Principles of Computer Systems Design - 236601

Prof. Liuba Shrira

MISSION

Teach students approaches, concepts, abstractions, and techniques
that underly the design and implementation of computer software systems.

METHOD

- Lectures cover the main concepts and techniques.
The lecture material is based on
Principles of Computer System Design, by Jerome H. Saltzer and M. Frans Kaashoek, Morgan Kaufmann, 2009 (ISBN 978-0-12-374957-4) . Background material appears in Modern Operating Systems, by Andrew Tannenbaum, 2nd Edition, Prentice Hall Publishers (ISBN: 0130313580) .

- Discussions go a level deeper by analysing and evaluating the design of important software systems in class, and by comparing and contrasting case studies of successful and unsuccessful systems.
The discussions are based on a list of selected papers in the literature.

- Hands-on assignments provide students the opportunity to poke at real systems from outside.

- Student presentations: Occasionally, we offer an independent topic study and in-class group presentations
to teach students topic research and oral communication skills.
Topics are matched to student interests.

OVERALL ORGANIZATION OF MATERIAL
(These topics will be covered at different depths depending on interests)

1. Complexity. The main challenge in designing computer system is controling complexity. The first topic is therefore a discussion of the sources of complexity in computer systems, and an overview of the techniques to control complexity.

2. System organization. To control the complexity of computer systems a solid organization that enforces hard modularity is needed. The topic introduces techniques of system organization that provide or utilize hard modularity: the client-server model, virtual memory, operating system kernels, and threads. Material reviews concepts acquired in a basic OS class.

3. Naming. Modularity requires support for naming to allow sharing. Computer systems employ many
different naming systems (file names, network addresses, virtual memory addresses, etc.), some simple, some complex. The topic covers the important issues in designing naming systems.

4. Performance. Computer systems exhibit performance problems due to incommensurate scaling. We introduce simple performance models and techniques to address performance bottlenecks in computer systems.

5. Networking. Computer systems function seldom in isolation; most of them are distributed over multiple nodes and interact with other computer systems. The topic addresses the modularity issues in designing network protocols in the light of scalability and fault tolerance considerations.

6. Reliability. Computer system experiences failures. This topic covers issues in designing systems that can tolerate failures. As a case study we look at reliable storage systems.

7. Security. To control who can access what in a computer system, security mechanism are needed
but, importantly, societal considerations need to be taken in account.
Security mechanisms and societal policies are the focus of this topic.

Abstractions, and techniques covered are not a recipe for designing computer systems. Just applying these ideas will not necessarily result in a good computer system. Unlike in many other subjects (e.g., math), many problems in computer systems do not have a clean solution. The reason is that the goals of a design are often conflicting (e.g., a system should be both blindingly fast and tolerate any failure); in addition, economic goals put additional constraints on a design (e.g., it has to be fast and cost only one dollar). To provide insight how the techniques learned can be applied and misapplied we present many cases studies of successful (and unsuccessful) computer system. Our hope is that you learn from these case studies, and develop an intuition and a taste for designing computer systems.

Since computer systems play a crucial role in society, there are also many social and ethical problems that often need to be addresses in designing a computer system. Sometimes these social issues are harder to address than the technical ones. Through each of the topics we discuss the interaction of the social and technical issues.

REQUIRED BACKGROUND

We assume you know basic programming abstractions (e.g., algorithm, procedure call) and implementation techniques for such abstraction (e.g., interpreters). In addition, we assume a basic knowledge of physical constructs to realize these abstractions (e.g., processor, memory, etc.) as covered by a basic OS course.
Finally, we assume you have some experience with building software for computer systems, or at least experience with using computer systems.

REQUIRED CLASS WORK

Classes meet once a week for 2 X 50 min.
Typically, students are expected to read a chapter of a book, and/or a paper in preparation for each class, and are required to submit short answers related to the material due for the class. Students are expected to actively participate in the recitation part of the class, and are evaluated on class participation. There are 2-3 quizes and 2-3 hands-on assignments.