CSC209: Software Tools and Systems Programming

Week 1: Intro & Bitwise Operations[1]

Kianoosh Abbasi

[1] Slides are mostly taken from Andi Bergen's in summer 2021.

What is This Course About?

- Systems programming with a UNIX focus:
  - The file system
  - Process management
  - System calls
  - Inter-process communication
- C is our programming language of choice
  - Other systems-level languages exist, e.g., C++, Go

What is This Course About? (2)

- Programming tools: Write code more efficiently
  - Debugging tools: gdb, valgrind, strace, ltrace
  - Build automation tools: make
  - Version control: git
- Interacting with a UNIX-like OS using the Bash shell

notes: With the help of these tools you will spend less time debugging, more time being productive.

After taking this course, you should be able to. . .

- Diagram and explain the memory usage/layout of an arbitrary C program
- Write programs that leverage the Linux kernel interface, GNU C library, and GNU C compiler to:
    - Perform low-level I/O, process management, and inter-process communication (local & networked)
    - Adhere to the UNIX philosophy
- Read the man page of an unfamiliar system call or library function and be able to understand and use it
- Use standard UNIX development tools and command-line utilities

A Typical Week in CSC209

- Prepare for class in advance
  - Watch videos on PCRS to familiarize yourself with concepts
  - Solve simple exercises (worth marks)
  - Note down your questions
- Participate in class to consolidate and deepen your knowledge by:
  - Practising more advanced exercises
  - Asking questions
- Apply your knowledge on assignments and tutorials

Slides

- Slides will NOT help you much
- They merely contain the topics and headlines
- Attend the lectures to learn things!

Course Evaluation Scheme

- 10%: Weekly lecture prep (due 11am on Monday)
- 10%: Assignment 1 (System Calls, Dynamic Memory)
- 10%: Assignment 2 (Processes)
- 10%: Assignment 3 (Communication)
- 20%: Midterm Exam
- 40%: Final Exam (Min. 40% required to pass course)

Assignment Submission

- Assignments will be submitted over git
- Repositories will be managed on MarkUs
- Each assignment will have its own directory in your repo
- Feedback and marks will be pushed to your repos

Assignment Grading

- Assignment grading will be automated
- Be careful with:
  - Required file names
  - Directory structure
  - Output format
    - “Hello, world!” is not equivalent to “Hello, World!” or “Hello, world!” (spot the differences)
- **Code that does not compile will receive a grade of zero**

Testing Your Assignments/Tutorials

- Ensure that your code compiles without warnings and errors
- It is necessary, but not sufficient, that your code runs without crashing the lab PCs
- Use the necessary debugging tools to ensure that your program is free of errors that may cause it to:
    - Work on one PC, crash on another PC
    - Work on some runs, crash on other runs, even on the same PC

Academic Integrity

The work you submit must be your own, done without participation by others. It is an academic offence to hand in anything written by someone else without acknowledgement.

Academic Integrity Don'ts:

- Looking at another student's assignment
- Using code that you haven't written, without attribution
- Asking someone else (e.g., classmate or stranger on Stack Overflow) to write your code or help you solve the problems
- You are not helping your friend when you give them a copy of your assignment
- You are hurting your friend when you ask them to give you a copy of their assignment

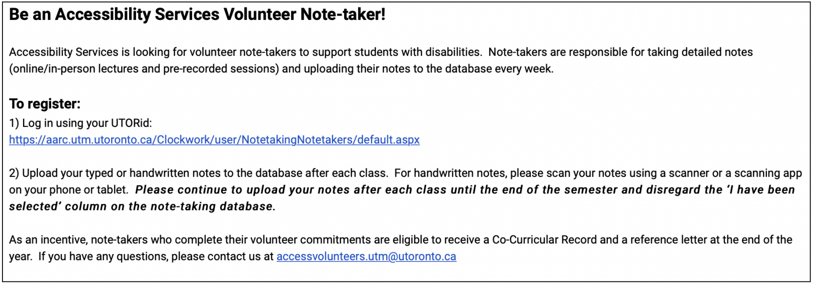
Do Help Each Other by:

- Explaining and/or clarifying concepts
- Reviewing/modifying/practicing exercises from PCRS, lectures, and previous weeks' tutorials together
- Helping each other understand documentation, error messages
  Give someone a fish and you feed them for a day. Teach someone to fish and you feed them for a lifetime.

How to Get Support

- Use the Discussion Board
  - Ask questions about course content here (not by e-mail), so all students can benefit
- Form an FSG
- Office Hours
- Go to TA office hours, ask questions to your TAs
  - Don't ask TAs "How do I do the assignment?"
  - Ask questions to understand tools/concepts needed for completing the assignment, common mistakes, debugging techniques, etc.

Note-taker requests



Be an Accessibility Services Volunteer Note-taker!

Accessibility Services is looking for volunteer note-takers to support students with disabilities. Note-takers are responsible for taking detailed notes (online/in-person lectures and pre-recorded sessions) and uploading their notes to the database every week.

To register:

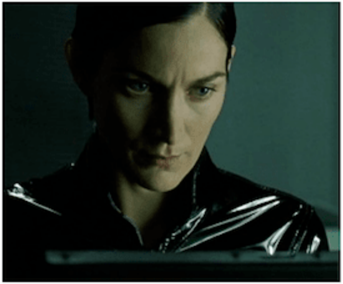
1) Log in using your UTORid:
https://aarc.utm.utoronto.ca/Clockwork/user/NotetakingNotetakers/default.aspx

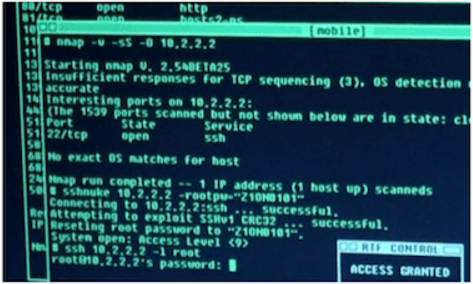
2) Upload your typed or handwritten notes to the database after each class. For handwritten notes, please scan your notes using a scanner or a scanning app on your phone or tablet. ***Please continue to upload your notes after each class until the end of the semester and disregard the 'I have been selected' column on the note-taking database.***

As an incentive, note-takers who complete their volunteer commitments are eligible to receive a Co-Curricular Record and a reference letter at the end of the year. If you have any questions, please contact us at accessvolunteers.utm@utoronto.ca

Preparing for CSC209: Linux

- This course assumes basic familiarity with Linux
- Get accustomed to using a text editor on Linux
- Familiarize yourself with basic shell commands/utilities
- Learn to ssh into the lab machines to work remotely
- Install a Linux virtual machine on your PC (even better: dual-boot)





```
80/tcp      open        http
81/tcp      open        hosts2-ns
```

```
# nmap -v -sS -O 10.2.2.2

Starting nmap V. 2.54BETA25
Insufficient responses for TCP sequencing (3). OS detection
accurate
Interesting ports on 10.2.2.2:
(The 1539 ports scanned but not shown below are in state: cl
Port        State       Service
22/tcp      open        ssh

No exact OS matches for host

Nmap run completed -- 1 IP address (1 host up) scanneds
# sshnuke 10.2.2.2 -rootpw="Z1ON0101"
Connecting to 10.2.2.2:ssh ... successful.
Attempting to exploit SSHv1 CRC32 ... successful.
Reseting root password to "Z1ON0101".
System open: Access Level <9>
# ssh 10.2.2.2 -l root
root@10.2.2.2's password: █
```

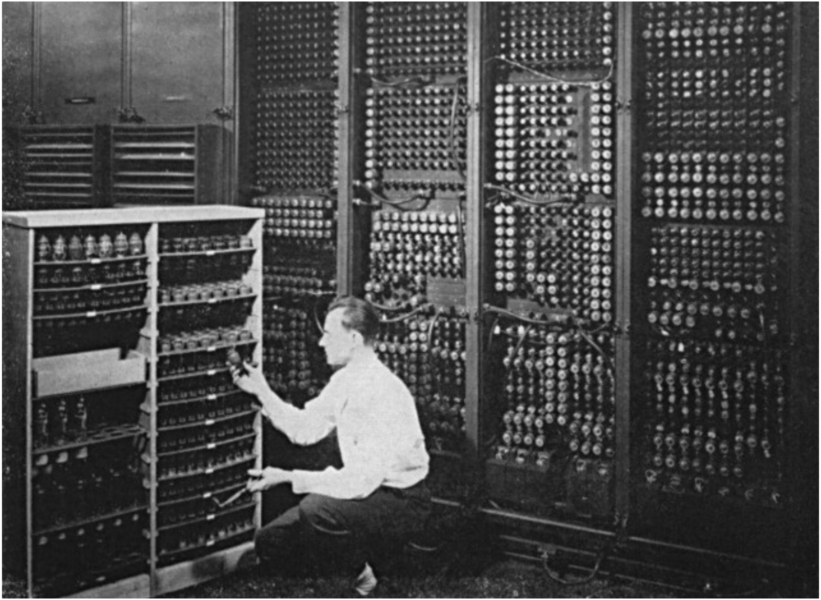
C Programming

- The first half of the course focuses on learning important concepts in C
- Next half will utilize C for system programming
- After taking this course, you should just as easily be able to learn another language (like Go) and accomplish the same goals
- Java (which you learned in CSC207) syntax was designed to be familiar to C programmers

Preview of What's Next

- This week: Bitwise operations, machine language, and assembly
- Next week: C syntax, UNIX shell, and compiling C programs
- The week after: Arrays and pointers (very important!!!)
- Class prep is due each Tuesday at 6pm (1 hour before the lecture)
- Assignment 1 will be posted in a few weeks

Computers



Binary logic

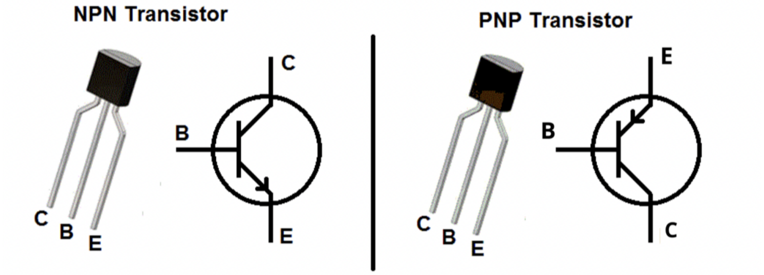
1. Representing numbers in base 2
2. All arithmetic is the same as base 10
3. How to represent negative numbers?

Negative numbers

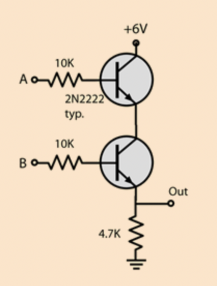
1. Setting one bit aside as the “sign bit”
2. 1’s complement
3. 2’s complement

Transistors

1. All the **smartness** that revolutionized our lives goes back to transistors!
2. Watch https://www.youtube.com/watch?v=J4oO7PT_nzQ



Binary gates

1. Now we can design binary operators
2. NOT, AND, OR, etc
3. 

Bitwise Operations

Working with file descriptors (FDs) and other low-level interfaces requires us to be familiar with some bit operations, namely:

1. Shift operators
2. Bitwise `AND`, `OR`, `NOT`, and `XOR` operators
3. Using bitwise operators to *set* and *clear* bits

Bitwise Left Shift Operator

`i << j` shifts bits in `i` to the left by `j` places

- The right-hand-side is filled with `0` bits if `i` is *positive*
    - It is undefined if `i` is *negative*

```
unsigned char i, j;
i = 13;       /* binary 00001101 */
j = i << 2;  /* binary 00110100 */
```

Bitwise Right Shift Operator

`i >> j` shifts bits in `i` to the right by `j` places

- The left-hand-side is filled with `0` bits if `i` is *positive*
- It is implementation-defined if `i` is *negative*

```
unsigned char i, j;
i = 13;      /* binary 00001101 */
j = i >> 2;  /* binary 00000011 */

char k, l;
k = -128;    /* binary 10000000 */
l = k >> 2;  /* binary 11100000 (maybe) */
```

Bitwise AND

```c
char i, j, k;
i = 21;    /* binary 00010101 */
j = 56;    /* binary 00111000 */
k = i & j; /* binary 00010000 */
```

Bitwise OR

```
char i, j, k;
i = 21;      /* binary 00010101 */
j = 56;      /* binary 00111000 */
k = i | j;  /* binary 00111101 */
```

Bitwise NOT

```
char i, j;
i = 21;  /* binary 00010101 */
j = ~i;  /* binary 11101010 */
```

Bitwise XOR

```
char i, j, k;
i = 21;     /* binary 00010101 */
j = 56;     /* binary 00111000 */
k = i ^ j;  /* binary 00101101 */
```

Convention for Numbering Bits

Convention for referring to specific bits is to number them from the right-hand-side, e.g., a char consists of 8 bits, starting from `b0` on the right-most (i.e., *least significant* or *low-order*) bit:

```
b7 b6 b5 b4 b3 b2 b1 b0
```

Setting Bits

To *set* bit 4 of integer `n` (i.e., set value of bit 4 to `1`):

```
n = n | (1 << 4)
```

or (16 is less clear, though!)

```
n = n | 16
```

or

```
n |= 16
```

Clearing Bits

To *clear* bit 3 of integer `n` (i.e., set value of bit 3 to 0):

```
n = n & ~(1 << 3)
```

or

```
n = n & ~8
```

Exercise

How can you test whether bit `i` is set in integer `n`?

Exercise

1. Calculate whether a signed integer `a` is greater than `b` or not (with bitwise operators)
2. Find a bitwise statement that returns `a` if `a > b` and zero otherwise
3. Calculate the maximum of two signed integers `a` and `b` only with bitwise operators and without branching.