Ense352–Course Outline–Fall 2022

`Version: $HGdate: Mon, 29 Aug 2022 12:17:33 -0600 $`

Vital Info

Instructor	Karim Naqvi
Office	ED479
Phone	337.2279
Email	karim.naqvi@uregina.ca
URCourses site	`https://urcourses.uregina.ca`
Office Hours	Mon/Wed, 15h00-1630
Lectures	ed106.1, Mon/Wed, 11h30–12h45
Labs	092: ed485.1, Tue 08h30–11h15
	091: ed485.1, Thu 08h30–11h15
Midterm	Wed 2022-11-02 Room: lecture theatre
Final Exam	Mon 19 Dec 2022, 09h00-12h00. Room: TBA

Course

Ense352 — Fundamentals of Computer Systems Architecture[1]

Description

The goal of this course is to give you knowledge of how computer systems work at a very low level. Modern hardware and software is so complex that becoming an expert in even a small subset of the field requires years of intense effort. This inescapable complexity has a huge downside: it makes it really easy to build powerful systems which you, the builder, do not understand. The results:

- you can't understand, much less evaluate, the engineering tradeoffs you are making
- the systems may underperform, or may be too expensive
- the systems are fragile and full of bugs
- the systems cannot be debugged easily

[1] See the calendar description.

Everything in this course is focused on increasing your understanding of the tools you use. We'll focus on several, carefully selected topics, which I've listed below. On this journey, we'll be taking the programmer's perspective, rather than that of becoming a domain expert. That is, we'll try to maximize the benefit to you *as a programmer* (or if you're not a programmer, as a professional deeply involved with computer systems).

This course may also serve as a foundation for later, in-depth, electives such as computer architecture, computer design, compilers, systems programming, operating systems, embedded systems, high-performance computing, low-power computing, software tooling, etc.

Course Texts

- *Computer systems: a programmers perspective/3e*, R. E. Bryant and D. R. OHallaron, Pearson Education Inc., 2015. You can rent or buy this book from vitalsource.com; although I have not used this service, apparently it works. The previous edition (2e) is also acceptable, although chapter 6 (The Memory Hierarchy) differs between 2e and 3e.

- *The definitive guide to the Arm Cortex-M3 and Cortex-M4 Processors/3e*, Joseph Yiu, Elsevier, 2014. This text is also extensively used in enel351 and enel452. You can read the ebook online after signing in with your `uregina.ca` credentials. The previous edition (2e) is also fine.

Course Topics

- Instruction set architecture: Machine instructions, data types, register set. Our focus is on the ARM Cortex-M3, less so on x64 and ia32. We'll also take a simplified look system architecure.

- The compiler toolchain: The translation steps from "high-level" C/C++ to assembly language and to an executable program.

  - We will use the Linux command-line interface (CLI) as a way to peer beyond the veil to see (and invoke) these steps in isolation, as well automate them using `make`.

- Bits and Bytes: Representation, storage, access, and manipulation of data in assembly language and C: numbers, pointers, strings, code, etc.

- Integer representation and arithmetic

- Floating point representation and arithmetic

- Writing assembly language programs for the Cortex-M3 architecture
- Device I/O: interfacing with the real world
- Machine-level control flow: if-else, loops, switch statements
- Machine-level procedures: stack, procedure call and return, recursion
- Machine level aggregate data access: Arrays and Structures
- The memory hierarchy and the crucial role played by caching
- Linking
- Virtual memory.
- Comparing C and Java. Virtual machines

Some of the later topics may be elided if time does not permit their inclusion.

Evaluation

The evaluations will consist of quizzes, assignments, labs, project, midterm exam, and final exam. You must pass the exam component to pass the class. (*i.e.* the midterm and final marks collectively must be no less than 50%)

Assign.	10%	
Quizzes	0%	
Lab	20%	
Project	20%	
Midterm	15%	
Final	35%	
Total	100%	
Instructor's Discretion	±5%	

Students are being graded and assigned a mark based not merely on the materials handed in but on their ability and knowledge they have demonstrated in these materials. The work submitted and the mark received is only valid if it is based upon this ability and knowledge.

In all cases you may be asked to demonstrate in person that the work you turned in is your own, and that you have the knowledge to solve the problems that have been assigned for any portion of the class. Regardless of the quality and correctness of your written work, your grade depends ultimately on your ability to demonstrate your understanding in person. From time to time you may be asked to demonstrate this in person.

Policies

Assignments

Individual work is required for all assignments and projects. This means you should not share code, or discuss detailed specifics of your solutions with each other.

Handing in late work

In general the penalty for handing in any late work is 20% per day, up to a maximum of 2 days (i.e. 48 hours). This rule will be applied to assignments, labs, and the project. After 2 days the mark will be zero.

I will allow up to 3 "allowed late" days over the semester. The use of a late day allows you to submit an assignment or project component up to 24h late with no consequent penalty. The allowed late will come into effect automatically if you submit an assignment late; you need not request this in advance. Even with the use of allowed late days, you must adhere to the 2 day limit on any hand-in item; you are not allowed to go beyond this threshold as it would restrict discussing the solution details in class.

The allowed late days are there to help you deal with "crunch" times during the semester, or illness. Use them wisely and don't waste them because additional extensions of deadlines will not be granted.

Long-term personal issues

The above "late days" system should help in most cases, but sometimes dramatic and unforeseen events arrive: such as health issues requiring extended hospitalization, or a family crisis requiring travel outside the country. In such cases, the best course of action is to contact the course instructor as well as the engineering office and inform them of the situation.

Special Needs Policy

Any student in this course who, because of a disability, has a need for special accommodations, should come and discuss this with me in the first week of classes, as well as contacting the Disability Resource Office at 585-4631.

Cheating

Students are responsible for familiarizing themselves with the university calendar regarding academic regulations and standards. In particular the regu-

lations relating to academic misconduct in section 5.14.2 are required reading.

What is cheating?

- Sharing code in any way: allowing fellow students to examine your code or examining theirs is cheating. You must be careful to keep your code secure: don't leave code in unprotected storage, and don't leave paper listings lying about.
- Using code from other sources: previous offerings of this and other classes, code written by your classmates, code found on the Internet.
- Even submitting code *you wrote* and received credit for in another class is also cheating unless you have specifically arranged with the instructor to do so.
- Sharing written assignments or exams is cheating.
- Not being able to explain the code you submitted is evidence of cheating.
- **Note well: if you submit code that is copied from an illegitimate source, even if you take pains to conceal the original structure of the code, you are quite likely to be caught.**

What is *not* cheating?

- Discussing *in general terms* high-level issues such as design alternatives: provided you do not share code (see above).
- Discussing assignments in order to resolve ambiguities, and questions of interpretation. In these cases (which occur very often!) it's recommended to also seek help from the instructor.
- Helping fellow students use course resources: computers, compiler, assembler, linker, debugger, editor, profiler, etc.
- Helping fellow students understand the *meaning* of language constructs.
- Using code that has been *specifically provided* by the course instructor for your use.
- Working with other students, studying together, doing homework together, provided you don't share code.

- Seeking out other sources of information and studying them with the goal of understanding the underlying principles, then independently implementing your own solutions.
- It is possible for two student's work to bear a marked similarity even though they did not share code or discuss details. This is particularly true for small "toy" programs. In this case the ability of each student to fully explain their code would constitute evidence of independent work.