Announcements

- Teaching Assistant:
  - Asim Shrestrah
  - Email: ashres1@lsu.edu

- All of you should be now in the class mailing list.
  - Let me know if you haven’t received any messages yet.

- Lecture notes are available on the course web site.

Roadmap

- OS Structures
  - Multiprogramming and Timesharing
  - Storage Structure
  - System Calls

- OS Design and Implementation
  - Different Design Approaches

Operating System Structure

- **Multiprogramming** needed for efficiency
  - Single process cannot keep CPU and I/O devices busy at all times
  - Multiprogramming organizes jobs (code and data) so CPU always has one to execute
  - How it works:
    - A subset of total jobs in system is kept in memory simultaneously
    - One job selected and run via job scheduling
    - When it has to wait (for I/O for example), OS switches to another job

- **Timesharing** is logical extension in which CPU switches jobs so frequently that users can interact with each job while it is running, creating interactive computing
  - Response time should be < 1 second
  - Each user has at least one program loaded in memory and executing ⇒ process
  - If several jobs ready to be brought into memory ⇒ job scheduling
  - If several jobs ready to run at the same time ⇒ CPU scheduling
  - If processes don’t fit in memory, **swapping** moves them in and out to run
  - Virtual memory allows execution of processes not completely in memory
### Storage Structure

- **Main memory** - only large storage media that the CPU can access directly.
- **Secondary storage** - extension of main memory that provides large nonvolatile storage capacity.
- Magnetic disks - rigid metal or glass platters covered with magnetic recording material
  - Disk surface is logically divided into *tracks*, which are subdivided into *sectors*.
  - The *disk controller* determines the logical interaction between the device and the computer.
- **Tertiary Storage**: low cost, high capacity storage
  - eg. tape libraries, CD, DVD, floppy disks
- **Tape** is an economical medium for purposes that do not require fast random access, e.g., backup copies of disk data, holding huge volumes of data.
- Large tape installations typically use robotic tape changers that move tapes between tape drives and storage slots in a tape library.
  - stacker - library that holds a few tapes
  - silo - library that holds thousands of tapes

### Disk Architecture

- **Disk surface** is logically divided into *tracks*, which are subdivided into *sectors*.
- The *disk controller* determines the logical interaction between the device and the computer.

### Storage Hierarchy

- Storage systems organized in hierarchy.
  - Speed
  - Cost
  - Volatility*

*volatile: loses its content when the power is off.

### Performance of Various Levels of Storage

- Movement between levels of storage hierarchy can be explicit or implicit

<table>
<thead>
<tr>
<th>Level</th>
<th>Name</th>
<th>Cache</th>
<th>Main Memory</th>
<th>Disk Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>registers</td>
<td>cache</td>
<td>main memory</td>
<td>disk storage</td>
</tr>
<tr>
<td>2</td>
<td>1 - 512</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>512 - 16 MB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>&gt; 16 GB</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Caching

- **Caching** - copying information into faster storage system; main memory can be viewed as a last cache for secondary storage.
- Important principle, performed at many levels in a computer (in hardware, operating system, software)
- Information in use copied from slower to faster storage temporarily
- Faster storage (cache) checked first to determine if information is there
  - If it is, information used directly from the cache (fast)
  - If not, data copied to cache and used there
- Cache smaller than storage being cached
  - Cache management important design problem
  - Cache size and replacement policy
Migration of Integer A from Disk to Register

- Multitasking environments must be careful to use most recent value, not matter where it is stored in the storage hierarchy

- Multiprocessor environment must provide cache coherency in hardware such that all CPUs have the most recent value in their cache

- Distributed environment situation even more complex
  - Several copies of a datum can exist

System Calls

- Programming interface to the services provided by the OS
- Typically written in a high-level language (C or C++)
- Mostly accessed by programs via a high-level Application Program Interface (API) rather than direct system call use
  - Ease of programming
  - Portability
- Three most common APIs are Win32 API for Windows, POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X), and Java API for the Java virtual machine (JVM)

Example of System Calls

- System call sequence to copy the contents of one file to another file

System Call Implementation

- Typically, a number associated with each system call
  - System-call interface maintains a table indexed according to these numbers
- The system call interface invokes intended system call in OS kernel and returns status of the system call and any return values
- The caller need know nothing about how the system call is implemented
  - Just needs to obey API and understand what OS will do as a result call
  - Most details of OS interface hidden from programmer by API
    - Managed by run-time support library (set of functions built into libraries included with compiler)

API – System Call – OS Relationship

Standard C Library Example

- C program invoking printf() library call, which calls write() system call
**Operating System Design and Implementation**

- Start by defining goals and specifications
- Affected by choice of hardware, type of system
  - Batch, time shared, single user, multi user, distributed
- **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
- No unique solution for defining the requirements of an OS
  - Large variety of solutions
  - Large variety of OS

**Operating System Design and Implementation (Cont.)**

- Important principle: to separate policies and mechanisms
  - **Policy**: What will be done?
  - **Mechanism**: How to do something?
- Eg. to ensure CPU protection
  - Use Timer construct (mechanism)
  - How long to set the timer (policy)
- The separation of policy from mechanism is allows maximum flexibility if policy decisions are to be changed later

**OS Design Approaches**

- Simple Structure
- Layered Approach
- Microkernels
- Modules

**Simple Structure**

- No well defined structure
- Start as small, simple, limited systems, and then grow
- MS-DOS - written to provide the most functionality in the least space
  - Not divided into modules
  - Its interfaces and levels of functionality are not well separated
  - e.g. application programs can access low level drivers directly
    - Vulnerable to errant (malicious) programs
MS-DOS Structure

UNIX

- UNIX - limited by hardware functionality, 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

Layered Approach

- The operating system is divided into a number of layers (levels), each built on top of lower layers.
  - The bottom layer (layer 0), is the hardware;
  - The highest (layer N) is the user interface.
- With modularity, layers are selected such that each uses functions (operations) and services of only lower-level layers
  - GLUnix: Global Layered Unix

Layered Operating System

Microkernel System Structure

- Move all non-essential components from the kernel into “user” space
- Main function of microkernel: Communication between client programs and various services which are run in user space
  - Uses message passing (never direct interaction)
- Benefits:
  - Easier to extend the OS
  - Easier to port the OS to new architectures
  - More reliable (less code is running in kernel mode)
  - More secure
- Detriments:
  - Performance overhead of user space to kernel space communication
- Examples: QNX, Tru64 UNIX
Layered OS vs Microkernel

Modular Approach

- Most modern operating systems implement kernel modules
  - 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 more flexible
  - Any module can call any other module

Solaris Modular Approach

Mac OS X Structure - Hybrid

- BSD: provides support for command line interface, networking, file system, POSIX API and threads
- Mach: memory management, RPC, IPC, message passing

Summary

- OS Structures
  - Multiprogramming and Multitasking
  - Storage Structure
    - Primary, secondary & tertiary storage
    - System Calls
- OS Design and Implementation
  - Different Design Approaches
    - Unstructured, Layered, Modular, Microkernel,
    - and Hybrid approaches
- Next Lecture: Processes
- Reading Assignment: Chapter 2 from Silberschatz.

Acknowledgements

- “Modern Operating Systems” book and supplementary material by A. Tanenbaum
- R. Doursat and M. Yuksel from UNR