Lab Tutorial

1. Introduction

In this tutorial we review the basics of Linux, C programming and Data Structures in C. You should already have encountered most of this material in prerequisite classes, but now is a good time to brush up your skills, if you are uncomfortable with any of this material you will likely not be successful with the class projects.

2. Linux

2.1. Background

Linux is an open-source Unix-like operating system, first released by Linus Torvalds in 1991 [Wikipedia (https://en.wikipedia.org/wiki/Linux)]. Ken Thompson and Dennis Ritchie are the main founders of the Unix operating system that was developed at AT&T Bell Labs in the 1970s [Wikipedia (https://en.wikipedia.org/wiki/Unix)]. Today Unix, Linux and other Unix-like operating systems are among the most widely used and influential operating systems.

2.2. Pyrite

The class projects will target the Linux operating systems, specifically, the projects will be graded on the pyrite server. If your code does not compile and execute on pyrite then it will be considered not working. You can develop on a local machine, but it is important to test on pyrite before submitting. Accessing the server requires being on a campus network or using VPN. Linux and Mac users can ssh into pyrite using a terminal, while Windows users can connect using putty. Information about VPN and putty can be found here (https://www.cs.iastate.edu/how-access-your-files-cs-file-server). Assuming Linux or Mac, login using:

```
> ssh username@pyrite.cs.iastate.edu
```

Transferring a file from your local machine to pyrite:

```
> scp ~/os-project1/shell.c username@pyrite.cs.iastate.edu:~/os-project1/shell.c
```

2.3. Linux Utilities

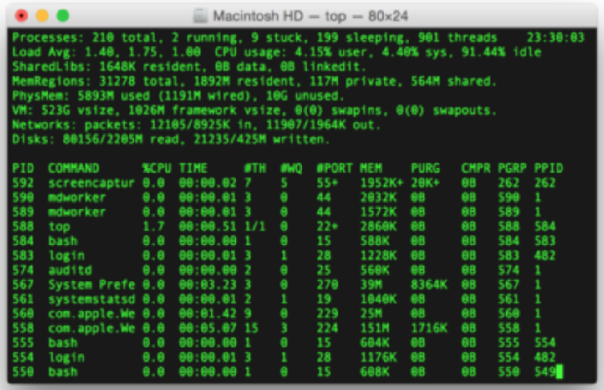
It would be good to familiarize yourself with a text editor (e.g., vim or nano) and basic utilities.

```
ls - list the contents of a directory
tree - list the contents of a directory in an expanded tree form
mkdir - make a directory
cd - change the current directory
cp - copy a file
rm - remove a file
rmdir - remove a directory
pwd - show the current path
ps - lists running processes, will be useful later in the class, typical usage: ps -A
man - read the manual for system programs, shell commands and function calls
grep - search for lines containing a regular expression
```

2.4. Terminal and Shell

In the first project we will implement a shell, so it is important to understand what exactly the terminal (or counsel) and shell are. Historically, a **terminal** is a device for user to computer input and output, an example is shown below.

Today we use virtual terminals like this.



A **shell** is a program that provides a command line interface (CLI) to an operating system. Although shells are typically accessed through a terminal, the terminal and shell are distinct entities. Examples of shells are bash, csh, ksh and zsh. The main purpose of the shell is to execute user and system programs. It is important to realize that ls, cd, rm etc. are just programs, they are not built in shell commands. This was the part of the genius of the UNIX operating system, rather than having one large monolithic shell, commands are simple (single-purpose) individual programs that can be easily strung together to perform complex tasks. The way we string commands together is through pipes, which stream the output of one command into the input of another command. For example, suppose we want to count how many files in the current directory are owned by the root user. One approach is to list all files in the directory, use cut to focus on just the owner of each file, use grep to find only the lines containing the word "root" and finally use wc to count the number of lines.

```
> ls -la * | cut -d ' ' -f 3 | grep root | wc -l
```

An excellent demonstration of the Unix philosophy can be found **here** **(https://www.youtube.com/watch?v=XvDZLjaCJuw)**.

3. C Programming

3.1 Background

C was developed at Bell Labs by Dennis Ritchie between 1972 and 1973 for the purpose of making Unix utilities and it was used to re-implement the Unix kernel [**Wikipedia** **(https://en.wikipedia.org/wiki/C_(programming_language))**]. C quickly became a dominate language because it enabled high performance with direct control of memory and machine specific features while still providing

high-level programming abstractions. Today, C and C++ are still widely used in systems programming, most operating systems, compilers, system libraries and virtual machines (e.g., Python's PVM, Java's JVM, C#'s CLR...) contain C and C++ code.

3.2 Communication

Because of C's close association to Unix, it was designed to follow the Unix philosophy by providing easy access to command line arguments and pipes. Command line arguments are passed to main as an array of strings.

```
int main(
    int argc /* number of white space separated tokens on the command line */,
    char *argv[] /* array of pointers to each white space separated token on command line */
)
```

Output, for example printf, is directed to the the text stream stdout. Input, for example fgets, reads from stdin. The pipe in most Unix shells connects the stdout of one program to the stdin of another program. The default piping of stdout and stdin can be easily changed, for example, stdin can be directed to read from a file like this:

```
freopen("in.txt", "r", stdin);
```

3.3 Make and GDB

This **Lab Tutorial** **(http://pages.cs.wisc.edu/~remzi/OSTEP/lab-tutorial.pdf)** covers the basics of C programming in a UNIX environment. In this class you will be expected to provide a simple make file (no more complicated than the one shown in the tutorial) with your projects. It would be a good idea to familiarize yourself with using gdb to debug C programs.

Here is a simple makefile that would be appropriate for the purposes of the projects in this class.

```
# First target is default, this is what the graders will use.
build:
    gcc -o pass pass.c
debug:
    gcc -o pass -g pass.c
clean:
    rm pass
```

3.4 Valgrind

Another important debugging tool, not described in the linked tutorial, is valgrind. Because C has no memory protections bugs can cause misleading errors. I had a student who couldn't figure out how to fix their multi-theaded code, depending on where they placed calls to the thread library the program would mysteriously crash. After hours of trying every way to fix the threads they asked for help, the error had nothing to do with threads. Can you find the error in the following code?

```
#include <stdlib.h>
int* values;

void foo(void) {
    values = malloc(10 * sizeof(char));
    values[5] = 0;
}

int main(void) {
    foo();
    // rest of program...
    return 0;
}
```

There are actually potentially two errors. We compile the program and run valgrid like this (compiling with -g helps valgrind report line numbers):

```
> gcc -o myprog -g myprog.c
> valgrind --leak-check=yes ./myprog

==397028== Invalid write of size 4
==397028==    at 0x401146: foo (myprog.c:7)
==397028==    by 0x401157: main (myprog.c:11)
==397028==  Address 0x4a32054 is 10 bytes after a block of size 10 alloc'd
==397028==    at 0x483A809: malloc (vg_replace_malloc.c:307)
==397028==    by 0x401133: foo (myprog.c:6)
==397028==    by 0x401157: main (myprog.c:11)
==397028==
==397028==
==397028== HEAP SUMMARY:
```

```
==397028==     in use at exit: 10 bytes in 1 blocks
==397028==   total heap usage: 1 allocs, 0 frees, 10 bytes allocated
==397028==
==397028== LEAK SUMMARY:
==397028==    definitely lost: 0 bytes in 0 blocks
==397028==    indirectly lost: 0 bytes in 0 blocks
==397028==      possibly lost: 0 bytes in 0 blocks
==397028==    still reachable: 10 bytes in 1 blocks
==397028==         suppressed: 0 bytes in 0 blocks
==397028== Reachable blocks (those to which a pointer was found) are not shown.
==397028== To see them, rerun with: --leak-check=full --show-leak-kinds=all
```

Valgrind is telling us that on line 7 we have a buffer overflow. 5 seems as though it should be in range, so clearly we have not allocated the array correctly, in this case we calculated the array size based char (1 byte) but the array is indexed by int.

The second issue is that we have not freed as much memory as has been allocated, which means there is a memory leak.

Here is a fixed version of the code.

```
#include <stdlib.h>
int* values;

void foo(void) {
    values = malloc(10 * sizeof(int));
    values[5] = 0;
}

int main(void) {
    foo();
    // rest of program...
    free(values);
    return 0;
}
```

4. Data Structures in C

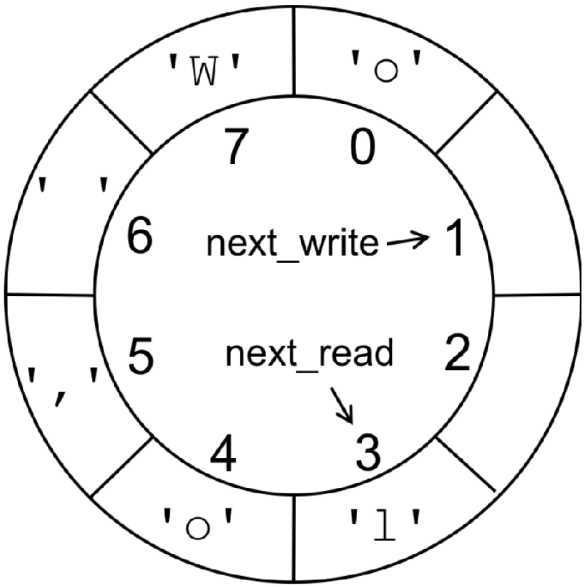
Data structures are often learned in a theoretical context with exercises using a memory-managed programming language. Using C

to implement data structures forces us to think more carefully about the performance cost of data structure implementations. Both linked list and ring buffer can be used to implement a queue, but the linked list requires dynamic memory operations and can result in thrashing (something we will learn about in Module 9). As another example, a heap is an efficient way to implement a priority queue, but in an embedded operating system that typically has few processes, it may be better to implement a priority queue using a sorted list.

Operating systems are among the most complicated software every created, so it is not surprising that they use many types of data structures, but there are a few data structures that are particularly useful for this class.

Abstract Data Structure	**Implementation**	**Common Usage**
queue	ring buffer	device drivers, I/O buffering
	ping-pong buffer	device drivers, DMA
	linked list (single, double and circular)	process scheduling
priority queue	heap	process scheduling
stack		call stack
table		memory management
tree		file system directory structure

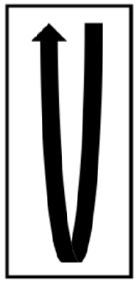
A ring buffer uses a fixed contiguous block of memory, it is commonly used for buffering I/O.



The ring buffer results in sequential memory accesses, this is better for performance than the potentially random accesses of a linked list. Random access can lead to a condition know as thrashing, a sever memory condition we will investigate in Module 9 of this class.

Typical Memory Access Pattern

Ring Buffer
(sequential)



Linked List
(random)