UNIVERSITY OF CALIFORNIA AT BERKELEY
College of Engineering
Department of Electrical Engineering and Computer Sciences
Computer Science Division

CS 162
Fall 1988

John Ousterhout

Mini-Sermon #2: Building High-Performance Systems

- You learn all sorts of fancy high-falutin' algorithms in Computer Science classes, but most of the time you don't need them.

- Dilemma:
  - In operating systems, performance is usually a critical issue.
  - Usually, only a few pieces of OS are peformance-critical (e.g. context switch time).
  - Making performance optimizations makes code messier, harder to understand and maintain. Don't want to do it unless absolutely necessary.

- Option #1: Measurement. Grab all data that can be gotten from existing systems. Measure process sizes, file distributions, etc etc. Try to identify areas where performance does and doesn't matter.

- What about things with no existing data?

- Option #2: Modelling. Build model of system, write simulator to simulate system behavior in order to see how well your algorithms will work. Problem: it's often hard to figure out right system model: extrapolate from current systems.

- Option #3: Tuning.
  - Make system as simple as possible, but build in mechanisms for evaluating performance once system is running.
  - Get system running, then make performance measurements.
  - Tune those portions of the system that didn't perform up to snuff. This means that strategy algorithms must be designed to be changeable.

- Above all, don't worry about performance unless you're SURE there's a problem. In most places it doesn't matter.

- Remember that it's a zero-sum game: every line of code you write is one line you didn't write. Every feature or change should be justified. Don't do things just because they're ''cute'' or ''neat''.