COMP 3280 – Computer Systems

Course Description

Calendar entry

The design and use of computing machinery. May not be held with the former COMP 3370. Prerequisites: [COMP 2080 and COMP 2280] and [one of STAT 1150, STAT 2000 (B), STAT 2001 (B), STAT 2220, or PHYS 2496].

General Course Description

This course builds on the memory and CPU organizations introduced in second year by investigating advanced computer architectures that go beyond a simple Von Neumann machine. Rather than being able to design computer hardware, this course aims to make students better software developers through a deeper understanding of how computer organization affects code.

We will look at techniques that try to maximize the amount of concurrent and parallel work a computer can do within a CPU, across multiple levels of memory, and through independent input/output operations. All of these techniques not only have a computer doing multiple things at the same time; they all change how and *when* code is executed. What you expect a computer to do with code you write is *not* what the computer will do with those instructions. How does this happen in a way that guarantees correct code execution? How does this change what is meant by good code?

Detailed Prerequisites

Before entering this course, a student should be able to:

- Demonstrate how the code they write gets interpreted and executed by a computer.
- Convert high-level code, including code representing data structures and recursive operations, into assembly.
- Implement common assembly provided by a compiler: including the run-time stack, parameter passing, local variables, and dynamic data structures.
- Implement data structures such as lists, stacks, and binary trees in a high-level language.
- Apply divide-and-conquer, greedy, randomized (including Monte Carlo and Las Vegas algorithms), and dynamic programming algorithmic design techniques.
- Analyze the worst-case running time of typical divide-and-conquer, greedy, randomized, and dynamic programming algorithms.
- Describe how memory is used and safely managed within code they write.
- Build a memory management system.

Course Goals

By the end of this course students will:

- See how memory organization can both positively and negatively impact the performance of their code.
- Gain an understanding of modern CPU design and how those design choices lead to our ability to implement advanced operating systems and sophisticated applications.
- Be prepared to apply a greater understanding of CPU/device interactions to the implementation of device drivers.
- Show that a given code segment does or does not correctly guarantee mutual exclusion.
- Implement atomic operations to provide mutual exclusion.
- Use semaphores to coordinate concurrent access to a shared buffer.

Learning Outcomes

Performance

Students should be able to:

1. Calculate the performance of a computer in terms execution time.
2. Compare the performance of two computers running the same code.
3. Analyze attempts to improve computer performance using cycles per instruction relative to the number of instructions and CPU cycle time.
4. Explain Amdahl's law.

Memory

Students should be able to:

1. Compare cache memory implementation strategies and their effectiveness at different levels of the memory hierarchy.
2. Analyze the effect of a caching strategy on a memory intensive algorithm.
3. Explain the architectural designs needed to support the virtual memory algorithms used by operating systems.
4. Explain how a process' entire address space can be provided/supported via the concept of virtual memory.
5. Translate virtual addresses into physical addresses.
6. Describe the algorithms that work with hardware to manage paging and virtual memory.

CPU Pipelining

Students should be able to:

1. Show how pipelining affects the execution of a sequence of instructions.
2. Implement simple branch prediction algorithms and show how each algorithm works to minimize control hazards for a given piece of code.
3. Show how a given piece of code would have its instructions reordered to minimize data hazards.
4. Explain the software and hardware design choices used to minimize structural hazards.

I/O

Students should be able to:

1. Use timing diagrams to describe synchronous bus transfer operations [maybe].
2. Use interrupts and memory mapped I/O to implement the transfer of blocks of data between a storage device and memory independent of the CPU.
3. Demonstrate the impact of multiple units accessing memory in parallel.

Synchronization

Students should be able to:

1. Identify the critical section(s) requiring mutually exclusive access in a piece of code that will be run concurrently.
2. Identify correct and incorrect implementations that attempt to protect a critical section.
3. Identify code protecting a critical section that would result in deadlock and propose solutions that avoid and/or prevent said deadlock.
4. Use atomic hardware instructions to build atomic operations such as a lock.
5. Write code that uses atomic operations to implement a semaphore.
6. Write code where concurrent threads of execution communicate and coordinate activity through shared memory.