

CSE 451: Operating Systems Autumn 2003

Lecture 1 Course Introduction

Hank Levy
levy@cs.washington.edu
596 Allen Center

Today's agenda

- Administrivia
 - overview of course
 - projects, assignments, exams, ...
 - sections
 - overloading
- OS overview
 - functional
 - resource mgmt, major issues
 - historical
 - batch systems, multiprogramming, time shared OS's
 - PCs, networked computers

10/3/2003

© 2003 Hank Levy

2

Course overview

- Everything you need to know will be on the course web page:

<http://www.cs.washington.edu/education/courses/451/CurrentQtr>

10/3/2003

© 2003 Hank Levy

3

Overloading

- If you intend on dropping this course
 - please do it soon!
- If you want to get into this course
 - plan for the worst case (i.e. you don't get in)
 - but, make sure you've sent me email

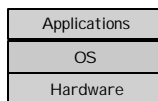
10/3/2003

© 2003 Hank Levy

4

What is an Operating System?

- An operating system (OS) is:
 - a software layer to abstract away and manage details of hardware resources
 - a set of utilities to simplify application development



- "all the code you didn't write" in order to implement your application

10/3/2003

© 2003 Hank Levy

5

The OS and Hardware

- An OS mediates programs' access to hardware resources
 - Computation (CPU)
 - Volatile storage (memory) and persistent storage (disk, etc.)
 - Network communications (TCP/IP stacks, ethernet cards, etc.)
 - Input/output devices (keyboard, display, sound card, etc.)
- The OS abstracts hardware into logical resources and well-defined interfaces to those resources
 - processes (CPU, memory)
 - files (disk)
 - programs (sequences of instructions)
 - sockets (network)

10/3/2003

© 2003 Hank Levy

6

Why bother with an OS?

- **Application benefits**
 - programming simplicity
 - see high-level abstractions (files) instead of low-level hardware details (device registers)
 - abstractions are reusable across many programs
 - portability (across machine configurations or architectures)
 - device independence: 3Com card or Intel card?
- **User benefits**
 - safety
 - program “sees” own virtual machine, thinks it owns computer
 - OS protects programs from each other
 - OS fairly multiplexes resources across programs
 - efficiency (cost and speed)
 - share one computer across many users
 - concurrent execution of multiple programs

10/3/2003

© 2003 Hank Levy

7

The Major OS Issues

- **structure**: how is the OS organized?
- **sharing**: how are resources shared across users?
- **naming**: how are resources named (by users or programs)?
- **security**: how is the integrity of the OS and its resources ensured?
 - **protection**: how is one user/program protected from another?
- **performance**: how do we make it all go fast?
- **reliability**: what happens if something goes wrong (either with hardware or with a program)?
- **extensibility**: can we add new features?
- **communication**: how do programs exchange information, including across a network?

10/3/2003

© 2003 Hank Levy

8

More OS issues...

- **concurrency**: how are parallel activities (computation and I/O) created and controlled?
- **scale**: what happens as demands or resources increase?
- **persistence**: how do you make data last longer than program executions?
- **distribution**: how do multiple computers interact with each other?
- **accounting**: how do we keep track of resource usage, and perhaps charge for it?

10/3/2003

© 2003 Hank Levy

9

OS History

- In the very beginning...
 - OS was just a library of code that you linked into your program; programs were loaded in their entirety into memory, and executed
 - interfaces were literally switches and blinking lights
- And then came **batch systems**
 - OS was stored in a portion of primary memory
 - OS loaded the next job into memory from the card reader
 - job gets executed
 - output is printed, including a dump of memory (why?)
 - repeat...
 - card readers and line printers were very slow
 - so CPU was idle much of the time (wastes \$\$)

10/3/2003

© 2003 Hank Levy

10

Spooling

- Disks were much faster than card readers and printers
- **Spool** (Simultaneous Peripheral Operation On-Line)
 - while one job is executing, spool next job from card reader onto disk
 - slow card reader I/O is overlapped with CPU
 - can even spool multiple programs onto disk
 - OS must choose which to run next
 - job scheduling
 - but, CPU still idle when a program interacts with a peripheral during execution

10/3/2003

© 2003 Hank Levy

11

Multiprogramming

- To increase system utilization, multiprogramming OSs were invented
 - keeps multiple runnable jobs loaded in memory at once
 - overlaps I/O of a job with computing of another
 - while one job waits for I/O completion, OS runs instructions from another job
 - to benefit, need asynchronous I/O devices
 - need some way to know when devices are done
 - interrupts
 - polling
 - goal: optimize system throughput
 - perhaps at the cost of response time...

10/3/2003

© 2003 Hank Levy

12

Timesharing

- To support interactive use, create a timesharing OS:
 - multiple terminals into one machine
 - each user has illusion of entire machine to him/herself
 - optimize response time, perhaps at the cost of throughput
- Timeslicing
 - divide CPU equally among the users
 - if job is truly interactive (e.g. editor), then can jump between programs and users faster than users can generate load
 - permits users to interactively view, edit, debug running programs (why does this matter?)
- MIT Multics system (mid-1960's) was the first large timeshared system
 - nearly all OS concepts can be traced back to Multics

10/3/2003

© 2003 Hank Levy

13

Distributed OS

- distributed systems to facilitate use of geographically distributed resources
 - workstations on a LAN
 - servers across the Internet
- supports communications between jobs
 - interprocess communication
 - message passing, shared memory
 - networking stacks
- sharing of distributed resources (hardware, software)
 - load balancing, authentication and access control, ...
- speedup isn't the issue
 - access to diversity of resources is goal

10/3/2003

© 2003 Hank Levy

14

Parallel OS

- Some applications can be written as multiple parallel threads or processes
 - can speed up the execution by running multiple threads/processes simultaneously on multiple CPUs
 - need OS and language primitives for dividing program into multiple parallel activities
 - need OS primitives for fast communication between activities
 - degree of speedup dictated by communication/computation ratio
 - many flavors of parallel computers
 - SMPs (symmetric multi-processors)
 - MPPs (massively parallel processors)
 - NOWs (networks of workstations)
 - computational grid (SETI @home)

10/3/2003

© 2003 Hank Levy

15

Embedded OS

- Ubiquitous computing
 - cheap processors embedded everywhere
 - how many are on your body now? in your car?
 - cell phones, PDAs, network computers, ...
- Typically very constrained hardware resources
 - slow processors
 - very small amount of memory (e.g. 8 MB)
 - no disk
 - typically only one dedicated application

10/3/2003

© 2003 Hank Levy

16

CSE 451

- In this class we will learn:
 - what are the major components to most OSs?
 - how are the components structured?
 - what are the most important (common?) interfaces?
 - what policies are typically used in an OS?
 - what algorithms are used to implement policies?

10/3/2003

© 2003 Hank Levy

17