1a. Project: Information and Suggestions

Project information

This section is copied from handout 1, with a few edits.

During the last half of the course there is a project in which students will work in groups of three or so to apply the methods of the course to their own research projects. Groups of two are possible, but usually a larger group works better. A group of one is not allowed. Each group will pick a real system. Ideally, this will be one that some member of the group is actually working on, but it can one from a published paper or from someone else's research. If you can't think of a suitable system, there are some suggestions below. The goal of the project is to write:

A specification for the system.

High-level code that captures the novel or tricky aspects of the actual implementation.

The abstraction function and key invariants for the correctness of the code. This is not optional; if you can't write these things down, you don't understand what you are doing.

Depending on the difficulty of the specification and code, the group may also write a correctness proof for the code.

Projects may range in style from fairly formal, like handout 18 on consensus, in which the 'real system' is a simple one, to fairly informal (at least by the standards of this course), like the section on copying file systems in handout 7. These two handouts, along with the ones on naming, sequential transactions, concurrent transactions, and caching, are examples of the appropriate size and possible styles of a project.

The result of the project should be a write-up, in the style of one of these handouts. During the last two weeks of the course, each group will give a 25-minute presentation of its results. We have allocated three class periods for these presentations, which means that there will be nine or fewer groups.

The projects will have five milestones. The purpose of these milestones is not to assign grades, but to make it possible for the instructors to keep track of how the projects are going and give everyone the best possible chance of a successful project

1. We will form the groups around March 11, to give most of the people that will drop the course a chance to do so.
2. Each group will write up a 2-3 page project proposal, present it to one of the instructors around spring break, and get feedback about how appropriate the project is and suggestions on how to carry it out. Any project that seems to be seriously off the rails will have a second proposal meeting a week later.
3. Each group will submit a 5-10 page interim report in the middle of the project period.

4. Each group will give a presentation to the class during the last two weeks of classes.
5. Each group will submit a final report, which is due on Friday, May 14, the last day allowed by MIT regulations. Of course you are free to submit it early.

One third of the groups will be 'early' ones; the others will be 'late' ones that give their presentations one week later. The due dates of proposals and interim reports will be spread out over two weeks in the same way. If there are few enough groups, we will dispense with the early ones.

Schedule

Thurs., Mar. 18	Early proposals
Thurs., Apr. 1	Late proposals
Tues., Apr. 15	Early interim reports
Thurs., Apr. 22	Late interim reports
Thurs., May 6	**Early project presentations**
Tues., May 11	**Late project presentations**
Thurs., May 13	**Late project presentations**

Project Suggestions

Ad-Hoc Mobile Networking

Creating a wireless network infrastructure is costly and requires the involvement of large organizations to look after installed base. And all of this seems so unnecessary, since there are always lots of people wandering around with wireless devices that are perfectly capable of routing messages from a sender to a desired recipient. Why not just build a network out of whatever devices seem to be available at the time? Potential issues you might want to consider:

a) How do you deal with the fact that devices are always moving around?

b) How does the system route a message to a device if the sender has no idea where the receiver is located or what route it might be able to use?

c) How much power are you going to consume from the devices of unsuspecting passers-by?

Relaxed Memory Consistency

It turns out that in a multiprocessor with caching, computer architects believe that they can get better performance by building machines that are not guaranteed to preserve the observed order of writes. More specifically, if a one thread performs a sequence of writes such as x=1; y=2; z=3; starting from a state in which x, y, and z =0, it may be possible for other threads to observe states in which x=0, y=0, and z=3, and so on. To enable programmers to develop programs, these machines typically guarantee that if a program uses synchronization correctly, it will not observe

any of these reorderings. There have been many papers published on this topic, but there is still a feeling in the community that we don't have a complete handle on this phenomenon. It would be nice to develop enough formal specification of what is going on to better understand the problem.

Striped File Systems

Traditional file systems have been implemented on top of a single disk. But this has drawbacks: large files must be read sequentially, and if the disk fails, you lose everything.

Another way to implement a file system is to somehow distribute the data across many disks. This way a system can read the data in parallel. And if you put some redundancy in the writes to disk, your system can even tolerate the failure of one or more disks before it loses data. Some questions to think about:

a) How would you build such a system using a collection of machines, each with a single disk?

b) How would you make the data secure against eavesdroppers attempting to determine the contents of the file?

c) What kind of system issues would affect whether a machine would observe any performance increase from reading or writing data in parallel?

d) What interface to the file system would you provide?

Package tracking system

Write a spec and a high-level implementation for the Fedex package tracking system. You might starting by going to the Fedex web site and examining how the system works. The tricky part of this project is separating the tracking from other functions, notable package routing.

You should treat these independently: tracking has as input the information that the bar code scanners pick up about the package identity at various points in the Fedex system, as well as the package source and intended destination. Routing is an entirely separate function driven by information about current locations and destinations from the tracking system, as well as knowledge of available transport.

Work on the tracking system first, and then go on to routing if you have time.

Garbage collection

Write one or more specs for garbage collectors, and one or more high-level implementations. The specs should first deal with reachability, assuming an infinite pool of memory to draw from. Then it should take a finite pool into account, and say something about when a request won't be satisfied. You might look at mark-and-sweep, copying, and generational collectors. Note that collectors are used in log-structured file systems and object-oriented data bases as well as in main memory allocators.

Server farms

Write specs and high-level implementations for a collection of machines used to implement a web server farm. Such farms are usually organized into:

* Stateful machines that store persistent data
* Stateless machines that do front-end processing
* Switches that send requests to the right places

Things to think about are scaling, initialization, partitioning of data, replication for availability and for load balancing. An important goal is to minimize the amount of manual intervention needed to handle load balancing and system failures.