CSC3150 Assignment 1

Li Dongming, 119020023

Program Design

Program 1

1. Fork a child process
2. if the fork is successful, execute the compiled binary program file in the child process using `execv`
3. using `waitpid` to wait for the child process to terminate or stop
4. print out the signal returned by the child process

Program 2

1. create a child process using `kernel_clone` to execute a custom function `my_exec`
2. in `my_exec`, use `do_execve` to execute the program at `/tmp/test`
3. in the parent process, use `do_wait` to wait for the child process to terminate or stop
4. print out the signal returned by the child process

Bonus

1. parse the input argument into a `struct` for further usage (implemented arguments include: `-A`, `-p`, `-g`, `-n`, `-V`. And `-T` is implemented and **CANNOT be opted**)
2. get the paths for directories of each process under `/proc`
3. parse pid, parent pid, command, group pid from `/proc/[pid]/stat`
4. build a process tree from the parsing result. The children of the tree are implemented as a singly linked list.
5. if necessary, sort the children of each node by the command name
6. print the process tree using recursion according to arguments specified from user input

Environment Setup

In program 1 and bonus, a ordinary linux system is enough, no special setup is needed.

For program 2,

1. download linux system of version 5.10.146
2. modify the source code to export `do_wait`, `getname_kernel`, `do_execve`, `kernel_clone` using `EXPORT_SYMBOL`. If the function is declared as static, remove the `static` identifier
3. compile and install the new linux kernel according to tutorial slide.
4. in program 2, declare such kernel functions using `extern`

Screenshots

Program 1

```
alarm.c
I am the Parent Process, my pid = 7821
I am the Child Process, my pid = 7822
------------CHILD PROCESS START------------
This is the SIGALRM program

Parent process receives SIGCHLD signal
CHILD PROCESS TERMINATED with signal "Alarm clock"
bus.c
I am the Parent Process, my pid = 7844
I am the Child Process, my pid = 7845
------------CHILD PROCESS START------------
This is the SIGBUS program

Parent process receives SIGCHLD signal
CHILD PROCESS TERMINATED with signal "Bus error"
floating.c
I am the Parent Process, my pid = 7847
I am the Child Process, my pid = 7848
------------CHILD PROCESS START------------
This is the SIGFPE program

Parent process receives SIGCHLD signal
CHILD PROCESS TERMINATED with signal "Floating point exception"
hangup.c
I am the Parent Process, my pid = 7850
I am the Child Process, my pid = 7851
------------CHILD PROCESS START------------
This is the SIGHUP program

Parent process receives SIGCHLD signal
CHILD PROCESS TERMINATED with signal "Hangup"
illegal_instr.c
I am the Parent Process, my pid = 7852
I am the Child Process, my pid = 7853
------------CHILD PROCESS START------------
This is the SIGILL program

Parent process receives SIGCHLD signal
CHILD PROCESS TERMINATED with signal "Illegal instruction"
```

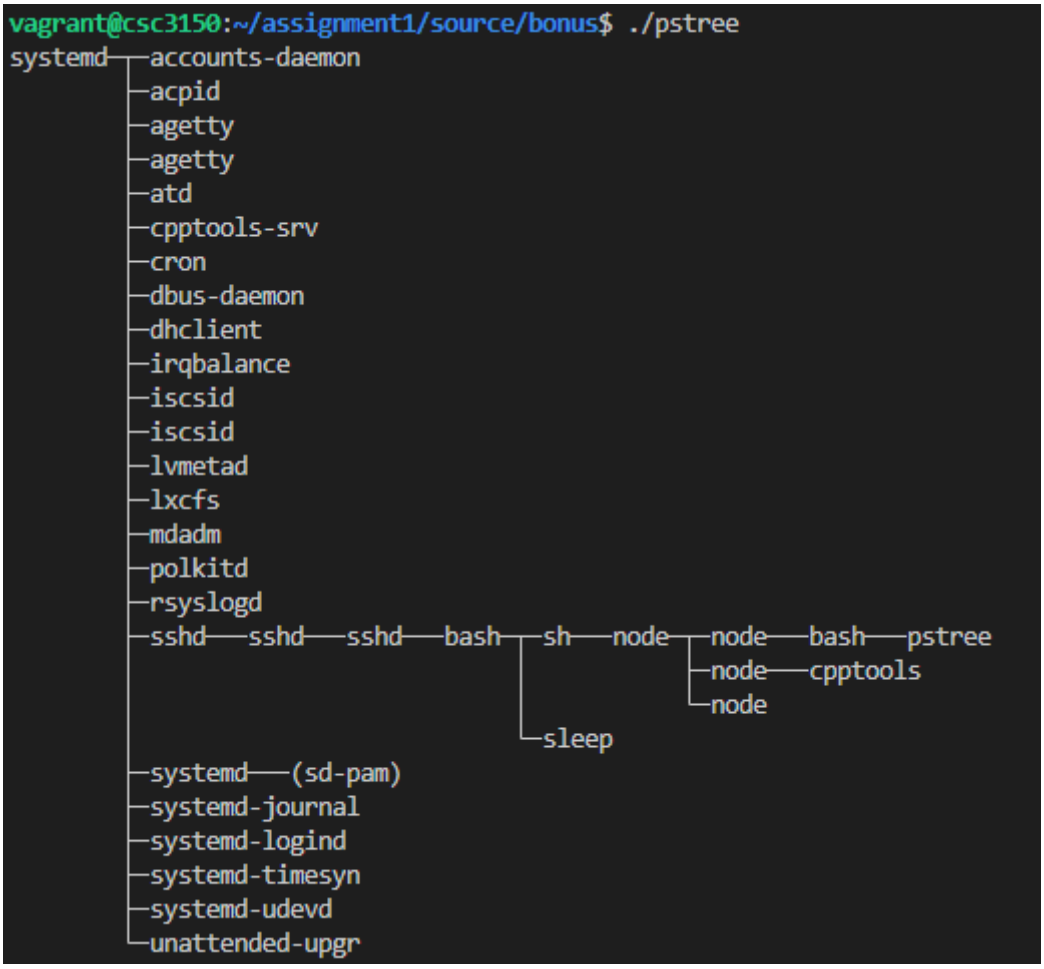
```
interrupt.c
I am the Parent Process, my pid = 7855
I am the Child Process, my pid = 7856
------------CHILD PROCESS START------------
This is the SIGINT program

Parent process receives SIGCHLD signal
CHILD PROCESS TERMINATED with signal "Interrupt"
kill.c
I am the Parent Process, my pid = 7857
I am the Child Process, my pid = 7858
------------CHILD PROCESS START------------
This is the SIGKILL program

Parent process receives SIGCHLD signal
CHILD PROCESS TERMINATED with signal "Killed"
normal.c
I am the Parent Process, my pid = 7859
I am the Child Process, my pid = 7860
------------CHILD PROCESS START------------
This is the normal program

------------CHILD PROCESS END------------
Parent process receives SIGCHLD signal
Normal termination with EXIT STATUS 0
pipe.c
I am the Parent Process, my pid = 7861
I am the Child Process, my pid = 7862
------------CHILD PROCESS START------------
This is the SIGPIPE program

Parent process receives SIGCHLD signal
CHILD PROCESS TERMINATED with signal "Broken pipe"
quit.c
I am the Parent Process, my pid = 7863
I am the Child Process, my pid = 7864
------------CHILD PROCESS START------------
This is the SIGQUIT program

Parent process receives SIGCHLD signal
CHILD PROCESS TERMINATED with signal "Quit"
```

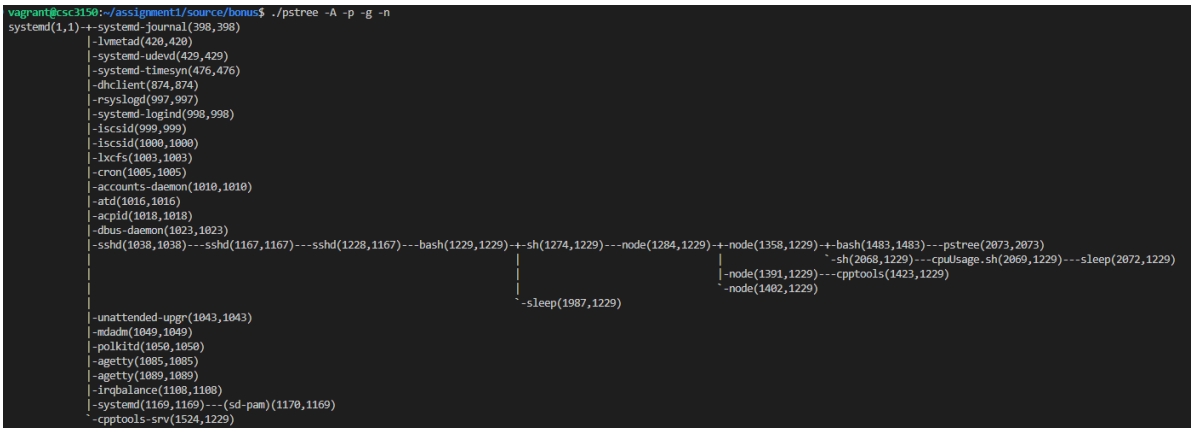
```
segment_fault.c
I am the Parent Process, my pid = 7866
I am the Child Process, my pid = 7867
------------CHILD PROCESS START------------
This is the SIGSEGV program

Parent process receives SIGCHLD signal
CHILD PROCESS TERMINATED with signal "Segmentation fault"
stop.c
I am the Parent Process, my pid = 7869
I am the Child Process, my pid = 7870
------------CHILD PROCESS START------------
This is the SIGSTOP program

Parent process receives SIGCHLD signal
CHILD PROCESS STOPPED
terminate.c
I am the Parent Process, my pid = 7871
I am the Child Process, my pid = 7872
------------CHILD PROCESS START------------
This is the SIGTERM program

Parent process receives SIGCHLD signal
CHILD PROCESS TERMINATED with signal "Terminated"
trap.c
I am the Parent Process, my pid = 7873
I am the Child Process, my pid = 7874
------------CHILD PROCESS START------------
This is the SIGTRAP program

Parent process receives SIGCHLD signal
CHILD PROCESS TERMINATED with signal "Trace/breakpoint trap"
```

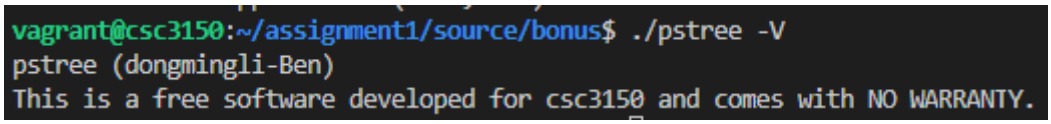
Program 2

```
[10833.783705] [program2] : Module_init {Li Dongming} {119020023}
[10833.783707] [program2] : Module_init create kthread start
[10833.783764] [program2] : Module_init kthread start
[10833.783775] [program2] : This is the parent process, pid = 8356
[10833.783798] [program2] : child process
[10833.783799] [program2] : This is the child process, pid = 8357
[10833.858727] [program2] : get SIGTERM signal
[10833.858729] [program2] : child process terminated
[10833.858730] [program2] : child process TERMINATED with signal 7
[10837.618567] [program2] : Module_exit
```

Bonus







What I learnt from the tasks

1. documentation is very important and it is extremely difficult for a person not familiar with some code (e.g. Linux kernel) to start playing with them without detailed documentation on what the functions expect and how they work.
2. Don't mess with kernel unless you know what you are doing.
3. Everything is a file in linux.