What are the objectives of an Operating System?

- **Convenience & Abstraction**
  - The OS should facilitate the task of application and system programmer.
  - Hardware details should be hidden, uniform interface for different I/O devices provided.

- **Efficiency**
  - Should take up few resources, make good use of resources, and be fast.

- **Protection**
  - Fairness, security, safety.

Services Provided by the Operating System

- **Program execution**
  - Load instructions and data into main memory.
  - Initialise I/O devices, etc.

- **Access to I/O devices**
  - Provides a uniform interface for various devices.

- **Controlled access to files**
  - Abstracts over structure of data on I/O device.
  - Provides protection mechanisms.
**Slide 5**

**SERVICES PROVIDED BY THE OPERATING SYSTEM**

- System access: provides protection of
  - data
  - system resources; and
  - resolves access conflicts
- Program development
  - Editors, compilers, and debuggers: not part of the core, but usually supplied with the OS.

**Slide 6**

**SERVICES PROVIDED BY THE OPERATING SYSTEM**

- Error detection and response
  - Possible errors:
    - Internal and external hardware errors
      - Memory error
      - Device failure
    - Software errors
      - Arithmetic overflow
      - Access forbidden memory locations
    - Operating system cannot grant request of application
  - The OS has to
    - Clear error condition
    - Minimise effect on other applications

**Slide 7**

**SERVICES PROVIDED BY THE OPERATING SYSTEM**

- Accounting
  - Collect statistics
  - Monitor performance
  - Used to anticipate future enhancements
  - Used for billing users

**Slide 8**

**OPERATING SYSTEM**

The operating system controls the

- Movement, storage, and processing of data

But it is not always ‘in control’:

- Functions same way as ordinary computer software
- It is just a program (or a set of programs) that is executed
- Relinquishes control of the processor to execute other programs
- Must depend on the processor to regain control
Kernel
- Portion of operating system that is running in privileged (or “kernel” or “supervisor”) mode
- Usually resident in main memory
- Implements protection
- Contains fundamental functionality required to implement other services
- Also called the nucleus or supervisor

Evolution of Operating Systems
Serial Processing: late 1940s to mid 1950s
- No operating system
- Machines run from a console with display lights and toggle switches, input device, and printer
- Manual schedule
- Setup for each user included
  - loading the compiler, source program,
  - saving compiled program,
  - loading and linking
Improvements: libraries of common functions, linkers, loaders, compilers, debuggers available to all users.

Evolution of Operating Systems
Simple Batch Systems: mid 1950s, by GM for IBM 701
- The monitor controls the execution of programs:
  - it batches jobs together
  - the program branches back to monitor when finished
  - resident monitor is in main memory and available for execution
- Instructions to monitor via Job Control Language (JCL)
  - the monitor contains a JCL interpreter
  - each job includes instructions in JCL to tell the monitor
    - what compiler to use
    - what data to use
  - predecessor of shell
Monitor takes up main memory and CPU time but improves utilization of computer

Evolution of Operating Systems
Serial Processing: late 1940s to mid 1950s
- No operating system
- Machines run from a console with display lights and toggle switches, input device, and printer
- Manual schedule
- Setup for each user included
  - loading the compiler, source program,
  - saving compiled program,
  - loading and linking
Improvements: libraries of common functions, linkers, loaders, compilers, debuggers available to all users.
**Hardware Features**

New hardware features support development of OS features

- **Memory protection**
  - do not allow the memory area containing the monitor to be altered

- **Timer**
  - prevents a job from monopolizing the system

- **Privileged instructions**
  - for example, I/O instructions

- **Interrupts**
  - relinquishing control to and gaining control from user program

**Uniprogramming**

**Problem:**

- Processor must wait for I/O instruction to complete before preceding
- I/O instructions are very slow compared to computations

**Solution:** Interleave the execution of multiple jobs!

**Multiprogramming**

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

- Increased throughput
- Increased utilisation

**Example**

<table>
<thead>
<tr>
<th>Job 1</th>
<th>Job 2</th>
<th>Job 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type of Job</td>
<td>CPU bound</td>
<td>I/O bound</td>
</tr>
<tr>
<td>Duration</td>
<td>5 min</td>
<td>15 min</td>
</tr>
<tr>
<td>Memory req't</td>
<td>50k</td>
<td>100k</td>
</tr>
<tr>
<td>Disk?</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Terminal?</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Printer?</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
**Slide 17**

**Effects of Multiprogramming**

<table>
<thead>
<tr>
<th></th>
<th>Uniprogramming</th>
<th>Multiprogramming</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor util.</td>
<td>22%</td>
<td>43%</td>
</tr>
<tr>
<td>Memory util.</td>
<td>30%</td>
<td>67%</td>
</tr>
<tr>
<td>Disk util.</td>
<td>33%</td>
<td>67%</td>
</tr>
<tr>
<td>Printer util.</td>
<td>33%</td>
<td>67%</td>
</tr>
<tr>
<td>Elapsed time</td>
<td>30 min</td>
<td>15 min</td>
</tr>
<tr>
<td>Throughput</td>
<td>6 jobs/h</td>
<td>12 jobs/h</td>
</tr>
<tr>
<td>mean resp. time</td>
<td>18 min</td>
<td>10 min</td>
</tr>
</tbody>
</table>

**Slide 18**

**Time Sharing**

Batch multiprogramming improves the utilisation of static jobs, but what about interactive jobs?

- Using multiprogramming to handle multiple interactive jobs
- Processor’s time is shared among multiple users
- Multiple users simultaneously access the system through terminals

**Slide 19**

**Batch Multiprogramming versus Time Sharing**

Different requirements for interactive execution

<table>
<thead>
<tr>
<th></th>
<th>Batch Multiprogramming</th>
<th>Time Sharing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Principal objective</td>
<td>Maximise CPU utilisation</td>
<td>Minimise response time</td>
</tr>
<tr>
<td>Control</td>
<td>JCL with job</td>
<td>Interactive commands</td>
</tr>
</tbody>
</table>

One of the first systems: Compatible Time-Sharing System (CTSS), 1961, IBM 709 & IBM 7094

- a system clock creates interrupts in regular intervals
- system switches to a new user
- old user’s program and data saved to disk
**Primitive Time Sharing (CTSS)**

Job1: 15,000    Job3: 5000
Job2: 20,000    Job4: 10,000

<table>
<thead>
<tr>
<th>Job</th>
<th>0</th>
<th>5000</th>
<th>10000</th>
<th>15000</th>
<th>20000</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Slide 21**