Operating Systems

H. Levy

What is an Operating System?

- "all the code that you didn't write"
- the code that manages physical (hardware) resources
- provides users with "logical" well-behaved environment
- O.S. defines a set of logical resources (objects) and a set of well-defined operations on those objects (i.e., an interface to the use of those objects)
- provides *mechanisms* and *policies* for the control of objects/resources
- controls how different users and programs interact

What resources need to be managed?

- the CPU (computation)
- primary memory
- secondary memory devices (disks, tapes, etc)
- networks
- input devices (keyboard, mouse)
- various I/O devices (printers, display, cameras, speakers)

The O.S. must be able to service all of these devices, simultaneously, and support safe sharing of devices among all users and programs demanding service.

Major issues in Operating Systems

- structure -- how is an operating system organized?
- sharing -- how are resources shared among users
- naming -- how are resources named (by users or programs)
- protection -- how is one user/program protected from another
- security -- how to restrict the flow of information
- performance -- why is it so slow?
- reliability and fault tolerance -- when something goes wrong
- extensibility -- how do we add new features?
- communication -- how and with whom can we communicate (exchange information)

Major issues in OS (2)

- concurrency -- how are parallel activities created and controlled?
- scale and growth -- what happens as demands or resources increase?
- persistence -- how to make data last longer than programs
- compatibility -- can we ever do anything new?
- distribution -- accessing the world of information
- accounting -- who pays the bills, and how do we control resource usage?

A brief history of operating systems

- "in the beginning", the OS was just code to which you linked your program, loaded the whole thing into memory, and ran your program; basically, just a run-time library
- simple batch systems were first real operating systems:
  - os was stored in part of primary memory
  - it loaded a single job (from card reader) into memory
  - ran that job (printed its output, etc.)
  - loaded the next job...
  - *control cards* in the input file told the os what to do
- *Spooling* and *buffering* allowed jobs to be read ahead of time onto tape/disk or into memory.

Multiprogramming

- ***Multiprogramming*** **systems provided increased utilization**
  - keeps multiple runnable jobs loaded in memory
  - overlaps I/O processing of a job with computes of another
  - benefits from I/O devices that can operate asynchronously
  - requires the use of interrupts and DMA
  - tries to optimize *throughput*

Timesharing

- ***Timesharing*** **supported** ***interactive*** **use of**
  - each user feels as if he/she has the entire machine (at least late at night!)
  - timesharing tries to optimize *response time*
  - based on time-slicing -- dividing CPU equally among the users
  - permitted active viewing, editing, debugging, participation of users in the execution process
- **MIT Multics system (mid-late 1960s) was first large timesharing system**

Real-Time Operating Systems

- **real-time os are used for specialized applications: subway systems, flight control, factories, nuclear power plants...**
- **basic idea: os must guarantee response to physical events in a fixed time interval**
- **problem is to schedule all activities in order to meet all of their critical requirements**
- **with the use of video applications on modern PCs, *all* operating systems will soon have read-time requirements**

Distributed Operating Systems

- **distributed systems facilitate use of geographically distributed resources**
- **supports communication between parts of a job or different jobs**
- **sharing of distributed resources, hardware and software**
- **permits some parallelism, but speedup is not the issue**

Parallel Operating Systems

- **Support parallel applications wishing to get speedup of computationally complex tasks**
- **Needs basic primitives for dividing one task into multiple parallel activities**
- **Supports efficient communication between those activities**
- **Supports synchronization of activities to coordinate sharing of information**

Embedded Operating Systems

- **The decreased cost of processing makes computers ubiquitous. Each "embedded" application needs its own operating system or control software, e.g.:**
  - cell phones (e.g., Geoworks)
  - U. S. Robotics Palm Pilot or Apple Newton (PDAs)
  - "network terminals" (internet interfaces)
- **In the future, your house will have 100s of these things.**

CSE 451

- **In this class we will learn:**
  - what are the parts of an O.S.
  - how is the o.s. and each sub-part structured
  - what are the important interfaces
  - what are the important policies
  - what algorithms are typically used
- **We will do this through reading, lectures, and programming.**
- **You will need to keep up with all three of these.**