# CSE-381: Systems 2

# Homework #3

**Due: Wednesday September 9 2020 before 11:59 PM**

Late-submission (75% points): Up to Thu Sept 10 before 11:59 PM

**Email-based help Cutoff: 5:00 PM on Tue, Sept 8 2020**

Maximum Points for This Part: 25

**Objective**

The objective of this homework is to develop 1 C++ program to:

- Print process hierarchy for a given PID (process ID)
- Gain familiarity with developing a C++ class
- Continue to gain familiarity with development and testing of C++ programs
- Continue to bolster concepts of stream/file processing.
- Review basics of string processing & problem solving
- Continue to gain familiarity with the use of `std::unordered_map`

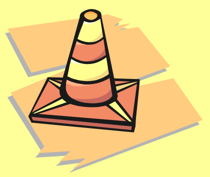
**Submission Instructions**

This part of the homework assignment must be turned-in electronically via Canvas using the CODE plugin. Ensure your program compiles without any warnings or style violations. Ensure you have tested operations of your program as indicated. Once you have tested your implementation, upload the following onto Canvas:

- Just the one C++ header file and 1 C++ source file with the naming convention `MUID_hw3.h` and `MUID_hw3.cpp`, where `MUID` is your Miami unique ID.

**General Note**: Upload each file associated with homework (or lab exercises) individually to Canvas. Do not upload archive file formats such as zip/tar/gz/7zip/rar etc.

## Grading Rubric:



The programs submitted for this homework **must pass necessary base case test(s) in order to qualify for earning any score at all**. Programs that do not meet base case requirements will be assigned zero score! **Program that do not compile, have a method longer than 25 lines, or just some skeleton code, or bad formatting like "} }" or "} } }" will be assigned zero score.**

- **Base case points**: 8 points
- **Additional tests**: 12 points (full-points only for simple recursive solution)
- **Overall C++ class design, code quality, conciseness/code reuse, documentation etc.**: 6 points. These points are typically the hardest to earn.
- **-1 Points**: for each warning generated by the compiler (warnings are most likely sources of errors in C++ programs)
- **Do not use global variables** – that is not good programming practice.

**DUE DATE:** **Wed, September 9 2020 Before 11:59 PM (EST)**

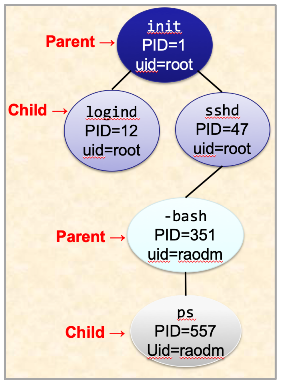
## Develop a C++ program to print full process hierarchy

### *Objective*

The objective of this program (developed as a C++ class) is to print the full hierarchy of processes for a given PID (process ID, specified as the 2nd command-line argument). The data about processes is read from a given text file (as 1st command-line argument).

Essentially this program automates the manual process of tracing the Linux process hierarchy that was done as part of an earlier lab exercise. Refer to the earlier lab exercise of using the output of `ps -fe` to trace processes to recollect the manual process of tracing the process hierarchy.

### *Background*



In Linux, processes are organized as a tree, rooted at `/sbin/init` or `/lib/systemd/system` which is the first process that the Linux kernel starts running. Each process is identified by a unique ID called PID (short for process ID). Furthermore, each process has a PPID (short for parent process ID). The process hierarchy can be determined manually, by recursively tracing the PPID in the output of `ps -fe` command. However, this program is designed to automate this task of tracing the process hierarchy.

### *Data file formats*

Prior to solving any problem is important to study the supplied data. So, ensure you view the data files (yes, of course you can do this in `NetBeans`). The supplied data files are pretty much the output of `ps -fe` command, also reviewed in lab exercise(s).

### *Program requirements:*

- This program should be developed as a C++ class with the following 2 files (other than the `main` function, all methods and instance variables should be part of the class):
    - **`MUID_hw3.h`**: This header file should contain the class declaration. You must have 2 public methods – ❶ a method that loads process data from a given file into 2 `unordered_maps` (discussed below) ❷ a method that prints the process tree for a given PID. You may add any private helper methods as you see fit. It is up to you to decide meaningful names for methods and their parameters. **Each method and instance variable must be documented using Javadoc/doxygen style comments**.
    - **`MUID_hw3.cpp`**: This source file should contain the implementation for the methods you have defined in the header file (ensure you `#include "MUID_hw3.h"` in this source file). This file should document implementation details as comments. This file will also contain the implementation for the `main` method. Your `main` method should create an `MUID_hw3` object and calls methods on the object with suitable parameters.

## *Tips and Notes:*

- In your source file (`.cpp`), remember to prefix each method with your class name, as in: `void MUID_hw3::doSomething()` to let the compiler know that this implementation is for the `doSomething` method in `MUID_hw3` class.
- To ease printing the process hierarchy, in your class use 2 `unordered_map`s as instance variables in your class to store the following:
  1. `pid` ⇔ `ppid` information to ease look-up of parent process ID.
  2. `pid` ⇔ `cmd` information to ease look-up the command associated with a PID.
- Note that each input text file has the 1st line as a header line, which should be ignored.
- Use `std::istringstream` to process each line. Even if you don't use a specific column of data, it is still easier to read it and simply not use it. After that, to read the full command (which has spaces in it) use `std::getline` method after reading specific columns. Refer to your C++ reference sheet and keep I/O simple and straightforward. Excluding variable definitions, processing the input file is just 7-to-8 lines of code.
- Note the order in which processes are printed in the sample output. It is top-down (and not bottom-up). Each column is separated by 1-tab (*i.e.*, `"\t"`)
- For printing the process hierarchy in a top-down manner, you may use an iterative or a recursive solution as you see fit. However, the recursive solution will most likely be shorter than an iterative solution and is preferred. The reference solution uses recursion and prints the process tree is 7-to-8 lines of simple code without any additional containers. **Hence, the full 12-points will be assigned only for a simple recursive solution**.

## *Sample outputs*

One you have completed your program you can test its operation using the command shows below and compare your output to the output shown below. Note that for the 3-column output, each column is separated by 1-tab (*i.e.*, `"PID\tPPID\tCMD"`) character

**Base case #1 [Must pass to earn any points]:**

```
$ ./raodm_hw3 proc_info1.txt 1
Process tree for PID: 1
PID     PPID    CMD
1       0       /sbin/init
```

**Base case #2 [Must pass to earn any points]:**

```
$ ./raodm_hw3 proc_info2.txt 1
Process tree for PID: 1
PID     PPID    CMD
1       0       /lib/systemd/systemd  --system   --deserialize    19
```

**Test case #3 [Must pass to earn full points]:**

```
$ ./raodm_hw3 proc_info1.txt 27426
Process tree for PID: 27426
PID     PPID    CMD
1       0       /sbin/init
949     1       /usr/sbin/sshd   -D      -d
```

```
27282  949    sshd:  salvucwa    [ priv]
27323  27282  sshd: salvucwa@pts/31
27324  27323  -bash
27425  27324  script
27426  27425  bash -i
```

**Test case #4 [Must pass to earn full points]:**

```
$ ./raodm_hw3 proc_info1.txt 27535
Process tree for PID: 27535
PID    PPID   CMD
1      0      /sbin/init
949    1      /usr/sbin/sshd   -D      -d
25640  949    sshd: raodm [priv]
25681  25640  sshd: raodm@pts/22
25697  25681  -bash   --posix    --norc
27535  25697  ps -fe
```

**Test case #5 [Must pass to earn full points]:**

```
$ ./raodm_hw3 proc_info2.txt 25474
Process tree for PID: 25474
PID    PPID   CMD
1      0      /lib/systemd/systemd  --system   --deserialize    19
917    1      /usr/sbin/sshd -D
25150  917    sshd:  raodm    [priv]
25175  25150  sshd: raodm@pts/0
25176  25175  -bash
25474  25176  ps -fea
```

**Test case #6 [Must pass to earn full points]:**

```
$ ./raodm_hw3 proc_info2.txt 25464
Process tree for PID: 25464
PID    PPID   CMD
1      0      /lib/systemd/systemd  --system   --deserialize    19
917    1      /usr/sbin/sshd -D
25150  917    sshd:  raodm    [priv]
25175  25150  sshd: raodm@pts/0
25176  25175  -bash
25454  25176  ./a.out
25455  25454  ./a.out -10   --wait  20
25456  25455  ./a.out
25457  25456  ./a.out
25458  25457  ./a.out
25459  25458  ./a.out
25460  25459  ./a.out  --one    --two -3
25461  25460  ./a.out
25462  25461  ./a.out
25463  25462  ./a.out       --2
25464  25463  ./a.out
```

## Submit to Canvas

This homework assignment must be turned-in electronically via Canvas via the `CODE Plugin`. Ensure your program compiles without any warnings or style violations and operates correctly, at least for the base case. **Ensure you have Javadoc/doxygen style comments in the header file**. Once you have tested your implementation, upload just one C++ source file via the `CODE plugin`.