CMSC 216
Introduction to Computer Systems

Akilesh Praveen
Dr. Ilchul Yoon • Spring 2020 • University of Maryland
http://cs.umd.edu/class/spring2020/cmsc216/

Last Revision: February 4, 2020

Table of Contents

---

1 Notes & Preface

This is a compilation of my notes for CMSC216 as a TA for the Spring 2020 offering of the course at the University of Maryland. All content covered in these notes was created by Dr. Ilchul Yoon and Dr. A.U. Shankar at the University of Maryland.

The actual content of this note repository is the content that I cover as a TA during my discussion section, combined with my personal insights for the course. I believe that together, these will serve as great **supplementary material** for CMSC216, but I would still highly recommend attending all of your lecture and discussion sections to achieve success in CMSC216.

The notes template in use is Alex Reustle's template, which can be found on his github at the following location: `https://github.com/Areustle/CMSC351SP2016FLN`

I maintain this repository and as such, take responsibility for any mistakes. Please send errors to `apraveen@cs.umd.edu`

2 Week 1 - Introduction to CMSC216

CMSC216 is where you learn how a computer works on a much lower level than you've experienced before. There are 3 main components that the course will explore.

Overview

- **UNIX** Threads, processes, and pipes as the building blocks of much bigger applications. We will be working with the UNIX operating system on the development environment at `grace.umd.edu`
- **C** is a high-performance language that works at a much lower level than Java. Things like memory management and advanced data structures are left up to the user. We'll cover concepts like memory management, pointers, and system calls.
- **Assembly** is even lower-level than C, and studying it will reveal how processors process instructions, store data, and maintain a stack and a heap. It's the lowest level you'll go in this class. For this semester's 216, you will be using MIPS assembly.

Grace

In this class, we will be using the `Grace` system to do all of our work. It's a little confusing to understand at first, so here's my way of thinking about it. In CMSC132, we did all of our work on our own computers. We pulled the skeleton code for the projects from the 132 website/repository, edited the code on our computers, and then uploaded our code to the submit server (via Eclipse) in order to test it.

In CMSC216, we have been given access to this big computer that UMD CS owns known as `Grace`. You, as a student, have been given a small chunk of that machine to call your own (for the semester). In this class, we will access your files on the `Grace` system using a program known as `ssh` (that's how MobaXTerm works) and do all of our editing + running code on `Grace` itself. In fact, we will also be submitting our projects from `Grace` to the UMD CS submit server.

Here are the relevant links for getting it all set up. You'll need to setup `Grace` and `gcc` (the C compiler that we'll be using within `Grace`).

- `http://www.cs.umd.edu/ nelson/classes/resources/GraceSystem.shtml`
- `http://www.cs.umd.edu/ nelson/classes/resources/setting_gcc_alias.shtml`

Useful UNIX Commands

Although the UNIX environment may seem confusing at first, learning it is essential to navigating the Grace environment. Below are some of the basic commands that you may find useful when getting started.

- **`ssh`** → If you are not using MobaXTerm, you will have to access grace using the `ssh` command. For the purpose of logging in for CMSC216, I recommend adding the `-y` flag in order to bypass the warning it will give you. E.g. `ssh -y yourdirectoryid@grace.umd.edu`
- **`ls`** → The `ls` command lists all the files in your current directory. You can use the `-l` flag to get more detailed information. E.g. `ls`, `ls -l`
- **`cd`** → The `cd` command changes the directory you're currently in, mainly to directories that you can see with `ls`. Typing `cd ..` will navigate one directory 'up' from your current directory, and `cd` without anything else will return you to your home directory. E.g. `cd 216public`
- **`pwd`** → This command displays your current directory. Useful for finding out where exactly you are in the UNIX file hierarchy. E.g. `pwd`
- **`rm`** → This command stands for 'remove'. It can be used to remove singular files, or can alternatively be used with the `-r` flag to recursively remove directories. E.g. `rm hello.c`, `rm -r project1` (project1 would be a folder.

Lots of these UNIX commands are super useful once you get to know them, but it may be hard becoming acquainted with how they work from the outset. It's a far cry from the GUI you had in CMSC132, so here are a few tips.

- If you're just starting out and still need a graphical representation of the filesystem, I'd highly recommend setting up **MobaXTerm**. The program provides just a little more graphical representation than just a pure terminal, and allows you to navigate the Grace filesystem more freely. I like to think of it as training wheels as you get acquainted with Grace.
- I'd highly recommend getting used to making folders, deleting folders, deleting files, and navigating up and down through the filesystem with rapid sequences of `ls` and `cd`. As with all things, practice makes perfect, and pretty soon you'll be a command line wizard.

Machine

A computer is composed of several parts, but a great way to think about it is a few main components connected by a **bus**.

- **Memory** can just be thought of as a contiguous array of bytes. At the end of the day, this is the stuff that has to be written to/read from.
- **I/O Devices** are connected to the CPU via a bus, like mentioned above. By performing read/write operations to the right adaptor, the CPU is able to interface with different I/O devices.
- **CPU** is the central processing unit of the computer. It handles computational operations (arithmetic, logic, etc.) and interfaces with the memory and I/O devices via the bus. The CPU is also responsible for performing the **fetch-execute cycle**.

A bus is like one main connector that's responsible for making sure the CPU, memory, and I/O devices are all able to interface with each other.

Note that in this course, we won't be going too in-depth into hardware (that's more Computer Engineering), but it's great background knowledge to have as you approach this class, which is why I have included it here.

3 Week 2

4 A Main Section (Template)

Here are some cool notes that will no doubt be very helpful to students.

Here is some cool code in Maryland colors!

```
int main() {
  int x = 1;
  int y = 2;
  printf("hello world");
  return x + y;
}
```

A Subsection

A subheading's text