Introduction of OS

(SGG 3.1-3.2; USP 2)

Instructor: Tongping Liu

Lecture Outline

- Definition of Operating System (OS)
- Evolution of Computer Systems and OS Concepts
- Different Types of OS
  - Parallel/Distributed/Real-time/Embedded OS
- OS Services
- OS Structures and Basic Components
  - Task Scheduling/Memory Management/File Management/IO Management

First Computer: ENIAC in 1940s

- Big: 27 tons, 680 ft², and use 150kW
- Slow: tens instructions/second
- Limited functions: addition/multiplication
- Hard to use: button switch or punching card I/O
First Computer: ENIAC (cont.)

- Got a problem with a program? → You are in trouble 🙄

- Process one job at a time
  - A slow speed
  - CPU time is precious

- Batch systems
  - Read in more jobs
  - Process one by one
  - I/O devices are still slow?

Another Problem of ENIAC

Earlier computers, e.g. ENIAC, were hard-wired to do one task. If the computer had to perform a different task, it had to be rewired, which was a tedious process.

Von Neumann Architecture (1945)

- Since then computers more or less based on the same basic design, the Von Neumann Architecture!
- With a stored-program computer, a general purpose computer could be built to run different programs.
Desktop Systems: 1980s

- **Personal computers dedicated to a single user.**
- **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).
- I/O devices – display, keyboard, mouse and printers
- Today, desktop computers may run several different types of operating systems (Windows, MacOS, Linux)

Parallel High-Performance Systems

- **Goals:**
  - Increased performance/throughput
  - Increased reliability, e.g. fault tolerance
- Multiprocessor systems: more than one CPUs
  - **Tightly coupled system** – processors share memory, bus, IO, and clock; communication usually takes place through the shared memory
- **Symmetric multiprocessing (SMP) vs. asymmetric**
  - SMP: each processor runs an identical copy of the operating system, and all processors are peers
  - Asymmetric: master-slave
Why do you need an OS?

- Yes: you should have an OS to employ a modern computer
  - Otherwise, a set of silicon circuits do nothing good for you!
- OS provides user-friendly interfaces for using computers

Do I have to buy the OS?

OS (Windows) $300

Definition of Operating System

- A program manages computer hardware and software resources and provides common services for user programs.
- Different names for OS:
  - Kernel – the program running at all times (different from application programs)
  - Control program – controls the execution of user programs and operations of I/O devices
  - Resource allocator – manages and allocates resources
- Goals of Operating system
  - **Convenience**: Make the computer convenient to use.
  - **Efficiency**: Manage system resources in an efficient manner

Lecture Outline

- Definition of Operating System (OS)
- Evolution of Computer Systems and OS Concepts
- Different Types of OS
  - Parallel/Distributed/Real-time/Embedded OS
- OS Services
- OS Structures and Basic Components
  - Task Scheduling/Memory Management/File Management/I/O Management

Multiprogrammed Systems

- Several jobs run “concurrently”
  - Job: computing → input → computing → … → output
  - Take turns to use CPU and I/O devices
- But **which** job uses **what** and **when**?
  - Need a manager/supervisor → OS
- **How** to use the hardware (e.g., I/O devices)?
  - Resource manager/interface → OS

A single program cannot keep busy for CPU and I/O, thus wasting resources
Time Sharing Systems

- Extension of multi-programmed systems
- Multiple interactive users
  - Allow on-line interaction with users;
  - Response time for each user should be short
- CPU is multiplexed among several jobs of several users that are kept in memory
  - CPU is allocated to jobs in Round-Robin manner
  - All active users must have a fair share of the CPU time:
    - e.g. with 10 ms time quantum
- Example systems: IBM 704 and 7090

Multiprogramming vs. Time-Sharing

- Multiprogramming is the effective utilization of CPU time, by allowing several programs to use the CPU at the same time but time sharing is the sharing of a computing facility by several users that want to use the same facility at the same time.

Distributed Systems

- Loosely coupled system – each processor has its own local memory, communicating with another one through various communications lines
- Advantages of distributed systems.
  - Resource sharing
  - Computation speed up – load sharing
  - Reliability
  - Communications

Peer-to-Peer Computing Systems

- One type of distributed system
- P2P does not distinguish clients and servers
  - All nodes are considered as peers
  - Each may act as the client, the server or both
  - A node must join P2P network
    - Registers its service with the central lookup service on network, or
    - Broadcasts requests for service and responds to requests for service via discovery protocol
  - Examples include Napster and Gnutella
Special Purpose Systems

A **real-time** system is used when there are some strict time requirements on the operations of a processor or the flow of data

- **Hard** real-time: critical tasks must be completed on time
- **Soft** real-time: no absolute timing guarantees (e.g., multimedia applications)

An **embedded** system is a component of a more complex system

- Controlling a nuclear plant or a missile
- Controlling home and car appliances (e.g., microwave, DVD players, car engines, …)

Example: VxWorks and eCos; Android and iOS

Lecture Outline

- Definition of Operating System (OS)
- Evolution of Computer Systems and OS Concepts
- Different Types of OS
  - Parallel/Distributed/Real-time/Embedded OS
- OS Services
  - OS Structures and Basic Components
    - Task Scheduling/Memory Management/File Management/IO Management

OS Services

An Operating System provides services to both the programs and to the users.

- It provides programs an environment to execute.
- It provides users the services to execute the programs in a convenient manner.

Common services:

- Program execution
- I/O operations
- File System manipulation
- Communication
- Error Detection
- Resource Allocation
- Protection

System Calls

System calls provide the services of the operating system to user programs.

- System calls are the only entry points into the kernel system.
  - Generally available in routines written in C and C++
  - Certain low-level tasks may be written using assembly language
System Calls vs. APIs

- **Application programming interface (API)**
  - An application programming interface (API) is (as the name suggests) an interface for your application to use code that does not belong to your application (e.g., a library or a system call).
  - Some APIs might have system calls within them, e.g., read, write. They have side effects outside the scope of a program.
  - Some APIs might not invoke system calls, e.g., memcpy.

System Call Control Flow

- User application calls a user-level library routine ( getpid(), read(), exec(), etc.)
- Invokes system call through the stub, which specifies the system call number defined in unistd.h:

  ```
  #define __NR_getpid 172
  __SYSCALL(__NR_getpid, sys_getpid)
  ```

  This generally causes an interrupt, trapping to kernel
  - Kernel looks up system call number in syscall table, and calls appropriate syscall handler
  - Syscall handler executes, and returns the results to the userspace process

Operation Mode

- Computers (supported by mode bit) have at least two modes of operations
  - **User mode** – execution on behalf of a user
  - **Kernel mode** (also monitor mode or system mode or privileged mode) – executing on behalf of operating system
- E.g., Syscalls → switches to kernel mode.

Privileged instructions can be issued only in kernel mode.
Types of System Calls

- Process control
- File management
- Device management
- Information maintenance
- Communications
- Protection

Examples of Unix System Calls

<table>
<thead>
<tr>
<th>Process management</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pid = fork()</td>
<td>Create a new process identical to the parent</td>
</tr>
<tr>
<td>pid = wait()</td>
<td>Wait for a child to terminate</td>
</tr>
<tr>
<td>s = exeve(name, argv, envirp)</td>
<td>Replace process' core image</td>
</tr>
<tr>
<td>exit(status)</td>
<td>Terminate process execution and return status</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>File management</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fd = open(file, how...)</td>
<td>Open a file for reading, writing or both</td>
</tr>
<tr>
<td>s = close(fd)</td>
<td>Close an open file</td>
</tr>
<tr>
<td>n = read(fd, buffer, nbyte)</td>
<td>Read data from a file into a buffer</td>
</tr>
<tr>
<td>n = write(fd, buffer, nbyte)</td>
<td>Write data from a buffer into a file</td>
</tr>
<tr>
<td>position = lseek(fd, offset, whence)</td>
<td>Move the file pointer</td>
</tr>
<tr>
<td>s = stat(name, &amp;buf)</td>
<td>Get a file's status information</td>
</tr>
</tbody>
</table>

Windows vs. Unix System Calls

<table>
<thead>
<tr>
<th>Windows</th>
<th>Unix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process Control</td>
<td>CreateProcess()</td>
</tr>
<tr>
<td></td>
<td>CreateProcessObj()</td>
</tr>
<tr>
<td></td>
<td>Wait()</td>
</tr>
<tr>
<td>File Manipulation</td>
<td>CreateFile()</td>
</tr>
<tr>
<td></td>
<td>WriteFile()</td>
</tr>
<tr>
<td></td>
<td>CloseHandle()</td>
</tr>
<tr>
<td>Device Manipulation</td>
<td>SetCurrentProcessDebugCheck()</td>
</tr>
<tr>
<td></td>
<td>ReadKernelFile()</td>
</tr>
<tr>
<td>Information Maintenance</td>
<td>GetPriority()</td>
</tr>
<tr>
<td></td>
<td>SetUse()</td>
</tr>
<tr>
<td>Communication</td>
<td>CreatePipe()</td>
</tr>
<tr>
<td></td>
<td>CreateFileMapping()</td>
</tr>
<tr>
<td></td>
<td>MapFileFile()</td>
</tr>
<tr>
<td>Protection</td>
<td>SetFileSecurity()</td>
</tr>
<tr>
<td></td>
<td>InitializeSecurityDescriptor()</td>
</tr>
<tr>
<td></td>
<td>SetSecurityDescriptorGroup()</td>
</tr>
</tbody>
</table>

An API Example

- A program may invoke the printf() library call, which calls write() system call.
(Hardware) Interrupt

- An interrupt is a signal to the processor emitted by hardware or software, indicating an event that needs immediate attention.
- An interrupt transfers the control to the interrupt service routine (ISR), generally through Interrupt Vector Table containing the addresses of all service routines.
- ISR: a piece of code determines what actions should be taken for each interrupt.
- Once the interrupt has been serviced by the ISR, the control is returned to the interrupted program.

Basic Interrupt Processing

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

What if another interrupt occurs during interrupt processing?

Types 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, like scheduling
- **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
  - User request for an operating system service (e.g., system calls)
  - Runtime errors

Lecture Outline

- Definition of Operating System (OS)
- Evolution of Computer Systems and OS Concepts
- Different Types of OS
  - Parallel/Distributed/Real-time/Embedded OS
- OS Services
- OS Structures and Basic Components
  - Task Scheduling/Memory Management/File Management/IO Management
Components in Operating System

- **Process/Thread Management**
  - CPU (processors): most precious resource

- **Memory Management**
  - Main memory

- **File Management**
  - File Management → data / program

- **Secondary-Storage Management** → disk

- **I/O System Management** → I/O devices

- **Protection and Security** → access management

---

Process Management

- A **process** is a program in execution (active),
  - Dynamic concept

- A process needs resources: execution environment
  - including CPU time, registers, memory, files, and I/O devices to accomplish its task

- OS provides mechanism to
  - Create/delete processes
  - Run/Suspend/resume processes (scheduling/signal)
  - Process communication and synchronization
  - Deadlock handling

---

Main Memory Management

- The main memory is
  - a large array of words/bytes, each with its own address
  - a volatile storage device: content lost when power is off

- The operating system will
  - Keep track of which parts of memory are currently being used and by whom
  - Decide which process to load when the memory becomes available
  - Allocate and de-allocate memory space as needed

---

File Management

- A file is a collection of related information (logic unit)
  - Format is defined by its creator.

- Represent programs (source/object forms) and data

- 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
Secondary-Storage Management

- The secondary storage backs up main memory and provides additional storage.
- Most common secondary storage type: disks
- The operating system is responsible for:
  - Free space management
  - Storage allocation
  - Disk scheduling

I/O System Management

- The Operating System will hide the details of I/O hardware from the user.
- In Unix, the I/O subsystem consists of:
  - A buffering and caching 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

Storage Hierarchy

- Storage systems organized in hierarchy:
  - Speed
  - Cost
  - Volatility
- Caching: copying information into faster storage system; main memory can be viewed as a last cache for secondary storage

Caching

- Important principle, performed at many levels in a computer (in hardware, operating system, software)
- Information in use is 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 is an important design problem
  - Cache size and replacement policy
Components in Operating System

- **Process/Thread Management**
  - CPU (processors): most precious resource

- **Memory Management**
  - Main memory

- **File Management** ➔ data / program

- **Secondary-Storage Management** ➔ disk

- **I/O System Management** ➔ I/O devices

- **Protection and Security** ➔ access management

---

OS Interface: Shell and GUI

- For *programmers* and *end-users*

  - Two main interfaces:
    - Command-line 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: click-and-play

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

---

Bourne Shell Command Interpreter

The Mac OS X GUI
Lecture Outline

- Definition of Operating System (OS)
- Evolution of Computer Systems and OS Concepts
- Different Types of OS
  - Parallel/Distributed/Real-time/Embedded OS
- OS Services
- OS Structures and Basic Components
  - Task Scheduling/Memory Management/File Management/IO Management