CSE 550: Computer Systems

Instructor: Tom Anderson (tom@cs.washington.edu)
TA: Dane Brandon (dane@cs.washington.edu)
Class meetings: Monday/Wednesday 10:30 – 11:50, EEB 031
Sections: Thursdays, 4:30 – 5:20, BLD 286
http://www.cs.washington.edu/550

[Note: to work around conference schedules, class will occasionally meet on Fridays 10:30 – 11:50. Location for those classes, TBD.]

Catalog Description: Advanced techniques in computer system software design, including network systems, operating systems, web servers, parallel computing, and databases. Prerequisite: CSE major and CSE 451.

Motivation: This course will provide the common intellectual foundation for systems research, suitable as a terminal course for those not interested in further study in systems, or as a gateway course to the various specialized systems courses the department offers. The course will cover the common foundation for research in operating systems, databases, cluster and wide area distributed systems, networking, and parallel systems. These topics include:

Concurrency
Transactions and serializability
Virtualization/isolation
Network protocol design
Reliability out of unreliable components
Parallelism and cache coherence
System evolution
Workload characterization
Security

In addition, the course will also teach students some of the mechanics of doing systems research:

How to read and critique a systems research paper
How to set up and present a quantitative experiment

Student Deliverables:

Foundational readings in computer systems
Blog entry answering one thought question per paper
Three problem sets
Two design exercises
One student-defined team project, with intermediate milestones

Readings, Blogs and Presentation: The core of the course is the reading and discussion of a set of foundational papers in computer systems. These papers cover the most important ideas in computer systems research, ones that are necessary to understand the context for any later work you might encounter. The reading is to be done before class, as we will take the papers as a starting point, not the end point of the class discussion. The approach we take is historical and comparative. While an undergraduate course in these subjects typically covers "what" computer systems do, we will focus on "why" – what other approaches were taken and what we can learn from their success and failure. Background reading is optional, included for those students who might be missing a particular topic from their preparation.

For each class other than the first, we will post a small set of discussion questions, linked off the course web page, based on the reading. You are required to add a **unique** comment to the discussion of **one** of the questions by 9am on the day of the class. (This is to give time for everyone to read the blog entries before class.) Note that the earlier you post, the easier it is to be unique. Please keep blog entries short: they can be anything that provides insight into the question being asked. We will automatically grant two mulligans during the quarter.

Project and Problem Sets: The course project is to be done in small teams (2-3 people), and should illustrate some interesting concurrent or distributed algorithm or system, and it must include a quantitative evaluation of the work. The project can be either student defined or one defined by the instructor. Sample projects would be to implement and quantitatively evaluate Paxos (week 6), optimal congestion control (week 9), or a multithreaded, replicated web storage service supporting both read and write operations (week 2, week 4). We have some software (PlanetLab, Seattle) to make it easier to develop and and test your project. We will give a tutorial on these in section in the first week of class, but their use in the project is optional. In addition, every 2-3 weeks, you will be required to write a status update on your project, and to meet as a group with the TA to discuss it.

There will also be three problem sets handed out during the quarter and two extended design exercises. The problem sets and design exercises are to be done individually. The problem sets are intended to reinforce the basic material covered in the class – e.g., can you write a correct concurrent program, while the design exercises are intended to apply the ideas in the class to novel domains. The design exercises will be done in two steps: an initial draft due at noon (by email) on the day of the section, followed by a revised draft due one week later (in section). This is so that student work can benefit from the discussion in section. Your writeups should include what and why for your proposal, as well as a sketch of how you would quantatatively evaluate your approach over the alternatives (what experiments you would run).

The problem sets are due in section. Project milestone writeups are due at noon the day before your scheduled meeting with the TA to discuss the milestone. The final project presentations and writeup are due 9am Monday of finals week.

Collaboration/Cheating: In our experience, students often learn more from each other than they do from the instructor. Thus, we encourage you to collaborate with your classmates, in all aspects of the course. The grading in the class is emphatically not curved; we would like nothing better than for all of you to get a 4.0.

To draw a very clear line, you may use any idea from any other person or group in the class or out, provided you clearly state what you have borrowed and from whom. If you do not provide a citation -- that is, you turn other people's work in as your own -- that's cheating. Anything else is fair game. Of course, we'll be grading you on the ideas you've added, but you should always borrow as much as you can as a starting point – there's no point in reinventing the wheel.

Final: This class will have no final exam.

Grading: Blogs: 10%; Problem sets and design exercises: 50%; Project: 40%

Introduction

We kick off with a case study of a successful system design.

(9/29) Dennis M. Ritchie and Ken Thompson. The UNIX Timesharing System. Communications of the ACM 17(7), July 1974.

Section (9/30): Project discussion and PlanetLab tutorial.

Concurrency

Please read this first if your grasp of concurrency is rusty.

Background reading: Andrew Birrell, An Introduction to Programming with Threads, DEC SRC Research Report 35, January 1989.

These papers contrast two different approaches to concurrency, the first tries to accommodate it, while the second tries to avoid it.

(**No class on 10/4, due to OSDI; instead we'll meet 10/1.)**

(10/1) Butler Lampson and David Redell. Experience with Processes and Monitors in Mesa. Communications of the ACM, volume 23, issue 2, February 1980.

(10/6) Pai et al., Flash: An Efficient and Portable Web Server, USENIX 1999.

Section (10/7): Thread programming practice. Problem set #1 out.

Parallelism

These papers grapple with several of the difficulties of obtaining good application performance when using multiple processors.

(10/11) Thomas Anderson, Brian Bershad, Edward Lazowska, and Henry Levy. Scheduler Activations: Effective Kernel Support for the User-Level management of Parallelism. ACM TOCS, February 1992, pp. 53-79.

(10/13) Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. OSDI 2004.

Project outline due at noon on **Wednesday October 13.**

Section (10/14): More thread programming practice.

Transactions

Please read the background chapter if your grasp of transactions is rusty or absent.

Background reading: Michael J. Franklin. Concurrency Control and Recovery. In Tucker (ed.), The Computer Science and Engineering Handbook, 1997.

The first paper introduces the concept of transactions and how they can be implemented; the second reading shows how to extend the transaction concept to distributed systems.

(10/18) Gray et al. The Recovery Manager of the System R Database Manager. ACM Computing Surveys, 1981.

(10/20) Bernstein, Hadzilacos, and Goodman. Distributed Recovery. Chapter 7 in Concurrency Control and Recovery in Database Systems. [Read only up to, and not including, three phase commit.]

Section (10/21): Design exercise #1: PCM-based storage system. Problem set #1 due. Problem set #2 out.

File Systems

File system design did not start and end with the Berkeley UNIX FFS. Here are two case studies of how file system design is influenced by application and hardware constraints.

(10/25) Mendel Rosenblum and John K. Ousterhout. The Design and Implementation of a Log-Structured File System. ACM Trans. on Computer Systems 10(1), February 1992, pp. 26-52.

(10/27) Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google File System. SOSP 2003.

Section (10/28): Individual meetings with project teams during section. Project progress report due Wednesday at noon. Design exercise #1 final draft due.

Consensus

These two papers provide some of the background for the theory of distributed systems. The first paper introduces the notion of state machine replication; the second shows how to make state machine replication non-blocking for most failure conditions.

(11/1) Lamport, Time, Clocks and the Ordering of Events in a Distributed System, CACM 1978.

(11/3) Lamport, Paxos Made Simple, ACM SIGACT News, 2001.

Section (11/4): Design exercise #2: High availability web service.

Virtualization

These two papers explain how to implement virtual memory and virtual machines.

(11/8) A. Bensoussan et al. The Multics virtual memory: concepts and design. Communications of the ACM, volume 15, issue 5, May 1972.

(11/10) Paul Barham et al.. Xen and the Art of Virtualization SOSP 2003.

(**No section on 11/11; instead it will meet 11/12 at 10:30**)

Section (11/12): Internet basics (TCP/IP). Design exercise #2 final draft due. Problem set #2 due and problem set #3 out.

Caching and Cache Coherence

Caching is central to most areas of computer science; these two papers extend caching to work in a distributed system.

(11/15) Kai Li and Paul Hudak. Memory Coherence in Shared Virtual Memory Systems. ACM Trans. on Computer Systems 7(4), November 1989, pp. 321-359.

(11/17) Jay Kistler and M. Satyaranayaman. Disconnected Operation in the Coda File System. ACM Trans. On Computer Systems, 1992.

Section (11/18): Internet basics continued.

Networking

For those who lack an undergraduate background in networking, these two chapters may be helpful; another option is to read chapters from Peterson & Davies, Computer Networking: A Systems Approach

Background reading: Salzer and Kaashoek. The Network as a System. Chapter 7 in Principles of Computer System Design.

Background reading: Hari Balakrishnan. An Introduction to Wide-Area Internet Routing. Unpublished

The Clark paper outlines what the Internet designers were trying to accomplish. The other two papers are case studies of a "systems" approach to network protocol design.

(11/22) David Clark. The Design Philosophy of the DARPA Internet Protocols. SIGCOMM 88.

(**No class on 11/24 and no section on 11/25; Thanksgiving.)**

(11/29) Thomas Anderson, Andy Collins, Arvind Krishnamurthy, and John Zahorjan. PCP: Efficient Endpoint Congestion Control, NSDI 2006.

(12/1) John John, Ethan Katz-Bassett, Arvind Krishnamurthy, Thomas Anderson and Arun Venkataramani. Consensus Routing: The Internet as a Distributed System. NSDI 2008.

Section (12/2): Individual meetings with project teams during section; project progress report due at noon on Wednesday.

Week 10: Security and Wrapup

The first paper outlines how to design distributed systems to be secure; the second distills rules of thumb for designing computer systems.

(12/7) Lampson, Computer Security in the Real World, 2001.

(12/8) Lampson. Hints for Computer System Design. ACM Operating Systems Review, Vol. 15, No. 5, October 1983.

Section (12/10): Problem set #3 due; discussion in section.

December 14, 9am: Final project presentations