Operating Systems –
OS Architecture Models

ECE 344
OS Architecture

• Designs that have been tried in practice
  – Monolithic systems
  – Layered systems
  – Virtual machines
  – Client/server a.k.a. Microkernels

• Many of the concepts governing these architectures apply to software architectures in general
Monolithic Systems

- A.k.a., “The Big Mess” or spaghetti code
- Prominent in the early days
- The structure consists of no-structure
- The system is a collection of procedures
- Each procedure can call any other procedure
- **No information hiding** (as opposed to modules, packages, classes)
Monolithic Systems (cont.’d.)

- **A little structure**, imposed by exposing a set of system calls to the outside
- Supporting these system calls through utility procedures (check data passed to system call, move data around …)
  1. a **main procedure** requesting the services
  2. a **set of service procedures** that carry out system calls
  3. a set of **utility procedures** supporting the system calls

```plaintext
{ simple structure (e.g. fetching data from user space) }
```

Diagram:
- System calls
  - ... service procedures
    - ... utility procedures
      - (e.g. fetching data from user space)
A Monolithic OS
Pros & Cons

• **Tightly integrated code** in one address space
• **Unreliable**, as a bug anywhere in the kernel can bring down the whole system
• Tight integration has high potential for **efficient** use of resources and for **efficient** code
• Early designs lacked potential for **extension**
• Modern designs can load executable modules dynamically (i.e., **extensible**)
  – E.g., Linux, FreeBSD, Solaris
MS-DOS System Structure

• 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
MS-DOS Structure

- Application program
- Resident system program
- MS-DOS device drivers
- ROM BIOS device drivers

ECE 344 Operating Systems
UNIX System Structure

- UNIX – limited by hardware functionality (at its time of origin), the original UNIX operating system **had limited structuring.**
- The UNIX OS consists of two separable parts.
  - Systems programs
  - The **kernel**
    - Consists of **everything below the system-call interface** and **above the physical hardware**
    - Provides the **file system**, **CPU scheduling**, **memory management**, and other operating-system functions; a large number of functions for one level.
## UNIX System Structure

<table>
<thead>
<tr>
<th>(the users)</th>
</tr>
</thead>
<tbody>
<tr>
<td>shells and commands</td>
</tr>
<tr>
<td>compilers and interpreters</td>
</tr>
<tr>
<td>system libraries</td>
</tr>
</tbody>
</table>

**system-call interface to the kernel**

- signals terminal handling character I/O system terminal drivers
- file system swapping block I/O system disk and tape drivers
- CPU scheduling page replacement demand paging virtual memory

**kernel interface to the hardware**

- terminal controllers terminals
- device controllers disks and tapes
- memory controllers physical memory

---

ECE 344 Operating Systems
Layered Systems

- **Generalization** of previous scheme
- Organization into a *hierarchy of layers*
- **Layer n+1 uses** services (exclusively) supported by layer n
- Easier to **extend** and **evolve**
- A call may have to propagate through lots of layers
  - At occasions (optimization) layer $n+1$ may also access layers $n-k$ directly
- **Upcall**, layer $n-k$ calls into layer $n$ has also been proposed (e.g., in the context of thread scheduling)
Operating System Layers

- new operations
- hidden operations
- existing operations

layer $M$

layer $M-1$
# THE (Dijkstra, 1968)

|----------|--------------|-------------------|----------------------|-------------|-----------------------------------|

ECE 344 Operating Systems
Ring-based protection

System calls

Increasing privilege level
Microkernel System Structure

- Moves as much as possible from the kernel into “user” space
- 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 (a server crashing in userspace)
- Not clear what should go into the microkernel
- Mach, QNX, NT, L4
Microkernel
Microkernel Examples

- AIX
- AmigaOS
- Amoeba
- Chorus microkernel
- EROS
- Haiku
- K42
- LSE/OS (a nanokernel)
- KeyKOS (a nanokernel)
- The L4 microkernel family
- Mach, used in GNU Hurd,
- NEXTSTEP, OPENSTEP, and Mac OS X
- MERT
- Minix
- MorphOS
- NewOS
- QNX
- Phoenix-RTOS
- RadiOS
- Spring operating system
- VSTa
- Symbian OS
- OSE

Nanokernel is a very light-weight microkernel
Windows NT Client-Server Structure
Monolithic vs. Microkernel

- *Mon.* tend to be **easier to design**, therefore **faster development cycle** and more potential for growth (see Linux)
- *Mon.* tend to be **more efficient** due to use of shared kernel memory (instead of IPC)
  - However, very efficient micro kernels have been designed in research and laboratory settings
- *Micro.* tend to be used for embedded systems (e.g., robotic, medical etc.)
- In *Micro.* Many OS components reside in their own, private protected address space (not possibly in *Mon.* designs)
Others

- Hybrid kernels (a.k.a. modified microkernel)
  - Add more code to the kernel for efficiency
    - Windows 2000, Windows XP
  - Based on message passing
  - Based on minimalist concept
  - Not based on loading modules
Exokernel

• Traditionally
  – Kernel hides hardware from application
  – Based on conceptual model (files systems, virtual address space, schedulers, sockets)
• Generally, this eases application development
• Sometimes problematic (e.g., security / privacy of stored data upon deletion)
  – Security-oriented application requires file system to delete data
  – Reliability-oriented application requires file system to keep data for failure recovery purposes
Exokernel cont’d.

- Kernel allocates physical resources to application
- Application decides what to do with the resources
  - Application can link to a libOS to emulate a conventional OS
  - Application uses resources as it wishes
- Exokernel could emulate several OSes, one for each application it runs
Exokernel Summary

• Provide as few abstractions as possible
• Kernel is relatively small
  – Allocates, protects, and multiplexes resources
• Low-level hardware access enables to build custom abstractions on per application basis
  – Performance reasons
• Does not force the layering of abstractions
Virtual Machines

- A virtual machine takes the layered approach to its logical conclusion. **Hardware is simulated in software; all resources are virtualized; individual OS run on virtualized resources**

- A virtual machine provides an interface *identical* to the underlying bare hardware

- The operating system creates the illusion of multiple processes, each executing on its own processor with its own (virtual) memory
Virtual Machines (Cont.)

• The resources of the physical computer are shared to create the virtual machines.
  – CPU **scheduling** can create the appearance that **users have their own processor**.
  – Spooling and a file system can provide **virtual disks, virtual memory** and **virtual printers**.
  – A normal user time-sharing terminal serves as the **virtual machine operator’s console**
System Models

Processes

Kernel

Hardware

Programming Interface

Kernel

VM1

Virtual Machine Implementation

Hardware

ECE 344 Operating Systems
Pros/Cons of Virtual Machines

• VM model provides **complete protection**
• At the cost of **not enabling any direct resource sharing**
• A virtual-machine system is a **perfect vehicle for operating-systems research** and development.
  – System development is done on the virtual machine, instead of on a physical machine and so does not disrupt normal system operation.
• The virtual machine concept is difficult to implement due to the effort required to provide an **exact** duplicate to the underlying machine.
• Available in practice (IBM, VMWare, Sys161)