Look "under the hood" to understand how computer systems work

- Understand some of the tradeoffs in systems design
- Understand what makes a "good" system
- Embedded system: special-purpose OS tightly coupled to application software
- Understand why a program that looks alright might be badly broken

---

**Example 1: Displaying Time of Day**

```c
#include <timer.h>
void ShowTime (void) {
    int hour, mins, secs;
    hour = Timer.hour;
    mins = Timer.mins;
    secs = Timersecs;

    printf("time = %02d:%02d:%02d\n", hour, mins, secs);
}
```

Where is the problem?

---

**Example 2: Initialising an Array**

```c
void ResetArray (int array[10000][10000]) {
    int i, j;
    for (i=0; i<10000; i++) {
        for (j=0; j<10000; j++) {
            array[i][j] = 0;
        /* OR array[j][i] = 0 ??? */
        }
    }
}
```

What difference does it make?

---

**Example 3: Password Verification**

---
**Example 3: Password Verification**

```c
int CheckPassword (char *given, char *passwd) {
    int i;

    for (i=0; i<14; i++) {
        if (passwd[i] != given[i]) {
            return EXIT_FAILURE;
        }
    }
    return EXIT_SUCCESS;
}
```

What is the problem?

---

**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.

---

**Layers of Computer System**

- End User
- Programmer
- Application Programs
- Utilities
- Operating System
- Computer Hardware

---

**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 9

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 10

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 11

SERVICES PROVIDED BY THE OPERATING SYSTEM

→ Accounting
  - collect statistics
  - monitor performance
  - used to anticipate future enhancements
  - used for billing users

Slide 12

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
**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

**Multiprogramming**

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

- Increased throughput
- Increased utilization

**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!

**Example**

<table>
<thead>
<tr>
<th>Type of Job</th>
<th>Job 1</th>
<th>Job 2</th>
<th>Job 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duration</td>
<td>5 min</td>
<td>15 min</td>
<td>10 min</td>
</tr>
<tr>
<td>Memory req't</td>
<td>50k</td>
<td>100k</td>
<td>80k</td>
</tr>
<tr>
<td>Disk?</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Terminal?</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Printer?</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
### Slide 21

**Effects of Multiprogramming**

- **Uniprogramming**
  - Processor utilis.: 22%
  - Memory utilis.: 30%
  - Disk utilis.: 33%
  - Printer utilis.: 33%
  - Elapsed time: 30 min
  - Throughput: 6 jobs/h
  - mean resp. time: 18 min

- **Multiprogramming**
  - Processor utilis.: 43%
  - Memory utilis.: 67%
  - Disk utilis.: 67%
  - Printer utilis.: 67%
  - Elapsed time: 15 min
  - Throughput: 12 jobs/h
  - mean resp. time: 10 min

### Slide 22

**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 23

**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  
Job2: 20,000  
Job3: 5000  
Job4: 10,000

---

**Processes**

- Problems occurring in multiprogramming batch systems, time-sharing systems required a closer look at "jobs".
- What exactly is a Process?

Exact definition is differs from to textbook to textbook:

- A program in execution
- An instance of a program running on a computer
- A unit of execution characterised by
  - a single, sequential thread of execution
  - a current state
  - an associated set of system resources (memory, devices, files)

We define a Process to be an **unit of resource ownership**

---

The OS has to:

- Load the executable from hard disk to main memory
- Keep track of the states of every process currently executed
- Make sure
  - no process monopolises the CPU
  - no process starves
  - interactive processes are responsive