CSCI 4061: Introduction

Chris Kauffman

Week 1

Logistics

Goals Today

- Motivation
- Unix Systems Programming
- C programs
- Course Mechanics

In and Out of Class

- Common Misconception: Everything you need to know happens in lecture
- Truth: Much of what you'll learn will be when you're reading and doing things on your own

Reading

EITHER

Robbins and Robbins, Unix Systems Programming Ch 1

- Official textbook
- A bit harder to read at times
- Will go somewhat in order of chapters

OR

Stevens and Rago, Advanced Programming in the UNIX Environment Ch 1

- Optional textbook, similar coverage
- Somewhat more readable
- Will go somewhat out of order

Plethora of Operating Systems

What is the job of the operating systems?

What do all these have in common?

Responsibilities of the OS?

Create a "virtual machine" on top of hardware

- OS creates an abstraction layer
- Similar programming interface regardless of underlying hardware environment:
- Phones, Laptops, Cars, Planes, Nuclear Reactors
- all see Processes, Memory, Files, Network

Enforce Discipline / Referee

- Limit damage done by one party to another
- Processes communicate along fixed lines
- Multiple users must explicitly share info
- Shared resources are managed

Why Unix?

Which of theses is NOT Unix-like?

Unix is Popular

Just because it's popular, doesn't mean it's good. However, Unix is pretty great.

Unix is Old, Tested, and often Open

- Developed from the 70s, honed under pressure from academia and industry for widely varying uses
- Among the first projects to benefit from shared source code
- Philosophy: Simple, Sharp tools that Combine Flexibly
- Keep the Kernel functionality small but useful
- Abstractions provided in Unix are well-studied, nearly universal

The Unix "Virtual" Machine

Unix Kernel provides basic facilities to manage its high level abstractions of hardware, translate to actual hardware

- Link: Interactive Map of the Linux Kernel
- Examples Below

Processes: Executing Code

- Create new processes
- Status of other processes
- Pause until events occur
- Create/Manage threads within process

File System: Storage / Devices

- Create / Destroy Files
- `read()` / `write()`
- Special files for communication, system manipulation

Process Communication

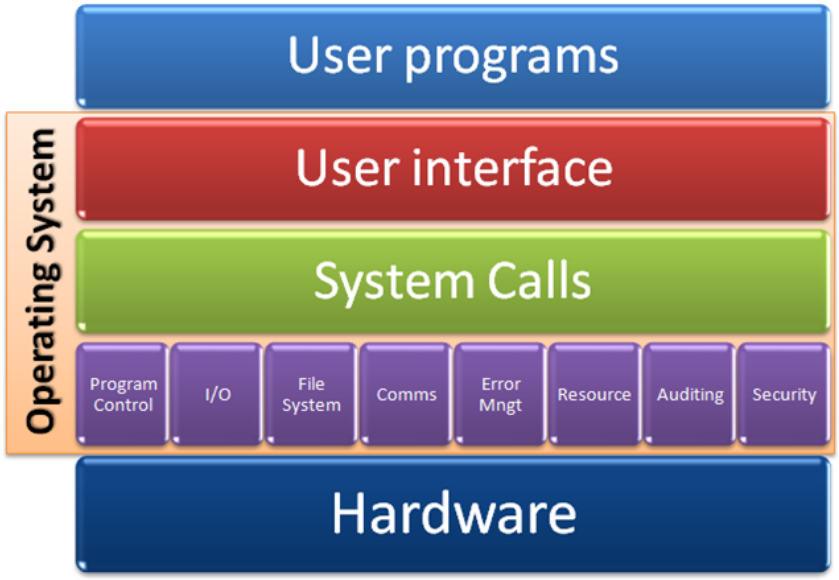
- Messages between processes
- Share memory / resources
- Coordinate resource use

Networking

- Open sockets which connect to other machines
- `send()`/`recv()` data over

The Outsides of the OS vs the Insides

- Operating Systems are layered like everything else in computer science
- 4061: outer layer
- 5103: inner layers
- EE Degree: bottom layer



CSCI 4061

- Systems Programming
- Use functionality provided by kernel
- Gain some knowledge of internals but focus on external practicalities

CSCI 5103

- Creation of a kernel / OS internals
- Theory and practice of writing / improving operating systems
- Implement system calls

System Calls : The OS's Privilege

- User programs will never actually read data from a file
- Instead, will make a request to the OS to read data from a file
- Usually done with a C function like in

  ```
  int nbytes_read = read(file_des, in_buf, max_bytes);
  ```

- After a little setup, OS takes over
- Elevates the CPU's privilege level to allow access to resources not normally accessible using assembly instructions
  - Modern CPUs have security models with normal / super status
  - Like `sudo make me a sandwhich` for hardware
- At completion of `read()` CPU drops back to normal level
- User program now has stuff in `in_buf` or an error to deal with
- Same for process creation, communication, I/O, memory management, etc.

Question: Why do it this way?

Distinction of Application vs Systems Programming

> *The primary distinguishing characteristic of systems programming when compared to application programming is that application programming aims to produce software which provides services to the user directly (e.g. word processor), whereas systems programming aims to produce software and software platforms which provide services to other software, are performance constrained, or both.*
>
> *System programming requires a great degree of hardware awareness. Its goal is to achieve efficient use of available resources, either because the software itself is performance critical (AAA video games) or because even small efficiency improvements directly transform into significant monetary savings for the service provider (cloud based word processors).*
>
> – *Wikipedia: Systems Programming*

In short: systems programmers write the code between the OS and everything else. *But*, systems vs application is more of a continuum than a hard boundary.

General Topics Associated with Systems Programming

Concurrency	Multiple things can happen, order is unpredictable
Asynchrony	An event can happen at any point
Coordination	Multiple parties must avoid deadlock / starvation
Communication	Between close entities (threads/processes) or distant entities (network connection)
Security	Access to info is restricted
File Storage	Layout of data on permanent devices, algorithms for efficient read/write, buffering
Memory	Maintain illusion of a massive hunk of RAM for each process (pages, virtual memory)
Robustness	Handle unexpected events gracefully
Efficiency	Use CPU, Memory, Disk to their fullest potential as other programs are built from here

In our projects, we'll hit on most of these.

Assumption: You know some C

- CSCI 2021 is a prereq, covers basic C programming
- Assume that you know syntax, basic semantics

Why C vs other languages?

Computers are well-represented in C

> *You just have to know C. Why? Because for all practical purposes, every computer in the world you'll ever use is a von Neumann machine, and C is a lightweight, expressive syntax for the von Neumann machine's capabilities.*
>
> *–Steve Yegge, Tour de Babel*

C and Unix Go Way Back

> *Aside from the modular design, Unix also distinguishes itself from its predecessors as the first portable operating system: almost the entire operating system is written in the C programming language that allowed Unix to reach numerous platforms.*
>
> *– Wikipedia: Unix*

Exercise: Recall these C things

- Two different syntaxes to loop
- Stack arrays
- The meaning of `void`
- `struct`: aggregate, heterogeneous data
- `malloc()` and `free()`
- Pointers to and Address of variables
- Dynamically allocated arrays and `structs`
- `#define` : Pound define constants
- Local scope, global scope
- Pass by value, pass by reference
- `printf()` / `fprintf()` and format strings
- `scanf()` / `fscanf()` and format strings
- Commands to compile, link, execute

Exercise: Actual C Code

```
#include <stdio.h>
#include <stdlib.h>
int main(){
  long n = 1;
  void *mem = NULL;
  while( (mem = malloc(n)) != NULL){
    printf("%12ld bytes: Success\n",n);
    free(mem);
    n *= 2;
  }
  printf("%12ld bytes: Fail\n",n);
  n /= 2;

  long kb = n / 1024;
  long mb = kb / 1024;
  long gb = mb / 1024;

  printf("\n");
  printf("%12ld b  limit\n",n);
  printf("%12ld KB limit\n",kb);
  printf("%12ld MB limit\n",mb);
  printf("%12ld GB limit\n",gb);
  return 0;
}
```

- Describe at a high level what this C program does
- Explain the line

  `while( (mem = malloc(n)) != NULL){`

  in some detail
- What kind of output would you expect on your own computer?

Exercise: C Program with Input

```
typedef struct int_node_struct {
  int data;
  struct int_node_struct *next;
} int_node;
int_node* head = NULL;

int main(int argc, char **argv){
  int x;
  FILE *input = fopen(argv[1], "r");
  while(fscanf(input,"%d",&x) != EOF){
    int_node *new = malloc(sizeof(int_node));
    new->data = x;
    new->next = head;
    head = new;
  }
  int_node *ptr = head;
  int i=0;
  printf("\nEntire list\n");
  while(ptr != NULL){
    printf("list(%d) = %d\n",i,ptr->data);
    ptr = ptr->next;
    i++;
  }
  fclose(input);
  return 0;
}
```

- What data structure is being used?
- Are there any global variables?
- What's going on here:

  ```
  new->data = x;
  new->next = head;
  ```

- Where do input numbers come from?
- In what order will input numbers be printed back?
- Does the program have a memory leak? (What is a memory leak?)

Answers: On the Course Site

- Canvas has links to course materials
- Lecture slides will be available either before lecture or soon after
- Code we use in class will also be available
- Take your own notes but know that resources are available

Course Mechanics

See separate slides for specific course mechanics