COS 284
Introduction to Computer Systems

Department of Computer Science and Engineering
Taylor University

Three Credit Hours
Euler Science 200 • MWF 11:00–11:50

Final Exam: W/20-May-2020, 1:00–3:00

1 Instructor

Dr. Tom Nurkkala
Associate Professor, Computer Science and Engineering
Director, Center for Missions Computing

Office	Euler Science Complex 211	
Email	`tnurkkala@cse.taylor.edu`	
Phone	765/998-5163	
Hours	Mon, Wed	1:00–2:00
	Tue, Thu	10:00–12:00
	Or by appointment	

2 Description

Prerequisites: COS 121, MAT 215

From the catalog: An integrated introduction to computer hardware architecture, operating systems, and their interaction. Assembly language and operating system programming are emphasized.

3 Credit

This course is based on the Introduction to Computer Systems course at Carnegie Mellon University (CMU). Professors Randal Bryant and David O'Hallaron, who developed the course and wrote the textbook (Section 5), have generously made available course material (including notes, syllabi, laboratory exercises, and lecture slides) for use at other institutions. I make considerable use of their excellent materials throughout the course (including in this document!) and wish to credit and thank them.

I have also drawn liberally from previous offerings of this course at Taylor University, and wish to thank Professor Jonathan Geisler for his kind assistance.

4 Course Overview

The aim of the course is to help you become a better programmer by teaching you the basic concepts underlying all computer systems. I want you to learn what really happens when your programs run, so that when things go wrong (as they always do) you will have the intellectual tools to solve the problem.

Why do you need to understand computer systems if you do all of your programming in high-level languages? In most of computer science, we're pushed to make abstractions and stay within their frameworks. But, any abstraction ignores effects that can become critical. As an analogy, Newtonian mechanics ignores relativistic effects. The Newtonian abstraction is completely appropriate for bodies moving at less than $0.1c$, but higher speeds require working at a greater level of detail.

The following "realities" are some of the major areas where the abstractions you've learned in previous classes break down:

1. **An `int` isn't an integer; a `float` isn't a real**. Our finite representations of numbers have significant limitations, and because of these limitations we sometimes have to think in terms of bit-level representations.

2. **You've got to know assembly language**. Even if you never write programs in assembly, the behavior of a program sometimes cannot be understood based purely on the abstraction of a high-level language. Furthermore, understanding the effects of bugs requires familiarity with the machine-level model.

3. **Memory matters**. Computer memory is not unbounded. It must be allocated and managed. Memory referencing errors are especially pernicious. An erroneous updating of one object can cause a change in some logically unrelated object. Also, the combination of caching and virtual memory provides the functionality of a uniform unbounded address space, but not the performance.

4. **There is more to performance than asymptotic complexity**. Constant factors also matter. There are systematic ways to evaluate and improve program performance.

5. **Computers do more than execute instructions**. They also need to get data in and out and they interact with other systems over networks.

By the end of the course, you will understand these realities in some detail. As a result, you will have learned skills and knowledge that will help you throughout your education and career as a computer scientist.

5 Learning Objectives

By the end of the semester, you should be able to:

1. Describe how a computer represents integers internally.
2. Convert twos complement binary values to decimal.
3. Convert binary to decimal and hexadecimal.
4. Understand how a computer represents fractional values.
5. Convert a fractional decimal value to its IEEE single- or double-precision representation.
6. Read x86 assembly code.
7. Write simple x86 assembly functions.
8. Describe how a computer executes a function call including parameters, return values, and the stack.
9. Describe how compilers convert C code to x86 assembly.
10. Explain how various C data structures are accessed in x86 assembly.
11. Describe the motivation for a memory hierarchy.
12. Explain how a memory hierarchy works.
13. Understand how a cache is organized internally.
14. Create programs that take advantage of the memory hierarchy.
15. Explain how a linker works.
16. Contrast static and dynamic linking.
17. Explain what a system call is and how system calls work.
18. Explain the fork system call.
19. Explain the exec family of system calls.
20. Describe how hardware and the operating system cooperate to deal with exceptional situations.
21. Measure the performance of an application accurately.
22. Improve the performance of an application based on good measurement techniques.

23. Understand virtual memory and how it works.
24. Explain the benefits of virtual memory.
25. Explain the role of the translation look-aside buffer.
26. Describe the virtual memory organization of a process running with Linux on an x86 processor.
27. Explain how a dynamic memory allocator works.
28. Be familiar with garbage collection.
29. Be familiar with common memory bugs and how to avoid them.
30. Explain what concurrent computation is and how to achieve it.
31. Contrast process-based concurrency with I/O multiplexing-based concurrency.
32. Describe why synchronization is necessary.
33. Be familiar with how to achieve synchronization.

6 Texts

The text by Bryant and O'Hallaron [1] is required. We will refer to it as `CS:APP`. You are *encouraged* to purchase—or have readily available—the updated edition of "K&R," the classic reference on C.[2]

It is *very important* that you read carefully the assigned readings in the `CS:APP` book. You will get the most benefit if you read assigned passages *before* they are covered in class. A detailed guide to reading assignments can be found in the course schedule.

The textbook includes many practice problems throughout the body of each chapter. These are straightforward exercises that help you understand the material you have just read by using it *immediately.* For self-study, solutions for all practice problems appear at the end of each chapter.

In short, the best way to use the textbook to enhance your learning is as follows:

1. Read the assigned readings *before* the corresponding class.
2. Work the practice problems *immediately* as you encounter them in the text.
3. Check your work—and your understanding—with the answer key.

7 Topics

These are the topics that could potentially be covered in the course. It is unlikely that we will have time to cover all this material.

- Course Overview
- Bits and Bytes
  - Basics
  - Integers
  - Floating Point
- Machine Programming
  - Basics
  - Control
  - Procedures
  - Data
  - Advanced
- Program Optimization
- Memory Hierarchy
- Cache Memory
- Linking
- Exceptional Control Flow
  - Exceptions and Processes
  - Signals and Non-Local Jumps
- Virtual Memory
  - Concepts
  - Systems
- Dynamic Memory
  - Basic
  - Advanced
- Internetworking

  - Network Programming
  - Web Servers

- Concurrent Programming

  - Sychronization

8 Labs

The labs come directly from the textbook authors. They have a very nice system set up so that you can test and submit all your programs online and will already know how you are doing on the assignment prior to submission. This should make it very easy to know when you have a correct solution and when you need to keep working.

Please do not look for solutions online. Doing so constitutes cheating. The authors have introduced randomness into the assignments so solutions posted by others may be wrong anyway.

You will use Linux for all lab work. For some labs, you will be given a binary that works on the CSE machines, but should probably work on any recent vintage of Linux. For others, you are required to develop code for a Linux box. You may use any machine to develop, but *it must run correctly on the CSE machines.* You must check your work on the machines in the laboratories before submitting it.

Most low-level systems code is written in C, and you will be required to do the same. You will also be required to read, understand, generate, or otherwise fiddle with x86 assembly on a Linux box.

9 Evaluation

The grading scheme for the course will be announced in the next few days.

10 Course Expectations

Following are my expectations regarding the course.

10.1 Attendance

You are required to attend all class sessions. I will be in class each day, and I expect you to be there also.

In general, I am very understanding about students who must miss class due to a sanctioned Taylor activity, medical appointment, job interview, family emergency, and the like. If possible, let me know in advance that you will not be in class; I will work with you to arrange make-up instruction, homework, exams, etc.

10.2 Late Work

All course assignments will include an unambiguous due date. Usually, assignments are due at the beginning of class on the due date. If there are multiple sections of a class, the assignment is due at the beginning of the earliest such section. Barring exceptional circumstances like those mentioned in section 10.1, I expect your work to be submitted *on the due date*. Late work will *not* be accepted.

This policy on late work is intended to prepare you for real-world experience after graduation. In the marketplace, late work is not merely an inconvenience. Missing a deadline may alienate your customer, upset your manager, ruin your project, or terminate your employment! *Now* is the time to learn the self discipline and time management skills required to complete your work when it is due.

10.3 Conduct

I expect you to be prepared, awake, aware, and participatory during class. I will not hesitate to ask you to stand or move if you are distracted or sleepy.

I expect you to join in discussions, respond to questions from me and from your colleagues, and ask questions of me. I expect you to hold my feet to the fire if I am being unclear, unkind, or contradictory.

10.4 Gizmos

You may not use a laptop, tablet, or similar device to check e-mail, engage in social networking, surf the web, or any other activity not directly relevant to current classroom activity. If you use an electronic gizmo during class for legitimate academic purposes (e.g., note taking), be prepared to demonstrate relevant use on demand at any time.

11 Course Management

We use several systems to help manage the course and for on-line communication.

11.1 Email

Electronic mail is an official channel of communication between all members of the university community. You are responsible to check your email regularly (daily) for information related to the course.

11.2 Canvas

The Computer Science and Engineering department uses Canvas as our Learning Management System. The URL for Canvas is `https://canvas.cse.taylor.edu`.

You are responsible for checking Canvas regularly to keep up with assignment due dates and other announcements. For due dates, *the Canvas calendar is your friend*.

11.3 Slack

This course will use Slack for informal communication, Q&A, last minute announcements, jokes, and the like. Find the *TU CSE Student* slack team at `tucsestudents.slack.com`. Look there for a *channel* dedicated to the course.

12 Academic Integrity

As a student at an institution whose goal is to honor Christ in all that it does, I expect you to uphold the strictest standards of academic integrity. You must do your own work, cite others when you present their work, and never misrepresent your academic performance in any way. Violation of these standards stains the reputations of you as a student, Taylor as an institution, and Jesus as our Lord.

Every assignment should indicate clearly that it is either:

- An **individual** assignment, to be done *entirely by you*, without any direct participation from other students.
- A **group** assignment, to be done *collectively with a group*

Unless otherwise stated, assignments are **individual** assignments.

You are *always* welcome to get help from the instructor on *any* homework assignment or project, whether an individual or group assignment.

12.1 What Constitutes Academic Dishonesty?

For purposes of this course, the following are *non-exhaustive* examples of violations of academic integrity.

1. Sharing code or other electronic files by copying, retyping, looking at, or supplying a copy of a file from this or a previous semester.
2. Sharing written assignments or exams by looking at, copying, or supplying an assignment or exam.
3. Using another student's code. Using code from this or previous offerings of the class, from courses at other institutions, or from any other source (e.g., software found on the Internet).

4. Looking at another student's code. Although mentioned above, it bears repeating: looking at other students' code or allowing others to look at yours is academic dishonesty. There is no notion of looking "too much," since no looking is allowed at all.

12.2 What Does Not Constitute Academic Dishonesty?

In contrast, the following are *non-exhaustive* examples of activities that *do not* violate academic integrity.

1. Clarifying ambiguities or vague points in class handouts or textbooks.
2. Helping others use the computer systems, networks, compilers, debuggers, profilers, or other system facilities without regard to a particular assignment or project.
3. Helping others with high-level design issues.
4. Helping others with high-level (*not* code-based) debugging.
5. Using code provided by the instructor from the course web site or elsewhere.

12.3 From the Provost

Taylor's Provost[1] defines *plagiarism* as follows:

> In an instructional setting, plagiarism occurs when a person presents or turns in work that includes someone else's ideas, language, or other (not common-knowledge[2]) material without giving appropriate credit to the source. Plagiarism will not be tolerated and may result in failing this course, and may also result in further consequences.

The Provost goes on to say:

> Academic dishonesty constitutes a serious violation of academic integrity and scholarship standards at Taylor that can result in substantial penalties, at the sole discretion of the University, including but not limited to, denial of credit in a course as well as dismissal from the University.
>
> In short, a student violates academic integrity when he or she claims credit for any work not his or her own (words, ideas, answers, data, program codes, music, etc.) or when a student misrepresents any academic performance.

For more information, see the Taylor University Undergraduate Catalog.

[1] At Taylor, the *Provost* is our Chief Academic Officer.

[2] *Common knowledge* means any knowledge or facts that could be found in multiple places or as defined by a discipline, department, or faculty member.

13 Support Services

Be aware of the following support services available to you as a Taylor student.

13.1 Academic Assistance

The Academic Enrichment Center (AEC), located in the Zondervan Library, provides individualized academic skills help (e.g. test preparation, note taking, planning, etc.). Contact **Dr. Scott Gaier**, `scgaier@taylor.edu`.

13.2 Tutoring

Peer Tutoring Services, located in the AEC in Zondervan Library, provides free help to students in most content areas. For further information, contact **Darci Nurkkala**, drnurkkala@taylor.edu.

13.3 Students with Special Needs

The Academic Enrichment Center provides a variety of services for students who have physical or other disabilities. If you need accommodations due to a disability, please contact **Ken Taylor**, `kntaylor@taylor.edu`.

References

[1] Randal Bryant and David O'Hallaron. *Computer Systems: A Programmer's Perspective*. 3rd ed. Pearson, 2015. ISBN: 978-0134092669.

[2] Brian Kernighan and Dennis Ritchie. *The C Programming Language*. 2nd ed. Prentice-Hall, 1988. ISBN: 978-0-13-110362-7.

This document last updated February 1, 2020.