1 Introduction

What is an operating system?
Layer of software between the hardware and application programs. Two main functions:

- Resource manager
- Extended (abstract, virtual) machine

OS as resource manager

- mediator/coordinator: resolve conflicting resource demands
- protect users from each others (and from themselves)
- mechanisms and policies for resource sharing, information flow

OS as extended machine

- provides stable, portable, reliable, safe, well-behaved environment (ideally)
- Magician: makes computer appear to be more than it really is
- Single processor appears like many separate processors
- Single memory made to look like many separate memories, each potentially larger than the real memory
• Abstraction that is easier to program and reason about than the hardware

<table>
<thead>
<tr>
<th>Physical machine</th>
<th>Extended machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor(s)</td>
<td>Threads</td>
</tr>
<tr>
<td>Memory</td>
<td>Processes</td>
</tr>
<tr>
<td>Disks</td>
<td>Files</td>
</tr>
<tr>
<td>Networks</td>
<td>Operating System</td>
</tr>
<tr>
<td>Monitors/UIs</td>
<td>Comm. channels</td>
</tr>
<tr>
<td>Speakers</td>
<td>System</td>
</tr>
<tr>
<td>Microphones</td>
<td>Events</td>
</tr>
<tr>
<td>Clock/Timer</td>
<td></td>
</tr>
</tbody>
</table>

What resources need to be managed?

• processors/cores (computation)
• main memory
• secondary memory (disks, non-volatile solid state)
• network links
• I/O devices (monitors/UIs, printers, audio, video)

OS must

• service all of these devices simultaneously,
• support safe, efficient, and fair resource sharing and information flow among users and programs.

Major issues in operating systems

• concurrency —how are parallel activities created and controlled?
• sharing —how are resources shared among users?
• naming —how are resources named (by programs and users)
• protection —how is one user/program protected from another?
• security —how to restrict information flow and prevent misuse?
• performance —why is it so slow?
• structure —how is an operating system organized?
• reliability and fault tolerance —how to handle failures
• extensibility —how do we add new features?
• communication —how and with whom can we communicate?
• scale and growth —what happens as demand and resources increase?
• persistence —how to make data last longer than programs
• distribution —how to integrate a world of information and resources?
• accounting —who does what, and how do we control resource usage?

Brief history of operating systems

• In the beginning, one user/program at a time, no overlap of computation and I/O. OS first appeared as a subroutine library shared by all users.

• simple batch systems were first real OS:
  – OS stored in part of main memory
  – it loaded a single job (from card reader) into memory
  – ran the job, printed its output, etc.
  – loaded next job

• Spooling and buffering allowed jobs to be read ahead of time onto tape/disk or into memory.

• Multiprogramming systems provided increased utilization
  – multiple runnable jobs loaded in memory
  – overlap I/O processing of one job with computation of another
  – benefit from I/O devices that can operate asynchronously
– requires use of interrupts/DMA
– tries to optimize *throughput*

*Timesharing systems* support interactive use
– each user feels as if he/she has the entire machine (at least at night)
– tries to optimize response time
– based on time-slicing —dividing available CPU time equally among the users
– permits interactive work; participation of users in the execution process
– MIT Multics system was first large timesharing system (mid-late 1960s)
– requires periodic clock interrupts

*Distributed* operating systems
– facilitate use of geographically distributed resources
– supports communication between parts of a job or different jobs
– sharing of distributed resources, hardware and software
– permits parallelism, but speedup is not necessarily the main objective
– not covered in this course—check out Distributed Systems course

* Characteristics of current OS’es:*

– Large
  * Tens of millions of source lines of code (SLOC)
  * Tens of thousands of person-years
– Complex
  * asynchronous
  * concurrent/parallel
  * abundance of hardware platforms with different idiosyncrasies
  * conflicting needs of different application programs and users
  * performance and dependability are crucial
What we’ll cover in this course

- **Process management**
  - Threads and processes
  - Synchronization
  - Multiprogramming
  - CPU Scheduling
  - Deadlock

- **Memory management**
  - Dynamic storage allocation
  - Sharing main memory
  - Virtual memory

- **I/O management**
  - File storage management
  - Naming
  - Concurrency
  - Performance

- **Advanced topics (based on research papers)**
  - Virtual machines
  - Multi-core (OS structure, scalability)
  - Weak memory models
  - Energy management
  - Distributed systems