CSC3150 Assignment1 Report

Lai Wei, 120090485

Oct. 4th, 2022

program 1

Design

The design of this program is straight forward. At the beginning, we fork a child process. In the user mode, we use `fork()` function to for a child process.

In the child process, we use `execve()` to execute the program.

In the parent process, we use `waitpid()` to waiting for the child process to die. After child process die, we analyze the child process exit signals by a group of macro functions:

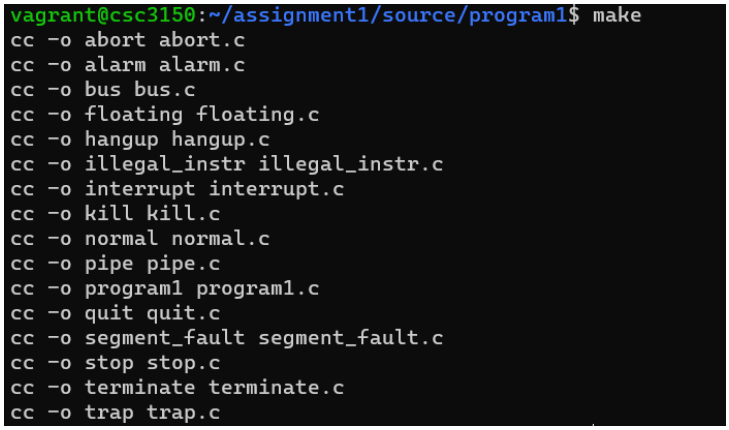
- first, `WIFEXITED()` to check if the process exited correctly.
- else, use `WIFSIGNALED()` to check if the process signaled. If so, use `WTERMSIG()` to get the signal code. Also, I write a function `formatSigName()` that use a big switch block to convert the signal code to human-readable signal name.
- else, use `WIFSTOPPED()` to check if the process stopped. If so, use `WSTOPSIG()` to get the signal code. Also, the function `formatSigName()` can use a big switch block to convert the signal code to human-readable signal name.

Setting-up

To compile the main program `program1.c` and its test programs,

```
cd source/program1
make
./program1 ./<test_program_name>
```

After `make`, you will see:



```
vagrant@csc3150:~/assignment1/source/program1$ make
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
```

All test cases screen shot are at the next section.

Sample output

All 15 test cases, `kill.c`, `interrupt.c`, `terminate.c`, `trap.c`, `abort.c`, `illegal_instr.c`, `stop.c`, `normal.c`, `bus.c`, `quit.c`, `pipe.c`, `segment_fault.c`, `alarm.c`, `floating.c`, `hangup.c` are tested, as shown below.

```
======1/15 [interrupt] starts ======
Process start to fork
I am the Parent Process, my pid = 3105
I am the Child Process, my pid = 3106
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGINT program

Parent process receives SIGCHLD signal
Parent process receives SIG-INTERRUPT (code: 2) signal
======1/15 [interrupt] finished ======
```

```
======2/15 [pipe] starts ======
Process start to fork
I am the Parent Process, my pid = 3108
I am the Child Process, my pid = 3109
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGPIPE program

------------CHILD PROCESS END------------
Parent process receives SIGCHLD signal
Normal termination with EXIT STATUS = 0
======2/15 [pipe] finished ======
```

```
======3/15 [kill] starts ======
Process start to fork
I am the Parent Process, my pid = 3111
I am the Child Process, my pid = 3112
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGKILL program

Parent process receives SIGCHLD signal
Parent process receives SIG-KILLED (code: 9) signal
======3/15 [kill] finished ======
```

```
======4/15 [stop] starts ======
Process start to fork
I am the Parent Process, my pid = 3114
I am the Child Process, my pid = 3115
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGSTOP program

Parent process receives SIGCHLD signal
Parent process receives SIG-STOPPED (SIGNAL) (code: 19) signal
======4/15 [stop] finished ======
```

```
======5/15 [trap] starts ======
Process start to fork
I am the Parent Process, my pid = 3117
I am the Child Process, my pid = 3118
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGTRAP program

Parent process receives SIGCHLD signal
Parent process receives SIG-TRACE/BREAKPOINT TRAP (code: 5) signal
======5/15 [trap] finished ======
```

```
======6/15 [abort] starts ======
Process start to fork
I am the Parent Process, my pid = 3121
I am the Child Process, my pid = 3122
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGABRT program

Parent process receives SIGCHLD signal
Parent process receives SIG-ABORTED (code: 6) signal
======6/15 [abort] finished ======
```

```
======7/15 [floating] starts ======
Process start to fork
I am the Parent Process, my pid = 3125
I am the Child Process, my pid = 3126
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGFPE program

Parent process receives SIGCHLD signal
Parent process receives SIG-FLOATING POINT EXCEPTION (code: 8) signal
======7/15 [floating] finished ======
```

```
======8/15 [segment_fault] starts ======
Process start to fork
I am the Parent Process, my pid = 3129
I am the Child Process, my pid = 3130
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGSEGV program

Parent process receives SIGCHLD signal
Parent process receives SIG-SEGMENTATION FAULT (code: 11) signal
======8/15 [segment_fault] finished ======
```

```
======9/15 [normal] starts ======
Process start to fork
I am the Parent Process, my pid = 3133
I am the Child Process, my pid = 3134
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
======9/15 [normal] finished ======
```

```
======10/15 [illegal_instr] starts ======
Process start to fork
I am the Parent Process, my pid = 3136
I am the Child Process, my pid = 3137
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGILL program

Parent process receives SIGCHLD signal
Parent process receives SIG-ILLEGAL INSTRUCTION (code: 4) signal
======10/15 [illegal_instr] finished ======
```

```
======11/15 [quit] starts ======
Process start to fork
I am the Parent Process, my pid = 3140
I am the Child Process, my pid = 3141
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGQUIT program

Parent process receives SIGCHLD signal
Parent process receives SIG-QUIT (code: 3) signal
======11/15 [quit] finished ======
```

```
======12/15 [hangup] starts ======
Process start to fork
I am the Parent Process, my pid = 3144
I am the Child Process, my pid = 3145
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGHUP program

Parent process receives SIGCHLD signal
Parent process receives SIG-HANGUP (code: 1) signal
======12/15 [hangup] finished ======
```

```
======13/15 [alarm] starts ======
Process start to fork
I am the Parent Process, my pid = 3147
I am the Child Process, my pid = 3148
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGALRM program

Parent process receives SIGCHLD signal
Parent process receives SIG-ALARM CLOCK (code: 14) signal
======13/15 [alarm] finished ======
```

```
======14/15 [bus] starts ======
Process start to fork
I am the Parent Process, my pid = 3150
I am the Child Process, my pid = 3151
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGBUS program

Parent process receives SIGCHLD signal
Parent process receives SIG-BUS ERROR (code: 7) signal
======14/15 [bus] finished ======
```

```
======15/15 [terminate] starts ======
Process start to fork
I am the Parent Process, my pid = 3154
I am the Child Process, my pid = 3155
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGTERM program

Parent process receives SIGCHLD signal
Parent process receives SIG-TERMINATED (code: 15) signal
======15/15 [terminate] finished ======
```

program 2

Design

The design of this program is straight forward. At the beginning, we fork a kernel thread by `kthread_create()`. In this new thread, we use `kernel_thread()` function to create a new child process.

In the child process, we use `do_execve()` to execute the program.

In the created kernel thread, we use `do_wait()` to wait for the child process die. After child process die, we analyze the child process exit signals by a group of functions. Since this is the kernel mode, we can't use the functions provided by the standard library directly. However, we can copy the macro functions and modified them to our own kernel mode functions. They are begun with `_`.

- first, `_WIFEXITED()` to check if the process exited correctly.
- else, use `_WIFSIGNALED()` to check if the process signaled. If so, use `_WTERMSIG()` to get the signal code. We don't have `sys_siglist[]` any more. So I implement a function called `_WSIG_TO_NAME` that can convert the status code to a readable string.
- else, use `_WIFSTOPPED()` to check if the process stopped. If so, use `_WSTOPSIG()` to get the signal code. we also use `_WSIG_TO_NAME` that can convert the status code to a readable string.

Setting-up

Modify and re-compile the kernel

(Assuming the kernel has been successfully installed before this step.)

To export some kernel functions so that we can use them in our code, we need to use `EXPORT_SYMBOL()` to export them from the local files.

For this task, I exported 5 functions. They are:

- `do_wait` from `kernel/exit.c`
- `do_execve` from `fs/exec.c`
- `getname_kernel` from `fs/namei.c`
- `kernel_clone` and `kernel_thread` from `kernel/fork.c`

After modifying the code, we recompile the kernel by the following commands, execute them one by one.

```
cd /path/of/your/kernel/source/code
make bzImage -j$(nproc)
make modules -j$(nproc)
make modules_install
make install
reboot
```

If everything is OK, the modified kernel has been compiled and installed.

Compile the code

Now we compile the code.

```
cd source/program2
make test
make
```

After two `make` instruction, you will see:

```
vagrant@csc3150:~/assignment1/source/program2$ make test
gcc test.c -o /tmp/test
vagrant@csc3150:~/assignment1/source/program2$ make
make -C /lib/modules/5.10.114/build M=/home/vagrant/assignment1/source/program2 modules
make[1]: Entering directory '/home/vagrant/kernel_test/linux-5.10.114'
  CC [M]  /home/vagrant/assignment1/source/program2/program2.o
  MODPOST /home/vagrant/assignment1/source/program2/Module.symvers
  CC [M]  /home/vagrant/assignment1/source/program2/program2.mod.o
  LD [M]  /home/vagrant/assignment1/source/program2/program2.ko
make[1]: Leaving directory '/home/vagrant/kernel_test/linux-5.10.114'
```

Then to run the code, type:

```
sudo insmod program2.ko  # insert the module
sudo rmmod program2      # remove the module
dmesg                    # get the info
```

All test cases output screen shot are at the next section.

Sample output

All 15 test programs, `kill.c`, `interrupt.c`, `terminate.c`, `trap.c`, `abort.c`, `illegal_instr.c`, `stop.c`, `normal.c`, `bus.c`, `quit.c`, `pipe.c`, `segment_fault.c`, `alarm.c`, `floating.c`, `hangup.c` are tested, as shown below.

```
======1/15 [kill] starts ======
[ 4218.012835] [program2] : Module_init {Lai Wei} {120090485}
[ 4218.012836] [program2] : module_init create kthread start
[ 4218.012901] [program2] : module_init kthread start
[ 4218.012913] [program2] : The child process has pid = 12546
[ 4218.012913] [program2] : This is the parent process, pid = 12545
[ 4218.012929] [program2] : child process
[ 4218.013219] [program2] : child process terminated.
[ 4218.013220] [program2] : get SIGKILL signal
[ 4218.013221] [program2] : The return signal is 9
[ 4220.523140] [program2] : Module_exit
======1/15 [kill] finished ======
```

```
======2/15 [interrupt] starts ======
[ 4220.574563] [program2] : Module_init {Lai Wei} {120090485}
[ 4220.574564] [program2] : module_init create kthread start
[ 4220.575006] [program2] : module_init kthread start
[ 4220.575018] [program2] : The child process has pid = 12566
[ 4220.575019] [program2] : This is the parent process, pid = 12564
[ 4220.575020] [program2] : child process
[ 4220.575256] [program2] : child process terminated.
[ 4220.575257] [program2] : get SIGINT signal
[ 4220.575258] [program2] : The return signal is 2
[ 4223.079536] [program2] : Module_exit
======2/15 [interrupt] finished ======
```

```
======3/15 [terminate] starts ======
[ 4223.133553] [program2] : Module_init {Lai Wei} {120090485}
[ 4223.133554] [program2] : module_init create kthread start
[ 4223.134090] [program2] : module_init kthread start
[ 4223.134103] [program2] : The child process has pid = 12585
[ 4223.134104] [program2] : This is the parent process, pid = 12583
[ 4223.134105] [program2] : child process
[ 4223.134375] [program2] : child process terminated.
[ 4223.134376] [program2] : get SIGTERM signal
[ 4223.134376] [program2] : The return signal is 15
[ 4225.641021] [program2] : Module_exit
======3/15 [terminate] finished ======
```

```
======4/15 [trap] starts ======
[ 4225.696441] [program2] : Module_init {Lai Wei} {120090485}
[ 4225.696442] [program2] : module_init create kthread start
[ 4225.696961] [program2] : module_init kthread start
[ 4225.696981] [program2] : The child process has pid = 12604
[ 4225.696981] [program2] : This is the parent process, pid = 12602
[ 4225.696983] [program2] : child process
[ 4225.771176] [program2] : child process terminated.
[ 4225.771178] [program2] : get SIGTRAP signal
[ 4225.771179] [program2] : The return signal is 5
[ 4228.231331] [program2] : Module_exit
======4/15 [trap] finished ======
```

```
======5/15 [abort] starts ======
[ 4228.284440] [program2] : Module_init {Lai Wei} {120090485}
[ 4228.284442] [program2] : module_init create kthread start
[ 4228.284710] [program2] : module_init kthread start
[ 4228.284726] [program2] : The child process has pid = 12623
[ 4228.284726] [program2] : This is the parent process, pid = 12622
[ 4228.284858] [program2] : child process
[ 4228.357537] [program2] : child process terminated.
[ 4228.357539] [program2] : get SIGABRT signal
[ 4228.357540] [program2] : The return signal is 6
[ 4230.847655] [program2] : Module_exit
======5/15 [abort] finished ======
```

```
======6/15 [illegal_instr] starts ======
[ 4230.902755] [program2] : Module_init {Lai Wei} {120090485}
[ 4230.902756] [program2] : module_init create kthread start
[ 4230.902921] [program2] : module_init kthread start
[ 4230.902945] [program2] : The child process has pid = 12643
[ 4230.902945] [program2] : This is the parent process, pid = 12642
[ 4230.902958] [program2] : child process
[ 4230.975907] [program2] : child process terminated.
[ 4230.975909] [program2] : get SIGILL signal
[ 4230.975910] [program2] : The return signal is 4
[ 4233.408587] [program2] : Module_exit
======6/15 [illegal_instr] finished ======
```

```
======7/15 [stop] starts ======
[ 4233.455787] [program2] : Module_init {Lai Wei} {120090485}
[ 4233.455788] [program2] : module_init create kthread start
[ 4233.456002] [program2] : module_init kthread start
[ 4233.456016] [program2] : The child process has pid = 12663
[ 4233.456016] [program2] : This is the parent process, pid = 12662
[ 4233.456018] [program2] : child process
[ 4233.456250] [program2] : child process terminated.
[ 4233.456251] [program2] : get SIGSTOP signal
[ 4233.456252] [program2] : The return signal is 19
[ 4236.002739] [program2] : Module_exit
======7/15 [stop] finished ======
```

```
======8/15 [normal] starts ======
[ 4236.002739] [program2] : Module_exit
[ 4236.049760] [program2] : Module_init {Lai Wei} {120090485}
[ 4236.049766] [program2] : module_init create kthread start
[ 4236.050051] [program2] : module_init kthread start
[ 4236.050065] [program2] : The child process has pid = 12682
[ 4236.050066] [program2] : This is the parent process, pid = 12681
[ 4236.050067] [program2] : child process
[ 4236.050363] [program2] : child process terminated.
[ 4236.050364] [program2] : Normal termination with EXIT STATUS = 0
[ 4238.560222] [program2] : Module_exit
======8/15 [normal] finished ======
```

```
======9/15 [bus] starts ======
[ 4238.608779] [program2] : Module_init {Lai Wei} {120090485}
[ 4238.608781] [program2] : module_init create kthread start
[ 4238.609301] [program2] : module_init kthread start
[ 4238.609311] [program2] : The child process has pid = 12703
[ 4238.609311] [program2] : This is the parent process, pid = 12701
[ 4238.609312] [program2] : child process
[ 4238.681805] [program2] : child process terminated.
[ 4238.681807] [program2] : get SIGBUS signal
[ 4238.681808] [program2] : The return signal is 7
[ 4241.117846] [program2] : Module_exit
======9/15 [bus] finished ======
```

```
======10/15 [quit] starts ======
[ 4241.169374] [program2] : Module_init {Lai Wei} {120090485}
[ 4241.169376] [program2] : module_init create kthread start
[ 4241.169606] [program2] : module_init kthread start
[ 4241.169620] [program2] : The child process has pid = 12722
[ 4241.169620] [program2] : This is the parent process, pid = 12721
[ 4241.169622] [program2] : child process
[ 4241.241705] [program2] : child process terminated.
[ 4241.241707] [program2] : get SIGQUIT signal
[ 4241.241708] [program2] : The return signal is 3
[ 4243.684900] [program2] : Module_exit
======10/15 [quit] finished ======
```

```
======11/15 [pipe] starts ======
[ 4243.736078] [program2] : Module_init {Lai Wei} {120090485}
[ 4243.736080] [program2] : module_init create kthread start
[ 4243.736320] [program2] : module_init kthread start
[ 4243.736337] [program2] : The child process has pid = 12742
[ 4243.736338] [program2] : This is the parent process, pid = 12741
[ 4243.736394] [program2] : child process
[ 4243.737195] [program2] : child process terminated.
[ 4243.737196] [program2] : get SIGPIPE signal
[ 4243.737196] [program2] : The return signal is 13
[ 4246.244169] [program2] : Module_exit
======11/15 [pipe] finished ======
```

```
======12/15 [segment_fault] starts ======
[ 4246.290907] [program2] : Module_init {Lai Wei} {120090485}
[ 4246.290909] [program2] : module_init create kthread start
[ 4246.291080] [program2] : module_init kthread start
[ 4246.291093] [program2] : The child process has pid = 12761
[ 4246.291094] [program2] : This is the parent process, pid = 12760
[ 4246.291095] [program2] : child process
[ 4246.372364] [program2] : child process terminated.
[ 4246.372366] [program2] : get SIGSEGV signal
[ 4246.372367] [program2] : The return signal is 11
[ 4248.800431] [program2] : Module_exit
======12/15 [segment_fault] finished ======
```

```
======13/15 [alarm] starts ======
[ 4248.847498] [program2] : Module_init {Lai Wei} {120090485}
[ 4248.847499] [program2] : module_init create kthread start
[ 4248.847693] [program2] : module_init kthread start
[ 4248.848977] [program2] : The child process has pid = 12782
[ 4248.848978] [program2] : This is the parent process, pid = 12780
[ 4248.849022] [program2] : child process
[ 4250.855383] [program2] : child process terminated.
[ 4250.855385] [program2] : get SIGALRM signal
[ 4250.855386] [program2] : The return signal is 14
[ 4251.356078] [program2] : Module_exit
======13/15 [alarm] finished ======
```

```
======14/15 [floating] starts ======
[ 4251.408104] [program2] : Module_init {Lai Wei} {120090485}
[ 4251.408105] [program2] : module_init create kthread start
[ 4251.408278] [program2] : module_init kthread start
[ 4251.408293] [program2] : The child process has pid = 12800
[ 4251.408294] [program2] : This is the parent process, pid = 12799
[ 4251.408348] [program2] : child process
[ 4251.481566] [program2] : child process terminated.
[ 4251.481568] [program2] : get SIGFPE signal
[ 4251.481569] [program2] : The return signal is 8
[ 4253.914970] [program2] : Module_exit
======14/15 [floating] finished ======
```

```
======15/15 [hangup] starts ======
[ 4253.968465] [program2] : Module_init {Lai Wei} {120090485}
[ 4253.968467] [program2] : module_init create kthread start
[ 4253.968694] [program2] : module_init kthread start
[ 4253.968711] [program2] : The child process has pid = 12820
[ 4253.968712] [program2] : This is the parent process, pid = 12819
[ 4253.968714] [program2] : child process
[ 4253.969326] [program2] : child process terminated.
[ 4253.969327] [program2] : get SIGHUP signal
[ 4253.969328] [program2] : The return signal is 1
[ 4256.475746] [program2] : Module_exit
======15/15 [hangup] finished ======
```

bonus

Design

The design follows a straight-forward rule:

1. Read the process information in the `/proc/[pid]` directory. Each `pid` process is in that corresponding directory. We can read the `status` file to get the process name, it's pid, and it's parent pid. It's worth-noting that some process even have some child process, namely, tid. They are located in the `/proc/[pid]/[tid]` directory. We also read the status file to get the pid and process name.
2. We first use a linked list to store the processes info. Also this node has two more pointers to be acted like a tree. Each node represents a process as follows:

```
struct ps_node{
    /*process info field*/

    struct ps_node* list_next;
    struct ps_node* child_head;
    struct ps_node* child_next;
}
```

3. After reading all process, we begin to create a tree. we go through from the head of the linked list, iterate one by one by the `ps_node.list_next`. We find the current node's parent node by current node's ppid value. After find it, assign the current node to the parent's node's child linked list (This linked list is maintained by `child_head` and `child_next`).
4. To display the tree, we take `pid=1` node as the starting node. In each layer of recursion, we:
    1. display the process information
    2. iterate its child linked list. If that child has children, call `display` function recursively.

Set-up

To compile and run the code, type following commands:

```
cd source/bonus
make
./pstree
```

You will see part of the output like this:

```
vagrant@csc3150:~/assignment1/source/bonus$ make
gcc pstree.c -o pstree
vagrant@csc3150:~/assignment1/source/bonus$ ./pstree
-systemd
   |-systemd-journal
   |-lvmetad
   |-systemd-udevd
   |-dhclient
   |-acpid
   |-dbus-daemon
   |-lxcfs
   |    `-2*[{lxcfs}]
   |-cron
   |-atd
   |-systemd-logind
```

Supported options and Sample output

Additional NOTE on 10/10: I received the test program for this task on the morning of 10/10, which is the last day for submission. I found that my work doesn't match the function supposed in the tester exactly. Then I decided to change it a little bit. **Now it can pass the test program. The only difference between mine and the real pstree is that the real pstree has a better formatted output**, which I have the constant indentation. The screen shot below are from my old program. Now what I submitted is the new program.

We support five options. They are:

- `--help`: Print out the help doc. (NOTE: in default Linux's `pstree`, there is no option called `--help`. But, when you type some mismatching options, the help doc would pop out. So I think this count as one.)

```
vagrant@csc3150:~/assignment1/source/bonus$ ./pstree --help
Future's pstree: welcome to the helper doc!
Usage: pstree [ -c ] [ -V ] [ -p ] [ --help ] [ PID ]
Display a tree of processes.

    -c, --compact     don't compact identical subtrees
    -p, --show-pids   show PIDs; implies -c
    -V, --version     display version information
    --help            display helper doc
    PID               start at this PID; default is 1 (init)
```

- `-c, --compact`: Extract the same name process

```
vagrant@csc3150:~/assignment1/source/bonus$ ./pstree -c
-systemd
   |-systemd-journal
   |-lvmetad
   |-systemd-udevd
   |-dhclient
   |-acpid
   |-dbus-daemon
   |-lxcfs
   |    |-lxcfs
   |    |-lxcfs
   |    `-lxcfs
   |-cron
   |-atd
   |-systemd-logind
   |-accounts-daemon
   |    |-gmain
   |    `-gdbus
   |-rsyslogd
   |    |-in
   |    |-in
   |    `-rs
```

- `-p, --show-pids`: Show pid of each process, in the extracted form.

```
vagrant@csc3150:~/assignment1/source/bonus$ ./pstree -p
-systemd (1)
   |-systemd-journal (374)
   |-lvmetad (396)
   |-systemd-udevd (399)
   |-dhclient (783)
   |-acpid (962)
   |-dbus-daemon (965)
   |-lxcfs (966)
   |    |-lxcfs (975)
   |    |-lxcfs (976)
   |    `-lxcfs (19234)
   |-cron (984)
   |-atd (987)
   |-systemd-logind (988)
   |-accounts-daemon (989)
   |    |-gmain (997)
   |    `-gdbus (1008)
   |-rsyslogd (990)
   |    |-in (1015)
   |    |-in (1016)
   |    `-rs (1017)
   |-unattended-upgr (998)
   |    `-gmain (1100)
   |-sshd (1004)
   |    |-sshd (1879)
   |    |    `-sshd (1918)
```

- `-V, --version`: Print our the version

```
vagrant@csc3150:~/assignment1/source/bonus$ ./pstree -V
pstree (By Future) 1.0 Copyright (C) 2022 Lai.

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.
```

- `PID`: Display the tree at the given `PID` node.

```
vagrant@csc3150:~/assignment1/source/bonus$ ./pstree 1974
-node
   |-10*[{node}]
   |-node
   |    |-12*[{node}]
   |    `-2*[bash]
   |-node
   |    |-11*[{node}]
   |    |-node
   |    |    `-6*[{node}]
   |    |-cpptools
   |    |    `-22*[{cpptools}]
   |    `-node
   |         `-7*[{node}]
   `-node
        `-12*[{node}]
```

- Multiple options: Passing `PID` and `-p` at the same time.

```
vagrant@csc3150:~/assignment1/source/bonus$ ./pstree 1974 -p
-node (1974)
   |-node (1981)
   |-node (1982)
   |-node (1983)
   |-node (1984)
   |-node (1985)
   |-node (1992)
   |-node (2024)
   |-node (2025)
   |-node (2026)
   |-node (2027)
   |-node (2030)
   |    |-node (2031)
   |    |-node (2032)
   |    |-node (2033)
   |    |-node (2034)
   |    |-node (2035)
   |    |-node (2039)
   |    |-node (2047)
   |    |-node (2048)
   |    |-node (2049)
   |    |-node (2050)
   |    |-node (17042)
   |    |-node (17124)
```

- Default: (incomplete screenshot).

```
vagrant@csc3150:~/assignment1/source/bonus$ ./pstree
-systemd
   |-systemd-journal
   |-lvmetad
   |-systemd-udevd
   |-dhclient
   |-accounts-daemon
   |    |-gmain
   |    `-gdbus
   |-2*[iscsid]
   |-lxcfs
   |    `-2*[{lxcfs}]
   |-rsyslogd
   |    |-2*[{in}]
   |    `-rs
   |-acpid
   |-cron
   |-dbus-daemon
   |-atd
   |-systemd-logind
   |-sshd
   |    |-sshd
   |    |    `-sshd
   |    |        `-bash
   |    |            `-sleep
   |    |-sshd
   |    |    `-sshd
   |    |        `-bash
   |    |            `-sleep
   |    `-sshd
   |        `-sshd
   |            `-bash
   |                `-pstree
   |-unattended-upgr
   |    `-gmain
   |-polkitd
   |    |-gmain
   |    `-gdbus
   |-mdadm
```

Things learnt

1. It's my first time get touch with the Linux source code, understanding what "codes to form an operation system" would be like. (That's really huge!) The comments says that Mr. Linus wrote this code in 1991. I can't believe that he can do it with poor computer and display at that time, writing a well-structured operating system.
2. No standard library when writing the OS. Things get more complicated...
3. Understand how to fork process in both user mode in C and in the kernel mode. There're some difference.
4. Know how to compile the Linux kernel, and even modify the kernel.
5. Debug with few knowledge online. By reading the source codes and discussing with classmates.
6. Know how to view process info in Linux. Practice how the data structure and string format problem.