CSC3150 Project1 Report

Name: Chen Huaxun

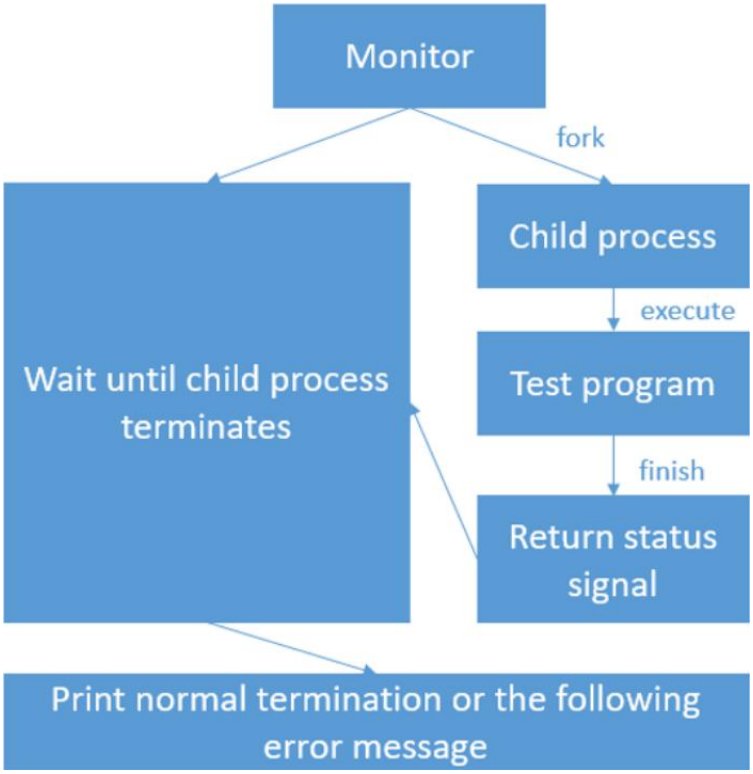
Student ID: 120090440

Design

Program1

1. Overall

In program 1, we are required to implement a program to (1) fork a child process, (2) make child process execute and send signal, (3) parent process wait for the child process, (4) parent process receives the signal and print the received signal. The following chart is the overall design of program 1.



2. Details

> Fork a child process:

pid_t fork (void)

The **fork()** function will create a child process which is an exact clone of the original parent process, except that it has its own process ID. Once it execute successfully, Unix will make two identical copies of address spaces, one for parent and the other for the child. Both process will start their execution at the next statement following the **fork()** call. In child process the return value of **fork()** is 0, in parent process the return value of **fork()** is pid of the child process.

> Execute a file:

int execve (const char *filename, char *const argv[], char *const env[])

The exec function used in the program to execute child process is **execve()**. The filename and arguments are passed to the main function of program1, which can be used to execute the child process.

> Wait for child process:

pid_t waitpid (pid_t pid, int *status_ptr, int options)

To make the parent process wait for the child process, call **wait()** function. However, if the child process failed, the **wait()** function will not report the stopped child process. As a result, use **waitpid()** and set the option to WUNTRACED in order to get reports on stopped child as well.

> Received signals classify and evaluate:

int WIFEXITED(), WIFSIGNALED(), WIFSTOPPED()

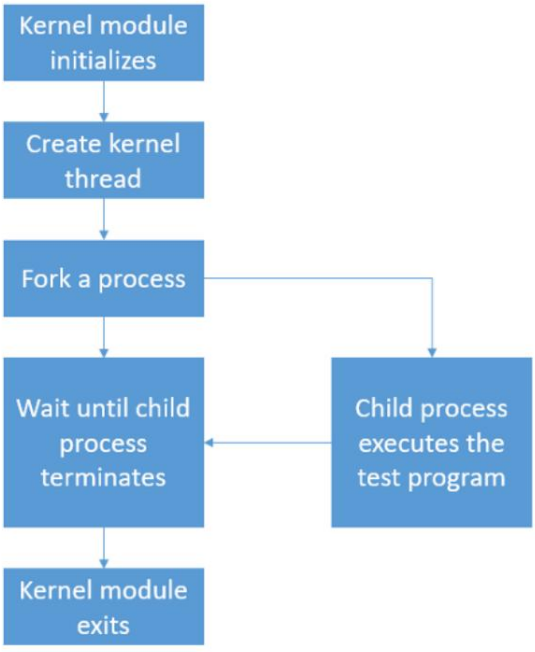
int WEXITSTATUS(), WTERMSIG(), WSTOPSIG()

When **waitpid()** returns with a valid pid, use first line macros to analyse the status referenced by the status argument. Use second line macros to get the exact values of the status argument.

Program2

1. Overall

In program 2 , we should create a kernel thread and fork a child process. We need (1) create a kernel thread and run my_fork function, (2) fork a child process in my_fork and execute test programs by my_exec, (3) parent process wait until child process terminates (use my_wait()), (4) verify and print the signals.The following chart is the overall design of program 2.



2. Details

> Kernel compile:

EXPORT_SYMPOL()

To use those APIs (**kernel_clone, do_wait, do_execve, getname_kernel**), (1) add **EXPORT_SYMPOL(function name);** behind those kernel functions, (2) type $make -j$(nproc) in command line and enter, (3) install kernel modules and install kernel ($make modules_install, $make install), (4) use 'extern' to clarify them before use in program2.c.

> Create and exit the kernel thread:

kthread_create()

In program 2, we need to initialize and exit a kernel thread. To implement these functions, we use the two methods: program2_init() and program2_exit(). In the program2_init method, we should create the kernel thread by calling the kthread_create().

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

> Fork a child process:

my_fork(), kernel_clone()

To fork a child process, we can call the **kernel_clone()** function in **my_fork()** function. It will create a child process and return the pid of child process if fork successfully.

```
pid = kernel_clone(&kargs);
```

The pointer to my_exec function should be passed to .stack in kargs so the execution of test file works well. Pass SIGCHLD to .flags and .exit_signal.

> Execute child process:

my_exec(), do_execve()

Program 2 uses **do_execve()** to execute the child process. In **my_exec()** function, use **do_execve()** and pass the absolute path to it, set argv and envp to **NULL**. It will return 0 if it execute successfully.

> Wait for child process:

my_wait(), do_wait()

do_wait() will make the parent process wait until child process terminates. Call **do_wait()** inside **my_wait()**, pass the reference of wait_opts to it, where set the .wo_flags = WEXITED | WUNTRACED to get reports on stopped child process. In **my_wait()**, also need to get and print the signal of child process. Bitwise operate 127 with wo.wo_stat to get the exact signal.

Bonus

1. Overall

In bonus, we need to design a pstree.c program to implement the linux command pstree. There are mainly four parts: (1) read the info of processes, (2) get name, pid, ppid, pgid, threads (3) read and process the input options, (4) dump tree.

Flow:

Read arguments -> set options values -> read process information -> dump tree

2. Details

> Get options:

getopt_long()

Include <getopt.h> to use it, **getopt_long()** could easily match the arguments passed in command line to a char array in C program so that I can use switch case to set specific tree to dump.

```
while ((c = getopt_long(argc, argv, optstr, options, NULL)) != -1)
    switch (c) {...}
```

> Read the info of process:

opendir(), readdir()

In linux, everything is a file. The info of all process are stored under "/proc" as files. The general path is like "proc/[pid]/task/[tid]/stat" or "proc/[pid]/stat", and we can get name and ppid at stat page. Things mainly done here is open file -> read file -> open file -> read file...

> Print tree:

Recursion. (1) print root process, (2) print the tree rooted at its parent process.

> Print name, id, ascii code:

Print_char, print_string, print_int

Simplify the process of output characters to screen

> Build the data structure of PROC

add_proc

(1) When we add a PROC whose parent is not in the PROC list, we add the parent PROC into the list by pid, which is unique for every process. (2) When we try to add a PROC, we first check whether it is added by its children . If it is, we update it to its name and pid, otherwise we add it.

find_proc, new_proc

Find process in the list, if find, return the process, else return NULL.

Update an old process or create a newly find process.

Environment

```
vagrant@csc3150:/home/seed/work/proj1/source$ cat /etc/issue
Ubuntu 16.04.7 LTS \n \l

vagrant@csc3150:/home/seed/work/proj1/source$ uname -r
5.10.146
vagrant@csc3150:/home/seed/work/proj1/source$ gcc --version
gcc (Ubuntu 5.4.0-6ubuntu1~16.04.12) 5.4.0 20160609
Copyright (C) 2015 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Kernel compile

EXPORT_SYMPOL()

To use those APIs (**kernel_clone, do_wait, do_execve, getname_kernel**), (1) add **EXPORT_SYMPOL(function name);** behind those kernel functions, (2) type $make -j$(nproc) in command line and enter, (3) install kernel modules and install kernel ($make modules_install, $make install), (4) use 'extern' to clarify them before use in program2.c.

How to run

Program1

1. Type 'make'
2. Type ./program1 + executable file name (abort, alarm...)

Program2

1. Recompile kernel
2. Type 'make'
3. Type 'sudo insmod program2.ko'
4. Type 'sudo rmmod program2.ko'
5. Type 'dmesg'

Program3

1. Type 'make'
2. Type './pstree' or './pstree -opt' where opt = V or n or p or g or c

---

Screen Shot

Program1

```
root@csc3150:/home/seed/work/proj1/source/program1# ./program1 abort
Process start to fork
I'm the Parent Process, my pid = 2350
I'm the Child Process, my pid = 2351
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
CHILD EXECUTION FAILED: 6
```

```
root@csc3150:/home/seed/work/proj1/source/program1# ./program1 alarm
Process start to fork
I'm the Parent Process, my pid = 2428
I'm the Child Process, my pid = 2429
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
CHILD EXECUTION FAILED: 14
```

```
root@csc3150:/home/seed/work/proj1/source/program1# ./program1 bus
Process start to fork
I'm the Parent Process, my pid = 2507
I'm the Child Process, my pid = 2508
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
CHILD EXECUTION FAILED: 7
```

```
root@csc3150:/home/seed/work/proj1/source/program1# ./program1 floating
Process start to fork
I'm the Parent Process, my pid = 2594
I'm the Child Process, my pid = 2595
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
CHILD EXECUTION FAILED: 8
```

```
root@csc3150:/home/seed/work/proj1/source/program1# ./program1 hangup
Process start to fork
I'm the Parent Process, my pid = 2682
I'm the Child Process, my pid = 2683
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
CHILD EXECUTION FAILED: 1
```

```
root@csc3150:/home/seed/work/proj1/source/program1# ./program1 illegal_instr
Process start to fork
I'm the Parent Process, my pid = 2747
I'm the Child Process, my pid = 2748
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
CHILD EXECUTION FAILED: 4
```

```
root@csc3150:/home/seed/work/proj1/source/program1# ./program1 interrupt
Process start to fork
I'm the Parent Process, my pid = 2826
I'm the Child Process, my pid = 2827
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
CHILD EXECUTION FAILED: 2
```

```
root@csc3150:/home/seed/work/proj1/source/program1# ./program1 kill
Process start to fork
I'm the Parent Process, my pid = 2900
I'm the Child Process, my pid = 2901
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
CHILD EXECUTION FAILED: 9
```

```
root@csc3150:/home/seed/work/proj1/source/program1# ./program1 normal
Process start to fork
I'm the Parent Process, my pid = 2944
I'm the Child Process, my pid = 2945
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
```

```
root@csc3150:/home/seed/work/proj1/source/program1# ./program1 pipe
Process start to fork
I'm the Parent Process, my pid = 3000
I'm the Child Process, my pid = 3001
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
CHILD EXECUTION FAILED: 13
```

```
root@csc3150:/home/seed/work/proj1/source/program1# ./program1 quit
Process start to fork
I'm the Parent Process, my pid = 3074
I'm the Child Process, my pid = 3075
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
CHILD EXECUTION FAILED: 3
```

```
root@csc3150:/home/seed/work/proj1/source/program1# ./program1 segment_fault
Process start to fork
I'm the Parent Process, my pid = 3165
I'm the Child Process, my pid = 3166
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
CHILD EXECUTION FAILED: 11
```

```
root@csc3150:/home/seed/work/proj1/source/program1# ./program1 stop
Process start to fork
I'm the Parent Process, my pid = 3210
I'm the Child Process, my pid = 3211
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
CHILD PROCESS STOPPED: 19
```

```
root@csc3150:/home/seed/work/proj1/source/program1# ./program1 terminate
Process start to fork
I'm the Parent Process, my pid = 3276
I'm the Child Process, my pid = 3277
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
CHILD EXECUTION FAILED: 15
```

```
root@csc3150:/home/seed/work/proj1/source/program1# ./program1 trap
Process start to fork
I'm the Parent Process, my pid = 3341
I'm the Child Process, my pid = 3342
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
CHILD EXECUTION FAILED: 5
```

Program2

```
[ 6664.219353] [program2] : Module_init ChenHuaxun 120090440
[ 6664.219354] [program2] : Module_init create kthread start
[ 6664.219450] [program2] : Module_init kthread start
[ 6664.219595] [program2] : The child process has pid = 16020
[ 6664.219596] [program2] : This is the parent process, pid = 16019
[ 6664.219620] [program2] : child process
[ 6664.220053] [program2] : child process terminates normally
[ 6664.220054] [program2] : The return signal is 0
[ 6713.370879] [program2] : Module_exit./my
```

```
[ 6610.244758] [program2] : Module_init ChenHuaxun 120090440
[ 6610.244759] [program2] : Module_init create kthread start
[ 6610.244850] [program2] : Module_init kthread start
[ 6610.244938] [program2] : The child process has pid = 15200
[ 6610.244939] [program2] : This is the parent process, pid = 15199
[ 6610.244960] [program2] : child process
[ 6610.324020] [program2] : get SIGABRT signal
[ 6610.324021] [program2] : child process is aborted
[ 6610.324022] [program2] : The return signal is 6
[ 6628.223048] [program2] : Module_exit./my
```

```
[ 6939.091307] [program2] : Module_init ChenHuaxun 120090440
[ 6939.091309] [program2] : Module_init create kthread start
[ 6939.091373] [program2] : Module_init kthread start
[ 6939.091399] [program2] : The child process has pid = 18154
[ 6939.091400] [program2] : This is the parent process, pid = 18153
[ 6939.091436] [program2] : child process
[ 6941.165749] [program2] : get SIGALARM signal
[ 6941.165751] [program2] : child process releases alarm signal
[ 6941.165751] [program2] : The return signal is 14
[ 6949.558937] [program2] : Module_exit./my
```

```
[ 6972.396788] [program2] : Module_init ChenHuaxun 120090440
[ 6972.396789] [program2] : Module_init create kthread start
[ 6972.396983] [program2] : Module_init kthread start
[ 6972.397101] [program2] : The child process has pid = 18845
[ 6972.397101] [program2] : This is the parent process, pid = 18844
[ 6972.397111] [program2] : child process
[ 6972.480587] [program2] : get SIGBUS signal
[ 6972.480588] [program2] : child process has bus error
[ 6972.480589] [program2] : The return signal is 7
[ 6981.655104] [program2] : Module_exit./my
```

```
[ 7003.242844] [program2] : Module_init ChenHuaxun 120090440
[ 7003.242846] [program2] : Module_init create kthread start
[ 7003.242994] [program2] : Module_init kthread start
[ 7003.243325] [program2] : The child process has pid = 19496
[ 7003.243326] [program2] : This is the parent process, pid = 19495
[ 7003.243359] [program2] : child process
[ 7003.319182] [program2] : get SIGFPE signal
[ 7003.319183] [program2] : child process has floating point exception
[ 7003.319184] [program2] : The return signal is 8
[ 7010.014346] [program2] : Module_exit./my
```

```
[ 7046.675654] [program2] : Module_init ChenHuaxun 120090440
[ 7046.675655] [program2] : Module_init create kthread start
[ 7046.675709] [program2] : Module_init kthread start
[ 7046.675769] [program2] : The child process has pid = 20168
[ 7046.675770] [program2] : This is the parent process, pid = 20167
[ 7046.675780] [program2] : child process
[ 7046.676120] [program2] : get SIGHUP signal
[ 7046.676121] [program2] : child process is hung up
[ 7046.676122] [program2] : The return signal is 1
[ 7061.638858] [program2] : Module_exit./my
```

```
[ 7092.210254] [program2] : Module_init ChenHuaxun 120090440
[ 7092.210258] [program2] : Module_init create kthread start
[ 7092.210420] [program2] : Module_init kthread start
[ 7092.211089] [program2] : The child process has pid = 20841
[ 7092.211089] [program2] : This is the parent process, pid = 20838
[ 7092.211198] [program2] : child process
[ 7092.291484] [program2] : get SIGILL signal
[ 7092.291486] [program2] : child process has illegal instruction
[ 7092.291486] [program2] : The return signal is 4
[ 7105.510662] [program2] : Module_exit./my
```

```
[ 7127.754966] [program2] : Module_init ChenHuaxun 120090440
[ 7127.754967] [program2] : Module_init create kthread start
[ 7127.755060] [program2] : Module_init kthread start
[ 7127.755131] [program2] : The child process has pid = 21511
[ 7127.755133] [program2] : This is the parent process, pid = 21510
[ 7127.755148] [program2] : child process
[ 7127.755536] [program2] : get SIGINT signal
[ 7127.755537] [program2] : child process gets interrupt from keyboard
[ 7127.755537] [program2] : The return signal is 2
[ 7137.470281] [program2] : Module_exit./my
```

```
[ 7160.738761] [program2] : Module_init ChenHuaxun 120090440
[ 7160.738762] [program2] : Module_init create kthread start
[ 7160.738895] [program2] : Module_init kthread start
[ 7160.738922] [program2] : The child process has pid = 22223
[ 7160.738923] [program2] : This is the parent process, pid = 22222
[ 7160.738994] [program2] : child process
[ 7160.739438] [program2] : get SIGKILL signal
[ 7160.739439] [program2] : child process is killed
[ 7160.739439] [program2] : The return signal is 9
[ 7169.559499] [program2] : Module_exit./my
```

```
[ 7186.366333] [program2] : Module_init ChenHuaxun 120090440
[ 7186.366350] [program2] : Module_init create kthread start
[ 7186.366480] [program2] : Module_init kthread start
[ 7186.366548] [program2] : The child process has pid = 22894
[ 7186.366548] [program2] : This is the parent process, pid = 22893
[ 7186.366583] [program2] : child process
[ 7186.367254] [program2] : get SIGFPIPE signal
[ 7186.367255] [program2] : child process writes on a pipe with no reader
[ 7186.367255] [program2] : The return signal is 13
[ 7194.585820] [program2] : Module_exit./my
```

```
[ 7212.770158] [program2] : Module_init ChenHuaxun 120090440
[ 7212.770159] [program2] : Module_init create kthread start
[ 7212.770240] [program2] : Module_init kthread start
[ 7212.770300] [program2] : The child process has pid = 23565
[ 7212.770301] [program2] : This is the parent process, pid = 23564
[ 7212.770306] [program2] : child process
[ 7212.858739] [program2] : get SIGQUIT signal
[ 7212.858741] [program2] : child process has terminal quit
[ 7212.858741] [program2] : The return signal is 3
[ 7222.118248] [program2] : Module_exit./my
```

```
[ 7249.030940] [program2] : Module_init ChenHuaxun 120090440
[ 7249.030941] [program2] : Module_init create kthread start
[ 7249.031126] [program2] : Module_init kthread start
[ 7249.031231] [program2] : The child process has pid = 24290
[ 7249.031232] [program2] : This is the parent process, pid = 24289
[ 7249.031245] [program2] : child process
[ 7249.107175] [program2] : get SIGSEGV signal
[ 7249.107176] [program2] : child process has invalid memory segment access
[ 7249.107177] [program2] : The return signal is 11
[ 7258.629739] [program2] : Module_exit./my
```

```
[ 7299.790663] [program2] : Module_init ChenHuaxun 120090440
[ 7299.790664] [program2] : Module_init create kthread start
[ 7299.790753] [program2] : Module_init kthread start
[ 7299.790776] [program2] : The child process has pid = 25056
[ 7299.790806] [program2] : This is the parent process, pid = 25055
[ 7299.790840] [program2] : child process
[ 7299.791240] [program2] : get SIGTERM signal
[ 7299.791241] [program2] : child process terminates
[ 7299.791242] [program2] : The return signal is 15
[ 7310.141929] [program2] : Module_exit./my
```

```
[ 7333.943786] [program2] : Module_init ChenHuaxun 120090440
[ 7333.943788] [program2] : Module_init create kthread start
[ 7333.943895] [program2] : Module_init kthread start
[ 7333.943954] [program2] : The child process has pid = 25728
[ 7333.943959] [program2] : This is the parent process, pid = 25727
[ 7333.943964] [program2] : child process
[ 7334.030779] [program2] : get SIGTRAP signal
[ 7334.030780] [program2] : child process reach a breakpoint
[ 7334.030781] [program2] : The return signal is 5
[ 7348.554517] [program2] : Module_exit./my
```

```
[ 6808.279951] [program2] : Module_init ChenHuaxun 120090440
[ 6808.279952] [program2] : Module_init create kthread start
[ 6808.280162] [program2] : Module_init kthread start
[ 6808.280220] [program2] : The child process has pid = 17503
[ 6808.280254] [program2] : This is the parent process, pid = 17502
[ 6808.280272] [program2] : child process
[ 6808.280323] [program2] : get SIGSTOP signal
[ 6808.280324] [program2] : child process stops
[ 6808.280324] [program2] : The return signal is 19
[ 6923.407060] [program2] : Module_exit./my
```

Bonus

```
vagrant@csc3150:/home/seed/work/proj1/source/bonus$ ./pstree
systemd-+-accounts-daemon---2*[{accounts-daemon}]
        |-acpid
        |-2*[agetty]
        |-atd
        |-cron
        |-dbus-daemon
        |-dhclient
        |-irqbalance
        |-2*[iscsid]
        |-lvmetad
        |-lxcfs---2*[{lxcfs}]
        |-mdadm
        |-polkitd---2*[{polkitd}]
        |-rsyslogd---3*[{rsyslogd}]
        |-sshd---sshd---sshd---bash-+-sh---node-+-node-+-bash---pstree
        |                           |           |      `-11*[{node}]
        |                           |           |-node---11*[{node}]
        |                           |           |-node---12*[{node}]
        |                           |           `-10*[{node}]
        |                           `-sleep
        |-systemd---(sd-pam)
        |-systemd-journal
        |-systemd-logind
        |-systemd-udevd
        `-unattended-upgr---{unattended-upgr}
```

```
root@csc3150:/home/seed/work/proj1/source/bonus# ./pstree -V
pstree (PSmisc) ChenHuaxun
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.
```

```
root@csc3150:/home/seed/work/proj1/source/bonus# ./pstree -n
systemd-+-systemd-journal
        |-lvmetad
        |-systemd-udevd
        |-dhclient
        |-2*[iscsid]
        |-systemd-logind
        |-atd
        |-cron
        |-lxcfs---2*[{lxcfs}]
        |-dbus-daemon
        |-rsyslogd---3*[{rsyslogd}]
        |-acpid
        |-accounts-daemon---2*[{accounts-daemon}]
        |-sshd---sshd---sshd---bash-+-sh---node-+-10*[{node}]
        |                           |           |-node-+-11*[{node}]
        |                           |           |      `-bash---sudo---su---bash---pstree
        |                           |           |-node---11*[{node}]
        |                           |           `-node---12*[{node}]
        |                           `-sleep
        |-unattended-upgr---{unattended-upgr}
        |-mdadm
        |-irqbalance
        |-polkitd---2*[{polkitd}]
        |-2*[agetty]
        `-systemd---(sd-pam)
```

What I learn?

How to use fork(): understand the return value of fork and its meanings. The relationship between child process and parent process.

How to use wait(): how to make parent process wait until child process finish executing.

Modify kernel source code and do task2: have a better understanding on how syscalls work. Enhance ability on reading source code and debugging. And understand the permission difference between user and kernel.

Clang-format: how to use clang-format to formulate my code.

pstree.c: memory allocation of processes. Better understanding of pointers and structs. Ability to get information through Google.