Operating System Structures

Jinkyu Jeong (jinkyu@skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu
OS Design and Implementation

- Design and Implementation of OS not “solvable”
  - Some approaches have proven successful
- Internal structure of different OSs can vary widely
- Start design by defining goals and specifications
- Affected by choice of hardware, type of system

**User goals and System goals**

- User goals – operating system should be convenient to use, easy to learn, reliable, safe, and fast
- System goals – operating system should be easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient
OS Design and Implementation

- Important principle to separate
  - Policy: What will be done?
  - Mechanism: How to do it?

- Mechanisms determine how to do something, policies decide what will be done

- Separation of policy from mechanism
  - Allows maximum flexibility if policy decisions are to be changed later

- Specifying and designing an OS is highly creative task of software engineering
Operating System Design Paradigms

- Monolithic kernel
- Microkernel
- Hybrid kernel
- Exokernel
- Virtual machines
- Distributed operating systems
Operating System Design Paradigms

Monolithic Kernel based Operating System
- Application
- VFS, System call
- IPC, File System
- Scheduler, Virtual Memory
- Device Drivers, Dispatcher, ...
- Hardware

Microkernel based Operating System
- Application
- Application IPC
- UNIX Server
- Device Driver
- File Server
- Basic IPC, Virtual Memory, Scheduling
- Hardware

"Hybrid kernel" based Operating System
- Application
- Application IPC
- UNIX Server
- Device Driver
- File Server
- Basic IPC, Virtual Memory, Scheduling
- Hardware
Operating System Structures

- MS-DOS – written to provide the most functionality in the least space
  - Not divided into modules
  - Although MS-DOS has some structure, its interfaces and levels of functionality are not well separated
### UNIX System (monolithic kernel)

<table>
<thead>
<tr>
<th>Kernel interface to the hardware</th>
<th>System-call interface to the kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>terminal controllers terminals</td>
<td>signals terminal handling</td>
</tr>
<tr>
<td>device controllers disks and tapes</td>
<td>character I/O system</td>
</tr>
<tr>
<td>memory controllers physical memory</td>
<td>file system swapping block I/O</td>
</tr>
<tr>
<td></td>
<td>system</td>
</tr>
<tr>
<td></td>
<td>disk and tape drivers</td>
</tr>
<tr>
<td></td>
<td>CPU scheduling</td>
</tr>
<tr>
<td></td>
<td>page replacement</td>
</tr>
<tr>
<td></td>
<td>demand paging</td>
</tr>
<tr>
<td></td>
<td>virtual memory</td>
</tr>
</tbody>
</table>

- (the users)
  - shells and commands
  - compilers and interpreters
  - system libraries
Monolithic Kernel

- All OS services operate in kernel space
- Good performance
- Disadvantages
  - Dependencies between system component
  - Complex & huge (millions(!) of lines of code)
  - Larger size makes it hard to maintain
- Examples
  - Multics, Unix, BSD, Linux
Microkernel System Structure

- Application Program
- File System
- Device Driver

Interprocess Communication
- User mode
- Kernel mode

Memory Management
- Messages

CPU Scheduling

Microkernel

Hardware
Microkernel

- Minimalist approach
  - IPC, virtual memory, thread scheduling
- Put the rest into user space
  - Device drivers, networking, file system, user interface
- More stable with less services in kernel space
- Disadvantages
  - Lots of system calls and context switches
  - Slow inter-process communication
- Examples
  - Mach, L4, AmigaOS, Minix, K42
Microkernel vs. Monolithic Kernel

- Moves as much from the kernel into user space
- Mach example of microkernel
  - Mac OS X kernel (Darwin) partly based on Mach
- Communication takes place between user modules using message passing

Benefits
- Easier to extend a microkernel
- Easier to port the operating system to new architectures
- More reliable (less code is running in kernel mode)
- More secure

Detriments
- Performance overhead of user space to kernel space communication
Many modern operating systems implement loadable kernel modules

- Easy and dynamic kernel service extension
- Uses object-oriented approach
- Each core component is separate
- Each talks to the others over known interfaces
- Each is loadable as needed within the kernel

Overall, similar to layers but with more flexible

- Linux, Solaris, etc
Solaris Modular Approach

- device and bus drivers
- scheduling classes
- file systems
- loadable system calls
- executable formats
- STREAMS modules
- miscellaneous modules
Hybrid Kernels

- Most modern operating systems are actually not one pure model
  - Hybrid combines multiple approaches to address performance, security, usability needs
  - Linux and Solaris kernels in kernel address space, so monolithic, plus modular for dynamic loading of functionality
  - Windows mostly monolithic, plus microkernel for different subsystem personalities
Exokernel

- **Follows end-to-end principle**
  - Extremely minimal
  - Fewest hardware abstractions as possible
  - Just allocates physical resources to apps

- **Disadvantages**
  - More work for application developers

- **Examples**
  - Nemesis
  - ExOS
### Mac OS X

#### Graphical User Interface
- **Aqua**

#### Application Environments and Services
- **Java**
- **Cocoa**
- **Quicktime**
- **BSD**

#### Kernel Environment
- **Mach**
- **BSD**

#### I/O Kit
- **I/O Kit**

#### Kernel Extensions
- **Kernel Extensions**
Android

- Developed by Open Handset Alliance (mostly Google)
  - Open Source
- Similar stack to IOS
- Based on Linux kernel but modified
  - Provides process, memory, device-driver management
  - Adds power management
- Runtime environment includes core set of libraries and Dalvik virtual machine
  - Apps developed in Java plus Android API
    - Java class files compiled to Java bytecode then translated to executable than runs in Dalvik VM
- Libraries include frameworks for web browser (webkit), database (SQLite), multimedia, smaller libc
Android Applications

Application Framework

Libraries

- SQLite
- openGL
- surface manager
- media framework
- webkit
- libc

Android runtime

- Core Libraries
- Dalvik virtual machine

Linux kernel
Virtual Machines

Non-virtual machine

Virtual machine

processes

kernel

hardware

programming interface

kernels

VM1

VM2

VM3

virtual machine manager

hardware
Distributed Operating Systems

- It is cheaper to buy multiple ordinary computers than buy a super computer
- Basic philosophy
  - Let’s group multiple computers and use them like one
- Single system image
  - Although it consists of multiple nodes, it appears to users and applications as a single-node
- Kernel must control local resource while providing global services by collaborating with other nodes
Working Together

Fig. 1-14. General structure of a multicomputer operating system.
Fig. 1-19. General structure of a network operating system.
Cloud Computing

- On-demand self-service
- Ubiquitous network access
  - Anywhere, anytime, anydevice
- Location-independent resource pooling
- Rapid elasticity
- Pay as you go

- E.g., Amazon EC2, Microsoft Azure, Google AppEngine, ...
Mobile-Cloud Computing

- Combination of cloud computing, mobile computing and wireless networks
- To bring rich computational resources to mobile users, network operators

![Diagram of Mobile-Cloud Computing System]
IoT Operating Systems

- Collaboration among intelligent things
- Tiny low-cost operating system design
- What will come?