CS 354 Machine Organization & Programming

Course Description:

An introduction to fundamental structures of computer systems and the C programming language with a focus on the low-level interrelationships and impacts on performance. Topics include the virtual address space and virtual memory, the heap and dynamic memory management, the memory hierarchy and caching, assembly language and the stack, communication, and interrupts/signals, compiling and assemblers/linkers.

Learning Objectives:

- Implement and interpret C programs using standard tools and relate C language constructs to both x86 assembly language and fundamental computer system structures.
- Differentiate the memory segments of a process's virtual address space and explain how each is used in C programs.
- Describe and diagram how the heap works internally, analyze the performance of heap allocation strategies, and implement in C a dynamic memory manager.
- Demonstrate how the memory hierarchy functions, differentiate different memory cache implementations, and appraise the effects of memory caching on the performance of C programs.
- Diagram a stack trace of execution for C programs and explain how the compiler implements the stack using x86 assembly language code.
- **revised: Formulate C programs and explain the underlying mechanisms that enable asynchronous execution.** ECE Proposed change and **Deb said yes on 4/21/22**
  From: Formulate C programs that send and receive signals, respond to exceptional circumstances, and explain the underlying mechanism that enables asynchronous execution.
- Identify and summarize the steps of C program compilation and describe the processes of linking object code modules to form an executable and loading executables to run them.

Textbooks:

- Required: Computer Systems: A Programmer's Perspective, Bryant and O'Hallaron, 2nd Edition (or 3rd Edition)
- Recommended: C Programming Language, Kernighan and Ritchie, 2nd Edition

Main Topics:

- C Programming (focus on pointers and address arithmetic)
- Virtual Memory and C's Abstract Memory Model
- Heap (dynamic memory allocators)
- Memory Hierarchy and Caching
- x86 Assembly and the Stack
- Exceptions and Signals
- Linking and Loading

Detailed Topics

- C Programming
  - Compiling (preproc->compiler->assembler->linker)
  - Programming Basics (variables; control structures; functions; basic I/O)
  - Static vs. Dynamic Allocation (arrays, structs, combined; strings; pointers, address arithmetic, and caveats; memory diagrams)

    - Command Line Arguments
- **Virtual Memory**
    - C Memory Areas (characteristics, what goes where)
    - Process View (virtual addresses and virtual address space)
    - Hardware View (memory hierarchy, physical addresses, and address space)
    - System View (page table - mapping virtual to physical addresses/pages)
- **Dynamic Memory (Heap)**
    - C's Allocator (malloc, calloc, realloc, free)
    - Program Break (unistd.h: brk, sbrk)
    - Allocator Design (goals/requirements, block/payload/overhead, allocation steps, fragmentation, headers/footers)
    - Placement Policies (first fit, next fit, best fit)
    - Free Lists (implicit vs explicit, splitting/coalescing, address, and LIFO ordering, simple and fitted segregation)
    - Memory Layout and Diagrams
- **Memory Hierarchy (Cache)**
    - Hierarchy (registers through network storage)
    - Locality (spatial, temporal)
    - Cache Operation (hit, miss kinds, victim, working set, ...)
    - Designing a Cache (C = (S,E,B,m); address breakdown; valid and dirty bits, tags and lines)
    - Direct Mapped, Set Associative, Fully Associative (replacement policies - LFU/LRU) Writing (write back/write allocate vs. write through/no write allocated)
    - Metrics and Performance
    - Coding for Caches
- **Assembly Language Part I (basics and instructions)**
    - compiling (c->assembly->machine code)
    - data formats (suffixes)
    - registers (pc, gen purpose, condition codes)
    - operand formats and types (addressing modes & effective addresses)
    - instructions:
        - data movement (vs. leal), NOTE operand order S,D
        - arithmetic and shift (including leal)
        - comparison and test (condition codes, set instruction), NOTE operand order S2,S1
        - unconditional and conditional jumps (absolute and relative encoding)
- **Assembly Language Part II (the stack and composites)**
    - function calling - the stack
    - param passing in caller's frame, return value in %eax
    - transfering control (call, leave, ret instructions)
    - stack frame (%esp, %ebp)
    - saving registers (caller vs. callee saved)
    - recursion
    - allocating and accessing composites on the stack
    - 1D and 2D arrays, structures, unions
    - alignment
    - pointers (including function pointers)
    - hacking code and stack smashing with buffer overflows
- **Exceptions and Signals**
    - exception table (exception number) and control flow (events, processor state)

- exceptions - interrupts (asynchronous), traps (system calls), aborts, halts
- processes (pids & pgids), user/kernel modes, context switching & concurrent execution
  signals
  - sending - kill(), alarm()
  - delivering - pending, blocking, bit vectors
  - receiving - signal handlers & sigaction()
  - multiple signal issues

- **Linking and Loading**
  - forward declaration (declaring vs. defining)
  - multifile coding (header & src files), compilation (makefiles)
  - static vs. dynamic linking
  - relocatable object files (ELF format)
  - symbols, symbol tables
  - symbol resolution (extern, static, strong vs. weak globals)
  - relocation
  - executable object files (ELF format)
  - loading and execution