CSC 360: Operating Systems

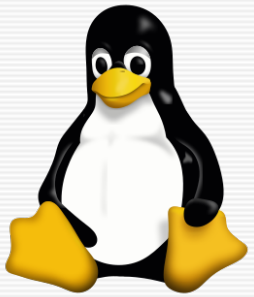
Tutorial #1

What Are Some Common Operating Systems?

What are Operating Systems?

- An operating system is a program that provides an easy-to-use and efficient platform for the execution of user programs
- It manages other program's access to the system hardware
- There is no universal definition and there are many costs and tradeoffs associated with OS design
- The kernel is used to refer to the program that is always running on the system
- Associated with the kernel may be other system programs and application programs

Programming Environment



- All assignments will be graded on the linux.csc.uvic.ca computers
- I recommend developing on Vim through the terminal. It's a good skill to learn.
- On Linux: ssh username@linux.csc.uvic.ca
- On Mac: ssh username@linux.csc.uvic.ca
- On Windows: Use Putty or something similar
- No Mercy if your code doesn't run on linux.csc.uvic.ca.
- https://www.uvic.ca/engineering/ece/faculty-and-staff/home/computing/remote-access/index.php
- http://www.viemu.com/a_vi_vim_graphical_cheat_sheet_tutorial.html

Why Program in C?

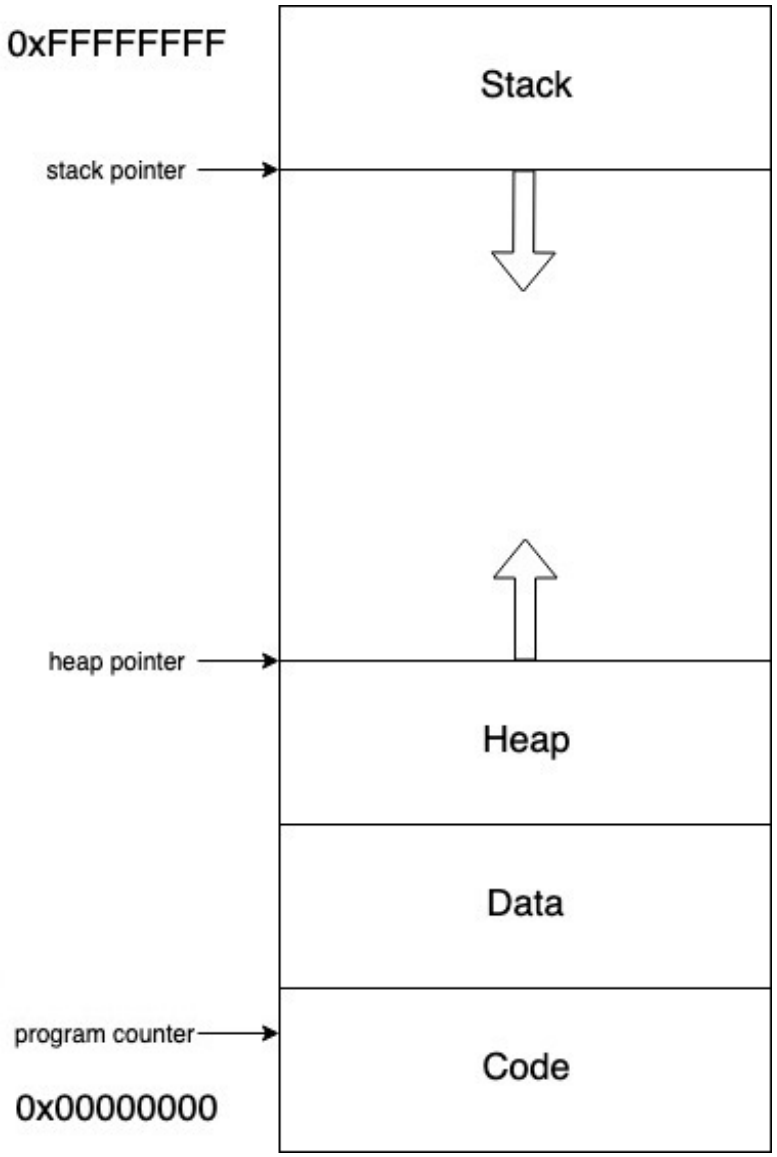
- The language was developed in order to create UNIX
- Most operating systems and systems software are still written in C
- Direct access to memory (pointers, addresses etc.)
- Relatively compact and simple syntax and grammar
- Forms the basis for other languages such as C++, C#, Java
- Your assignments in this class give you insight into the design of these early operating systems and the choices that were made
- You will be simulating what it was like to work on systems in the olden days of the 1970s and 1980s to further your understanding

The Basics of C

- The core C language is very small. It's like a more readable way of writing assembly code
- The language must be turned into machine instructions by a compiler
- Primitive datatypes – char, int, float, double, short, long etc. (signed/unsigned)
- Loops, functions, if-else, switch etc.
- Arrays store an aggregate of one datatype sequentially in memory.
- System calls are used to request the OS to perform actions on our behalf
- In most cases these syscalls aren't called directly but are encapsulated inside API functions
- APIs give us pre-written functions, so we don't have to rewrite basic tools from scratch
- The POSIX (Portable Operating System Interface) provides a set of standard libraries for UNIX based operating systems. On Windows, the Win32 API is used.

Process Address Space

- A process is an instance of a program that is in execution on the system
- The operating system allocates each process its own address space in primary storage (RAM)
- The program's instructions are in the code segment
- Global and static variables are stored in the data section
- The stack grows and shrinks as values are pushed and popped from the stack during function calls. Variables created inside of functions are stored here.
- The heap stores data that is allocated at runtime by invoking system calls to `malloc()` or `alloc()`
- What mistake might cause a stack overflow to occur?



Memory Allocation

- Variables that are declared outside of any functions are included in the data section of the address space for a process. They are “static”. You declare how much space you need before the program runs.
- When variables are passed to a function in C they are passed by value. Pointers allow for passing by reference, allowing functions to change the value of the original variable.
- These argument variables are pushed onto the stack at the beginning of a function call and popped from the stack when the function returns. This is the reason for local scope. We should never create a variable inside of a function and pass back a pointer. This will point to a stack location that may change.
- It is possible you have input data in a program, but you don’t know how much space it will use…
- You can dynamically allocate memory on the heap using the malloc() and alloc() system calls. It must be deallocated when no longer needed using free(). If you don’t you may get a stack and heap collision when you run out of room…
- This means you request the operating system to provide memory at runtime.

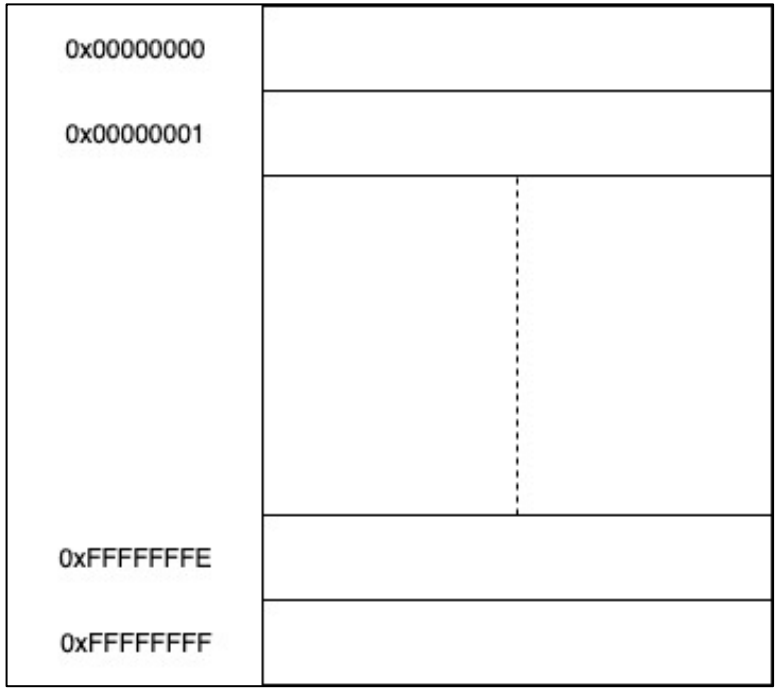
Structures

- Structs aggregate the storage of multiple data items, of potentially differing data types, into one memory block referenced by a single variable.
- Process control blocks (PCB) can be implemented as a structure
- Some people think of structures as like a class in Java but without any methods. i.e. It's just a container for different types of data that can be get and set.
- We could implement a database using structs…

```c
#include <stdio.h>
#include <stdlib.h>

struct university {
  char *name;
  int year_established;
};

typedef struct node { //typedef allows reference by "node" type
  int data;
  struct node *next;
  struct node *prev;
} node;


int main()
{
  //Have to refer to struct university each time...
  //This is stack allocation
  struct university UVIC = { .name = "University of Victoria", .year_established = 1963 };

  //Stack allocation for a struct
  //We use dot notation to access variables
  node my_node1 = { .data = 5, .next = NULL, .prev = NULL};

  //Heap allocation for a struct - we get a pointer to memory location
  //We use arrow notation to access variables
  node *my_node2 = malloc(sizeof(node));
  my_node2->data = 6;
  my_node2->prev = &my_node1;
  my_node2->next = NULL;

  my_node1.next = my_node2;
  printf("my_node1 data: %d next: %p prev: %p\n", my_node1.data, my_node1.next, my_node1.prev);
  printf("my_node2 data: %d next: %p prev: %p\n", my_node2->data, my_node2->next, my_node2->prev);
}
```

Pointers

Memory is just an array…



- A pointer is a variable which stores an address to a datatype

  `int *num;` num is a variable which stores an address to an integer

  `char *p;` p is a variable which stores an address to a character

- We can get the address of a variable using & "& for address"

  `int c = 5;` c is assigned the value of 5

  `num = &c;` num is assigned the address stored in c

  Now num points to the same address as c

- We can get the value stored at an address by "dereferencing" the pointer

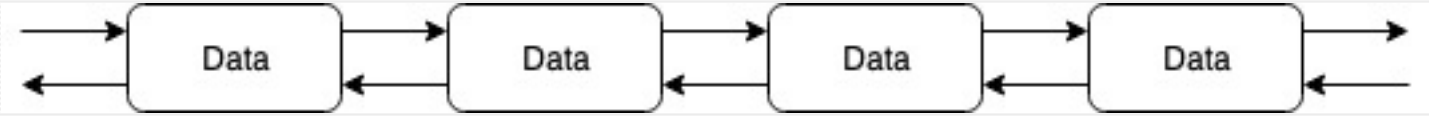
  `printf("The value stored at address %p is: %d\n", num, *num);`

  `The value stored at address 0xD1FB884A is: 5`

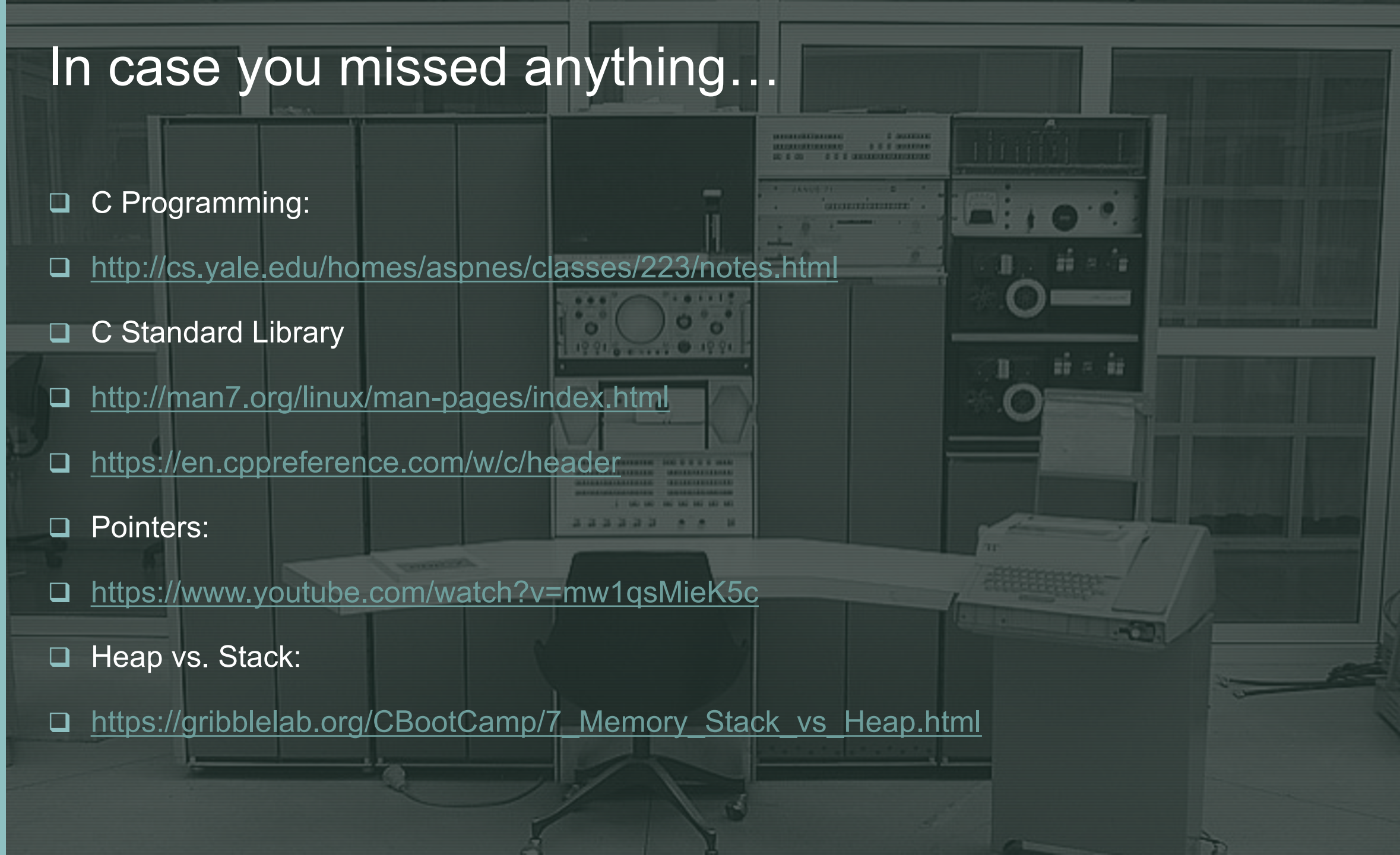
  Don't be confused by the use of the * symbol when both creating a pointer and dereferencing a pointer.

Linked Lists

- Many operating system components are implemented using simple data structures like queues, stacks and linked-lists. For example, the scheduler may deal with queues of processes.
- Linked Lists are used to store data that is non-contiguous in memory. This means that unlike an array, the nodes can be stored in different memory locations instead of sequentially.
- Each node in the list consists of some data and a link to the next node.
- This allows for dynamic (at runtime) addition and deletion of nodes in a list.
- In C we can implement nodes as *structs* and the links as *pointers to a struct*.
- Your warm-up assignment is to implement a linked list in C. This is to help you solidify your understanding of pointers and C programming in general.



In case you missed anything…



- C Programming:
- http://cs.yale.edu/homes/aspnes/classes/223/notes.html
- C Standard Library
- http://man7.org/linux/man-pages/index.html
- https://en.cppreference.com/w/c/header
- Pointers:
- https://www.youtube.com/watch?v=mw1qsMieK5c
- Heap vs. Stack:
- https://gribblelab.org/CBootCamp/7_Memory_Stack_vs_Heap.html