
Report for Assignment I

CSC-3150

OCTOBER 10, 2022

QI XIXIAN

120090691

1 Program 1

1.1 Ideas and Implementation

In this task, the program is executed under the user mode. First, the *program1.c* will fork a child process to execute a test program and wait for its signal in the parent process. After receiving the terminated (SIGCHLD) signal in the child process, the parent process will print out the termination information. In practice, I first used the `fork()` function to fork a child process, and determined which process it is using the return `pid_t` value:

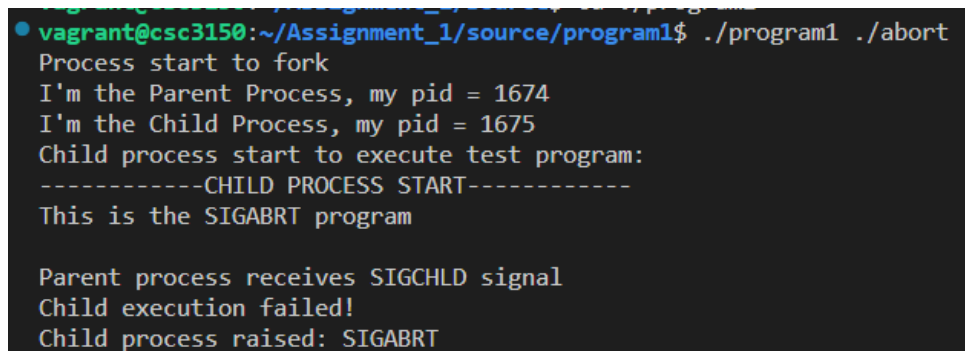
- -1: Error
- 0: Child process
- Others(> 0): Parent process

Within the child process, I used `execve()` function to replace and execute `./test` test. Then, I used `waitpid()` function to wait for the termination in the child process, while receiving the returned signal using `status` variable. Finally, after using macro-functions to transform, I use a switch-case to link it with all possible signals and output the information.

1.2 Environment Deploy

1. Goto the workspace `~/Assignment_1/source/program1` using `cd` command.
2. Compile all the c-program under the directory by typing `make` command.
3. Run the program with its argument a test file: `./program1 ./abort`
4. Replace the test file and repeat process 3.

1.3 Screenshots



```
● vagrant@csc3150:~/Assignment_1/source/program1$ ./program1 ./abort
Process start to fork
I'm the Parent Process, my pid = 1674
I'm the Child Process, my pid = 1675
Child process start to execute test program:
-----CHILD PROCESS START-----
This is the SIGABRT program

Parent process receives SIGCHLD signal
Child execution failed!
Child process raised: SIGABRT
```

Figure 1: Abort

```

● vagrant@csc3150:~/Assignment_1/source/program1$ ./program1 ./alarm
Process start to fork
I'm the Parent Process, my pid = 1704
I'm the Child Process, my pid = 1705
Child process start to execute test program:
-----CHILD PROCESS START-----
This is the SIGALRM program

Parent process receives SIGCHLD signal
Child execution failed!
Child process raised: SIGALRM

```

Figure 2: Alarm

```

● vagrant@csc3150:~/Assignment_1/source/program1$ ./program1 ./bus
Process start to fork
I'm the Parent Process, my pid = 1758
I'm the Child Process, my pid = 1759
Child process start to execute test program:
-----CHILD PROCESS START-----
This is the SIGBUS program

Parent process receives SIGCHLD signal
Child execution failed!
Child process raised: SIGBUS

```

Figure 3: Bus

```

● vagrant@csc3150:~/Assignment_1/source/program1$ ./program1 ./floating
Process start to fork
I'm the Parent Process, my pid = 1785
I'm the Child Process, my pid = 1786
Child process start to execute test program:
-----CHILD PROCESS START-----
This is the SIGFPE program

Parent process receives SIGCHLD signal
Child execution failed!
Child process raised: SIGFPE

```

Figure 4: Floating

```

● vagrant@csc3150:~/Assignment_1/source/program1$ ./program1 ./hangup
Process start to fork
I'm the Parent Process, my pid = 1812
I'm the Child Process, my pid = 1813
Child process start to execute test program:
-----CHILD PROCESS START-----
This is the SIGHUP program

Parent process receives SIGCHLD signal
Child execution failed!
Child process raised: SIGHUP

```

Figure 5: Hangup

```

● vagrant@csc3150:~/Assignment_1/source/program1$ ./program1 ./illegal_instr
Process start to fork
I'm the Child Process, my pid = 1876
I'm the Parent Process, my pid = 1875
Child process start to execute test program:
-----CHILD PROCESS START-----
This is the SIGILL program

Parent process receives SIGCHLD signal
Child execution failed!
Child process raised: SIGILL

```

Figure 6: Illegal_instr

```

● vagrant@csc3150:~/Assignment_1/source/program1$ ./program1 ./interrupt
Process start to fork
I'm the Parent Process, my pid = 1938
I'm the Child Process, my pid = 1939
Child process start to execute test program:
-----CHILD PROCESS START-----
This is the SIGINT program

Parent process receives SIGCHLD signal
Child execution failed!
Child process raised: SIGINT

```

Figure 7: Interrupt

```

● vagrant@csc3150:~/Assignment_1/source/program1$ ./program1 ./kill
Process start to fork
I'm the Parent Process, my pid = 1952
I'm the Child Process, my pid = 1953
Child process start to execute test program:
-----CHILD PROCESS START-----
This is the SIGKILL program

Parent process receives SIGCHLD signal
Child execution failed!
Child process raised: SIGKILL

```

Figure 8: Kill

```

● vagrant@csc3150:~/Assignment_1/source/program1$ ./program1 ./normal
Process start to fork
I'm the Parent Process, my pid = 1978
I'm the Child Process, my pid = 1979
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.

```

Figure 9: Normal

```

● vagrant@csc3150:~/Assignment_1/source/program1$ ./program1 ./pipe
Process start to fork
I'm the Parent Process, my pid = 1992
I'm the Child Process, my pid = 1993
Child process start to execute test program:
-----CHILD PROCESS START-----
This is the SIGPIPE program

Parent process receives SIGCHLD signal
Child execution failed!
Child process raised: SIGPIPE

```

Figure 10: Pipe

```

● vagrant@csc3150:~/Assignment_1/source/program1$ ./program1 ./quit
Process start to fork
I'm the Parent Process, my pid = 2036
I'm the Child Process, my pid = 2037
Child process start to execute test program:
-----CHILD PROCESS START-----
This is the SIGQUIT program

Parent process receives SIGCHLD signal
Child execution failed!
Child process raised: SIGQUIT

```

Figure 11: Quit

```

● vagrant@csc3150:~/Assignment_1/source/program1$ ./program1 ./segment_fault
Process start to fork
I'm the Parent Process, my pid = 2063
I'm the Child Process, my pid = 2064
Child process start to execute test program:
-----CHILD PROCESS START-----
This is the SIGSEGV program

Parent process receives SIGCHLD signal
Child execution failed!
Child process raised: SIGSEGV

```

Figure 12: Segment_fault

```

● vagrant@csc3150:~/Assignment_1/source/program1$ ./program1 ./stop
Process start to fork
I'm the Parent Process, my pid = 2114
I'm the Child Process, my pid = 2115
Child process start to execute test program:
-----CHILD PROCESS START-----
This is the SIGSTOP program

Parent process receives SIGCHLD signal
Child execution stopped
Child process raised: SIGSTOP

```

Figure 13: Stop

```

● vagrant@csc3150:~/Assignment_1/source/program1$ ./program1 ./terminate
Process start to fork
I'm the Parent Process, my pid = 2149
I'm the Child Process, my pid = 2150
Child process start to execute test program:
-----CHILD PROCESS START-----
This is the SIGTERM program

Parent process receives SIGCHLD signal
Child execution failed!
Child process raised: SIGTERM

```

Figure 14: Terminate

```

● vagrant@csc3150:~/Assignment_1/source/program1$ ./program1 ./trap
Process start to fork
I'm the Parent Process, my pid = 2179
I'm the Child Process, my pid = 2180
Child process start to execute test program:
-----CHILD PROCESS START-----
This is the SIGTRAP program

Parent process receives SIGCHLD signal
Child execution failed!
Child process raised: SIGTRAP

```

Figure 15: Trap

1.4 Knowledge Acquired

In this task, I acquired some basic knowledge about how to fork and execute another process under the user mode, which is really useful in programming. Also, I learnt the mechanism of signals in Unix systems.

2 Program 2

2.1 Ideas and Implementation

The works to do in this task is quite similar to the previous one, except in kernel mode. Therefore, we need to implement user-mode functions from the bottom up. First, a thread is created to execute `my_fork()`. Within the function, a process is forked to execute the test program (with strict path `/tmp/test`). After child process termination, `my_wait()` function embedded in the parent process receives a signal and alters the `wo_stat` member variable representing the signal. After dealt properly, information is printed out in kernel log.

The uniqueness of this task lies in the implementation of kernel-mode and self-defined

functions:

- **kthread_create()** & **wake_up_process()**
These two functions create a new thread under the kernel and wake it up, respectively. The `int (*threadfn)(void *data)` should be set as the address of `my_fork()` to run such function in the new thread.
- **my_fuct()**
This function forks the child process in `kernel_clone()` and waits to receive a signal in `my_wait()`. Also, it outputs the PID of parent and child process, as well as related signals after transformed in `sig_conv_out()`
- **kernel_clone()** This function forks a child process in `my_exec()`, so the argument `.slack` should be set as the address of that function. Also, argument `.exit_signal` should be set as 'SIGCHLD' to inform the parent process it is terminated.
- **my_exec()**
This function is exclusive to the child process. First, it gets the filename using `getname_kernel()`. Then, it executes the test program by invoking `do_execve()`.
- **sig_conv_out()**
This function first use macros (bit operations) to transform the raw status to the valid signal code. The macros are found in the header-file in of the user-mode, so I define them myself. Then, a switch-case structure is used to output related signal based on status.

2.2 Environment Deploy

1. Use **EXPORT_SYMBOL** to export all functions needed in the kernel source, and recompile it by typing `make -j$(nproc)`.
2. Go to the workspace `~/Assignment_1/source/program2` using `cd` command.
3. Compile 'program2.c' by typing `make` command. This creates multiple new files related to kernel module.
4. Compile the test file by typing `gcc -o test test.c`
5. Install the module using `insmod` command.
6. `dmesg` command can be used to display the message the program printed out.
7. use `rmmmod` command to remove the module.
8. Replace the test file and repeat process 5.

2.3 Screenshots

```
[ 9834.945653] [program2] : Module_init QiXixian 120090691
[ 9834.945654] [program2] : Module_init create kthread start
[ 9834.945919] [program2] : Module_init kthread start
[ 9834.946194] [program2] : The child process has pid = 3463
[ 9834.946195] [program2] : This is the parent process, pid = 3462
[ 9834.946196] [program2] : Child process
[ 9835.051818] [program2] : get SIGABRT signal
[ 9835.051820] [program2] : Child process terminated
[ 9835.051821] [program2] : The return signal is 6
[ 9857.564561] [program2] : Module_exit
```

Figure 16: Abort

```
[ 9881.220282] [program2] : Module_init QiXixian 120090691
[ 9881.220283] [program2] : Module_init create kthread start
[ 9881.220718] [program2] : Module_init kthread start
[ 9881.221239] [program2] : The child process has pid = 3632
[ 9881.221240] [program2] : This is the parent process, pid = 3630
[ 9881.221240] [program2] : Child process
[ 9883.222201] [program2] : get SIGALRM signal
[ 9883.222220] [program2] : Child process terminated
[ 9883.222221] [program2] : The return signal is 14
[ 9884.837438] [program2] : Module_exit
```

Figure 17: Alarm

```
[ 9899.159621] [program2] : Module_init QiXixian 120090691
[ 9899.159623] [program2] : Module_init create kthread start
[ 9899.159970] [program2] : Module_init kthread start
[ 9899.160509] [program2] : The child process has pid = 3714
[ 9899.160510] [program2] : This is the parent process, pid = 3713
[ 9899.160511] [program2] : Child process
[ 9899.246912] [program2] : get SIGBUS signal
[ 9899.246913] [program2] : Child process terminated
[ 9899.246913] [program2] : The return signal is 7
[ 9901.915282] [program2] : Module_exit
```

Figure 18: Bus

```
[ 9913.763883] [program2] : Module_init QiXixian 120090691
[ 9913.763884] [program2] : Module_init create kthread start
[ 9913.764158] [program2] : Module_init kthread start
[ 9913.764247] [program2] : The child process has pid = 3780
[ 9913.764248] [program2] : This is the parent process, pid = 3779
[ 9913.764248] [program2] : Child process
[ 9913.853451] [program2] : get SIGFPE signal
[ 9913.853452] [program2] : Child process terminated
[ 9913.853453] [program2] : The return signal is 8
[ 9918.871799] [program2] : Module_exit
```

Figure 19: Floating


```

[ 9931.141037] [program2] : Module_init QiXixian 120090691
[ 9931.141039] [program2] : Module_init create kthread start
[ 9931.141335] [program2] : Module_init kthread start
[ 9931.142039] [program2] : The child process has pid = 3865
[ 9931.142040] [program2] : This is the parent process, pid = 3862
[ 9931.142040] [program2] : Child process
[ 9931.142583] [program2] : get SIGHUP signal
[ 9931.142584] [program2] : Child process terminated
[ 9931.142585] [program2] : The return signal is 1
[ 9935.824376] [program2] : Module_exit

```

Figure 20: Hangup

```

[10150.396759] [program2] : Module_init QiXixian 120090691
[10150.396760] [program2] : Module_init create kthread start
[10150.397069] [program2] : Module_init kthread start
[10150.399544] [program2] : The child process has pid = 4105
[10150.399545] [program2] : This is the parent process, pid = 4104
[10150.399546] [program2] : Child process
[10150.490637] [program2] : get SIGILL signal
[10150.490639] [program2] : Child process terminated
[10150.490639] [program2] : The return signal is 4
[10155.030149] [program2] : Module_exit

```

Figure 21: Illegal_instr

```

[10164.733066] [program2] : Module_init QiXixian 120090691
[10164.733068] [program2] : Module_init create kthread start
[10164.733465] [program2] : Module_init kthread start
[10164.735190] [program2] : The child process has pid = 4166
[10164.735192] [program2] : This is the parent process, pid = 4164
[10164.735192] [program2] : Child process
[10164.736218] [program2] : get SIGINT signal
[10164.736218] [program2] : Child process terminated
[10164.736219] [program2] : The return signal is 2
[10167.629299] [program2] : Module_exit

```

Figure 22: Interrupt

```

[10175.300673] [program2] : Module_init QiXixian 120090691
[10175.300674] [program2] : Module_init create kthread start
[10175.300958] [program2] : Module_init kthread start
[10175.301147] [program2] : The child process has pid = 4218
[10175.301148] [program2] : This is the parent process, pid = 4217
[10175.301167] [program2] : Child process
[10175.302176] [program2] : get SIGKILL signal
[10175.302177] [program2] : Child process terminated
[10175.302177] [program2] : The return signal is 9
[10178.509896] [program2] : Module_exit

```

Figure 23: Kill

```

[10190.470055] [program2] : Module_init QiXixian 120090691
[10190.470057] [program2] : Module_init create kthread start
[10190.470469] [program2] : Module_init kthread start
[10190.470709] [program2] : The child process has pid = 4276
[10190.470710] [program2] : This is the parent process, pid = 4275
[10190.470710] [program2] : Child process
[10190.471382] [program2] : Normal termination with EXIT STATUS = 0.
[10190.471382] [program2] : Child process terminated
[10190.471383] [program2] : The return signal is 0
[10195.082203] [program2] : Module_exit

```

Figure 24: Normal

```

[10204.146775] [program2] : Module_init QiXixian 120090691
[10204.146776] [program2] : Module_init create kthread start
[10204.147147] [program2] : Module_init kthread start
[10204.147321] [program2] : The child process has pid = 4331
[10204.147322] [program2] : This is the parent process, pid = 4330
[10204.147322] [program2] : Child process
[10204.148865] [program2] : get SIGPIPE signal
[10204.148865] [program2] : Child process terminated
[10204.148866] [program2] : The return signal is 13
[10208.123925] [program2] : Module_exit

```

Figure 25: Pipe

```

[10400.285110] [program2] : Module_init QiXixian 120090691
[10400.285112] [program2] : Module_init create kthread start
[10400.285354] [program2] : Module_init kthread start
[10400.286768] [program2] : The child process has pid = 4541
[10400.286769] [program2] : This is the parent process, pid = 4540
[10400.286769] [program2] : Child process
[10400.374881] [program2] : get SIGQUIT signal
[10400.374882] [program2] : Child process terminated
[10400.374882] [program2] : The return signal is 3
[10403.611038] [program2] : Module_exit

```

Figure 26: Quit

```

[10411.662541] [program2] : Module_init QiXixian 120090691
[10411.662543] [program2] : Module_init create kthread start
[10411.662944] [program2] : Module_init kthread start
[10411.664809] [program2] : The child process has pid = 4609
[10411.664810] [program2] : This is the parent process, pid = 4608
[10411.664810] [program2] : Child process
[10411.751380] [program2] : get SIGSEGV signal
[10411.751382] [program2] : Child process terminated
[10411.751382] [program2] : The return signal is 11
[10413.787558] [program2] : Module_exit

```

Figure 27: Segment_fault

```

[10422.460813] [program2] : Module_init QiXixian 120090691
[10422.460815] [program2] : Module_init create kthread start
[10422.461104] [program2] : Module_init kthread start
[10422.461479] [program2] : The child process has pid = 4665
[10422.461480] [program2] : This is the parent process, pid = 4664
[10422.461481] [program2] : Child process
[10422.462132] [program2] : get SIGSTOP signal
[10422.462133] [program2] : Child process terminated
[10422.462133] [program2] : The return signal is 19
[10426.122970] [program2] : Module_exit

```

Figure 28: Stop

```

[10435.195721] [program2] : Module_init QiXixian 120090691
[10435.195723] [program2] : Module_init create kthread start
[10435.196134] [program2] : Module_init kthread start
[10435.196306] [program2] : The child process has pid = 4717
[10435.196307] [program2] : This is the parent process, pid = 4716
[10435.196307] [program2] : Child process
[10435.197148] [program2] : get SIGTERM signal
[10435.197149] [program2] : Child process terminated
[10435.197150] [program2] : The return signal is 15
[10438.714731] [program2] : Module_exit

```

Figure 29: Terminate

```

[10448.810541] [program2] : Module_init QiXixian 120090691
[10448.810543] [program2] : Module_init create kthread start
[10448.810871] [program2] : Module_init kthread start
[10448.811623] [program2] : The child process has pid = 4788
[10448.811624] [program2] : This is the parent process, pid = 4787
[10448.811624] [program2] : Child process
[10448.901656] [program2] : get SIGTRAP signal
[10448.901657] [program2] : Child process terminated
[10448.901658] [program2] : The return signal is 5
[10451.184726] [program2] : Module_exit

```

Figure 30: Trap

2.4 Knowledge Acquired

In this project, I learnt how to modify and recompile the kernel source. Also, this task deepened my understanding of operating system mechanisms.

3 Bonus

3.1 Ideas and Implementation

In this task, I implement the 'pstree' program **with -A -U -l arguments**. Information of processes and threads are stored in `/proc/PID/Stat` folder in linux. Therefore, to get a 'pstree', we need to get the information of each process and thread, construct a tree

to store, and output the tree in a certain type. However, the details are much more than that. To ease in constructing data structure, C++ is used to include STL template classes. Some key implementations are showed below:

1. Difference in Thread and Process

Opendir tools in `<direct.h>` can only detect the PID folder of processes. Luckily, `/proc/PID/task` contains directories of threads the process uses. Therefore, I read process in the `main` function, while read other threads in `read_thread()`, respectively.

2. Merge Nodes or Links

'pstree' command combines nodes and links that have exactly same names. To implement this feature, `pre_hashmap()` can go through the tree and compute the hash value of each node. The formula are given by:

$$hash_value(y) = depth(y) * int2char(y.name) + \sum_{x \in child \ of \ y} hash_value(x) \quad (1)$$

where `int2char()` is a function that compute the hash value of a string.

Therefore, we can easily evaluate if two links are identical by comparing their value.

3. Recursive output

The best solution to go through the whole tree is to use **Deep First Search** (DFS). Every time a twins (multiple siblings) are spotted, you can focus on the first one and leave others in the next lines. This logic enables to print the tree in order.

4. Output Line Control

Except the first line, every output line should output spaces or markers ("|"). Therefore, a vector called `Placeholder` is passed recursively to count how many spaces are ahead: so you can always print exact number of spaces when starting a new line. As for markers, it is only printed if the grandfathers are not terminated. So a vector, called `link_enabler` is used to store the termination status of each depth. When the value is false on certain depth, the `print_spaces` function will output spaces instead of markers.

5. Nodes

The information of nodes are given below:

- pid: store its pid
- child: a vector that stores pointers to each child
- name: a char list that stores its name
- father: a pointer to its parent
- hash_value: the hash value of itself
- is_link: whether the node is in a link

3.2 Environment Deploy

- Go to the workspace `~/Assignment_1/source/bonus` using `cd` command.
- Compile 'pstree.cpp' by typing `make` command.
- Run `./pstree` to display the process-tree. Then type `psree` and compare their output.

3.3 Screenshot

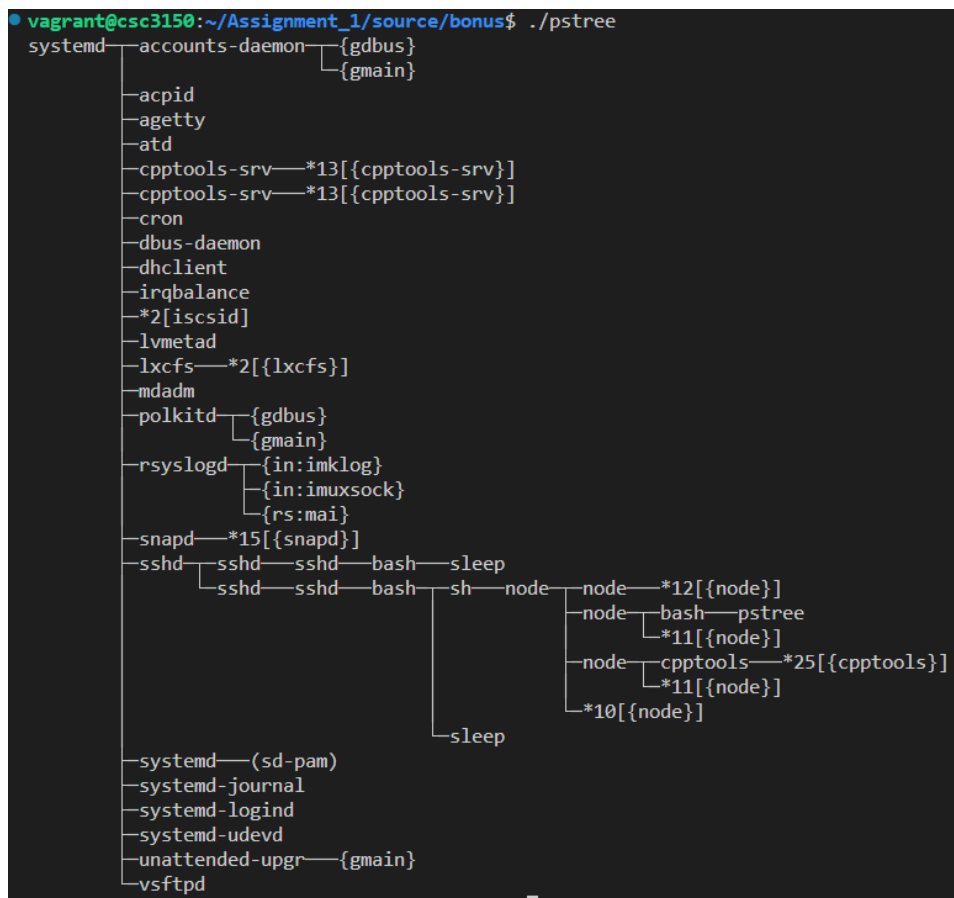


Figure 31: Example of pstree in my computer

```

vagrant@csc3150:~/Assignment_1/source/bonus$ ./pstree -A
systemd--accounts-daemon--{gdbus}
      |--{gmain}
      |--acpid
      |--agetty
      |--atd
      |--cpptools-srv---*13[{cpptools-srv}]
      |--cpptools-srv---*13[{cpptools-srv}]
      |--cron
      |--dbus-daemon
      |--dhclient
      |--irqbalance
      |--*2[iscsid]
      |--lvmtools
      |--lxcfs---*2[{lxcfs}]
      |--mdadm
      |--polkitd--{gdbus}
      |   |--{gmain}
      |   |--rsyslogd--{in:imklog}
      |   |   |--{in:imuxsock}
      |   |   |--{rs:mai}
      |   |--sshd--sshd---sshd---bash---sleep
      |   |   |--sshd---*2[sftp-server]
      |   |   |--sshd---sshd---bash--sh---node--node---*12[{node}]
      |   |   |   |--node--bash
      |   |   |   |   |--bash---pstree
      |   |   |   |   |   |--*12[{node}]
      |   |   |   |   |--node--cpptools---*25[{cpptools}]
      |   |   |   |   |   |--*11[{node}]
      |   |   |   |   |--*10[{node}]
      |   |   |   |--sleep
      |   |--*2[systemd---(sd-pam)]
      |--systemd-journal
      |--systemd-logind
      |--systemd-udev
      |--unattended-upgr---{gmain}
      |--vsftpd

```

Figure 32: Example of pstree with -A argument

3.4 Knowledge Acquired

In this task, I learnt how to get status of processes and threads in Linux. Also, this task makes me more proficient in using ADT types, such as map(dictionary), vector, hash_map and so on.