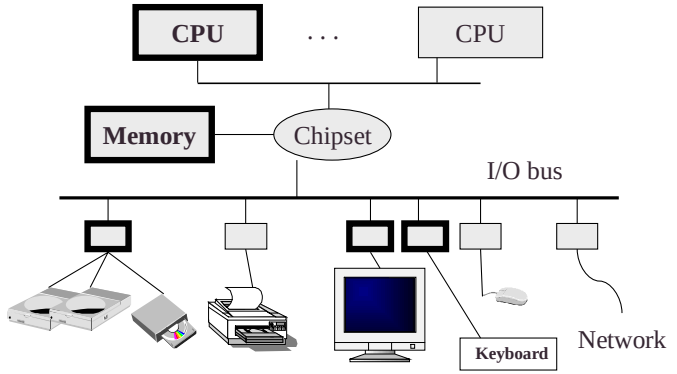
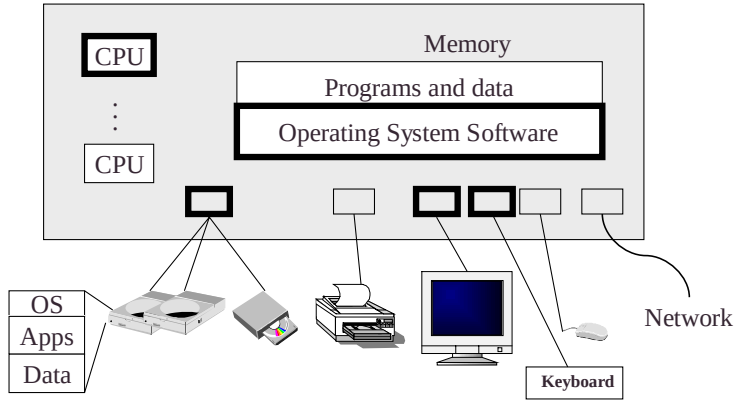
Operating System Overview

Otto J. Anshus
(including slides from Kai Li, Princeton University)
University of Tromsø

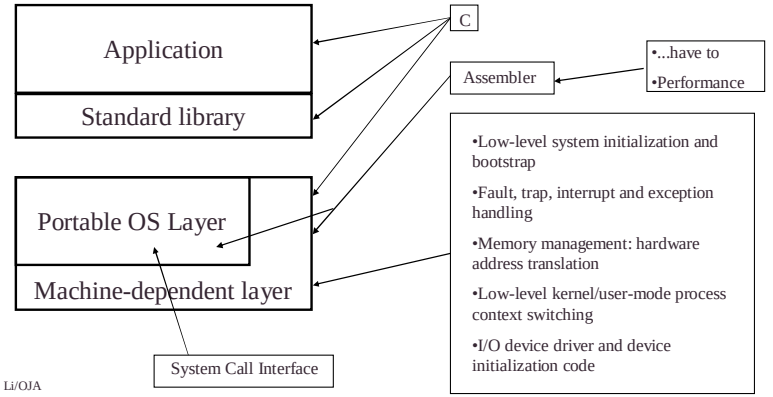
A Typical Computer from a Hardware Point of View



A Typical Computer System



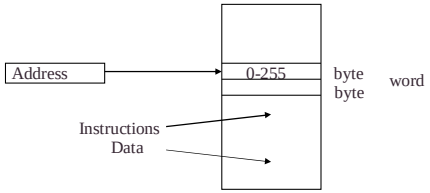
Typical Unix OS Structure



Linux Kernel version 2.0

- 500,000 lines of C code and 8000 lines of assembler
  - "Micro kernel" (process & memory management): 5%
  - Device drivers: 90%
  - Network, file systems, initialization, etc.: 5%

Memory



Intel architecture is "little endian"

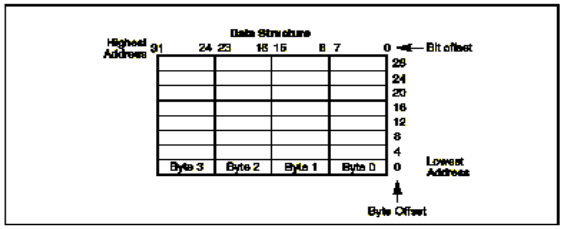
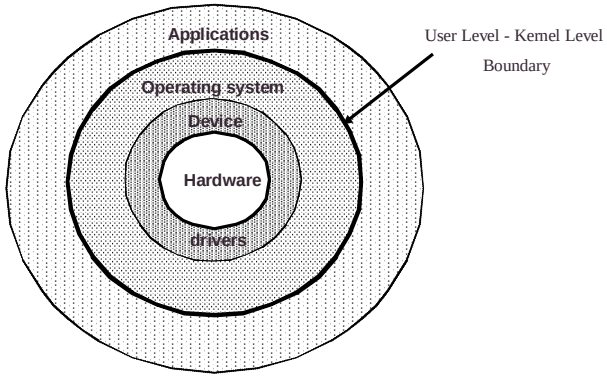


Figure 1-1. Bit and Byte Order

Hexadecimal

- **16 decimal is base**
  - 0, 1, 2,…,9, A, B, C, D, E, F
- **C4AFh=50351d**
  - $C*16^3+ 4*16^2+A*16^{1+} F*16^0$
  - $12*16^3+ 4*16^2+10*16^{1+} 15*16^0 = 50351d$
- **2^8-1=11111111b =255d =FFh**
- **2^{16}-1=1111111111111111b =65535d =FFFFh**
- **2^{32}-1=1111111111111111…1b =4294967295d =FFFFFFFFh**

Unix "Onion"



User level vs. Kernel level

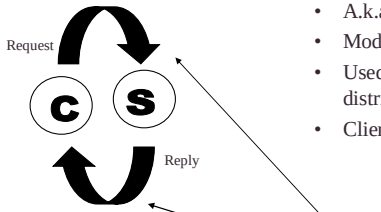
- Kernel (a.k.a. supervisory or privileged) level
  - All instructions are available
  - Total control possible so OS must say “Mine, all mine” (Daffy Duck)
- User level
  - Some instructions are not available any more
  - Programs can be modified and substituted by user

In theory, but not always in practice

Check it out: User vs. Kernel Level on Windows NT

- Start the PerfMon (start->administrative tools)
- add %user time and %privileged time
- Move the mouse around
  - %pt spikes
- Grab the perf mon window and move it around
  - %pt peaks

The Service Model



- A.k.a. the client/server model
- Model of interaction
- Used in both centralized and distributed systems
- Client can be Server and vice versa

Procedure Calls, System Calls, Traps, Interrupts, Shared Variables, or Message Passing

OS Service Examples

- Examples that are not provided at user level
  - File open, close, read and write
  - Control the CPU, or a single user takes over by doing
    ```
    while ( 1 ) ;
    ```
  - Protection:
    - Keep user programs from crashing OS
    - Keep user programs from crashing each other
- Examples that can be provided at user level
  - Read time of the day
  - Protected user level stuff

OS Responsibilities (1)

- Job control
  - Start, stop, kill
- User interface
  - Job Control Language (JCL) Interpretation
  - Window system
- Error handling
- Protection
- I/O handling
- Interrupt handling
  - Hardware
  - Software

Discussion topic: Window System part of the OS (1)?

- Yes:
  - Windows NT
    - Window Manager runs in Kernel Mode
    - Integrated with Graphic Device Drivers
    - Can not easily use several file systems at the same time or use another FS than NTFS
    - Performance benefit
- No:
  - Unix
    - X runs in User mode
    - Flexibility and “openness”
    - More overhead

Discussion topic: Window System part of the OS (2)?

- Solution space:
  - All in Kernel
  - All at user level
    - Must protect the display device or chaos possible
  - Split between Kernel and User level
    - Display drivers in Kernel
    - Rest at User level

OS Responsibilities

- Resource Control
  - Sharing
    - Scheduling
      - CPU
      - Memory (Registers, cache, memory (main, remote, disk)
      - I/O (network, interconnect, busses)
    - Multi access
  - Accounting

OS Characteristics

- Concurrency
  - Switching between tasks
  - Mutual exclusion
  - Condition synchronization
- Sharing
  - Allocation of resources
  - Concurrent access to data and other resources
  - Concurrent program execution
  - Protection of all resources including data and programs

Single process- single user
Single process - Multi user?
Multi process - Single user
Multi process - Multi user

OS Characteristics

- Efficiency
  - Overhead (low)
  - OS resource use (low)
  - CPU idle time (low)
  - Throughput (high)
  - Response time (short)
  - Fairness (yes, but what if not?)
- Reliability
  - Error isolation
  - Graceful degradation

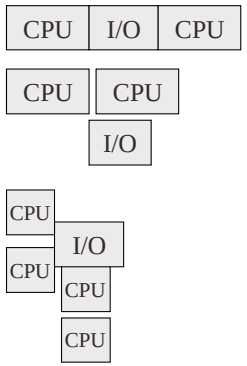
Multiprogramming
High & Long
Low and Short
Starvation

History: CPU waiting for I/O

- Assume
  - 1200 card program :-)
  - The assembler speed: 300 cards/sec
  - Card reader: 20 cards/sec
- Observations
  - 60 seconds to read program to memory
  - CPU runs the assembler for 4 seconds
- Implication
  - CPU is idle for 56 seconds = 93.3% of the time
  - CPU utilization is 6.7%
  - CPU and I/O device alternates, no overlap or interleaving

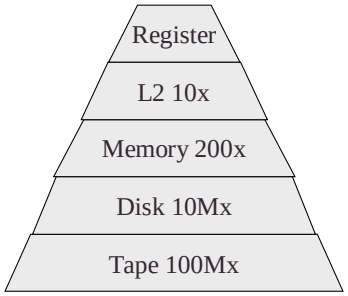
Processor Management

- Goals
  - Overlap between I/O and computation
  - Time sharing
  - Multiple CPU allocations
- Issues
  - Do not waste CPU resources
  - Synchronization and mutual exclusion
  - Fairness and deadlock free



Memory Management

- Goals
  - Support programs to run
  - Allocation and management
  - Transfers from and to secondary storage
- Issues
  - Efficiency & convenience
  - Fairness
  - Protection



Registers and memory

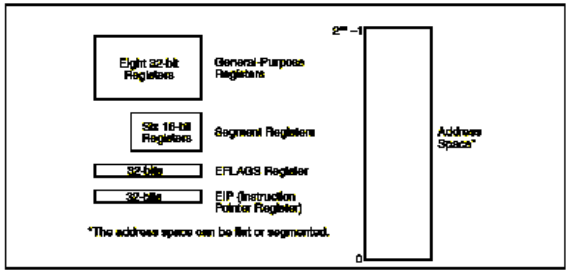


Figure 3-1. Pentium® Pro Processor Basic Execution Environment

System-level Registers

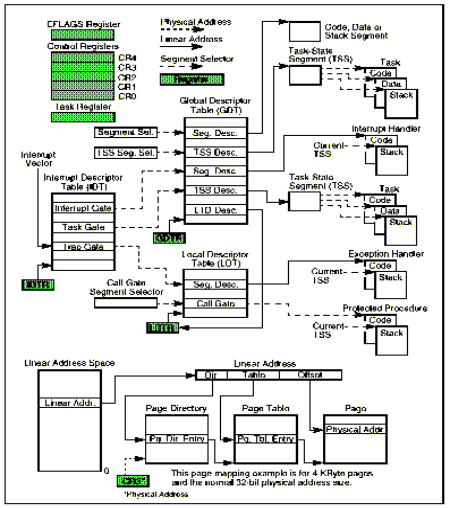
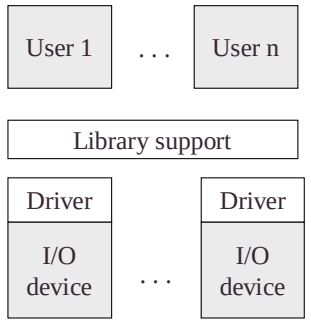


Figure 2-1. System-Level Registers and Data Structures

I/O Device Management

- Goals
  - Interactions between devices and applications
  - Ability to plug in new devices
- Issues
  - Efficiency
  - Fairness
  - Protection and sharing



OS Characteristics

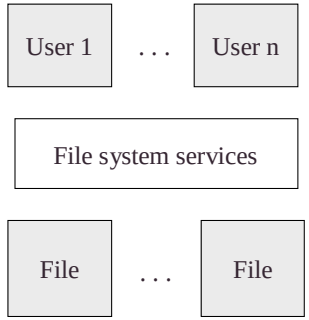
- Storing of data and programs
  - Simple and fast access
  - Protection against programs
    - failures
    - intentional
  - Protection against system
    - failures
    - intentional
- Non-deterministic
  - Time independence

Part of the OS?

User vs. Kernel Level?

File System

- A typical file system
  - Open a file with authentication
  - Read/write data in files
  - Close a file
- Can the services be moved to user level?



Discussion topic: User level FS?

- Yes: Minix
  - FS as a “server” at user level
    - almost a user process...
    - ...but booted together with OS
    - …and never terminates
    - …and gets higher CPU priority
    - …and a new server means recompiling the kernel
  - disk drivers at Kernel level
- NO: Unix and Windows NT
  - File system at Kernel level

OS Characteristics (4)

- Management
  - Simple
  - Modular
  - Well defined interfaces
  - Good documentation
- Small size
  - Simpler
  - Less bugs
  - Cheaper
  - Faster

Discussion topic: Network boot

- Need “netboot” code in EPROM
- Netboot must
  - Understand the network protocols (Ethernet, TCP/IP, something)
  - download boot code from a server
  - execute boot code to download OS
- Issues
  - Client must
    - have name of server and of self
  - Server must have a database of clients
  - Security, Protection, Restrictions

Ways to Develop An Operating System

- A hardware simulator
- A virtual machine
- A good kernel debugger
  - When OS crashes, always goes to the debugger
  - Debugging over the network