CS 471 Operating Systems

Yue Cheng

George Mason University
Fall 2017
Outline

- A brief introduction of OS evolution history

- Low-level OS concepts
  - Instruction execution
  - Interrupt processing

- OS design approaches
Co-evolution of Computer Systems

- Mainframe Systems
  - Batch Systems
  - Multi-programmed Systems
  - Time-sharing Systems
- Desktop Systems
- Modern Variants
  - Parallel Systems
  - Distributed Systems
  - Real-time and Embedded Systems
  - Handheld Systems
Mainframe Systems

- First computers to tackle many commercial and scientific applications

- Mainframes evolved through *batch*, *multi-programmed* and *time-shared systems*

- Early systems were afforded only by major government agencies or universities:
  - physically enormous machines run from a console.
  - the user submitted the job to the human operator in the form of punched cards.
  - The operator collects the output and returns it to the user.
Batch Systems

*Batch systems*: To speed up processing, operators *batched* together jobs with similar needs and ran them through the computer as a group.
Multiprogrammed Systems

Several (and not necessarily similar) jobs are kept in the main memory at the same time, and the CPU is switched to another job when I/O takes place.

*Objective:* ?
“Uniprogramming”
vs Multiprogramming

(a) Uniprogramming

(b) Multiprogramming with two programs
OS Features Needed for Multiprogramming

- **Job Scheduling** – must choose the processes that will be brought to memory

- **Memory Management** – must allocate the memory to several jobs

- **CPU Scheduling** – must choose among several jobs ready to run

- OS/360, developed by IBM to run on its System/360 series, was the first multiprogrammed operating system (1964).
Time-Sharing Systems: Interactive Computing

- Extension of multiprogrammed systems to allow on-line interaction with users.

- Each user is provided with an on-line terminal.

- Objective: *Response time* for each user should be short.

- The CPU is multiplexed among several jobs that are kept in memory and on disk.

- A job swapped in and out of memory to the disk.

- CPU is allocated to another job when I/O takes place.

- All active users must have a *fair* share of the CPU time (e.g. 10 ms for each user).
Desktop Systems – Hooray!

- *Personal computers* – computer system originally dedicated to a single user.
- I/O devices – keyboard, mouse, printers, …
- **Objective:** User convenience and responsiveness
  - Individuals have sole use of computers
  - A single user may not need advanced features of mainframe OS (maximize utilization, protection).
- Today, may run several different types of operating systems (Windows, MacOS, Linux)
Parallel Systems

- Multiprocessor systems with more than one CPU in close communication.
- *Tightly coupled system* – processors share memory and a clock; communication usually takes place through the shared memory.
- Advantages of parallel system
  - Increased *throughput*
  - Economy of scale
  - Increased reliability
    - graceful degradation
    - fault-tolerant systems
Parallel Systems

- **Symmetric multiprocessing (SMP)**
  - All processors are peers
  - Kernel routines can execute on different CPUs, in parallel

- **Asymmetric multiprocessing (AMP)**
  - Master/slave structure
  - The kernel runs on a particular processor
  - Other CPUs can execute user programs and OS utilities.
Parallel Systems (Cont.)

- Multi-core architectures
  - Include multiple computing cores on a single chip
  - Critical to address the *power wall*
  - Need to exploit *parallelism* at run-time
Moore’s Law

- “Number of transistors on a chip will double every 18 months”
  - “Prediction” by Gordon Moore, co-founder of Intel, in 1965
- The increase in chip complexity has been accompanied by significant increase in clock frequency speeds.
## CPU Clock Rates

Inflection point in computer history when designers hit the “Power Wall”

<table>
<thead>
<tr>
<th>Year</th>
<th>CPU</th>
<th>Clock rate (MHz)</th>
<th>Cycle time (ns)</th>
<th>Cores</th>
<th>Effective cycle time (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1980</td>
<td>8080</td>
<td>1</td>
<td>1000</td>
<td>1</td>
<td>1000</td>
</tr>
<tr>
<td>1990</td>
<td>386</td>
<td>20</td>
<td>50</td>
<td>1</td>
<td>50</td>
</tr>
<tr>
<td>1995</td>
<td>Pentium</td>
<td>150</td>
<td>6</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>2000</td>
<td>P-III</td>
<td>600</td>
<td>1.6</td>
<td>1</td>
<td>1.6</td>
</tr>
<tr>
<td>2003</td>
<td>P-4</td>
<td>3300</td>
<td>0.3</td>
<td>1</td>
<td>0.3</td>
</tr>
<tr>
<td>2005</td>
<td>Core 2</td>
<td>2000</td>
<td>0.5</td>
<td>1</td>
<td>0.5</td>
</tr>
<tr>
<td>2010</td>
<td>Core i7</td>
<td>2500</td>
<td>0.4</td>
<td>4</td>
<td>0.4</td>
</tr>
<tr>
<td>2010:1980</td>
<td>---</td>
<td>2500</td>
<td>2500</td>
<td>4</td>
<td>2500</td>
</tr>
</tbody>
</table>
Problems with Ever Increasing Clock Speed and Logic Density

- Power Wall

- CPU - Memory Performance Gap
Exponential Increase in Power Density

From Fred Pollack’s keynote address at Micro-32 conference
The CPU-Memory Gap
The gap widens between memory, disk, and CPU speeds.

The diagram shows the gap between disk seek time, flash SSD access time, DRAM access time, SRAM access time, CPU cycle time, and effective CPU cycle time from 1980 to 2010.
Real-Time and Embedded Systems

- A real-time system is used when rigid time requirements have been placed on the operation of a processor or the flow of data.

- An embedded system is a component of a more complex system
  - Control of a nuclear plant
  - Missile guidance
  - Control of home and car appliances (microwave oven, DVD players, car engines, …)

- Real-time systems
  - have well-defined time constraints.
  - may be either hard or soft real-time.
Distributed Systems

- Distribute the computation among several physical processors.

- *Loosely coupled clustered system* – each processor has its own local memory; processors communicate with one another through various communications lines

- Advantages of distributed systems
  - Resource and Load Sharing
  - Scalability
Handheld Systems

- Cellular telephones
- Smart X, where X = phones/watches
  - In 2011, more smart phones than PCs were shipped

![Worldwide smart phone and client PC shipments]

*Source: Canalys estimates © Canalys 2012*
Computer System Structures

- Prototypical System Architecture
- Instruction Execution
- I/O Operation
- Interrupt Processing
Prototypical System Architecture

- CPU
- Memory
- Memory Bus (proprietary)
- General I/O Bus (e.g., PCI)
- Graphics
- Peripheral I/O Bus (e.g., SCSI, SATA, USB)
Instruction Execution

- While executing a program, the CPU:
  - fetches the next instruction from memory (loading into IR)
  - decodes it to determine its type and operands
  - executes it
- May take multiple clock cycles to execute an instruction
- Examples:
  - LOAD R1, #3
  - LOAD R2, M2
  - STORE M3, R4
  - ADD R1, R2, R3

- Each CPU has a specific set of instructions that it can execute (instruction-set architecture).
How to Get Assembly Code From a Binary Executable?

- **Mac OS X**
  - `$ otool -t -v <binary>`

- **Linux**
  - `$ objdump -d <binary>`
Instruction Execution

- **Registers**
  - General registers (data/address)
  - **Program Counter (PC):** contains the memory address of the next instruction to be fetched.
  - **Stack Pointer (SP):** points to the top of the current stack in memory. The stack contains one frame for each procedure that has been entered but not yet exited.
  - **Program Status Word (PSW):** contains the condition code bits and various other control bits.
  - ...

- When *time multiplexing* the CPU, the operating system will often stop the running program to (re)start another one. In these cases, it must save the “state information” (e.g. values of the registers).
An I/O Operation

A canonical I/O device

- A basic protocol
  1. Process issues an I/O request
  2. OS waits by **polling** until device is not busy
  3. OS writes data to DATA Register
  4. OS writes command to COMMAND Register
  5. OS waits again until device is done with request
Lowering CPU Overheads with Interrupts

- An advanced protocol with interrupts
  1. Process issues an I/O request
  2. OS puts process to sleep waiting for I/O to complete
  3. Once I/O completes, device issues an interrupt trapping CPU jump into OS mode
  4. OS wakes up the sleeping process waiting for I/O
Lowering CPU Overheads with Interrupts

- Without interrupt, **no** I/O + compute overlap
  - I/O request

| CPU | 1 1 1 1 1 1 p p p p p p 1 1 1 1 1 |
| Disk | 1 1 1 1 1 1 |

- With interrupt, I/O + compute overlap ✓

| CPU | 1 1 1 1 1 1 2 2 2 2 2 2 1 1 1 1 1 |
| Disk | 1 1 1 1 1 1 |

Square: CPU cycle
1: Process 1
p: polling

Disk interrupts Process 2’s execution
Instruction Cycle with Interrupts

- Important hardware and software events are signaled to the CPU by an interrupt.
Interrupt Mechanism

- **Interrupt service routines (ISR) – Interrupt handler**
  - A piece of OS code that determine what action should be taken for each type of interrupt
  - E.g., reading data and return err code of device

- Once interrupt has been served, OS returns control over to the interrupted program
  - Context switch occurs

- **Modern OS are interrupt-driven**
Basic Interrupt Processing

1. The interrupt is issued
2. Processor finishes execution of current instruction
3. Processor signals acknowledgement of interrupt
4. Processor pushes PSW and PC onto control stack
5. Processor loads new PC value through the interrupt vector
6. ISR saves remainder of the process state information
7. ISR executes
8. ISR restores process state information
9. Old PSW and PC values are restored from the control stack

✓ What if another interrupt occurs during interrupt processing?
Incoming interrupts are disabled while another interrupt is being processed to prevent a lost interrupt.
Classes of Interrupts

- **I/O Interrupts**: Generated by an I/O controller, to signal normal completion of an operation or to signal a variety of error conditions.

- **Timer Interrupts**: Generated by a timer within the processor. This allows the operating system to perform certain functions on a regular basis.

- **Hardware Failure Interrupts**: Generated by a failure (e.g. power failure or memory parity error).

- **Traps (Software Interrupts)**: Generated by some condition that occurs as a result of an instruction execution
  - Errors
  - User request for an operating system service
Direct Memory Access (DMA)

- Used for high-speed I/O devices able to transmit information at (close to) memory speeds.
- Device controller transfers blocks of data from buffer storage directly to main memory without CPU intervention.
- Only one interrupt is generated per block, rather than one interrupt per byte/word.
DMA

- Without DMA support (c = \textit{copy} memory word)

<table>
<thead>
<tr>
<th>CPU</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>c</th>
<th>c</th>
<th>c</th>
<th>2</th>
<th>2</th>
<th>2</th>
<th>2</th>
<th>2</th>
<th>2</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disk</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

- With DMA support \(\checkmark\)

| CPU    | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 1 |
| DMA    |   |   |   |   |   | c | c | c |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| Disk   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   | 1 | 1 | 1 | 1 | 1 | 1 |

Only one interrupt generated
Dual-Mode Operation

- Operating System must protect *itself* and all other programs (and their data) from any malfunctioning program.

- Provide hardware support to differentiate between at least two modes of operations.
  1. *User mode* – execution done on behalf of a user.
  2. *Kernel mode* (also *monitor mode* or *system mode*) – execution done on behalf of operating system.
Dual-Mode Operation (Cont.)

- **Mode bit** added to computer hardware to indicate the current mode: kernel (0) or user (1).
- When an interrupt occurs hardware switches to kernel mode.

*Privileged instructions* can be issued only in kernel mode.
The system call can be executed by a generic trap instruction (or in some systems, by an instruction such as syscall).
Operating System Design Approaches

- Simple Structure
- Layered Approach
- Modular Approach
- Microkernels
- Virtual Machines
Simple System Structure

- Some operating systems do not have well-defined structures. Often, these started as simple systems and grew beyond their original scope.

- **MS-DOS** – written to provide the most functionality in the least space
  - not divided into modules
  - Although MS-DOS has some structure, its interfaces and levels of functionality are not well separated
MS-DOS Structure

application program

resident system program

MS-DOS device drivers

ROM BIOS device drivers
UNIX System Structure

- UNIX – limited by hardware functionality, the original UNIX operating system had limited structure. The UNIX OS consists of two separable parts.
  - System programs
  - The kernel (everything below the system-call interface and above the physical hardware)
    - Provides the file system, CPU scheduling, memory management, and other operating-system functions
    - A large number of functions for one level.
## UNIX System Structure

<table>
<thead>
<tr>
<th>(the users)</th>
</tr>
</thead>
<tbody>
<tr>
<td>shells and commands</td>
</tr>
<tr>
<td>compilers and interpreters</td>
</tr>
<tr>
<td>system libraries</td>
</tr>
</tbody>
</table>

**system-call interface to the kernel**

- signals terminal handling
- character I/O system
- terminal drivers
- file system
- swapping block I/O
- system
- disk and tape drivers
- CPU scheduling
- page replacement
- demand paging
- virtual memory

**kernel interface to the hardware**

- terminal controllers
- terminals
- device controllers
- disks and tapes
- memory controllers
- physical memory
Layered Approach

- The operating system is divided into a number of layers (levels), each built on top of lower layers. The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface.

- With *modularity*, layers are selected such that each uses functions (operations) and services of only lower-level layers.

- Simplifies debugging and system verification

- Disadvantages?
Modular Approach

- Modular kernel
  - The kernel has a set of core components
  - *Dynamically links* in additional services either during boot time or during run-time
  - Common in modern implementations of Unix such as Linux and Solaris
Microkernels

- Moves as much as possible from the kernel into the “user” space.
- Communication takes place between user modules using message passing (e.g. Mach operating system)
Microkernels (cont.)

- **Benefits**
  - easier to extend
  - more reliable (less code is running in kernel mode)
  - convenient for distributed architectures
  - Security (Maybe!)

- **Many modern OS are designed as Microkernels**
  - Apple Mac OS (based on Mach OS)
  - Many Smartphone OS
    - Android (L4 Microkernel family)
    - iPhone iOS (based on Mach)
Virtual Machines

- Originally proposed and implemented for VM Operating System (IBM)

- A virtual machine provides an interface *identical* to the underlying bare-metal hardware

- Each user is given her own virtual machine

- The operating system creates the illusion of multiple processes, each executing on its own processor with its own (virtual) memory
Virtual Machines (Cont.)

(a) Bare-metal Machine

(b) Virtual Machine

programming interface

kernel

hardware

virtual-machine implementation

kernel

VM1

VM2

VM3

hardware
OS History

MS/DOS
  ↓
Windows
  ↓
Windows NT
  ↓
Windows 8
  ↓
Windows 10
  ↓
Multics
  ↓
UNIX
  ↓
Linux
  ↓
Android
  ↓
iOS

MVS
  ↓
VMS
  ↓
VM/370
  ↓
BSD UNIX
  ↓
Mach
  ↓
NEXT
  ↓
Mac OS

Influence

Descendant