Chapter 2
Operating System Overview
Roadmap

- Operating System Objectives/Functions
  - The Evolution of Operating Systems
  - Major Achievements
  - Developments Leading to Modern Operating Systems
  - Microsoft Windows Overview
  - UNIX Systems
  - Linux
Operating System

• A program that controls the execution of application programs
• An interface between applications and hardware
• Main objectives of an OS:
  – Convenience
  – Efficiency
  – Ability to evolve
Layers and Views

Figure 2.1  Layers and Views of a Computer System
Services Provided by the Operating System

- Program development
  - Editors and debuggers.

- Program execution
  - OS handles scheduling of numerous tasks required to execute a program

- Access I/O devices
  - Each device will have unique interface
  - OS presents standard interface to users
Services cont...

• Controlled access to files
  – Accessing different media but presenting a common interface to users
  – Provides protection in multi-access systems

• System access
  – Controls access to the system and its resources
Services cont...

- Error detection and response
  - Internal and external hardware errors
  - Software errors
  - Operating system cannot grant request of application

- Accounting
  - Collect usage statistics
  - Monitor performance
The Role of an OS

- A computer is a set of resources for the movement, storage, and processing of data.
- The OS is responsible for managing these resources.
Operating System as Software

• The OS functions in the same way as an ordinary computer software
  – It is a program that is executed by the CPU
• Operating system relinquishes control of the processor
OS as Resource Manager

Figure 2.2 The Operating System as Resource Manager
Evolution of Operating Systems

- Operating systems will evolve over time
  - Hardware upgrades plus new types of hardware
  - New services
  - Fixes
Roadmap

– Operating System Objectives/Functions

The Evolution of Operating Systems

– Major Achievements

– Developments Leading to Modern Operating Systems

– Microsoft Windows Overview

– UNIX Systems

– Linux
Evolution of Operating Systems

• It may be easier to understand the key requirements of an OS by considering the evolution of Operating Systems

• Stages include
  – Serial Processing
  – Simple Batch Systems
  – Multiprogrammed batch systems
  – Time Sharing Systems
Serial Processing

• No operating system
• Machines run from a console with display lights, toggle switches, input device, and printer
• Problems include:
  – Scheduling
  – Setup time
Simple batch system

• Early computers were extremely expensive
  – Important to maximize processor utilization

• Monitor
  – Software that controls the sequence of events
  – Batch jobs together
  – Program returns control to monitor when finished
Monitor’s perspective

- Monitor controls the sequence of events
- *Resident Monitor* is software always in memory
- Monitor reads in job and gives control
- Job returns control to monitor

Figure 2.3 Memory Layout for a Resident Monitor
Job Control Language

- Special type of programming language to control jobs
- Provides instruction to the monitor
  - What compiler to use
  - What data to use
Desirable Hardware Features

- Memory protection for monitor
  - Jobs cannot overwrite or alter
- Timer
  - Prevent a job from monopolizing system
- Privileged instructions
  - Only executed by the monitor
- Interrupts
Modes of Operation

• User Mode
  – User program executes in user mode
  – Certain areas of memory protected from user access
  – Certain instructions may not be executed

• Kernel Mode
  – Monitor executes in kernel mode
  – Privileged instructions may be executed, all memory accessible.
Multiprogrammed Batch Systems

- CPU is often idle
  - Even with automatic job sequencing.
  - I/O devices are slow compared to processor

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read one record from file</td>
<td>15 $\mu$s</td>
</tr>
<tr>
<td>Execute 100 instructions</td>
<td>1 $\mu$s</td>
</tr>
<tr>
<td>Write one record to file</td>
<td>15 $\mu$s</td>
</tr>
<tr>
<td>TOTAL</td>
<td>31 $\mu$s</td>
</tr>
</tbody>
</table>

Percent CPU Utilization $= \frac{1}{31} = 0.032 = 3.2\%$

Figure 2.4 System Utilization Example
Uniprogramming

- Processor must wait for I/O instruction to complete before preceding
Multiprogramming

- When one job needs to wait for I/O, the processor can switch to the other job.
**Multiprogramming**

<table>
<thead>
<tr>
<th>Program A</th>
<th>Run</th>
<th>Wait</th>
<th>Run</th>
<th>Wait</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program B</td>
<td>Wait</td>
<td>Run</td>
<td>Wait</td>
<td>Run</td>
</tr>
<tr>
<td>Program C</td>
<td>Wait</td>
<td>Run</td>
<td>Wait</td>
<td>Run</td>
</tr>
<tr>
<td>Combined</td>
<td>Run A</td>
<td>Run B</td>
<td>Run C</td>
<td>Wait</td>
</tr>
</tbody>
</table>

(c) Multiprogramming with three programs
Table 2.1 Sample Program Execution Attributes

<table>
<thead>
<tr>
<th></th>
<th>JOB1</th>
<th>JOB2</th>
<th>JOB3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type of job</td>
<td>Heavy compute</td>
<td>Heavy I/O</td>
<td>Heavy I/O</td>
</tr>
<tr>
<td>Duration</td>
<td>5 min</td>
<td>15 min</td>
<td>10 min</td>
</tr>
<tr>
<td>Memory required</td>
<td>50 M</td>
<td>100 M</td>
<td>75 M</td>
</tr>
<tr>
<td>Need disk?</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Need terminal?</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Need printer?</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Utilization Histograms

Figure 2.6 Utilization Histograms
Time Sharing Systems

- Using multiprogramming to handle multiple interactive jobs
- Processor’s time is shared among multiple users
- Multiple users simultaneously access the system through terminals
## Batch Multiprogramming vs. Time Sharing

<table>
<thead>
<tr>
<th></th>
<th>Batch Multiprogramming</th>
<th>Time Sharing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Principal objective</td>
<td>Maximize processor use</td>
<td>Minimize response time</td>
</tr>
<tr>
<td>Source of directives to</td>
<td>Job control language commands provided with the job</td>
<td>Commands entered at the terminal</td>
</tr>
<tr>
<td>operating system</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2.3  Batch Multiprogramming versus Time Sharing
Early Example: CTSS

• Compatible Time-Sharing System (CTSS)
  – Developed at MIT as project MAC

• Time Slicing:
  – When control was passed to a user
  – User program and data loaded
  – Clock generates interrupts about every 0.2 sec
  – At each interrupt OS gained control and could assign processor to another user
CTSS Operation

Figure 2.7 CTSS Operation
Problems and Issues

• Multiple jobs in memory must be protected from each other’s data
• File system must be protected so that only authorised users can access
• Contention for resources must be handled
  – Printers, storage etc
Roadmap

– Operating System Objectives/Functions
– The Evolution of Operating Systems

**Major Achievements**

– Developments Leading to Modern Operating Systems
– Microsoft Windows Overview
– UNIX Systems
– Linux
Major Advances

• Operating Systems are among the most complex pieces of software ever developed

• Major advances include:
  – Processes
  – Memory management
  – Information protection and security
  – Scheduling and resource management
  – System
Process

• Fundamental to the structure of OS’s
• A process is:
  – A program in execution
  – An instance of a running program
  – The entity that can be assigned to and executed on a processor
  – A single sequential thread of execution, a current state, and an associated set of system resources.
Causes of Errors when Designing System Software

• Error in designing an OS are often subtle and difficult to diagnose

• Errors typically include:
  – Improper synchronization
  – Failed mutual exclusion
  – Non-determinate program operation
  – Deadlocks
Components of a Process

- A process consists of
  - An executable program
  - Associated data needed by the program
  - Execution context of the program (or “process state”)
- The execution context contains all information the operating system needs to manage the process
Process Management

Figure 2.8  Typical Process Implementation
Memory Management

- The OS has 5 principal storage management responsibilities
  - Process isolation
  - Automatic allocation and management
  - Support of modular programming
  - Protection and access control
  - Long-term storage
Virtual Memory

- File system implements long-term store
- Virtual memory allows programs to address memory from a logical point of view
  - Without regard to the limits of physical memory
Paging

- Allows process to be comprised of a number of fixed-size blocks, called pages.
- Virtual address is a page number and an offset within the page.
- Each page may be located anywhere in main memory.
Virtual Memory

Main Memory

Main memory consists of a number of fixed-length frames, each equal to the size of a page. For a program to execute, some or all of its pages must be in main memory.

Disk

Secondary memory (disk) can hold many fixed-length pages. A user program consists of some number of pages. Pages for all programs plus the operating system are on disk, as are files.

Figure 2.9 Virtual Memory Concepts
Virtual Memory Addressing

Figure 2.10  Virtual Memory Addressing
Information Protection and Security

• The problem involves controlling access to computer systems and the information stored in them.

• Main issues are:
  – Availability
  – Confidentiality
  – Data integrity
  – Authenticity
Scheduling and Resource Management

• Key responsibility of an OS is managing resources

• Resource allocation policies must consider:
  – Fairness
  – Differential responsiveness
  – Efficiency
Key Elements of an Operating System

![Diagram](image)

Figure 2.11 Key Elements of an Operating System for Multiprogramming
System Structure

• View the system as a series of levels
• Each level performs a related subset of functions
• Each level relies on the next lower level to perform more primitive functions
• This decomposes a problem into a number of more manageable subproblems
### Table 2.4 Operating System Design Hierarchy

<table>
<thead>
<tr>
<th>Level</th>
<th>Name</th>
<th>Objects</th>
<th>Example Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>Shell</td>
<td>User programming environment</td>
<td>Statements in shell language</td>
</tr>
<tr>
<td>12</td>
<td>User processes</td>
<td>User processes</td>
<td>Quit, kill, suspend, resume</td>
</tr>
<tr>
<td>11</td>
<td>Directories</td>
<td>Directories</td>
<td>Create, destroy, attach, detach, search, list</td>
</tr>
<tr>
<td>10</td>
<td>Devices</td>
<td>External devices, such as printers, displays, and keyboards</td>
<td>Open, close, read, write</td>
</tr>
<tr>
<td>9</td>
<td>File system</td>
<td>Files</td>
<td>Create, destroy, open, close, read, write</td>
</tr>
<tr>
<td>8</td>
<td>Communications</td>
<td>Pipes</td>
<td>Create, destroy, open, close, read, write</td>
</tr>
<tr>
<td>7</td>
<td>Virtual memory</td>
<td>Segments, pages</td>
<td>Read, write, fetch</td>
</tr>
<tr>
<td>6</td>
<td>Local secondary</td>
<td>Blocks of data, device channels</td>
<td>Read, write, allocate, free</td>
</tr>
<tr>
<td>5</td>
<td>Primitive processes</td>
<td>Primitive processes, semaphores, ready list</td>
<td>Suspend, resume, wait, signal</td>
</tr>
<tr>
<td>4</td>
<td>Interrupts</td>
<td>Interrupt-handling programs</td>
<td>Invoke, mask, unmask, retry</td>
</tr>
<tr>
<td>3</td>
<td>Procedures</td>
<td>Procedures, call stack, display</td>
<td>Mark stack, call, return</td>
</tr>
<tr>
<td>2</td>
<td>Instruction set</td>
<td>Evaluation stack, microprogram interpreter, scalar and array data</td>
<td>Load, store, add, subtract, branch</td>
</tr>
<tr>
<td>1</td>
<td>Electronic circuits</td>
<td>Registers, gates, buses, etc.</td>
<td>Clear, transfer, activate, complement</td>
</tr>
</tbody>
</table>

Gray shaded area represents hardware.
Roadmap

- Operating System Objectives/Functions
- The Evolution of Operating Systems
- Major Achievements
  
  Developments Leading to Modern Operating Systems
  
  - Microsoft Windows Overview
  - UNIX Systems
  - Linux
Different Architectural Approaches

- Various approaches have been tried, categories include:
  - Microkernel architecture
  - Multithreading
  - Symmetric multiprocessing
  - Distributed operating systems
  - Object-oriented design
Most early OS are a monolithic kernel
- Most OS functionality resides in the kernel.

A microkernel assigns only a few essential functions to the kernel
- Address spaces
- Interprocess communication (IPC)
- Basic scheduling
Multithreading

• Process is divided into threads that can run concurrently

• Thread
  – Dispatchable unit of work
  – executes sequentially and is interruptible

• Process is a collection of one or more threads
Symmetric multiprocessing (SMP)

• An SMP system has
  – multiple processors
  – These processors share same main memory and I/O facilities
  – All processors can perform the same functions

• The OS of an SMP schedules processes or threads across all of the processors.
SMP Advantages

- **Performance**
  - Allowing parallel processing

- **Availability**
  - Failure of a single process does not halt the system

- **Incremental Growth**
  - Additional processors can be added.

- **Scaling**
Multiprogramming and Multiprocessing

Time

Process 1

Process 2

Process 3

(a) Interleaving (multiprogramming; one processor)

Process 1

Process 2

Process 3

(b) Interleaving and overlapping (multiprocessing; two processors)

Figure 2.12 Multiprogramming and Multiprocessing
Distributed Operating Systems

- Provides the illusion of
  - a single main memory space and
  - single secondary memory space

- Early stage of development
Object-oriented design

- Used for adding modular extensions to a small kernel
- Enables programmers to customize an operating system without disrupting system integrity
Roadmap

- Operating System Objectives/Functions
- The Evolution of Operating Systems
- Major Achievements
- Developments Leading to Modern Operating Systems

Microsoft Windows Overview
- UNIX Systems
- Linux
Single-User Multitasking

• From Windows 2000 on Windows development developed to exploit modern 32-bit and 64-bit microprocessors

• Designed for single users who run multiple programs

• Main drivers are:
  – Increased memory and speed of microprocessors
  – Support for virtual memory
Windows Architecture

Figure 2.13  Windows and Windows Vista Architecture [RUSS05]
Client/Server Model

- Windows OS, protected subsystem, and applications all use a client/server model
  - Common in distributed systems, but can be used internal to a single system
- Processes communicate via RPC
Windows Objects

- Windows draws heavily on the concepts of object-oriented design.
- Key Object Oriented concepts used by Windows are:
  - Encapsulation
  - Object class and instance
Roadmap

– Operating System Objectives/Functions
– The Evolution of Operating Systems
– Major Achievements
– Developments Leading to Modern Operating Systems
– Microsoft Windows Overview

UNIX Systems
– Linux
Description of UNIX

Figure 2.14  General UNIX Architecture
Figure 2.15  Traditional UNIX Kernel
System V Release 4 (SVR4)

Figure 2.16 Modern UNIX Kernel
Roadmap

- Operating System Objectives/Functions
- The Evolution of Operating Systems
- Major Achievements
- Developments Leading to Modern Operating Systems
- Microsoft Windows Overview
- UNIX Systems

Linux
Modular Monolithic Kernel

- Although monolithic, the kernel is structures as a collection of modules
  - Loadable modules
  - An object file which can be linked and unlinked at run time

- Characteristics:
  - Dynamic Linking
  - Stackable modules
Linux Kernel Modules

Figure 2.17 Example List of Linux Kernel Modules
Linux Kernel Components

Figure 2.18  Linux Kernel Components