CSC3150 Assignment 1 Report

Basic Information

Name: Xing Qiliang **Student ID:** 123090669 **Date:** October 7, 2025

---

1. Program Design

1.1 Program 1: Signal Handling

1.1.1 Design Philosophy of Signal Handler

Program 1 adopts a parent-child process model to monitor and handle signals. The core design philosophy is:

- Parent process as monitor: responsible for creating child processes, waiting for their completion, and monitoring child process termination status
- Child process as executor: executes test programs through `execvp()` and may terminate due to various signals
- Status analyzer: parent process obtains child process status through `waitpid()` and parses different termination reasons

1.1.2 Implementation of Different Signal Processing Mechanisms

The program uses the status parameter of `waitpid()` system call to identify signal types:

- **Normal termination detection**: Use `WIFEXITED(status)` to check if the process exited normally
- **Signal termination detection**: Use `WIFSIGNALED(status)` to check if the process was terminated by a signal
- **Stop signal detection**: Use `WIFSTOPPED(status)` to check if the process was paused by a stop signal

- **Signal type extraction**: Obtain specific signal numbers through `WTERMSIG(status)` and `WSTOPSIG(status)`

1.1.3 Implementation of Process Control and Signal Transmission

Process control flow:

1. **Process creation**: `fork()` creates child process, return value distinguishes parent and child processes
2. **Program execution**: Child process uses `execvp(argv[1], &argv[1])` to execute target test program
3. **Status monitoring**: Parent process uses `waitpid(pid, &status, WUNTRACED)` to wait for child process status changes
4. **Signal analysis**: Parse child process termination reasons and signal types through status macros

1.1.4 Code Structure and Key Function Description

Main function structure:

- `main()`: Program entry point, implements complete process control flow
- `fork()`: Creates child process, returns 0 for child process, >0 for child process PID in parent
- `execvp()`: Executes target program in child process, replaces current process image
- `waitpid()`: Parent process waits for child process status changes, obtains termination status

Key signal handling logic:

```
// Supported signal types include:
SIGABRT, SIGFPE, SIGILL, SIGINT, SIGKILL, SIGPIPE,
SIGQUIT, SIGSEGV, SIGTERM, SIGTRAP, SIGBUS, SIGALRM,
SIGHUP, SIGSTOP, SIGTSTP
```

Test program design:

- Each test file (such as `alarm.c`, `interrupt.c`, `segment_fault.c`) specifically triggers a particular signal
- Uses `raise()` or system calls (such as `alarm()`) to actively generate signals

- Verifies the completeness of signal handling mechanism through different test programs

1.2 Program 2: Process Management

1.2.1 Design Strategy for Process Creation and Management

Program 2 is a kernel module that implements process management strategies in kernel space:

- **Kernel module architecture**: Implements process creation and management in kernel space through Linux kernel module mechanism
- **Modern kernel API**: Uses `kernel_clone()` to replace traditional `do_fork()`, conforming to modern kernel development standards
- **Kernel thread model**: Creates kernel threads through `kthread_create()` to execute process management tasks
- **Signal handler reset**: Resets all signal handlers to default state before creating child processes

1.2.2 Parent-Child Process Communication Mechanism

Process creation flow:

1. **Kernel thread creation**: `kthread_create(&my_fork, NULL, "MyThread")` creates kernel thread for executing process management
2. **Child process creation**: Uses `kernel_clone()` to create child process in kernel space
3. **Program execution**: Child process executes user space program through `kernel_execve()`
4. **Status monitoring**: Parent process uses `kernel_wait()` to wait for child process completion and obtain status

Kernel space characteristics:

- All operations are performed in kernel space, avoiding user space to kernel space switching overhead
- Uses kernel-specific system call interfaces (`kernel_*` series functions)
- Outputs log information to kernel log buffer through `printk()`

1.2.3 Implementation of Process Synchronization and Coordination

Synchronization mechanism:

- **Kernel wait**: `kernel_wait(pid, &status)` implements synchronous waiting of parent process for child process
- **Status analyzer design**: Implements object-oriented style status analyzer, encapsulating status parsing logic
- **Signal handler reset**: Resets all signal handlers by traversing `current->sighand->action[]`

Status analyzer architecture:

```
struct process_status_analyzer {
    int status;
    int (*get_exit_status)(struct process_status_analyzer *self);
    int (*get_term_signal)(struct process_status_analyzer *self);
    int (*is_exited)(struct process_status_analyzer *self);
    // ... other methods
};
```

1.2.4 Error Handling and Resource Management

Error handling strategy:

- **System call error checking**: Check return values of `kernel_clone()`, `kernel_execve()`, `kernel_wait()`
- **Resource cleanup**: Perform resource cleanup through `module_exit()` when module exits
- **Memory management**: Properly terminate kernel threads and child processes through `do_exit()`

Key function implementation:

- `my_fork()`: Core process management function, handles signal reset, process creation, status monitoring
- `my_exec()`: Child process execution function, calls `kernel_execve()` to execute target program
- `my_wait()`: Wait function, encapsulates `kernel_wait()` and handles error conditions

Kernel module lifecycle:

- **Initialization**: `program2_init()` creates and starts kernel thread
- **Execution**: Kernel thread executes process management tasks
- **Cleanup**: `program2_exit()` performs module cleanup work

1.3 Bonus: pstree Implementation

1.3.1 Core Algorithm Design of pstree Program

pstree program adopts a multi-stage processing algorithm to build and display process trees:

Algorithm flow:

1. **Process scanning phase**: Traverse `/proc` directory, read all process information
2. **Data structure construction**: Create `process_t` structure for each process, store process attributes
3. **Relationship establishment phase**: Build process tree based on PPID to establish parent-child process relationships
4. **Sorting optimization**: Sort processes and child processes according to options
5. **Tree display**: Recursively traverse process tree, generate tree visualization output

1.3.2 Data Structure Selection for Process Tree Construction

Core data structure:

```
typedef struct process {
    int pid, ppid, uid, pgid;           // Basic process information
    char comm[MAX_COMM];                // Process name
    char cmdline[MAX_CMDLINE];          // Command line arguments
    struct process **children;          // Child process pointer array
    int child_count, child_capacity;    // Child process count and capacity
    int thread_count;                   // Thread count
    int is_thread;                      // Thread identification flag
} process_t;
```

Storage strategy:

- **Global process table**: `process_t *processes[MAX_PROCESSES]` stores all processes
- **Dynamic child process array**: Each process maintains a dynamically expandable child process pointer array
- **Hash lookup optimization**: Quickly locate processes through PID to establish parent-child relationships

1.3.3 Implementation Strategy for Various Option Functions

Basic process tree display:

- Uses recursive algorithm `print_tree()` to traverse process tree
- Manages indentation levels and tree connectors through prefix strings
- Supports both Unicode (`└─├─│`) and ASCII (`|-|`) display modes

Thread aggregation display (-p, thread count):

- Scans `/proc/[pid]/task/` directory to count thread numbers
- Displays thread information in `──N*[{process_name}]` format after process name
- Implements association and counting between threads and main process

Command line argument display (-a):

- Reads `/proc/[pid]/cmdline` file to get complete command line
- Processes null character separators, converts to space-separated readable format
- Prioritizes displaying complete command line, degrades to displaying process name

Numeric sorting (-n):

- Implements `process_compare()` comparison function
- Supports sorting by PID numbers and alphabetical sorting by process names
- Sorts both the entire process table and child processes of each process

UID change display (-u):

- Compares UID between process and parent process to detect permission changes
- Gets username through `getpwuid()`, displays in `(user: username)` format

- Parses UID information from `/proc/[pid]/status` file

ASCII mode (-A):

- Provides ASCII character alternatives: `|-` and `|` replace Unicode characters
- Compatible with terminal environments that don't support Unicode

Process group ID display (-g):

- Reads PGID information from `/proc/[pid]/stat` file
- Displays process group identifier in `[pgid]` format after process name
- Automatically enables compact mode to avoid displaying redundancy

1.3.4 Design Philosophy of Process Chain Compression Algorithm

Compression strategy:

- **Single child process chain detection**: Identifies process nodes with only one child process
- **Chain compression**: Merges single child process chains for display on the same line, connected with `───`
- **Recursive compression**: Continues compression until encountering processes with multiple child processes or leaf nodes
- **Alignment calculation**: Precisely calculates the character length of compression chains to ensure correct alignment of child processes

Algorithm implementation:

```
// Core logic of compression algorithm
if (non_thread_children == 1 && !options.compact_not) {
    // Identify single child process and start compression
    printf("───%s", single_child->comm);
    // Continue checking if further compression is possible
    while (child_non_thread_count == 1) {
        // Recursively compress single child process chain
    }
}
```

1.3.5 Implementation of Indentation Alignment and Formatting

Indentation management:

- **Prefix string**: Uses recursively passed prefix strings to manage indentation at each level
- **Branch character selection**: Selects different branch characters based on whether it's the last child process
- **Compression alignment**: Calculates precise length of compression chains, adjusts indentation positions of subsequent child processes

Formatting features:

- **Highlight display**: Supports `-H PID` option to highlight specified process and its ancestors
- **Information integration**: Integrates PID, PGID, user information, thread count into unified format
- **Long line handling**: Supports `-l` option to control long line truncation behavior

Display mode selection:

- **Compact mode**: Uses `print_compact_tree()` by default to implement chain compression
- **Complete mode**: Uses `-c` option to disable compression, displays complete tree structure
- **Compatibility**: Mimics system pstree display behavior and formatting as much as possible

---

2. Development Environment Setup

2.1 System Information

2.1.1 Operating System Version and Kernel Information

Operating system environment:

- **OS**: Ubuntu 20.04.6 LTS (focal)
- **Kernel**: Linux 5.15.10 (Custom compiled on Oct 6, 2025)
- **Virtualization Platform**: VMware® Workstation 17 Pro 17.5.2 build-23775571
- **System Architecture**: x86_64

2.1.2 Development Toolchain Versions (gcc, make, etc.)

Compilation toolchain:

- **GCC Version**: 9.4.0 (Ubuntu 9.4.0-1ubuntu1~20.04.2)
- **Make Version**: GNU Make 4.2.1
- **VScode Version**: 1.104.3 (user setup)

2.2 Compilation Environment

2.2.1 How to Set Up Compilation Environment

Environment preparation:

```
# Install necessary development packages
sudo apt update
sudo apt install build-essential linux-headers-$(uname -r)
sudo apt install git make gcc libc6-dev
sudo apt-get install libncurses-dev gawk flex bison openssl libssl-dev dkms libelf-dev libudev-  dev libpci-dev libiberty-dev autoconf llvm dwarves
```

2.2.2 Makefile Configuration and Usage

Program 1 Makefile:

- Compiles all signal test programs and main program
- Supports `make all`, `make clean` commands
- Uses standard gcc compilation options

Program 2 Makefile:

- Kernel module compilation configuration
- Uses `$(MAKE) -C /lib/modules/$(shell uname -r)/build` for kernel module compilation
- Supports module installation and uninstallation

Bonus pstree Makefile:

- **Compiler setting**: Uses `gcc` as compiler
- **Compilation flags**: `-Wall -Wextra -std=c99` enables all warnings and C99 standard

- **Target file**: Generates executable file named `pstree`
- **Source file**: Compiles from single source file `pstree.c`
- **Clean function**: `make clean` deletes generated executable file
- **Test function**: `make test` automatically tests multiple pstree options
  - Basic function test: `./pstree | head -10`
  - PID display test: `./pstree -p | head -5`
  - Argument display test: `./pstree -a | head -5`
  - ASCII mode test: `./pstree -A | head -5`
  - Numeric sorting test: `./pstree -n -p | head -5`
- **Compilation command**: `$(CC) $(CFLAGS) -o $(TARGET) $(SOURCE)`

2.2.3 Compilation Options and Optimization Settings

Compilation options:

- **Program 1**: `-Wall -Wextra -std=c99` enables warnings and C99 standard
- **Program 2**: Uses kernel module standard compilation options
- **Bonus**: `-O2 -Wall` enables optimization and warning checks

2.3 Kernel Compilation (if applicable)

2.3.1 Preparation Work for Kernel Compilation

Program 2 needs to use three kernel functions: `kernel_clone`, `kernel_execve`, `kernel_wait`. These functions are not exported to modules by default, so custom kernel compilation is required.

2.3.2 Kernel Source Code Acquisition and Configuration

Kernel source code download:

```
# Download Linux 5.15.10 kernel source code
wget https://mirror.tuna.tsinghua.edu.cn/kernel/v5.x/linux-5.15.10.tar.xz
tar -xf linux-5.15.10.tar.xz
cd linux-5.15.10
```

Kernel configuration modification: Need to modify the following files to export necessary kernel symbols:

- `kernel/fork.c`: Add `EXPORT_SYMBOL(kernel_clone);`
- `kernel/exit.c`: Add `EXPORT_SYMBOL(kernel_wait);`
- `fs/exec.c`: Add `EXPORT_SYMBOL(kernel_execve);`

2.3.3 Specific Steps for Kernel Compilation

Kernel compilation process:

```
# Clean previous compilation results
make clean
make mrproper

# Configure kernel (use default configuration and save)
make menuconfig

# Compile kernel image and modules
make bzImage -j$(nproc)
make modules -j$(nproc)

# Install modules and kernel
sudo make modules_install
sudo make install

# Reboot to load new kernel
sudo reboot
```

2.3.4 Kernel Installation and Testing Process

Installation verification:

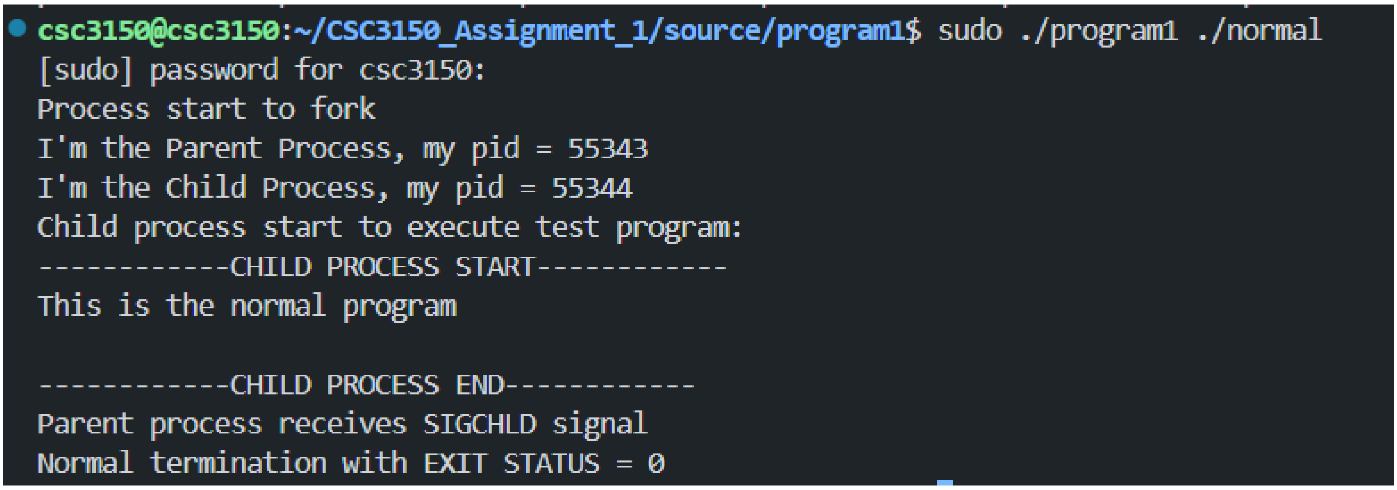
```
# Check kernel version after reboot
uname -r
# Should display: 5.15.10

# Check exported symbols
cat /proc/kallsyms | grep -E "(kernel_clone|kernel_execve|kernel_wait)"

# Verify kernel module compilation environment
ls /lib/modules/$(uname -r)/build
```

---

3. Program Output Screenshots

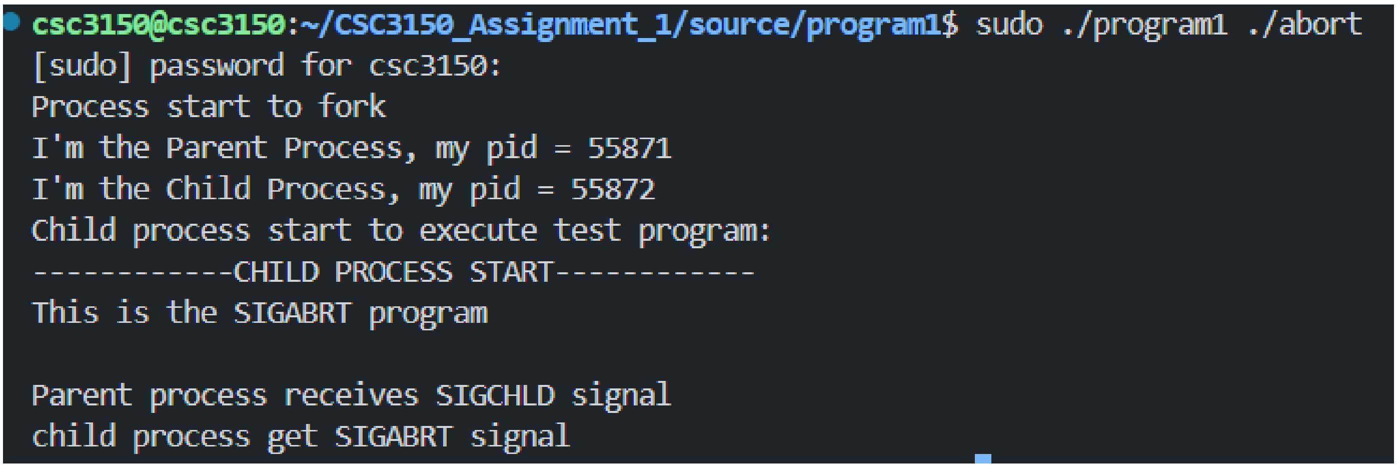
3.1 Program 1 Output

3.1.1 Normal Signal Processing Runtime Screenshots

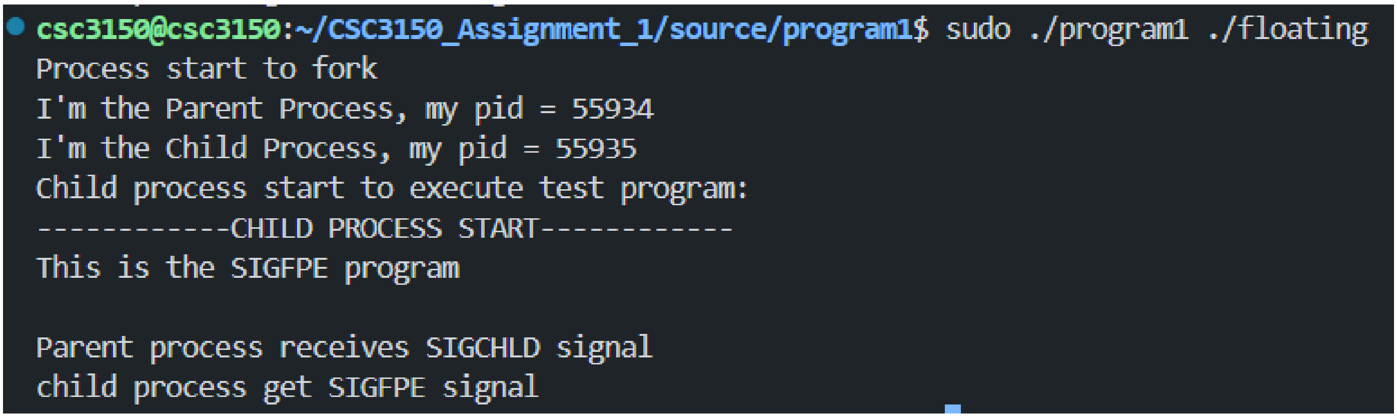


```
csc3150@csc3150:~/CSC3150_Assignment_1/source/program1$ sudo ./program1 ./normal
[sudo] password for csc3150:
Process start to fork
I'm the Parent Process, my pid = 55343
I'm the Child Process, my pid = 55344
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the normal program

------------CHILD PROCESS END------------
Parent process receives SIGCHLD signal
Normal termination with EXIT STATUS = 0
```

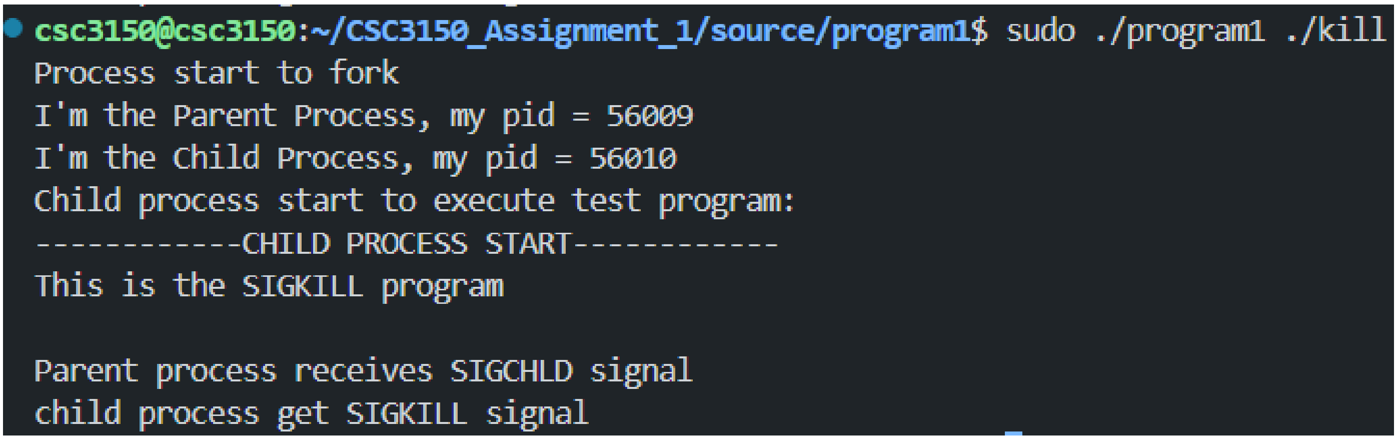
3.1.2 Processing Results of Different Signal Types



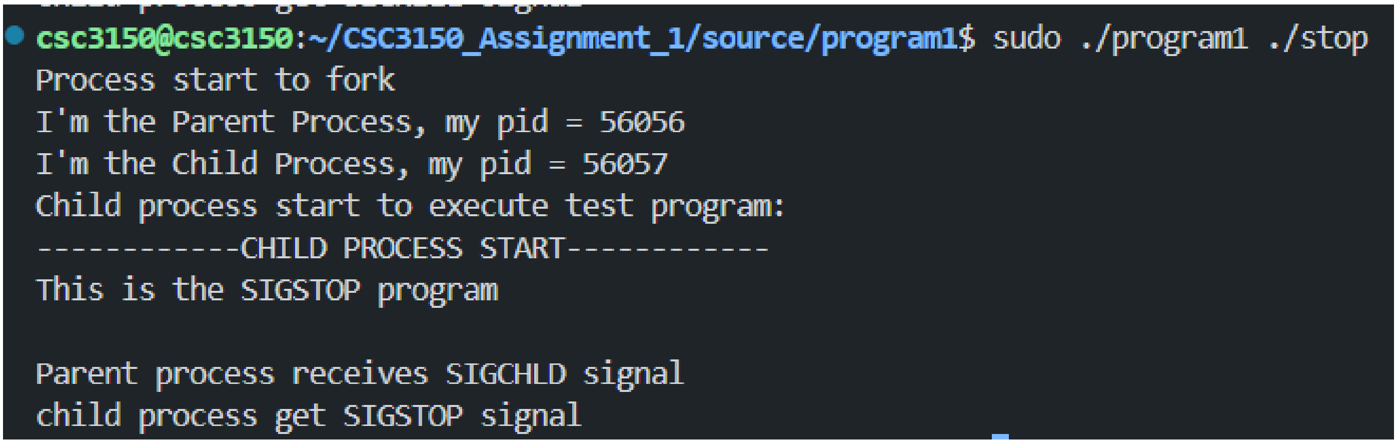
```
csc3150@csc3150:~/CSC3150_Assignment_1/source/program1$ sudo ./program1 ./abort
[sudo] password for csc3150:
Process start to fork
I'm the Parent Process, my pid = 55871
I'm the Child Process, my pid = 55872
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGABRT program

Parent process receives SIGCHLD signal
child process get SIGABRT signal
```



```
csc3150@csc3150:~/CSC3150_Assignment_1/source/program1$ sudo ./program1 ./floating
Process start to fork
I'm the Parent Process, my pid = 55934
I'm the Child Process, my pid = 55935
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
```



```
csc3150@csc3150:~/CSC3150_Assignment_1/source/program1$ sudo ./program1 ./kill
Process start to fork
I'm the Parent Process, my pid = 56009
I'm the Child Process, my pid = 56010
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
```



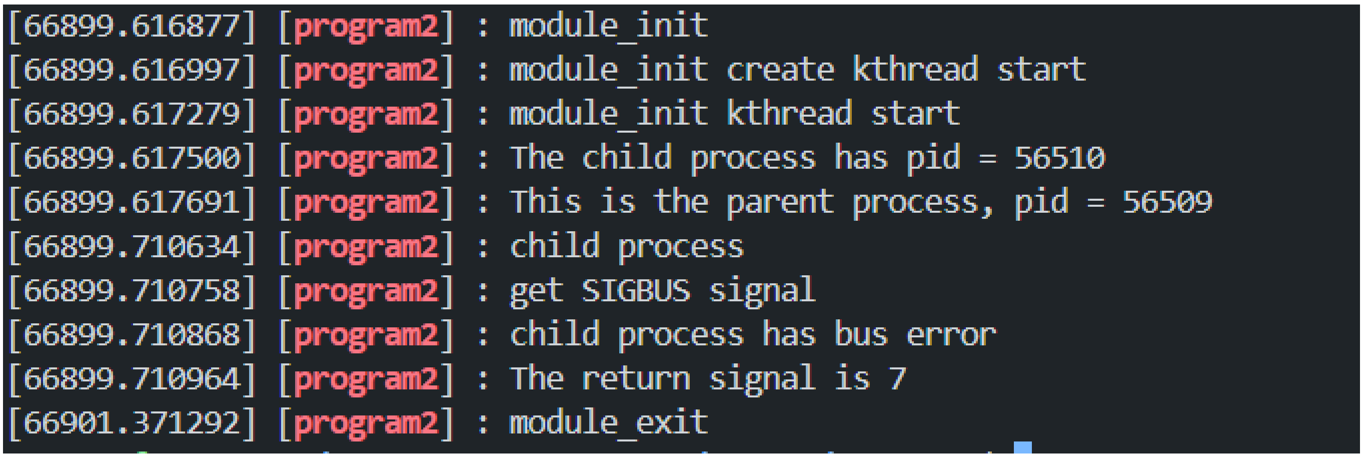
```
csc3150@csc3150:~/CSC3150_Assignment_1/source/program1$ sudo ./program1 ./stop
Process start to fork
I'm the Parent Process, my pid = 56056
I'm the Child Process, my pid = 56057
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGSTOP program

Parent process receives SIGCHLD signal
child process get SIGSTOP signal
```

To save space, only partial program runtime result screenshots are shown here.

3.2 Program 2 Output

```
sudo insmod program2.ko
sudo rmmod program2
dmesg | grep program2
```

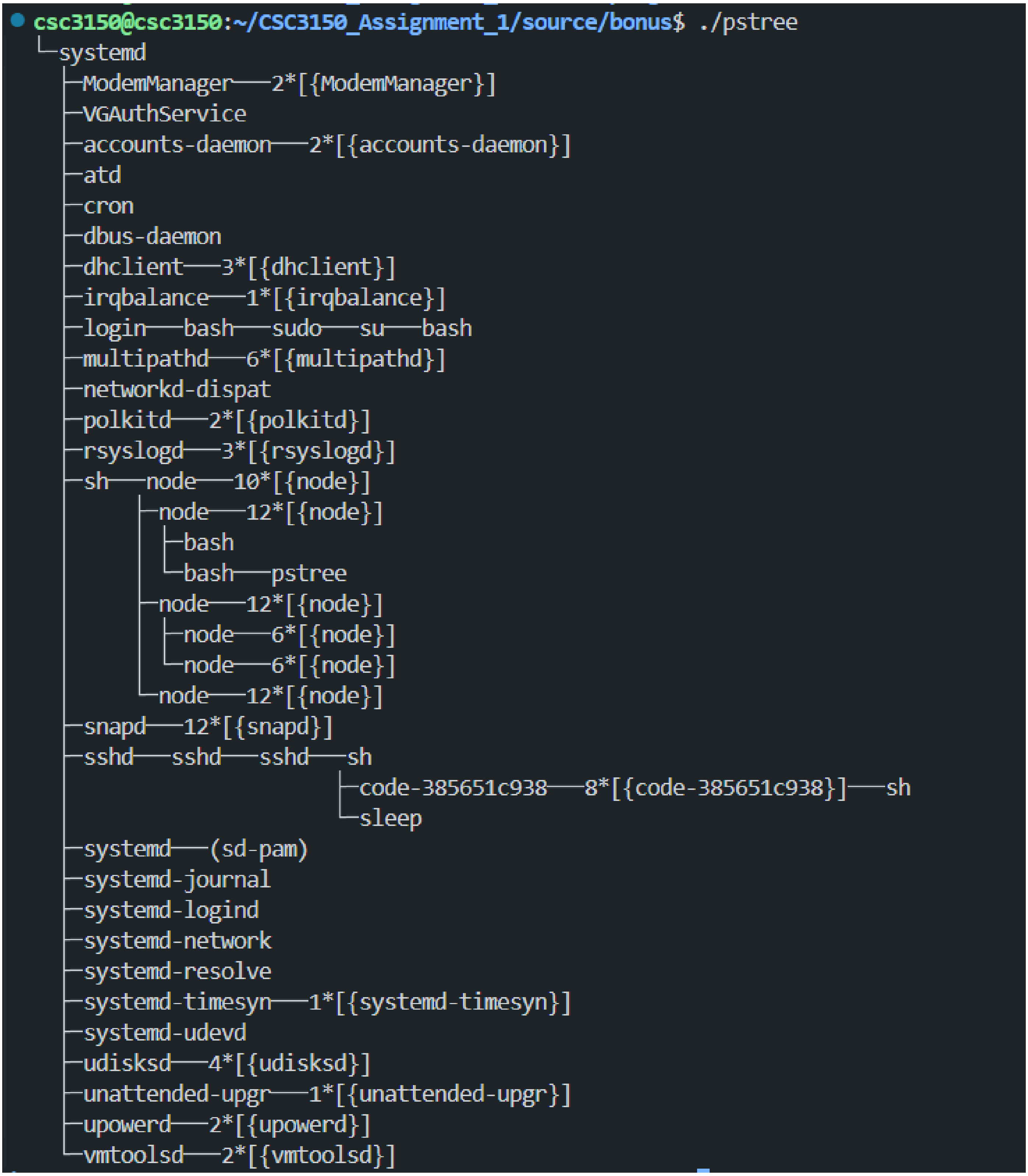
The result is as follows:



```
[66899.616877] [program2] : module_init
[66899.616997] [program2] : module_init create kthread start
[66899.617279] [program2] : module_init kthread start
[66899.617500] [program2] : The child process has pid = 56510
[66899.617691] [program2] : This is the parent process, pid = 56509
[66899.710634] [program2] : child process
[66899.710758] [program2] : get SIGBUS signal
[66899.710868] [program2] : child process has bus error
[66899.710964] [program2] : The return signal is 7
[66901.371292] [program2] : module_exit
```

3.3 Bonus Program Output

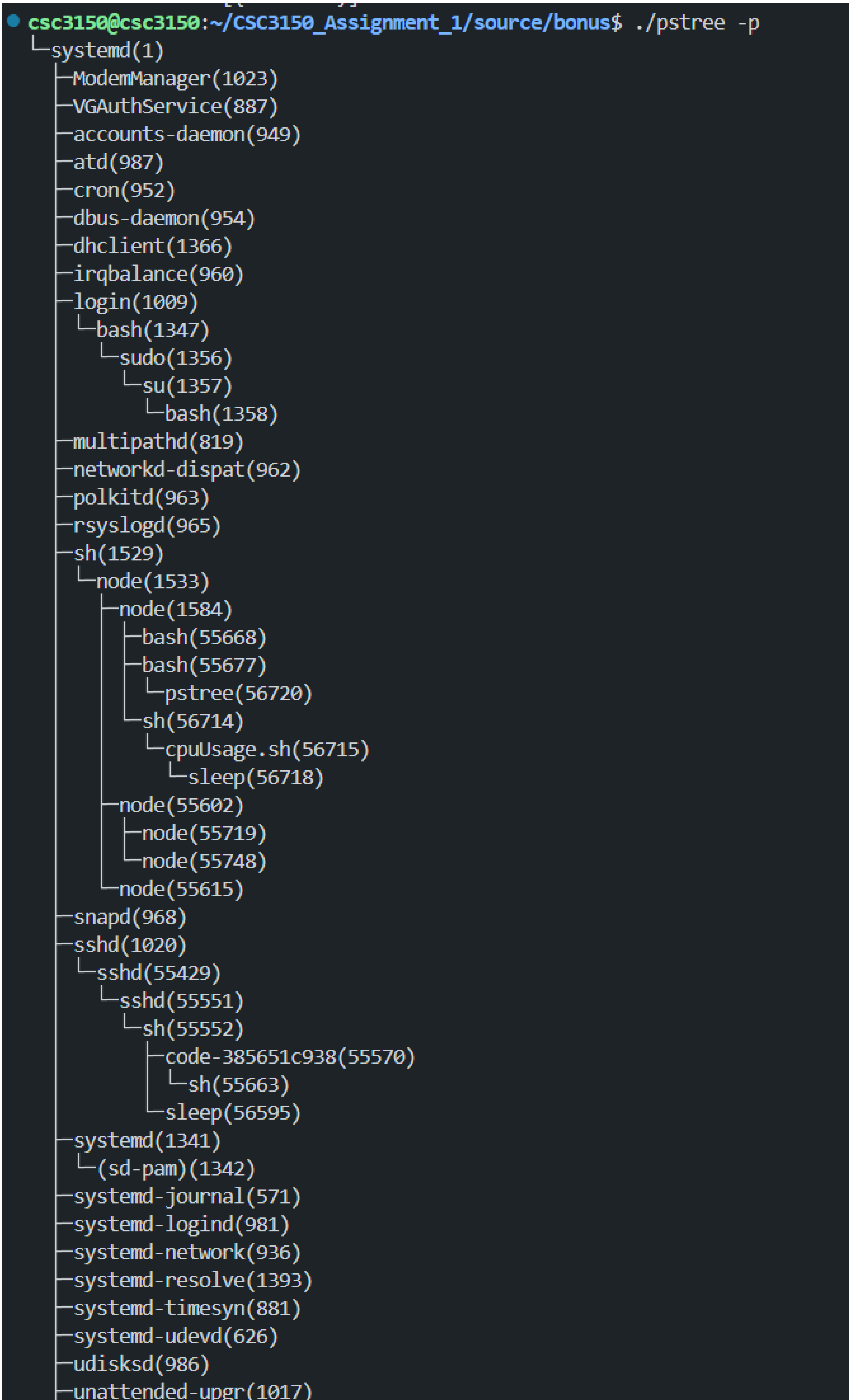
3.3.1 Basic pstree Function Demonstration



```
● csc3150@csc3150:~/CSC3150_Assignment_1/source/bonus$ ./pstree
└─systemd
  ├─ModemManager───2*[{ModemManager}]
  ├─VGAuthService
  ├─accounts-daemon───2*[{accounts-daemon}]
  ├─atd
  ├─cron
  ├─dbus-daemon
  ├─dhclient───3*[{dhclient}]
  ├─irqbalance───1*[{irqbalance}]
  ├─login───bash───sudo───su───bash
  ├─multipathd───6*[{multipathd}]
  ├─networkd-dispat
  ├─polkitd───2*[{polkitd}]
  ├─rsyslogd───3*[{rsyslogd}]
  ├─sh───node───10*[{node}]
  │     ├─node───12*[{node}]
  │     │ ├─bash
  │     │ └─bash───pstree
  │     ├─node───12*[{node}]
  │     │ ├─node───6*[{node}]
  │     │ └─node───6*[{node}]
  │     └─node───12*[{node}]
  ├─snapd───12*[{snapd}]
  ├─sshd───sshd───sshd───sh
  │                     ├─code-385651c938───8*[{code-385651c938}]───sh
  │                     └─sleep
  ├─systemd───(sd-pam)
  ├─systemd-journal
  ├─systemd-logind
  ├─systemd-network
  ├─systemd-resolve
  ├─systemd-timesyn───1*[{systemd-timesyn}]
  ├─systemd-udevd
  ├─udisksd───4*[{udisksd}]
  ├─unattended-upgr───1*[{unattended-upgr}]
  ├─upowerd───2*[{upowerd}]
  └─vmtoolsd───2*[{vmtoolsd}]
```

3.3.2 Output Comparison of Various Options

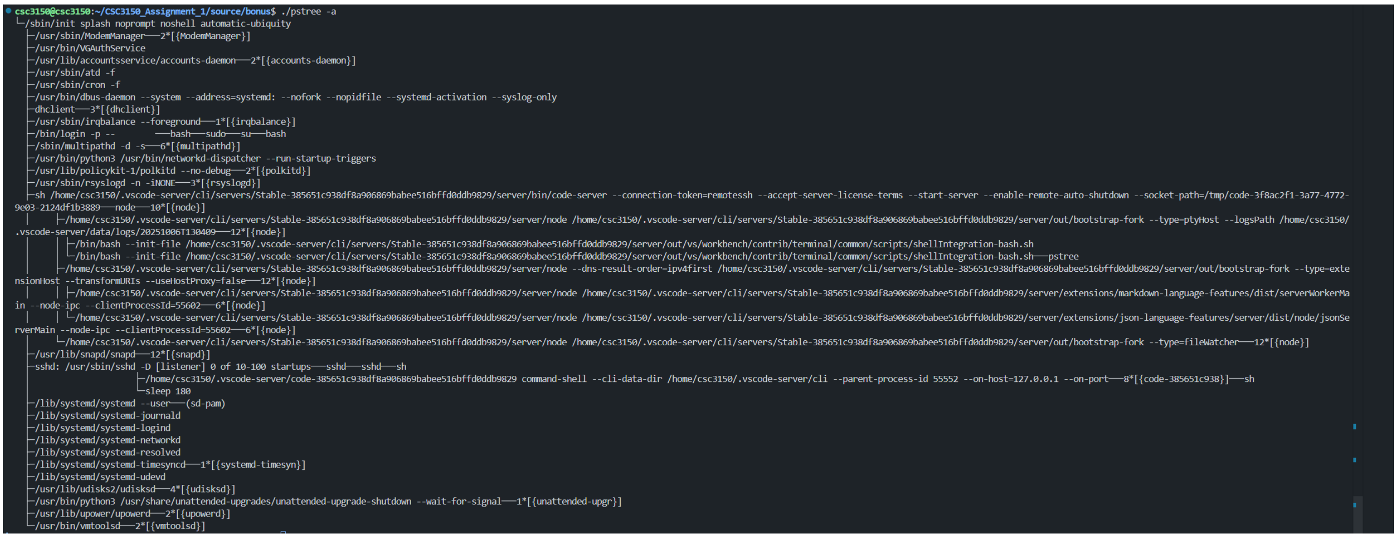
`./pstree -p` **(Display PID)**:



```
● csc3150@csc3150:~/CSC3150_Assignment_1/source/bonus$ ./pstree -p
└systemd(1)
 ├ModemManager(1023)
 ├VGAuthService(887)
 ├accounts-daemon(949)
 ├atd(987)
 ├cron(952)
 ├dbus-daemon(954)
 ├dhclient(1366)
 ├irqbalance(960)
 ├login(1009)
 │└bash(1347)
 │ └sudo(1356)
 │  └su(1357)
 │   └bash(1358)
 ├multipathd(819)
 ├networkd-dispat(962)
 ├polkitd(963)
 ├rsyslogd(965)
 ├sh(1529)
 │└node(1533)
 │ ├node(1584)
 │ │├bash(55668)
 │ │├bash(55677)
 │ ││└pstree(56720)
 │ │└sh(56714)
 │ │ └cpuUsage.sh(56715)
 │ │  └sleep(56718)
 │ ├node(55602)
 │ │├node(55719)
 │ │└node(55748)
 │ └node(55615)
 ├snapd(968)
 ├sshd(1020)
 │└sshd(55429)
 │ └sshd(55551)
 │  └sh(55552)
 │   ├code-385651c938(55570)
 │   │└sh(55663)
 │   └sleep(56595)
 ├systemd(1341)
 │└(sd-pam)(1342)
 ├systemd-journal(571)
 ├systemd-logind(981)
 ├systemd-network(936)
 ├systemd-resolve(1393)
 ├systemd-timesyn(881)
 ├systemd-udevd(626)
 ├udisksd(986)
 ├unattended-upgr(1017)
```

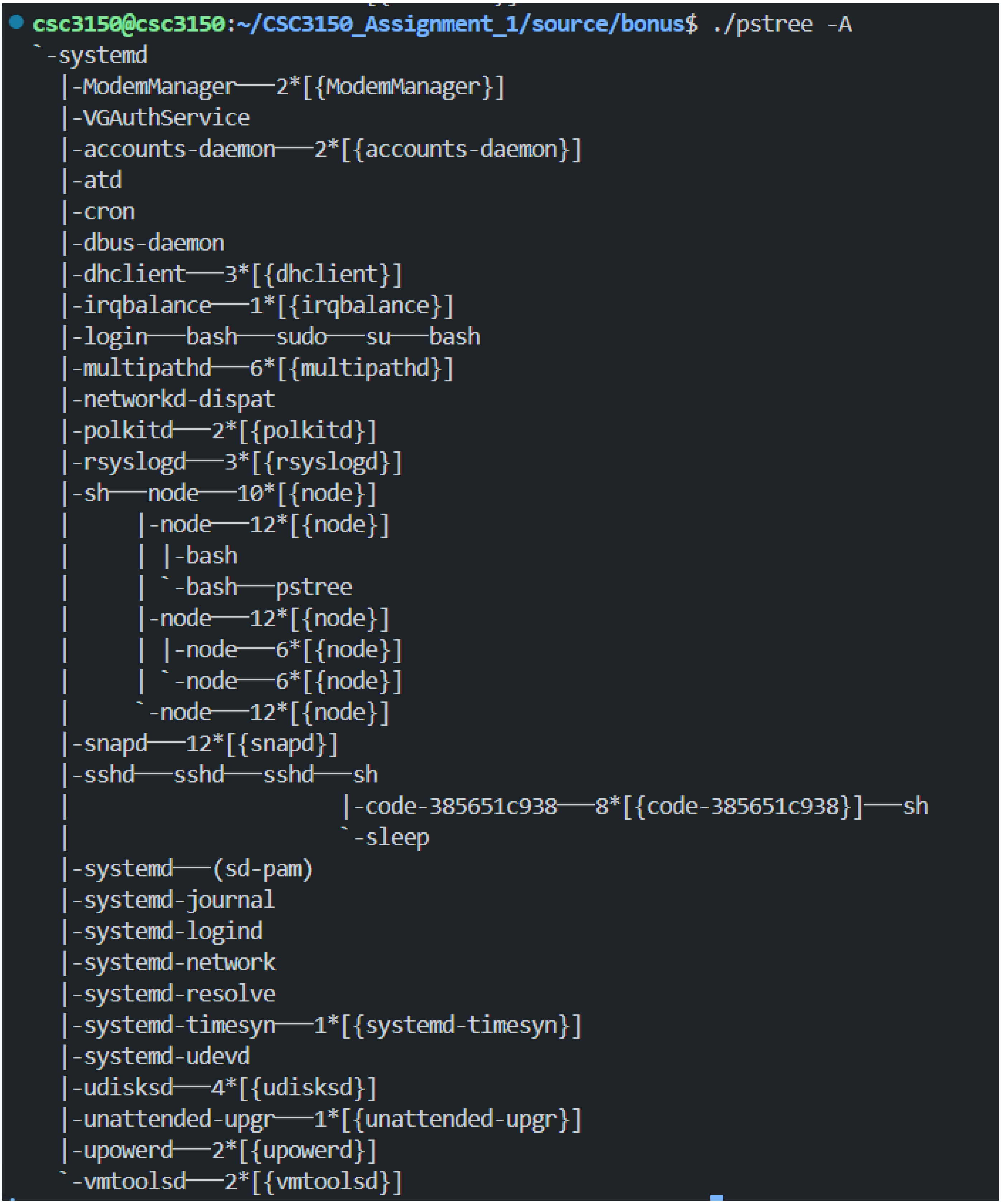
```
├─upowerd(34799)
└─vmtoolsd(888)
```

`./pstree -a` (Display Arguments):



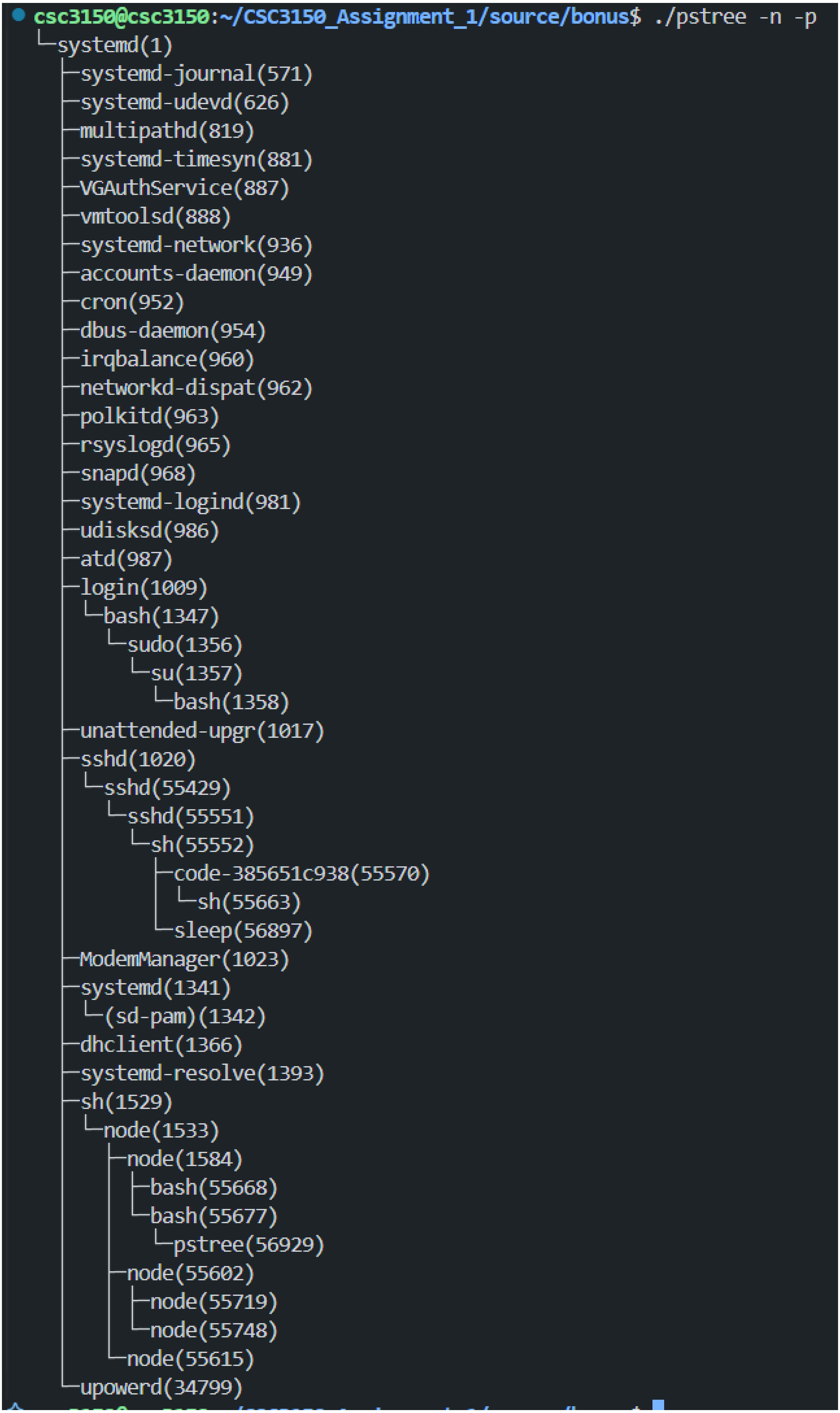
```
csc3150@csc3150:~/CSC3150_Assignment_1/source/bonus$ ./pstree -a
└─/sbin/init splash noprompt noshell automatic-ubiquity
  ├─/usr/sbin/ModemManager──2*[{ModemManager}]
  ├─/usr/bin/VGAuthService
  ├─/usr/lib/accountsservice/accounts-daemon──2*[{accounts-daemon}]
  ├─/usr/sbin/atd -f
  ├─/usr/sbin/cron -f
  ├─/usr/bin/dbus-daemon --system --address=systemd: --nofork --nopidfile --systemd-activation --syslog-only
  ├─dhclient──3*[{dhclient}]
  ├─/usr/sbin/irqbalance --foreground──1*[{irqbalance}]
  ├─/bin/login -p --        ──bash──sudo──su──bash
  ├─/sbin/multipathd -d -s──6*[{multipathd}]
  ├─/usr/bin/python3 /usr/bin/networkd-dispatcher --run-startup-triggers
  ├─/usr/lib/policykit-1/polkitd --no-debug──2*[{polkitd}]
  ├─/usr/sbin/rsyslogd -n -iNONE──3*[{rsyslogd}]
  ├─sh /home/csc3150/.vscode-server/cli/servers/Stable-385651c938df8a906869babee516bffd0ddb9829/server/bin/code-server --connection-token=remotessh --accept-server-license-terms --start-server --enable-remote-auto-shutdown --socket-path=/tmp/code-3f8ac2f1-3a77-4772-
 9e03-2124df1b3889──node──10*[{node}]
  │      ├─/home/csc3150/.vscode-server/cli/servers/Stable-385651c938df8a906869babee516bffd0ddb9829/server/node /home/csc3150/.vscode-server/cli/servers/Stable-385651c938df8a906869babee516bffd0ddb9829/server/out/bootstrap-fork --type=ptyHost --logsPath /home/csc3150/
 .vscode-server/data/logs/20251006T130409──12*[{node}]
  │      │ ├─/bin/bash --init-file /home/csc3150/.vscode-server/cli/servers/Stable-385651c938df8a906869babee516bffd0ddb9829/server/out/vs/workbench/contrib/terminal/common/scripts/shellIntegration-bash.sh
  │      │ └─/bin/bash --init-file /home/csc3150/.vscode-server/cli/servers/Stable-385651c938df8a906869babee516bffd0ddb9829/server/out/vs/workbench/contrib/terminal/common/scripts/shellIntegration-bash.sh──pstree
  │      ├─/home/csc3150/.vscode-server/cli/servers/Stable-385651c938df8a906869babee516bffd0ddb9829/server/node --dns-result-order=ipv4first /home/csc3150/.vscode-server/cli/servers/Stable-385651c938df8a906869babee516bffd0ddb9829/server/out/bootstrap-fork --type=exte
 nsionHost --transformURIs --useHostProxy=false──12*[{node}]
  │      │ ├─/home/csc3150/.vscode-server/cli/servers/Stable-385651c938df8a906869babee516bffd0ddb9829/server/node /home/csc3150/.vscode-server/cli/servers/Stable-385651c938df8a906869babee516bffd0ddb9829/server/extensions/markdown-language-features/dist/serverWorkerMa
 in --node-ipc --clientProcessId=55602──6*[{node}]
  │      │ └─/home/csc3150/.vscode-server/cli/servers/Stable-385651c938df8a906869babee516bffd0ddb9829/server/node /home/csc3150/.vscode-server/cli/servers/Stable-385651c938df8a906869babee516bffd0ddb9829/server/extensions/json-language-features/server/dist/node/jsonSe
 rverMain --node-ipc --clientProcessId=55602──6*[{node}]
  │      └─/home/csc3150/.vscode-server/cli/servers/Stable-385651c938df8a906869babee516bffd0ddb9829/server/node /home/csc3150/.vscode-server/cli/servers/Stable-385651c938df8a906869babee516bffd0ddb9829/server/out/bootstrap-fork --type=fileWatcher──12*[{node}]
  ├─/usr/lib/snapd/snapd──12*[{snapd}]
  ├─sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups──sshd──sshd──sh
  │                         ├─/home/csc3150/.vscode-server/code-385651c938df8a906869babee516bffd0ddb9829 command-shell --cli-data-dir /home/csc3150/.vscode-server/cli --parent-process-id 55552 --on-host=127.0.0.1 --on-port──8*[{code-385651c938}]──sh
  │                         └─sleep 180
  ├─/lib/systemd/systemd --user──(sd-pam)
  ├─/lib/systemd/systemd-journald
  ├─/lib/systemd/systemd-logind
  ├─/lib/systemd/systemd-networkd
  ├─/lib/systemd/systemd-resolved
  ├─/lib/systemd/systemd-timesyncd──1*[{systemd-timesyn}]
  ├─/lib/systemd/systemd-udevd
  ├─/usr/lib/udisks2/udisksd──4*[{udisksd}]
  ├─/usr/bin/python3 /usr/share/unattended-upgrades/unattended-upgrade-shutdown --wait-for-signal──1*[{unattended-upgr}]
  ├─/usr/lib/upower/upowerd──2*[{upowerd}]
  └─/usr/bin/vmtoolsd──2*[{vmtoolsd}]
```

`./pstree -A` (ASCII Mode):



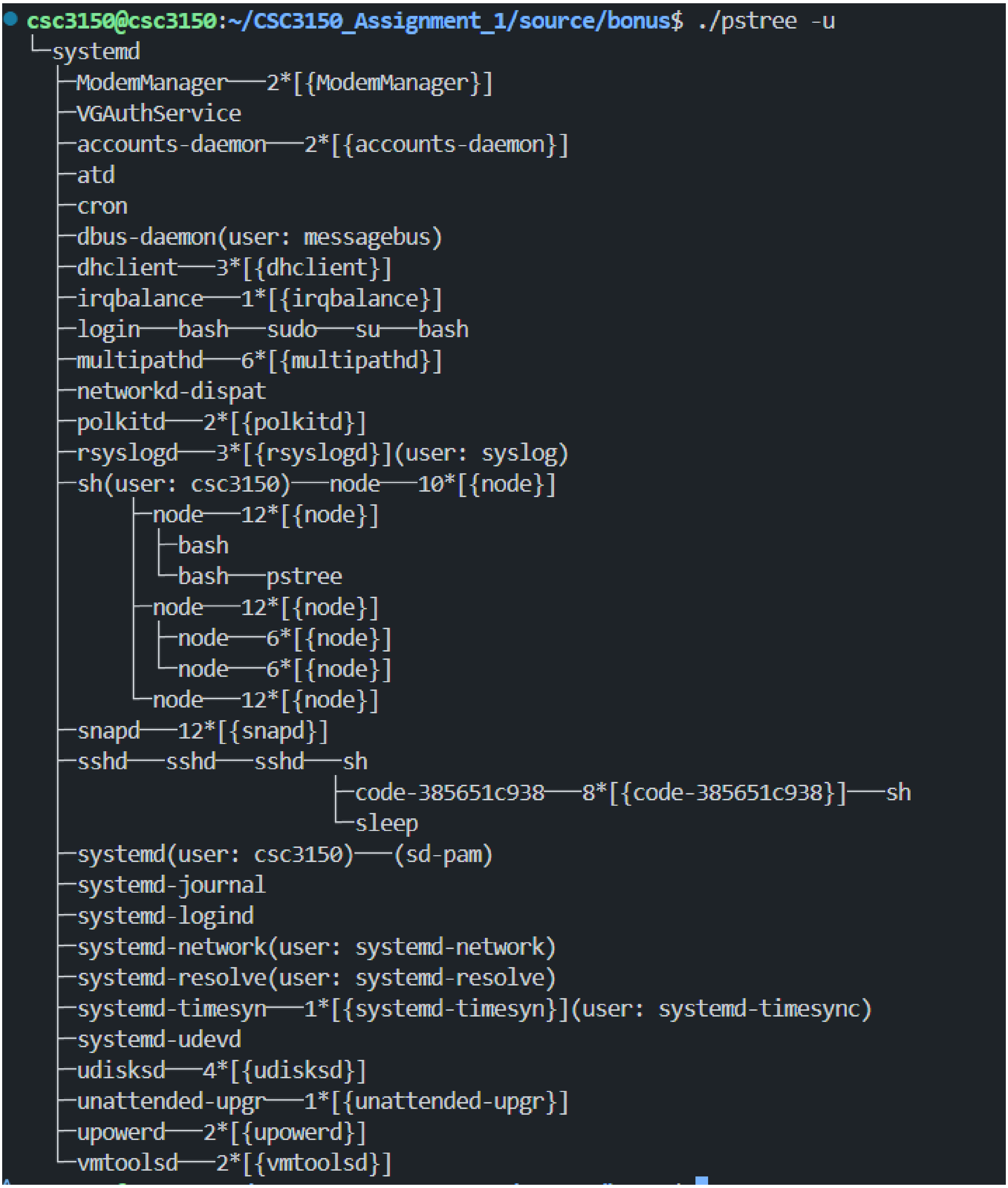
```
● csc3150@csc3150:~/CSC3150_Assignment_1/source/bonus$ ./pstree -A
  `-systemd
    |-ModemManager──2*[{ModemManager}]
    |-VGAuthService
    |-accounts-daemon──2*[{accounts-daemon}]
    |-atd
    |-cron
    |-dbus-daemon
    |-dhclient──3*[{dhclient}]
    |-irqbalance──1*[{irqbalance}]
    |-login──bash──sudo──su──bash
    |-multipathd──6*[{multipathd}]
    |-networkd-dispat
    |-polkitd──2*[{polkitd}]
    |-rsyslogd──3*[{rsyslogd}]
    |-sh──node──10*[{node}]
    |      |-node──12*[{node}]
    |      | |-bash
    |      | `-bash──pstree
    |      |-node──12*[{node}]
    |      | |-node──6*[{node}]
    |      | `-node──6*[{node}]
    |      `-node──12*[{node}]
    |-snapd──12*[{snapd}]
    |-sshd──sshd──sshd──sh
    |                     |-code-385651c938──8*[{code-385651c938}]──sh
    |                     `-sleep
    |-systemd──(sd-pam)
    |-systemd-journal
    |-systemd-logind
    |-systemd-network
    |-systemd-resolve
    |-systemd-timesyn──1*[{systemd-timesyn}]
    |-systemd-udevd
    |-udisksd──4*[{udisksd}]
    |-unattended-upgr──1*[{unattended-upgr}]
    |-upowerd──2*[{upowerd}]
    `-vmtoolsd──2*[{vmtoolsd}]
```

`./pstree -n` **(Numeric Sorting):**



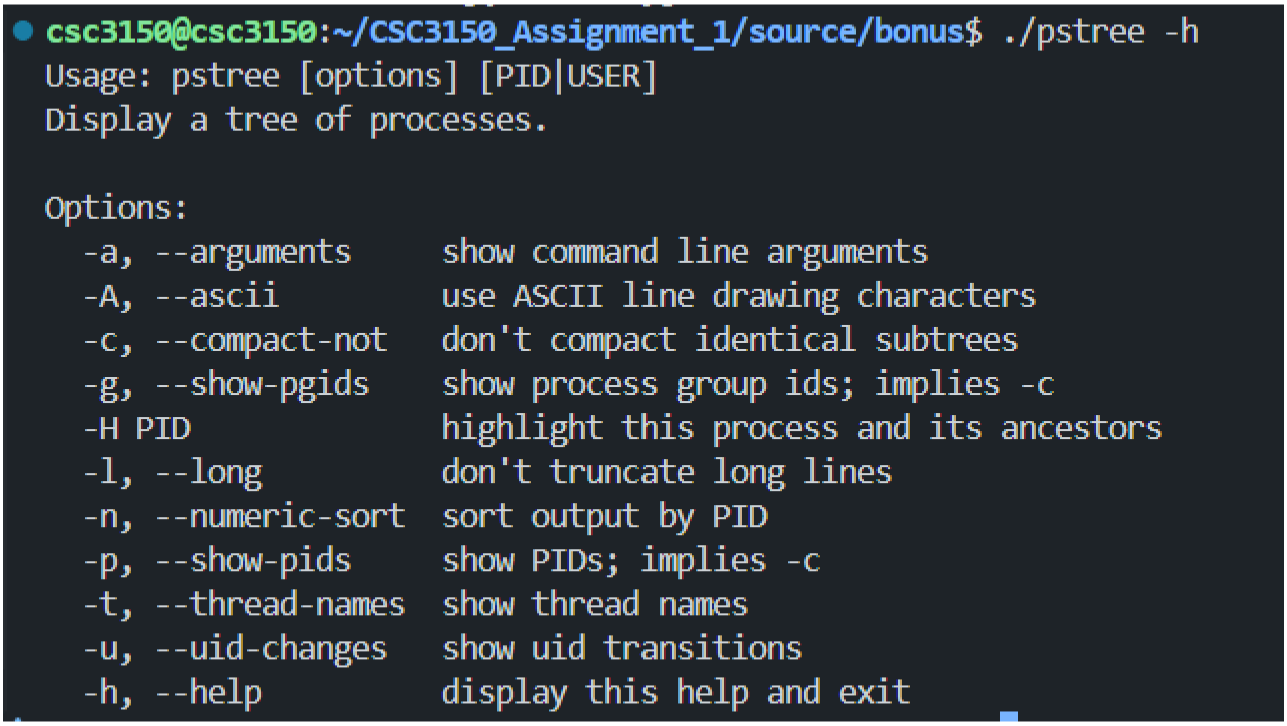
```
csc3150@csc3150:~/CSC3150_Assignment_1/source/bonus$ ./pstree -n -p
└─systemd(1)
  ├─systemd-journal(571)
  ├─systemd-udevd(626)
  ├─multipathd(819)
  ├─systemd-timesyn(881)
  ├─VGAuthService(887)
  ├─vmtoolsd(888)
  ├─systemd-network(936)
  ├─accounts-daemon(949)
  ├─cron(952)
  ├─dbus-daemon(954)
  ├─irqbalance(960)
  ├─networkd-dispat(962)
  ├─polkitd(963)
  ├─rsyslogd(965)
  ├─snapd(968)
  ├─systemd-logind(981)
  ├─udisksd(986)
  ├─atd(987)
  ├─login(1009)
  │ └─bash(1347)
  │   └─sudo(1356)
  │     └─su(1357)
  │       └─bash(1358)
  ├─unattended-upgr(1017)
  ├─sshd(1020)
  │ └─sshd(55429)
  │   └─sshd(55551)
  │     └─sh(55552)
  │       ├─code-385651c938(55570)
  │       │ └─sh(55663)
  │       └─sleep(56897)
  ├─ModemManager(1023)
  ├─systemd(1341)
  │ └─(sd-pam)(1342)
  ├─dhclient(1366)
  ├─systemd-resolve(1393)
  ├─sh(1529)
  │ └─node(1533)
  │   ├─node(1584)
  │   │ ├─bash(55668)
  │   │ └─bash(55677)
  │   │   └─pstree(56929)
  │   ├─node(55602)
  │   │ ├─node(55719)
  │   │ └─node(55748)
  │   └─node(55615)
  └─upowerd(34799)
```

`./pstree -u` **(UID Changes):**



```
csc3150@csc3150:~/CSC3150_Assignment_1/source/bonus$ ./pstree -u
 └─systemd
   ├─ModemManager───2*[{ModemManager}]
   ├─VGAuthService
   ├─accounts-daemon───2*[{accounts-daemon}]
   ├─atd
   ├─cron
   ├─dbus-daemon(user: messagebus)
   ├─dhclient───3*[{dhclient}]
   ├─irqbalance───1*[{irqbalance}]
   ├─login───bash───sudo───su───bash
   ├─multipathd───6*[{multipathd}]
   ├─networkd-dispat
   ├─polkitd───2*[{polkitd}]
   ├─rsyslogd───3*[{rsyslogd}](user: syslog)
   ├─sh(user: csc3150)───node───10*[{node}]
   │     ├─node───12*[{node}]
   │     │ ├─bash
   │     │ └─bash───pstree
   │     ├─node───12*[{node}]
   │     │ ├─node───6*[{node}]
   │     │ └─node───6*[{node}]
   │     └─node───12*[{node}]
   ├─snapd───12*[{snapd}]
   ├─sshd───sshd───sshd───sh
   │                     ├─code-385651c938───8*[{code-385651c938}]───sh
   │                     └─sleep
   ├─systemd(user: csc3150)───(sd-pam)
   ├─systemd-journal
   ├─systemd-logind
   ├─systemd-network(user: systemd-network)
   ├─systemd-resolve(user: systemd-resolve)
   ├─systemd-timesyn───1*[{systemd-timesyn}](user: systemd-timesync)
   ├─systemd-udevd
   ├─udisksd───4*[{udisksd}]
   ├─unattended-upgr───1*[{unattended-upgr}]
   ├─upowerd───2*[{upowerd}]
   └─vmtoolsd───2*[{vmtoolsd}]
```

Only partial functions are shown here. Other related functions can be queried through `./pstree -h`:



```
csc3150@csc3150:~/CSC3150_Assignment_1/source/bonus$ ./pstree -h
Usage: pstree [options] [PID|USER]
Display a tree of processes.

Options:
  -a, --arguments     show command line arguments
  -A, --ascii         use ASCII line drawing characters
  -c, --compact-not   don't compact identical subtrees
  -g, --show-pgids    show process group ids; implies -c
  -H PID              highlight this process and its ancestors
  -l, --long          don't truncate long lines
  -n, --numeric-sort  sort output by PID
  -p, --show-pids     show PIDs; implies -c
  -t, --thread-names  show thread names
  -u, --uid-changes   show uid transitions
  -h, --help          display this help and exit
```

4. Learning Outcomes

4.1 Program 1

4.1.1 Deepened Understanding of Linux System Programming

Through the implementation of Program 1, gained deep understanding of the following core concepts:

- **Process lifecycle management**: Mastered the collaborative working mechanism of `fork()`, `execvp()`, `waitpid()`
- **Signal handling mechanism**: Learned to use status macros like `WIFEXITED()`, `WIFSIGNALED()`, `WIFSTOPPED()` to parse process termination reasons
- **Parent-child process synchronization**: Understood dependency relationships and synchronous waiting mechanisms between processes
- **Error handling**: Learned error checking and exception handling best practices for system calls

4.1.2 Mastery of Signal Handling Mechanisms

Signal types with deep understanding:

- **Exception signals**: SIGSEGV (segmentation fault), SIGBUS (bus error), SIGFPE (floating point exception)
- **Termination signals**: SIGTERM (terminate), SIGKILL (force kill), SIGINT (interrupt)
- **Timing signals**: SIGALRM (alarm)
- **Stop signals**: SIGSTOP (stop), SIGTSTP (terminal stop)

Technical points mastered:

- Signal generation mechanisms (`raise()`, `alarm()`, etc.)
- Signal status parsing methods (bit operations and status macros)
- Differences in how different signals affect process behavior

4.2 Program 2

4.2.1 Kernel Modification and Compilation Operations

Kernel development experience:

- **Kernel module programming**: Learned basic structure of kernel modules, including usage of `module_init()`, `module_exit()`
- **Kernel API usage**: Mastered modern kernel APIs such as `kernel_clone()`, `kernel_execve()`, `kernel_wait()`
- **Symbol export**: Understood the role of `EXPORT_SYMBOL()` and kernel symbol table management
- **Kernel space programming**: Learned to use `printk()` for kernel log output, understood the difference between kernel space and user space

4.2.2 First Time Learning that Make Can Display a UI in Command Line to Generate Config Information

Kernel configuration and compilation process:

- **Using menuconfig**: Learned to use `make menuconfig` for graphical interface operations in kernel configuration
- **Compilation optimization**: Understood efficiency improvements of parallel compilation with `-j$(nproc)`
- **Module installation**: Mastered the difference and roles of `make modules_install` and `make install`

- **Version management**: Learned how to manage multiple kernel versions and switch boot options

4.3 Bonus

4.3.1 About pstree Usage

Advanced system programming skills:

- **File system operations**: Proficiently used `/proc` file system to obtain process information
- **Dynamic memory management**: Implemented dynamically expandable child process arrays, mastered advanced usage of `malloc()`, `realloc()`, `free()`
- **String processing**: Handled null character separators and format conversion in `/proc/[pid]/cmdline`
- **Data structure design**: Implemented efficient process tree data structures and traversal algorithms
- **Algorithm optimization**: Designed process chain compression algorithm, improved display effects and performance

---

Appendix

A. Compilation Commands

```
# Program 1
cd source/program1
make

# Program 2
cd source/program2
make

# Bonus Program
cd source/bonus
make
```