CS 571 Operating Systems

Dr. Songqing Chen

Outline

- Introduction
  - What is an Operating System?
  - Co-evolution of Computer Systems and Operating System Concepts

- Computer System Structures

- Operating System Structures
What is an Operating System?

- A program that acts as an *intermediary* between the user of a computer and the computer hardware.

- Operating system goals
  - **Convenience**: Make the computer system convenient to use.
  - **Efficiency**: Manage the computer system resources in an efficient manner

- “Everything a vendor ships when you order an operating system” is a good approximation

- “The one program running at all times on the computer” is the kernel. Everything else is either a system program (ships with the operating system) or an application program

- The *government* metaphor

---

Computer System Components

```
  User 1  User 2  User 3  ...  User n

  compiler  assembler  Text editor  Database system

  System and Application Programs

  Operating System

  Hardware
```
Computer System Components

1. **Hardware** – provides basic computing resources (CPU, memory, I/O devices).
2. **Operating system** – controls and coordinates the use of the hardware among various application programs for various users.
3. **Application programs** – define the ways in which the system resources are used to solve the computing problems of the users (compilers, database systems, video games, business programs).
4. **Users**

**Operating System Definitions:**

**System View**

- **Resource allocator** – manages and allocates resources.
- **Control program** – controls the execution of user programs and operations of I/O devices.
- **Kernel** – the one program running at all times (all else being application programs).

- I/O is a critical (and often under-valued) component of any computer system
  - For interaction with users/external world
  - For correct operation
  - For performance
Co-evolution of Computer Systems and Operating System Concepts

- Mainframe Systems
  - Batch Systems
  - Multi-programmed Systems
  - Time-sharing Systems
- Desktop Systems
- Modern Variants
  - Parallel Systems
  - Distributed Systems
  - Real-time and Embedded Systems
  - Handheld Systems

Mainframe Systems

- First computers to tackle many commercial and scientific applications
- Mainframes evolved through batch, multiprogrammed and time-shared systems
- Early systems were afforded only by major government agencies or universities:
  - physically enormous machines run from a console.
  - the user submitted the job to the human operator in the form of punched cards.
  - The operator collects the output and returns it to the user.
- Batch systems: To speed up processing, operators batched together jobs with similar needs and ran them through the computer as a group.
Batch Systems

Several (and not necessarily similar) jobs are kept in the main memory at the same time, and the CPU is switched to another job when I/O takes place.

Objective: ?

Multiprogrammed Systems

Several (and not necessarily similar) jobs are kept in the main memory at the same time, and the CPU is switched to another job when I/O takes place.

Objective: ?
“Uniprogramming” vs Multiprogramming

(a) Uniprogramming

(b) Multiprogramming with two programs

OS Features Needed for Multiprogramming

- **Job Scheduling** – must choose the processes that will be brought to memory
- **Memory Management** – must allocate the memory to several jobs
- **CPU Scheduling** – must choose among several jobs ready to run
- **OS/360**, developed by IBM to run on its System/360 series, was the first multiprogrammed operating system (1964).
Time-Sharing Systems: Interactive Computing

- Extension of multiprogrammed systems to allow on-line interaction with users.
- Each user is provided with an on-line terminal.
- **Objective:** *Response time* for each user should be short.

- The CPU is multiplexed among several jobs that are kept in memory and on disk.
- A job swapped in and out of memory to the disk.

- CPU is allocated to another job when I/O takes place.
- All active users must have a *fair* share of the CPU time (e.g. 10 ms for each user).

Desktop Systems

- *Personal computers* – computer system originally dedicated to a single user.
- I/O devices – keyboard, mouse, printers, ...
- **Objective:** User convenience and responsiveness.
  - Individuals have sole use of computers
  - A single user may not need advanced features of mainframe OS (maximize utilization, protection).
- Today, may run several different types of operating systems (Windows, MacOS, Linux)
Parallel Systems

- Multiprocessor systems with more than one CPU in close communication.
- *Tightly coupled system* – processors share memory and a clock; communication usually takes place through the shared memory.
- Advantages of parallel system
  - Increased *throughput*
  - Economy of scale
  - Increased reliability
    - graceful degradation
    - fault-tolerant systems

Parallel Systems

- *Symmetric multiprocessing (SMP)*
  - All processors are peers
  - Kernel routines can execute on different CPUs, in parallel

- *Asymmetric multiprocessing*
  - Master/slave structure
  - The kernel runs on a particular processor
  - Other CPUs can execute user programs and OS utilities.
Parallel Systems (Cont.)

- Multi-core architectures
  - Include multiple computing cores on a single chip
  - Critical to address the **power wall**
  - Need to exploit **parallelism** at run-time

![Diagram of multi-core architecture]

Intel Core i7 Cache Hierarchy

- **Core 0**
  - Regs
  - L1 d-cache
  - L1 i-cache
  - L2 unified cache
  - L3 unified cache (shared by all cores)
  - Main memory

- **Core 3**
  - Regs
  - L1 d-cache
  - L1 i-cache
  - L2 unified cache
  - L3 unified cache (shared by all cores)
  - Main memory

- **L1 i-cache and d-cache**:
  - 32 KB, 8-way,
  - Access: 4 cycles

- **L2 unified cache**:
  - 256 KB, 8-way,
  - Access: 11 cycles

- **L3 unified cache**:
  - 8 MB, 16-way,
  - Access: 30-40 cycles

- **Block size**: 64 bytes
Moore’s Law

- “Number of transistors on a chip will double every 18 months”
  - “Prediction” by Gordon Moore, co-founder of Intel, in 1965
- The increase in chip complexity has been accompanied by significant increase in clock frequency speeds.

### CPU Clock Rates

<table>
<thead>
<tr>
<th>Year</th>
<th>CPU</th>
<th>Clock Rate (MHz)</th>
<th>Cycle Time (ns)</th>
<th>Cores</th>
<th>Effective Cycle Time (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1980</td>
<td>8080</td>
<td>1</td>
<td>1000</td>
<td>1</td>
<td>1000</td>
</tr>
<tr>
<td>1990</td>
<td>386</td>
<td>20</td>
<td>50</td>
<td>1</td>
<td>50</td>
</tr>
<tr>
<td>1995</td>
<td>Pentium</td>
<td>150</td>
<td>6</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>2000</td>
<td>Pentium P-III</td>
<td>600</td>
<td>1.6</td>
<td>1</td>
<td>1.6</td>
</tr>
<tr>
<td>2003</td>
<td>Pentium P-4</td>
<td>3300</td>
<td>0.3</td>
<td>1</td>
<td>0.3</td>
</tr>
<tr>
<td>2005</td>
<td>Core 2</td>
<td>2000</td>
<td>0.50</td>
<td>2</td>
<td>1.0</td>
</tr>
<tr>
<td>2010</td>
<td>Core i7</td>
<td>2500</td>
<td>0.4</td>
<td>4</td>
<td>1.6</td>
</tr>
<tr>
<td>2010:1980</td>
<td>---</td>
<td>2500</td>
<td>2500</td>
<td>4</td>
<td>10,000</td>
</tr>
</tbody>
</table>

Inflection point in computer history when designers hit the “Power Wall”
Problems with Ever Increasing Clock Speed and Logic Density

- Power Wall

- CPU - Memory Performance Gap

Exponential Increase in Power Density

From Fred Pollack’s keynote address at Micro-32 conference
The CPU-Memory Gap

The gap widens between memory, disk, and CPU speeds.

Distributed Systems

- Distribute the computation among several physical processors.

- Loosely coupled system – each processor has its own local memory; processors communicate with one another through various communications lines

- Advantages of distributed systems
  - Resource and Load Sharing
  - Reliability
  - Communications
Real-Time and Embedded Systems

- A real-time system is used when rigid time requirements have been placed on the operation of a processor or the flow of data.

- An embedded system is a component of a more complex system
  - Control of a nuclear plant
  - Missile guidance
  - Control of home and car appliances (microwave oven, DVD players, car engines, …)

- Real-time systems
  - have well-defined time constraints.
  - may be either hard or soft real-time.

Handheld Systems

- Cellular telephones
- Smart Phones
  - In 2011, more smart phones than PCs were shipped
- How does this change the study of Operating Systems?
Computer-System Structures

- Computer System Organization
- Instruction Execution
- Computer System Operation
- Interrupt Processing
- Storage Structure
- Storage Hierarchy
Instruction Execution

- While executing a program, the CPU:
  - fetches the next instruction from memory (loading into IR)
  - decodes it to determine its type and operands
  - executes it
- May take multiple clock cycles to execute an instruction
- Examples:
  - LOAD R1, #3
  - LOAD R2, M2
  - STORE M3, R4
  - ADD R1, R2, R3
- Each CPU has a specific set of instructions that it can execute (instruction-set architecture).

Instruction Execution

- Registers
  - General registers (data/address)
  - Program Counter (PC): contains the memory address of the next instruction to be fetched.
  - Stack Pointer (SP): points to the top of the current stack in memory. The stack contains one frame for each procedure that has been entered but not yet exited.
  - Program Status Word (PSW): contains the condition code bits and various other control bits.
  - ...
- When time multiplexing the CPU, the operating system will often stop the running program to (re)start another one. In these cases, it must save the “state information” (e.g. values of the registers).
Computer-System Operation

- I/O devices and CPU can execute concurrently.
- Each device controller has local buffer(s).
- CPU moves data from/to main memory to/from local buffers.
- I/O is from/to the device to/from local buffer of controller.
- The device driver is special operating system software that interacts with the device controller.
- Typically, the device controller informs CPU that it has finished its operation by causing an interrupt.

Instruction Cycle with Interrupts

- Important hardware and software events are signaled to the CPU by an interrupt.
Classes of Interrupts

- **I/O Interrupts**: Generated by an I/O controller, to signal normal completion of an operation or to signal a variety of error conditions.

- **Timer Interrupts**: Generated by a timer within the processor. This allows the operating system to perform certain functions on a regular basis.

- **Hardware Failure Interrupts**: Generated by a failure (e.g. power failure or memory parity error).

- **Traps (Software Interrupts)**: Generated by some condition that occurs as a result of an instruction execution
  - Errors
  - User request for an operating system service

Interrupt Mechanism

- Interrupt transfers control to the interrupt service routine generally through the interrupt vector (e.g., Intel) which contains the addresses of all the service routines. (Alternatively, the machine has a status register or cause register that holds the reason for the interrupt – MIPS architecture)

- **Interrupt Service Routines (ISRs)**: Separate segments of code determine what action should be taken for each type of interrupt.

- Once the interrupt has been serviced by the ISR, the control is returned to the interrupted program. Need to save the “process state” (registers, PC, ...) before ISR takes over.

- Modern operating systems are *interrupt-driven*. 
Basic Interrupt Processing

1. The interrupt is issued
2. Processor finishes execution of current instruction
3. Processor signals acknowledgement of interrupt
4. Processor pushes PSW and PC onto control stack
5. Processor loads new PC value through the interrupt vector
6. ISR saves remainder of the process state information
7. ISR executes
8. ISR restores process state information
9. Old PSW and PC values are restored from the control stack

What if another interrupt occurs during interrupt processing?
Incoming interrupts are disabled while another interrupt is being processed to prevent a lost interrupt.

Direct Memory Access Structure

- Used for high-speed I/O devices able to transmit information at (close to) memory speeds.
- Device controller transfers blocks of data from buffer storage directly to main memory without CPU intervention.
- Only one interrupt is generated per block, rather than one interrupt per byte/word.
I/O Structure

- After I/O starts, control returns to user program only upon I/O completion.
  - Wait instruction idles the CPU until the next interrupt
  - Wait loop (contention for memory access).
  - At most one I/O request is outstanding at a time, no simultaneous I/O processing.

- After I/O starts, control returns to user program without waiting for I/O completion.
  - System call – request to the operating system to allow user to wait for I/O completion.
  - Device-status table contains entry for each I/O device indicating its type, address, and state.
  - Operating system indexes into I/O device table to determine device status and to modify table entry to include interrupt.

Device-Status Table

<table>
<thead>
<tr>
<th>device: card reader 1</th>
<th>status: idle</th>
</tr>
</thead>
<tbody>
<tr>
<td>device: line printer 3</td>
<td>status: busy</td>
</tr>
<tr>
<td>device: disk unit 1</td>
<td>status: idle</td>
</tr>
<tr>
<td>device: disk unit 2</td>
<td>status: idle</td>
</tr>
<tr>
<td>device: disk unit 3</td>
<td>status: busy</td>
</tr>
</tbody>
</table>

- request for line printer
  - address: 38546
  - length: 1372

- request for disk unit 3
  - file: xxx
    - operation: read
    - address: 43046
    - length: 20000
  - file: yyy
    - operation: write
    - address: 03458
    - length: 500

- ...
Dual-Mode Operation

- Operating System must protect itself and all other programs (and their data) from any malfunctioning program.

- Provide hardware support to differentiate between at least two modes of operations.
  1. *User mode* – execution done on behalf of a user.
  2. *Kernel mode* (also *monitor mode* or *system mode*) – execution done on behalf of operating system.

Dual-Mode Operation (Cont.)

- *Mode bit* added to computer hardware to indicate the current mode: kernel (0) or user (1).
- When an interrupt occurs hardware switches to kernel mode.

*Privileged instructions* can be issued only in kernel mode.
Transition From User to Kernel Mode

The system call can be executed by a generic trap instruction (or in some systems, by an instruction such as syscall).

Operating-System Structures

- System Components
- System Calls
- System Programs
- Operating System Design Approaches
Process Management

- A process is a program in execution. It is a unit of work within the system. A program is a passive entity, a process is an active entity.
- Process needs resources to accomplish its task
  - CPU, memory, I/O, files
  - Initialization data
- Process termination requires reclaim of any reusable resources
- Single-threaded process has one program counter specifying location of next instruction to execute
  - Process executes instructions sequentially, one at a time, until completion
- Multi-threaded process has one program counter per thread
- Typically system has many processes, some user, some operating system running concurrently on one or more CPUs
  - Concurrency by multiplexing the CPUs among the processes / threads

Process

- Introduced to obtain a systematic way of monitoring and controlling program execution
- A process is an executable program with:
  - associated data (variables, buffers…)
  - execution context: ie. all the information that
    • the CPU needs to execute the process
      - content of the processor registers
    • the OS needs to manage the process:
      - priority of the process
      - the event (if any) after which the process is waiting
      - other data (that we will introduce later)
A Simple Implementation of Processes

- The process index register contains the index into the process list of the currently executing process (B)
- A process switch from B to A consist of storing (in memory) B’s context and loading (in CPU registers) A’s context
- A data structure that provides flexibility (to add new features)

Memory Management

- All data in memory before and after processing
- All instructions in memory in order to execute
- Memory management determines what is in memory when
  - Optimizing CPU utilization and computer response to users
- Memory management activities
  - Keeping track of which parts of memory are currently being used and by whom
  - Deciding which processes (or parts thereof) and data to move into and out of memory
  - Allocating and deallocating memory space as needed
- **Virtual memory** management is an essential part of most operating systems
Storage Management

- OS provides uniform, logical view of information storage
  - Abstracts physical properties to logical storage unit - file
  - Each medium is controlled by device (i.e., disk drive, tape drive)
    - Varying properties include access speed, capacity, data-transfer rate, access method (sequential or random)
- File-System management
  - Files usually organized into directories
  - Access control on most systems to determine who can access what
  - OS activities include
    - Creating and deleting files and directories
    - Primitives to manipulate files and dirs
    - Mapping files onto secondary storage
    - Backup files onto stable (non-volatile) storage media

Storage-Device Hierarchy
Storage Structure

- **Main memory** – only large storage media that the CPU can access directly
  - Small cache memories are used to speed up average access time to the main memory at run-time.

- **Secondary storage** – extension of main memory that provides large nonvolatile storage capacity.
  - Magnetic disks
  - Solid state disks (SSDs)

Storage Hierarchy

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

- Faster access time, greater cost per bit
- Greater capacity, lower cost per bit
- Greater capacity, slower access speed
Performance of Various Levels of Storage

<table>
<thead>
<tr>
<th>Level</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>registers</td>
<td>cache</td>
<td>main memory</td>
<td>solid state disk</td>
<td>magnetic disk</td>
</tr>
<tr>
<td>Typical size</td>
<td>&lt; 1 KB</td>
<td>&lt; 16 MB</td>
<td>&lt; 64 GB</td>
<td>&lt; 1 TB</td>
<td>&lt; 10 TB</td>
</tr>
<tr>
<td>Implementation technology</td>
<td>custom memory with multiple ports CMOS</td>
<td>on-chip or off-chip CMOS SRAM</td>
<td>CMOS SRAM</td>
<td>flash memory</td>
<td>magnetic disk</td>
</tr>
<tr>
<td>Access time (ns)</td>
<td>0.25 - 0.5</td>
<td>0.5 - 25</td>
<td>80 - 250</td>
<td>25,000 - 50,000</td>
<td>5,000,000</td>
</tr>
<tr>
<td>Bandwidth (MB/sec)</td>
<td>20,000 - 100,000</td>
<td>5,000 - 10,000</td>
<td>1,000 - 5,000</td>
<td>500</td>
<td>20 - 150</td>
</tr>
<tr>
<td>Managed by</td>
<td>compiler</td>
<td>hardware</td>
<td>operating system</td>
<td>operating system</td>
<td>operating system</td>
</tr>
<tr>
<td>Backed by</td>
<td>cache</td>
<td>main memory</td>
<td>disk</td>
<td>disk</td>
<td>disk or tape</td>
</tr>
</tbody>
</table>

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

Caching

- 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, no matter where it is stored in the storage hierarchy.

![Diagram showing migration of integer A from disk to register]

- 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
  - Various solutions covered in Chapter 17

File Management

- A file is a collection of related information defined by its creator.
- Commonly, files represent programs (both source and object forms) and data.
- The operating system responsibilities:
  - File creation and deletion
  - Directory creation and deletion
  - Support of primitives for manipulating files and directories
  - Mapping files onto secondary storage
  - File backup on stable (non-volatile) storage media
I/O System Management

- The Operating System will hide the peculiarities of specific hardware from the user.

- In Unix, the I/O subsystem consists of:
  - A buffering, caching and spooling system
  - A general device-driver interface
  - Drivers for specific hardware devices

- Interrupt handlers and device drivers are crucial in the design of efficient I/O subsystems.

Protection and Security

- Protection – any mechanism for controlling access of processes or users to resources defined by the OS

- Security – defense of the system against internal and external attacks
  - Huge range, including denial-of-service, worms, viruses, identity theft, theft of service

- Systems generally first distinguish among users, to determine who can do what
  - User identities (user IDs, security IDs) include name and associated number, one per user
  - User ID then associated with all files, processes of that user to determine access control
  - Group identifier (group ID) allows set of users to be defined and controls managed, then also associated with each process, file
  - Privilege escalation allows user to change to effective ID with more rights
User Operating-System Interface

- Two main approaches
  - Command-line interpreter (a.k.a. command interpreter, or shell)
  - Graphical User Interfaces (GUI)

- The shell
  - allows users to directly enter commands that are to be performed by the operating system
  - is usually a system program (not part of the kernel)

- GUI allows a mouse-based window-and-menu system

- Some systems allow both (e.g. X-Windows in Unix)

System Calls

- System calls provide the interface between a running program and the operating system.
  - Generally available in routines written in C and C++
  - Certain low-level tasks may have to be written using assembly language.

- Typically, application programmers design programs using an application programming interface (API).

- The run-time support system (run-time libraries) provides a system-call interface, that intercepts function calls in the API and invokes the necessary system call within the operating system.

- Major differences in how they are implemented (e.g., Windows vs. Unix)
Example System-Call Processing

System Call: read (fd, buffer, nbytes)
### Major System Calls in Unix:
#### Process Management
- `pid = fork()`  
  - Create a child process identical to the parent
- `pid = waitpid(pid, &statloc, options)`  
  - Wait for a child to terminate
- `s = execve(name, argv, environp)`  
  - Replace a process’ core image
- `exit(status)`  
  - Terminate process execution and return status
- `s = kill(pid, signal)`  
  - Send a signal to a process

### System Programs
- System programs provide a convenient environment for program development.
- They can provide various services
  - Status information
  - File modification
  - Programming language support
  - Program loading and execution
  - Communications

*Most users’ view of the operating system is defined by system programs, not by the actual system calls.*
Operating System Design Approaches

- Simple Structure
- Layered Approach
- Modular Approach
- Microkernels
- Virtual Machines

Simple System Structure

- Some operating systems do not have well-defined structures. Often, these started as simple systems and grew beyond their original scope.

- **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 – limited by hardware functionality, the original UNIX operating system had limited structure. The UNIX OS consists of two separable parts.

- System programs
- The kernel (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.

- Simplifies debugging and system verification

- Disadvantages?
Modular Approach

- Modular kernel
  - The kernel has a set of core components
  - *Dynamically links* in additional services either during boot time or during run-time
  - Common in modern implementations of Unix such as Linux and Solaris

Microkernels

- Moves as much as possible from the kernel into the “user” space.
- Communication takes place between user modules using message passing (e.g. Mach operating system)
Microkernels (cont.)

- **Benefits**
  - easier to extend
  - more reliable (less code is running in kernel mode)
  - convenient for distributed architectures
  - Security (Maybe!)

- **Many modern OS are designed as Microkernels**
  - Apple Mac OS (based on Mach OS)
  - Many Smartphone OS
    - Android (L4 Microkernel family)
    - iPhone OS (based on Mach)
Virtual Machines

- Originally proposed and implemented for VM Operating System (IBM)

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

- Each user is given her own virtual machine

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

Virtual Machines (Cont.)

![Diagram showing non-virtual machine and virtual machine](image)
OS History

MS/DOS
  ↓
Windows
  ↓
Windows NT
  ↓
Windows 8
  ↓
Windows 10

MVS
  ↓
VMS
  ↓
VM/370
  ↓
Multics
  ↓
UNIX
  ↓
BSD UNIX
  ↓
Mach
  ↓
NEXT
  ↓
MacOS

Influence
Descendant

OS: Three Easy Pieces

- Virtualization
- Concurrency
- Persistency
OS Challenges

- Reliability
  - Does the system do what it was designed to do?

- Availability
  - What portion of the time is the system working?
  - Mean Time To Failure (MTTF), Mean Time to Repair

- Security
  - Can the system be compromised by an attacker?

- Privacy
  - Data is accessible only to authorized users

OS Challenges

- Portability
  - For programs:
    • Application programming interface (API)
    • Abstract virtual machine (AVM)
  - For the operating system
    • Hardware abstraction layer
OS Challenges

- Performance
  - Latency/response time
    - How long does an operation take to complete?
  - Throughput
    - How many operations can be done per unit of time?
  - Overhead
    - How much extra work is done by the OS?
- Fairness
  - How equal is the performance received by different users?
- Predictability
  - How consistent is the performance over time?