Operating Systems

Dickinson College
Computer Science 354
Spring 2012

 Professor Braught
✓ Email: braught@dickinson.edu
✓ Web: www.dickinson.edu/~braught
✓ Office: Tome 234
   • Office Hours: Monday 2:00-3:00
      Tuesday 10:00-11:00
      Wednesday 11:30-12:00
      Friday 3:00-4:00
✓ Class Meetings:
   • TF 1:30-2:45 Tome 231

 Syllabus:
✓ Homework Purpose and Marking
✓ Projects
✓ Exams
✓ Late Work Policy
✓ Class Schedule
✓ Suggested M.O.

 Course Outline:
✓ Joke
✓ What is an OS and why do we have them?
✓ Programs and Processes
✓ Process Scheduling
✓ Threads
✓ Thread Synchronization
✓ Memory Management
✓ File Systems

If OS’s Were Airlines

 Air Mac:
✓ All the stewards, captains, baggage handlers, and ticket agents look and act exactly the same. Every time you ask questions about details, you are gently but firmly told that you don't need to know, don't want to know, and everything will be done for you without your ever having to know, so just shut up.

 Air Windows:
✓ The terminal is pretty and colorful, with friendly stewards, easy baggage check and boarding, and a smooth take-off. After about 10 minutes in the air, the plane explodes with no warning whatsoever.

If OS’s Were Airlines

 Air Linux:
✓ Disgruntled employees of all the other OS airlines decide to start their own airline. They build the planes, ticket counters, and pave the runways themselves. They charge a small fee to cover the cost of printing the ticket, but you can also download and print the ticket yourself. When you board the plane, you are given a seat, four bolts, a wrench and a copy of the seat-HOWTO.html. Once settled, the fully adjustable seat is very comfortable, the plane leaves and arrives on time without a single problem, the in-flight meal is wonderful. You try to tell customers of the other airlines about the great trip, but all they can say is, "You had to do what with the seat?"
Operating Systems

What is an operating system?

Why do we have operating systems?

OS Purposes

An operating system serves two main purposes:

- It makes the computer easier to use.
  - Bridges the gap between the raw hardware and the user experience.
- It allows for more efficient use of the computer hardware.

Ease of Use (User)

User Perspective:

- OS makes it easier for users to do things like:
  - Launch programs
  - Use multiple programs concurrently
  - Keep track of files
  - Add additional hardware devices
  - Run really large programs

- As operating systems improve so does the user experience.

Ease of Use (Programmer)

Program / Programmer Perspective

- The OS makes it easier for programs (and programmers) to use the computer by providing support for common tasks:
  - Accessing hardware devices
  - Sharing system resources with other programs
  - Exchanging information and coordinating with other programs.

Efficient use of Hardware

Operating system advances that allow more efficient use of computer hardware than running a single program at a time:

- Task Switching
- Multiprogramming
- Timesharing

The use of these advances necessitate that the operating system also manage:

- Resource allocation
- Resource sharing
- Protection and Security

Operating System Design

An operating system will trade-off ease of use against efficient use of the hardware depending on the goals of the computer system for which it is designed.
**OS as a Bridge**

- Key elements of “user” experience:
  - Multiple concurrent programs
  - Large memory
  - Files and Directories
  - Protection from other programs / users
  - Communication / interaction with other programs / users
  - User Interface / GUI

**Raw Hardware Assumptions (CPU)**

- Single CPU Machine
  - Executes 1 instruction at a time
    - Fetch / Decode / Execute
      - Program Counter (PC): holds memory address for next fetch.
      - Instruction Register (IR): holds instruction for decode/execute.
  - Memory is array of bytes.
  - Instruction execution is “atomic”.
  - Programs store operands and results in general purpose registers.

**Process Context**

- A Process’ Context includes:
  - The register contents (GPRs and PC)
  - Contents of its memory
  - Files open for reading and writing
  - etc…

- A process’ context includes everything that is necessary to suspend and restart the process later.

**Raw Hardware Assumptions (Disk)**

- Hard Disks:
  - Basic hard disk controller can:
    - Read a sector (or block)
    - Write a sector (or block)
  - Sector to read/write is specified by a cylinder:head:sector (CHS) address.
  - Some disk controllers also use linear block addressing (LBA).

**Raw Hardware Assumptions (BIOS)**

- Basic Input/Output System (BIOS)
  - A number of small programs and subroutines:
    - Power on self test (POST)
    - System configuration utility
    - A set of routines for performing basic operations on common input/output devices. Such as:
      - Read / write a specified CHS from disk.
      - Read character from keyboard.
      - Display character on the screen.
    - OS bootstrap program
    - Stored on a Flash ROM that is part of the computer’s address space.
    - CMOS for configuration is also in address space.

**Bootstrap Process**

- In the beginning… there is only the BIOS.
  - The PC is initialized to the address of the POST program contained in the BIOS
  - The last instruction of the POST jumps to the address of the bootstrap program, also contained in the BIOS.
  - The bootstrap program uses the BIOS routines to load a program contained in the Master Boot Record (MBR) of the boot disk into memory at a known address.
  - MBR = first sector on the disk (512 bytes).
  - Boot disk is identified by data stored in the configuration CMOS.
  - The last instruction in the bootstrap program jumps to the address at which the MBR program was loaded.
  - The MBR program loads the OS kernel.
  - Often indirectly by loading another program (a secondary boot loader) that then loads the kernel.
Random OS Quote

➤ Saying that [Microsoft Windows] XP is the most stable MS OS is like saying that asparagus is the most articulate vegetable.

Dave Barry

OS Control of Shared Resources

➤ All use of shared system resources must be controlled by the operating system if it is to provide:
  ✔ Protection
  ✔ Multiprogramming
  ✔ Timesharing

➤ Additional hardware is required to ensure that the operating system can control all use of shared resources.

Hardware Support for OS

➤ The hardware support that is required is provided by:
  ✔ A mechanism for handling interrupts
  ✔ A mechanism for making system calls
  ✔ Dual mode processor operation
  ✔ Base and limit registers
  ✔ A Timer hardware device

Interrupts

➤ An interrupt is a signal from a device indicating that:
  ✔ An error has occurred.
  ✔ An event has occurred.
  ✔ Mouse has moved.
  ✔ Key has been pressed.
  ✔ An operation is complete.
  ✔ Data has been successfully written.
  ✔ Data is ready to be retrieved.

Interrupt Handling

➤ When an interrupt occurs:
  ✔ PC is loaded with address the address of an Interrupt Service Routine (ISR), which is part of the BIOS or the Operating System.
  ✔ Typically:
    ✔ ISR asks OS to suspend the process that was executing.
    ✔ ISR processes the interrupt.
      ✔ Reads mouse action / fetches character from keyboard etc…
      ✔ ISR asks the OS restart a process.

Vectored Interrupts

➤ Modern systems use vectored interrupts.
  ✔ Each device is assigned an interrupt request number (IRQ).
  ✔ The device’s IRQ is used as an index into the interrupt vector.
    ✔ The value at each index is the address of the ISR associated with the interrupt.
  ✔ The value from the interrupt vector is loaded into the PC.
Interrupts and Multiprogramming

- Interrupts enable multiprogramming via:
  - Interrupt driven I/O
  - Direct memory access (DMA)

System Calls

- System calls are the mechanism by which a user processes requests resources and services that are controlled by the operating system.
  - Conceptually, a system call is like a function call to a function that is part of the operating system.
  - The mechanism is just a little different.

System Calls - Abstractly

- When a process makes a system call, control is transferred to the operating system. Code in the operating system carries out the request and eventually control is returned to the process.

System Call Instructions

- A process makes a system call by executing a special machine/assembly language instruction:
  - SYSCALL
  - TRAP
  - SC
  - INT
  - Usually you do not see the system call instruction because it is wrapped inside a language library (java / c, c++ / etc).

System Calls are Interrupts

- A system call causes control to automatically transfer to OS using the interrupt vector.

System Call Parameters

- Parameters for a system call can be passed to the OS in three general ways:
  - On the system stack
  - In registers
  - In a block of memory
  - Different techniques are used for different system calls and even for individual parameters of the same system call.
    - E.g. Writing to a file. The file to write is usually indicated by an integer passed in a register. The data to be written is passed using a pointer to block of memory (the pointer can be passed in a register).
Dual Mode Operation

- To provide protection, modern processors have (at least) two different modes of operation:
  - **Kernel Mode**
    - a.k.a. [System | Supervisor | System | Privileged] Mode
    - Any of the machine’s instructions can be executed when CPU is in Kernel Mode.
  - **User Mode**
    - A limited sub-set of the machine’s instructions can be executed when the CPU is in User Mode.

Kernel Mode

- Machines are designed such that instructions that access shared resources are privileged (i.e., kernel mode) instructions.
  - Privileged instructions may be executed when the processor is in kernel mode.
  - An attempt to execute a privileged instruction when the processor is in user mode generates an interrupt.
  - The ISR, provided by the OS, that is invoked can then terminate the offending process.

Dual Mode, Interrupts and System Calls

- Every system call or interrupt automatically switches the processor to kernel mode before control transfers to the operating system code.
  - The OS then switches the kernel back to user mode before returning control to a user process.

Base and Limit Registers

- Base and Limit registers provide the simplest mechanism for protecting memory.

Timer Device

- Time sharing is enabled by the *timer device*.
  - The timer is usually implemented using a fixed rate clock and a counter.
    - The counter is set to a positive value.
    - The value of the counter is then decremented on each tick of the clock.
    - When the counter reaches 0 an interrupt is generated.
    - The ISR may then switch the running process.

Random OS Quote

- One of the main advantages of Unix over, say, MVS, is the tremendous number of features Unix lacks.
  - Chris Torek
  - Unix Guru
OS Responsibilities

- Operating systems are generally divided into several different sub-systems:
  - Process Management
    - Process = Running program
  - Memory Management
  - Storage Management

Process Management

- Process management includes:
  - Creating / Deleting processes
  - Scheduling processes
  - Blocking / Unblocking processes
  - Interprocess communication mechanisms
  - Synchronization mechanisms

Memory Management

- Memory management is responsible for:
  - Tracking used/free memory
  - Allocating/deallocating memory
  - Movement of processes to/from secondary storage
  - i.e. Virtual Memory

Storage Management

- Storage management is responsible for:
  - Creating / deleting files
  - Creating / deleting directories
  - File / directory manipulation
  - Read / write / change permissions
  - Mapping files and directories onto disk
  - Tracking free / used disk space.

Implementing Operating Systems

- Some of the design decisions faced in implementing an operating system include:
  - System software vs. OS kernel
  - Separation of mechanism and policy
  - Kernel architecture

System Software vs. Kernel

- Many services can be implemented either in the OS kernel or as processes that can be run in user mode.
**Mechanism and Policy**

- *Policies* are likely to change over time and thus should be separate from the *mechanisms* used to enforce them.
- An ideal mechanism is general enough to support a wide range of policies.

**Kernel Architecture**

- There are roughly 4 major architecture alternatives for OS Kernel design:
  - Monolithic (a.k.a. Simple) Structure
  - Layered Structure
  - Microkernel Structure
  - Modular Structure

---

**Monolithic Kernels**

- In a monolithic kernel nearly all OS functionality is contained in a single software module.
  - “... the ‘Big Mess’. The structure is that there is no structure.”
  - Tannenbaum
- Benefits?
- Drawbacks?
- Examples:
  - MS-DOS
  - Original UNIX

---

**The MS-DOS Monolithic Kernel**

- Application Programs
- MS-DOS Kernel
- MS-DOS Device Drivers
- BIOS Device Drivers
- Physical Hardware

---

**Original UNIX Kernel**

- User Applications and System Software
- System Call Interface
- Process Management
- Memory Management
- Device Management
- Device Drivers
- Physical Hardware

---

**Layered Kernels**

- OS is designed in layers such that:
  - Each layer uses only the services provided by the next lower layer.
  - The services provided by each layer are defined by a public interface.
Micro-Kernels

- With a micro-kernel only that functionality that actually requires kernel mode is included in the kernel.
  - Basic process and memory management
  - Message passing
  - Keep kernel policy free.
- All other functionality is implemented as separate processes that execute in user mode.

Modular Kernels

- Modular kernels have a core set of capabilities (almost a micro-kernel) but then also allow other modules to be dynamically added to the kernel during boot or during execution.

Virtual Machines

- Virtual machines provide a mechanism for hosting multiple independent operating systems on a single machine.

VMWare

- The VMWare virtualization layer runs on a host operating system.
- This application appears to the guest operating system as if it is a complete machine with its own CPU, memory and I/O devices.

Random OS Quote

- We just don't think a Linux partition on a mainframe makes a lot of sense. It's kind of like having a trailer park in the back of your estate.

  Scott McNealy
  Co-founder of Sun Microsystems