Introduction to Operating Systems

Instructor: Mark Llewellyn
markl@cs.ucf.edu
CSB 242, 823-2790

School of Electrical Engineering and Computer Science
University of Central Florida
What is an Operating System?

• In the most general sense an operating system is a collection of system software routines that sit between an application program and the computer hardware on which that application is to be executed.

The OS sits between the application program and the hardware.
What is an Operating System? (cont.)

- For now, we can think of an OS as:
  1) is the interface or intermediary between a user/application and the computer hardware
  2) provides an environment in which the user can execute programs conveniently and
     • application and/or system software
  3) manages the computer’s resources efficiently
     • memory, disk space, CPU time, I/O, software, etc.

- Often an OS is a tradeoff between convenience and efficiency
  - Windows (GUI) vs. Unix (command interpreter)
The OS As An Intermediary

- We’ll discuss hardware later in Chapter 2.
- What’s an application?
  - Software to accomplish a task
    - Spread sheet, word processor, browser, email
  - What about system software?
    - Depending on who you ask, can be considered application programs, a computer resource, or part of the OS
What Is A Process?

PROBLEM → ALGORITHM → PROGRAM

PROBLEM → ALGORITHM → PROGRAM

EXECUTABLE CODE

LINKER

OBJECT CODE

ASSEMBLER OR COMPILER

LIBRARIES

PROCESS

LOADER
What Is A Process (cont.)

• A *process*:  
  – is a program in execution.  
  – has a process control block (PCB)  
  – has a program counter (PC)

• A process can have one or more *threads*.  
  – A thread is sometimes known as a *lightweight process*
Types of Operating Systems

• Focus on two system resources
  – CPU (processor) Utilization
  – Main Memory Utilization

• Utilization is measure of busy time over total study time \((\frac{T_{\text{busy}}}{T_{\text{Total}}})\)

• In the old days computers were
  – physically very large
  – but very small in terms of resources and capabilities
  – also very, very expensive

• Important to achieve high utilization of resources
Early Systems

- Instructions and data written in binary
- Loaded using switches on front panel
- Computers also had a few buttons
  - Halt, Run, Load, Set PC (displayed contents), Increment PC
- Everything done by programmer the was no real “user” as we know them today
- Very slow set up time
- Very limited output (set PC and check lights)
- CPU sat idle much of the time.
- Very little wasted memory since RAM was so small.
  - Programmers always “squeezed” program into Main Memory
Early Systems - Hardware Innovations

• Needed way to speed up Input & Output (I/O)

• Paper Tape
  – Data entry difficult, splicing needed or recopy tape
  – Paper tape output faster than looking at lights but hard to read.

• Punch Cards
  – Faster form of input.
  – Offline Card-to-Printer improved readability of output

• Magnetic Tape
  – Input even faster

• Disk drives as a replacement for tapes
Early Systems - Software Innovations

• Assemblers
  – Symbolic programming rather than 1s and 0s
  – 1:1 relationship between assembler statements and machine instructions

• Linkers & Loaders
  – allowed the use of code libraries
  – didn’t have to rewrite common code

• Compilers
  – Programming in High Level Languages (Fortran, COBOL)
  – 1:n relationship between a program statement and machine instructions
  – Eased programming task and improved operational efficiency.
Early Systems - People/Procedural Changes

• Too much work for programmer
• Division of labor became necessary.
  – Divided tasks between a programmer and professional operator
  – Operator could now organize the work more effectively and “batch” jobs
• Batching
  – allows similar jobs to run sequentially
  – efficient use of system software
  – today, refers more to jobs which lack user interaction
    • Example, billing systems used by companies
  – still took long time to set up jobs
  – CPU frequently sat idle
Running Multiple Programs

• Serial/Sequential Execution
  – One job must finish before next job starts
  – Results in very low utilization of resources

• Concurrent Execution
  – Two or more processes executing at the same time but doing different activities
  – Processes take turns using single shared resource
  – Gives the illusion of parallel processing

• Parallel/Simultaneous Execution
  – Two or more processes performing the same activity at the same time
  – Requires two or more of the same resource (e.g., processors, printers, disk drives)
Resident Monitor

- Precursor to the Operating System
  - The beginnings of computer “self-governance”
- Resident in memory all the time, Monitored operations
- Primary task was job sequencing
  - In beginning read jobs sequentially from tape already prepared off-line
  - With disks, could select which jobs to run and when
- Job Scheduling
- Resident Monitors Improved:
  - CPU Utilization: Faster setup, less idle time.
  - Memory Utilization: Sharing of I/O drivers/code
  - Functionality: Accounting and run time & I/O limits.
The Beginnings of Multiprogramming

- With RM, had two programs running serially:
  - Application - RM - Application - RM - Application, etc.
  - Application ran to completion before RM took over

- Reduced CPU idle time between jobs but not between I/O operations
  - I/O speed very slow compared to CPU speed
  - Much of I/O deals with accessing data (tape, disk)
An Aside On The Memory Hierarchy

- **Registers**: Both special and general purpose. On processor chip.
- **Cache**: Buffers memory requests. Maybe multiple levels.
- **Primary/Main (RAM)**: Organized in word and bytes.
- **Secondary**: Fixed media, including hard disks, optical disks, etc.
- **Tertiary**: Removable media, including tapes, floppy disks, CDs, etc.

The memory hierarchy is organized from top to bottom:

- **Faster, more expensive, smaller amounts**
- **Slower, cheaper, larger amounts**
Multiprogramming (cont.)

- Need some way to perform CPU and I/O operations at the same time
- Keep more than one user program in memory
- Switch between programs during I/O operations

<table>
<thead>
<tr>
<th>Operating System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process 1</td>
</tr>
<tr>
<td>Process 2</td>
</tr>
<tr>
<td>Process 3</td>
</tr>
</tbody>
</table>
Multiprogramming (cont.)

- More efficient use of system, less idle time.
- Made possible because of reduced memory costs.
- Created a whole new set of problems:
  - Memory Management
  - Protection Mechanisms
  - Job Scheduling
  - CPU/Process Scheduling
- Improves resource utilization but not user interaction
  - “...manages the computer’s resources efficiently...”
- What about conveniences
  - “...the user can execute programs conveniently...”
Time Sharing

• Introduced to improve the interaction with computer
  – Use of terminals and teleprinters
  – Allows user to input data and interact with program
• Give each process a slice of CPU time
  – Don’t wait for next I/O operation
  – Similar to multiplexing
• Possible because user input is so slow
  – 1 char takes 1000 milliseconds to enter while only 2 milliseconds required for an interrupt handler
• Required the development of virtual memory, online file systems and directory structures
• Today’s systems generally support a combination of batch and time sharing.
Other Operating System Developments

- Real Time - Response time is critical
  - Hard Real Time
    - Industrial & Robotic controls.
    - Very small bounded delays.
    - Little use of swapping or secondary storage.
  - Soft Real Time
    - Immediate response not as critical.
    - Monitoring devices, etc.
    - Longer delays tolerated
    - Use of secondary storage & priority scheduling.
Other Developments (cont.)

- Personal Computers
  - Many of the OS features we’ll study are incorporated in today’s PCs.
  - Change in priority since hardware is cheap and only a single user.
  - Greater emphasis on convenience than on efficiency
  - Ultimate in interactive – in a way, we’ve gone back to the early days
  - Adds new demands also – networking and multimedia.

- Handheld Systems
Other Developments (cont.)

• Parallel vs. Distributed Systems vs. Clustered
  – Needed for various reasons: larger problems, larger data requirements,
  – Parallel/Tightly coupled – share resources (e.g., memory, clock, bus, OS, disks)
    • Asymmetric multiprocessing – master/slave
    • Symmetric multiprocessing – each processor has copy of OS
    • Typically used for scientific applications and simulations.
  – Distributed/Loosely coupled – Separate computers linked via network.
    • Used for client/server applications
    • Peer-to-Peer applications
  – Clustered Systems
    • Computers linked by network but sharing storage
Performance Measurements

- **Utilization (maximize)**
  - $U = \frac{T_{busy}}{T_{total}}$ where $T_{total}$ is total study time, or
  - $U = \frac{T_{used}}{T_{available}}$

- **Throughput (maximize)**
  - $X = \frac{C}{T}$ where $C$ is number of completed jobs/processes and $T$ is time frame
  - The rate at which requests are processed
Performance Measurements (cont.)

- **Turnaround Time** (minimize)
  - Typically used in reference to batch systems
  - The time it takes to complete/execute a job/process

- **Response Time** (minimize)
  - Typically used in reference to interactive systems
  - The time it takes for the system to respond to a user request from submission to start of a response