
[ 1608.043902] [program2] : Module_exit./my
```

terminate

```
[ 1656.776716] [program2] : Module_init XIAO Weizhao 120090588
[ 1656.778345] [program2] : Module_init create kthread start
[ 1656.780002] [program2] : Module_init kthread start
[ 1656.781488] [program2] : The child process has pid = 11874
[ 1656.782584] [program2] : This is the parent process, pid = 11873
[ 1656.783786] [program2] : Child process
[ 1656.784589] [program2] : get SIGTERM signal
[ 1656.801300] [program2] : Child process terminated
[ 1656.802390] [program2] : The return signal is 15
[ 1656.803240] [program2] : Child process terminated
[ 1658.337396] [program2] : Module_exit./my
```

trap

```
[ 1714.376999] [program2] : Module_init XIAO Weizhao 120090588
[ 1714.378085] [program2] : Module_init create kthread start
[ 1714.379229] [program2] : Module_init kthread start
[ 1714.380387] [program2] : The child process has pid = 12275
[ 1714.381824] [program2] : This is the parent process, pid = 12274
[ 1714.466254] [program2] : Child process
[ 1714.467525] [program2] : get SIGTRAP signal
[ 1714.469020] [program2] : Child process reached a trap
[ 1714.470379] [program2] : The return signal is 5
[ 1714.471553] [program2] : Child process terminated
[ 1716.586483] [program2] : Module_exit./my
```

3.3 Bonus

pstree -n

```
vagrant@csc3150:~/source/bonus$ ./pstree -n
(systemd─┬─(systemd-journal
         ├─(lvmetad
         ├─(systemd-udevd
         ├─(dhclient
         ├─(atd
         ├─(rsyslogd
         ├─(systemd-logind
         ├─(dbus-daemon
         ├─(accounts-daemon
         ├─(lxcfs
         ├─(acpid
         ├─(cron
         ├─(iscsid
         ├─(iscsid
         ├─(unattended-upgr
         ├─(mdadm
         ├─(polkitd
         ├─(sshd─┬─(sshd(sshd(bash─┬─(sh(node─┬─(node(bash(pstree
                                            ├─(node─┬─(cpptools
                                                    ├─(node
                                            ├─(node
                                  ├─(sleep
                 ├─(sshd(sshd(bash(sleep
                 ├─(sshd(sshd(bash(sleep
                 ├─(sshd(sshd(bash(sleep
                 ├─(sshd(sshd(bash(sleep
                 ├─(sshd(sshd(bash(sleep
                 ├─(sshd(sshd(bash(sleep
                 ├─(sshd(sshd(bash(sleep
         ├─(agetty
         ├─(agetty
         ├─(irqbalance
         ├─(systemd((sd-pam)
         ├─(stop
         ├─(cpptools-srv
```

pstree -V

```
vagrant@csc3150:~/source/bonus$ ./pstree -V
pstree (PSmisc) 22.21
Copyright (C) 1993-2009 Werner Almesberger and Craig Small
PSmisc comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it under
the terms of the GNU General Public License.
For more information about these matters, see the files named COPYING.
```

pstree

```
vagrant@csc3150:~/source/bonus$ ./pstree
(systemd─┬─(systemd-journal
         ├─(lvmetad
         ├─(systemd-udevd
         ├─(dhclient
         ├─(atd
         ├─(rsyslogd
         ├─(systemd-logind
         ├─(dbus-daemon
         ├─(accounts-daemon
         ├─(lxcfs
         ├─(acpid
         ├─(cron
         ├─(iscsid
         ├─(iscsid
         ├─(unattended-upgr
         ├─(mdadm
         ├─(polkitd
         ├─(sshd─┬─(sshd(sshd(bash─┬─(sh(node─┬─(node(bash(pstree
                                            ├─(node─┬─(cpptools
                                                    ├─(node
                                            ├─(node
                                  ├─(sleep
                 ├─(sshd(sshd(bash(sleep
                 ├─(sshd(sshd(bash(sleep
                 ├─(sshd(sshd(bash(sleep
                 ├─(sshd(sshd(bash(sleep
                 ├─(sshd(sshd(bash(sleep
                 ├─(sshd(sshd(bash(sleep
                 ├─(sshd(sshd(bash(sleep
         ├─(agetty
         ├─(agetty
         ├─(irqbalance
         ├─(systemd((sd-pam)
         ├─(stop
         ├─(cpptools-srv
```

pstree -A

```
vagrant@csc3150:~/source/bonus$ ./pstree -A
(systemd-+-(systemd-journal
          |-(lvmetad
          |-(systemd-udevd
          |-(dhclient
          |-(atd
          |-(rsyslogd
          |-(systemd-logind
          |-(dbus-daemon
          |-(accounts-daemon
          |-(lxcfs
          |-(acpid
          |-(cron
          |-(iscsid
          |-(iscsid
          |-(unattended-upgr
          |-(mdadm
          |-(polkitd
          |-(sshd-+-(sshd-(sshd-(bash-+-(sh-(node-+-(node-(bash-(pstree
                                                  |-(node-+-(cpptools
                                                          |-(node
                                                  |-(node
                                        |-(sleep
                  |-(sshd-(sshd-(bash-(sleep
                  |-(sshd-(sshd-(bash-(sleep
                  |-(sshd-(sshd-(bash-(sleep
                  |-(sshd-(sshd-(bash-(sleep
                  |-(sshd-(sshd-(bash-(sleep
                  |-(sshd-(sshd-(bash-(sleep
                  |-(sshd-(sshd-(bash-(sleep
          |-(agetty
          |-(agetty
          |-(irqbalance
          |-(systemd-((sd-pam)
          |-(stop
          |-(cpptools-srv
```

pstree -p

```
● vagrant@csc3150:~/source/bonus$ ./pstree -p
  (systemd(1)┬─(systemd-journal(420)
             ├─(lvmetad(438)
             ├─(systemd-udevd(449)
             ├─(dhclient(909)
             ├─(atd(1030)
             ├─(rsyslogd(1031)
             ├─(systemd-logind(1032)
             ├─(dbus-daemon(1033)
             ├─(accounts-daemon(1037)
             ├─(lxcfs(1041)
             ├─(acpid(1045)
             ├─(cron(1053)
             ├─(iscsid(1066)
             ├─(iscsid(1067)
             ├─(unattended-upgr(1079)
             ├─(mdadm(1087)
             ├─(polkitd(1091)
             ├─(sshd(1094)┬─(sshd(1597)(sshd(1658)(bash(1659)┬─(sh(1704)(node(1714)┬─(node(1782)(bash(16285)(pstree(16937)
                                                                                                  ├─(node(16062)┬─(cpptools(16140)
                                                                                                                ├─(node(16184)
                                                                                                  ├─(node(16075)
                                                                                   ├─(sleep(16908)
                          ├─(sshd(2030)(sshd(2065)(bash(2066)(sleep(16910)
                          ├─(sshd(3030)(sshd(3087)(bash(3088)(sleep(16907)
                          ├─(sshd(3933)(sshd(3990)(bash(3991)(sleep(16909)
                          ├─(sshd(5247)(sshd(5282)(bash(5283)(sleep(16882)
                          ├─(sshd(7516)(sshd(7551)(bash(7552)(sleep(16936)
                          ├─(sshd(7920)(sshd(7955)(bash(7956)(sleep(16880)
                          ├─(sshd(15972)(sshd(16007)(bash(16008)(sleep(16881)
             ├─(agetty(1127)
             ├─(agetty(1131)
             ├─(irqbalance(1148)
             ├─(systemd(1599)((sd-pam)(1600)
             ├─(stop(15066)
             ├─(cpptools-srv(16270)
```

pstree -U

```
vagrant@csc3150:~/source/bonus$ ./pstree -U
(systemd─┬─(systemd-journal
         ├─(lvmetad
         ├─(systemd-udevd
         ├─(dhclient
         ├─(atd
         ├─(rsyslogd
         ├─(systemd-logind
         ├─(dbus-daemon
         ├─(accounts-daemon
         ├─(lxcfs
         ├─(acpid
         ├─(cron
         ├─(iscsid
         ├─(iscsid
         ├─(unattended-upgr
         ├─(mdadm
         ├─(polkitd
         ├─(sshd─┬─(sshd(sshd(bash─┬─(sh(node─┬─(node(bash(pstree
                                              ├─(node─┬─(cpptools
                                                      ├─(node
                                              ├─(node
                                 ├─(sleep
                 ├─(sshd(sshd(bash(sleep
                 ├─(sshd(sshd(bash(sleep
                 ├─(sshd(sshd(bash(sleep
                 ├─(sshd(sshd(bash(sleep
                 ├─(sshd(sshd(bash(sleep
                 ├─(sshd(sshd(bash(sleep
                 ├─(sshd(sshd(bash(sleep
         ├─(agetty
         ├─(agetty
         ├─(irqbalance
         ├─(systemd((sd-pam)
         ├─(stop
         ├─(cpptools-srv
```

4. Knowledge from the tasks

4.1 Task 1

Task 1 is about user mode programming. I have learned to:

1. use **fork()** to fork new process;
2. use **execve()** to do execution;
3. use corresponding functions to **identify the signal** from the child process.

4.2 Task 2

Task 2 is about kernel mode programming. I have learned to

1. make a simple kernel module and knew how to insmod and rmmod.
2. configure the VM environment(compile the kernel);
3. use **vim** to modify the kernel, export the symbols;
4. check the references "https://elixir.bootlin.com/linux/v5.10/source" to gain the way in which I **set the arguments** for the **kernel_clone()** and **do_wait()**;
5. deal with the forking of new process in kernel mode, for example, using **kthread_create()** to create a kernel thread, using **kernel_clone()** to fork a new process;
6. check the signal from child process comparing wo.wo_stat&127 with corresponding signal(such as SIGABRT);
7. use **printk** to log things.

4.3 Bonus

I have learned to

1. use fopen() to read a file and use fscanf() to get information;
2. how to form a pstree according to the relationship array processList using recursive methods.