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

CS 162
Fall 1988

John Ousterhout

Topic 0: Introduction

- Readings for this topic:
  - Kernighan/Ritchie: the whole thing **** required ****.
  - Toy Operating System: up through Problem 1.1 **** required ****.
  - Silberschatz/Peterson: Chapters 1 and 2.
  - Kernighan/Pike: whatever you need.

- *Name, office, office hours, office phone, computer mailing address. Can review tapes in 205 McLaughlin?*

- *Mention lecture style with announcements in middle, then charge into material.*

- Why are operating systems interesting?
  - They combine things from many other areas of Computer Science: languages, hardware, data structures, algorithms.
  - OSes act as coordinators between users and programs, often with conflicting needs.
  - OS as magician: makes computer appear to be more than it really is (each process appears to have its own machine, with its own memory, etc.)
  - Large, complex systems: results in interesting problems of management, conflicting requirements.

  - Must both support users and protect them from each other (and themselves); security issues pose tough problems.
  - Not many of you will actually build OSes, but can learn general techniques and apply them elsewhere.

- Course Organization:
  - Lecture: discuss concepts, compare existing and proposed solutions.
  - Discussion: learn C programming language, analyze a very simple OS (Toy OS) in enormous detail to get a feel for a real system.

- *Go over class information sheet. Demand pictures. Explain course admission policy, if it's over-enrolled.*

- *Give first-day quiz to see what everybody knows.*

- *Lecture 1 ended here.*

- ''Operating system'' is a hard term to define. The discipline arose historically from a set of problems. It's easiest to introduce OS'es by discussing their history.

- History Phase 1 (hardware expensive, humans cheap): make more efficient use of the hardware, get humans farther from the machine.
  - In the beginning, user worked at console to debug. One user at a time, no overlap of computation and I/O. OS first appeared as a subroutine library shared by all users.
  - Simple batch monitor, get user away from the computer. OS = program to load and run user jobs, take dumps. Makes better use

of hardware, but more difficult to debug.

- Data channels, interrupts, overlap of I/O and computation.
  - Buffering and interrupt handling in OS.
  - Spool jobs on drum.
  - Still one job at a time, so utilization often bad.
  - No protection.
  - Short jobs wait for long ones.
- Memory protection + relocation.
  - Multiprogramming: several users share the system.
  - Small jobs finish fast.
  - OS must manage interactions between concurrent things.
  - OS'es finally begin to be an important science. *(People got interested in OS'es because they didn't work, e.g. Multics, OS/360)*

- History, Phase 2 (hardware cheap, humans expensive): make more efficient use of people time, get people back closer to hardware.
  - Interactive timesharing.
    - Terminals are cheap, so let the user interact with the system.
    - Fancy filing systems.
    - Problems of response time, thrashing.
  - Personal computing. Computers are cheap, put one in each terminal. The OS becomes a subroutine library again.
  - Networking: allow different machines to share resources easily.

- Characteristics of current OS'es:
  - Enormous:
    - 100k's of lines
    - 100-1000 man-years
  - Complex

        - Asynchronous
        - Hardware idiosyncracies
        - Conflicting needs of different users
        - Performance is crucial
    - Poorly understood:
        - The system outlives any of its builders
        - Never debugged *(OS/360 released with 1000 bugs)*
        - Behavior is hard to predict, tuning is often done by guessing
        - Often unreliable

- Operating systems have two general functions:
    - Coordinator: allow several things to work together in efficient and fair ways.
    - Standard library: provide standard facilities that everyone needs.

- Most of this course will deal with the coordination aspect: making many things work well together. This appears in several different areas:
    - Concurrency. OS allows several users to be working at the same time, as if each had a private personal machine. Or, one user can be doing many things at the same time. To keep track of everything, notion of process was invented. It's hard to keep processes from interacting in bad ways.
    - I/O devices. Don't want CPU to sit idle while an I/O device is working. *(especially something slow like terminal)* So, interrupts were invented:
        - Each I/O device has a little processor inside it so it can run autonomously.
        - CPU issues commands to I/O devices, then goes off to do other things.

    - When I/O device is finished, it issues *interrupt* to CPU: CPU stops whatever it was doing, and goes to special place to process I/O device request. *Ask how many know what an interrupt is.*
    - Interrupts complicate matters inside the operating system. How can OS make sure that an interrupt doesn't destroy the work that was interrupted? What happens when many interrupts occur at about the same time?
  - Memory. Each process needs to use some main memory. OS coordinates their usage so they can share it. It swaps information back and forth between disk and main memory so the system can run even if the total memory needed by all processes is greater than the size of the memory.
  - Files. Each user owns a collection of files. The OS coordinates how space is used for files so that they can all fit on the same disk.
  - Networks. Allow groups of workstations to work together.

- Another way of thinking of OS is wizard: OS presents illusions.
  - Single processor is made to look like many separate processors, one per user.
  - Single memory is made to look like many separate memories, one per process, each much larger than the real memory.

  Sometimes the illusions don't work.