The UNIVERSITY of VERMONT

Benjamin Fenster
CS-201 (Arslan)
5 November 2005

Introduction

I. History and Concepts
   a. Everybody has a different definition of an OS.
   b. Depends on what type of system too. For a desktop, don't care if the CPU is idle but expect good performance
   c. Programs should be easy to run
   d. Computer should be easy to use (but that expectation has changed over time)
   e. Hardware
      i. Basic computing resources
      ii. CPU, memory, I/O devices
   f. The goal of the OS is to coordinate use of hardware among user programs.
   g. Third layer: application programs. Fourth layer: users.

II. What Does it Do?
   a. Allocate Resources. How to allocate resources so there's no backlog?
   b. Control execution of user programs (which ones can run?)
   c. Kernel: The one program that's running all the time.
   d. Simple Batch System: OS + user program in memory
   e. Multiple Batch
      i. OS + Multiple programs
      ii. Need memory management so one job can't overwrite memory for another job
      iii. What if physical memory is insufficient?
   f. Desktops
      i. Only one user at a time (?)
      ii. Convenient for user to run programs the way s/he wants.
      iii. Want optimized response times. How to do that?
   g. Parallel Systems
      i. Want to improve performance
      ii. Need to coordinate – has some overhead
      iii. Also still has I/O bottleneck
      iv. SMP – symmetric multiprocessing. No host/slave relationship. More overhead
      v. AMP
         1. One CPU is host; delegates what the others should be doing.
         2. Means one CPU isn't doing any useful work.
         3. Less overhead.

III. Definition of OS
   a. Intermediary between user and computer
   b. Allocates resources; makes sure every process gets a fair share of resources
   c. Scheduling
   d. Keeps track of memory usage
   e. Have definitely evolved. Mainframes used to be the only timesharing systems. Now microcomputers are.

IV. Parallel Systems
   a. Multiprocessor systems
   b. "Tightly Coupled" – More than one processor shares the same clock and same memory. Communication is done through memory
   c. "Loosely coupled" – More autonomous processors. Communication done through comm. lines.
   d. Advantages
      i. More throughput! More processors = more work
      ii. Of course, not all problems have feasible multiprocessor solutions. Increase in speed may be far from the ideal case.
      iii. Economical when they share other components (like memory)
      iv. Increased reliability: One processor can take over if another fails. Fault tolerance.

e. Symmetric Multiprocessing (SMP). Each processor has its own operating system (an identical copy of the OS).
f. Asymmetric Multiprocessing
    i. Master-slave relationship
    ii. Master assigns tasks to slave processors
g. Distributed System
    i. Again, an environment with more than one processor – more than one computer
    ii. Loosely coupled system
    iii. Systems communicate with one another; assigns tasks, exchange data
    iv. Advantages
        1. Reliability, speed
        2. Resource sharing
    v. Requirements: Networking infrastructure
    vi. Can be client-server of peer-to-peer systems
h. Clustered System
    i. Multiple processors run together to accomplish a task / computation.
    ii. Distribute the work to clusters in UNIX.
    iii. Tightly coupled.
    iv. Don't need to be autonomous – accomplish a certain task
i. Real-Time Systems
    i. Used often in control devices
    ii. Increasingly found in everyday life: cars, medical equipment, Mars robot
    iii. Main characteristic: Certain time-critical applications that MUST meet their deadlines.
    iv. Hard real-time tasks: critical. Must meet their deadlines or that's the end of the system
    v. Soft real-time tasks: Higher priority, but not the end of the world if they miss their deadlines.
    vi. Many ordinary OS features cannot be implemented (e.g. virtual memory)
    vii. Runtime of OS tasks need to be deterministic
    viii. Generally embedded systems without much memory, so OS must be compact

V. Why do we Care
a. Different types of systems require different OSes
b. When processors share a clock, for example, everybody knows it's the same time.

VI. Computer System Structures
a. Consider memory as a base (with a memory controller)
b. CPU has direct access to memory
c. All other devices go through controllers.
d. Each controller has its own buffer space, so transfers can be done in blocks.

VII. Interrupts
a. Hardware: I/O is finished in a device, so the device sends a signal to the CPU that it's ready
b. Software: e.g. system call
c. Instruction execution cycle: fetch, decode, execute, interrupt
    i. If there is an interrupt, solve program counter and save registers
    ii. Go to subroutine to handle interrupt
d. Handling
    i. Polling: Continuously polls to see if there's an interrupt
    ii. Vectored interrupt system: Address is given in interrupt vector
e. Programs don't have I/O access. Makes a system call.
f. Two I/O Methods:
    i. Synchronous: Requesting process must wait until I/O has finished
    ii. Process requests I/O; waits
    iii. While waiting, device driver interacts with hardware.
    iv. As soon as I/O is finished, process resumes where it left off.
g. Device Status Table

i. Have device ID and its status (idle, busy)
ii. If it's busy that means there are processes using the device
    1. Table tells the address, length of data to be transferred (for a disk), or whatever other information is relevant to the request
    2. Want to know about the requesting process
    3. May have more than one queued request

h. Transfer to Memory
    i. Servicing interrupt for every keystroke doesn't cost much.
    ii. Wouldn't want to interrupt for every word transferred from disk to memory though
    iii. Direct Memory Access (DMA) structure
    iv. Used for high speed I/O devices
    v. Device controller transfers blocks of data from buffer storage directly to main memory without CPU intervention
    vi. Only one interrupt is generated per block, rather than one per byte
    vii. Not helpful if all processes are sequential, but if other processes are waiting we can do more useful processing while doing I/O

VIII. Storage Structures
a. Hierarchy: magnetic tape → optical disks → magnetic disks → electronic disk → main memory → cache → registers
b. Going up the hierarchy gets more expensive, faster, and more volatile.
c. Magnetic Disk
    i. Platter is one round sheet
    ii. Track is one circular ring within a platter
    iii. Sector is one portion of a track
    iv. Cylinder is the corresponding track across all platters.

IX. Caching
a. Used to hold recently accessed data (in high-speed memory) for slow I/O devices
b. One Use: From main memory to registers
c. Requires a cache-management policy: OS needs to know what data are available in the cache. This is just another level in the storage hierarchy.
d. When the same data are stored in at two different levels at the same time, that introduces some new problems. The data need to be consistent!
    i. If A is stored on disk it might be copied to main memory, cache, and a register.
    ii. When execution ends, need to update changes in a register inside the cache and main memory, then ultimately back on disk.

X. Hardware Protection
a. Dual Mode operation. Basic mechanism for protecting hardware.
b. Processes run in "user" or "monitor" (or "privileged") mode.
c. Certain instructions are privileged and can only be run in monitor mode
d. Means user is automatically protected against certain malicious applications when the user wants those actions performed s/he must make a system call
    i. System switches to monitor mode
    ii. Sets mode bit (1)
    iii. Sets user mode when executing user code
    iv. Sets = 0 (monitor) when interrupt / fault
e. What are privileged instructions?
    i. I/O operations
    ii. When user wants I/O, makes system call
    iii. Causes software interrupt
    iv. System performs the corresponding operation (a read, perhaps) and returns control to the user application
f. Memory Protection
    i. Shouldn't be able to write in OS's memory or in other processes' space
    ii. One solution is to use BASE and LIMIT registers to indicate what memory the application is allowed to use (limit = amount)

iii. Whenever there's a memory access, compare with base address. If K< BASE, generate addressing error – a software trap
iv. If address ≥ BASE + ILMIT, generate an error.
v. If both checks are passed, proceed witih the operation.
vi. Of course, loading / doing any operations on those registers would be privileged

g. CPU Protection
   i. Every process eventually needs to relinquish the CPU
   ii. Timer interrupts CPU after a certain amount of time has passed (just decrements a counter to zero). Again, these operations would be privileged.