CSSE 2310 7231

Computer Systems Principles + Programming

Week 3.1

Introduction to Operating Systems

School of Information Technology and Electrical Engineering
The University of Queensland

CSSE 2310 7231

Lecture Outline

- What is an operating system?
- Different views of operating systems
- History of operating systems
- Example operating systems
- Hardware support
- OS organisation

- Slide Credits
  - E.N. Elnozahy, U Texas
  - R. Chandra, Cornell University

CSSE 2310 7231

What is an Operating System?

- Write down what you think an operating system does.

- [Discussion to take place in class]

CSSE 2310 7231

CSSE 2310 7231

CSSE 2310 7231

Break

CSSE 2310 7231

The 5 Views of OS

- Your view of an OS depends on who you are:
  - The **hardware** view
  - The **operating system designer**’s view
  - The **application programmer**’s view
  - The **end-user**’s view
  - The **system administrator**’s view

CSSE 2310 7231

The Hardware View

- The operating system is the layer of software that interacts directly with the hardware, concerns revolve around:
  - The boot process
  - Devices and how the OS can use them
  - The interactions between H/W and OS

CSSE 2310 7231

The OS Designer’s View

- The interest revolves mainly about the OS itself, its internal structure, its efficiency, performance, data structures, etc..
  - How can we make the OS more efficient
  - How can we add more functionality?
  - How do we debug the OS? Make it more reliable, scalable, etc..

CSSE 2310 7231

The Application Programmer’s View

- The OS is like a library with a well defined set of API’s
  - What abstractions are available from the OS?
  - How well is the API structured? Not too low-level, or high-level.
  - How portable is the interface?
  - Protection of the intellectual investment--don’t want to keep rewriting the same program for each new OS release.
    - Explains why Windows has been so successful!

CSSE 2310 7231

The End-User’s View

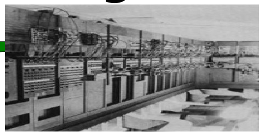
- The OS is just a program that happens to be pre-installed
  - Must not crash or externalize the ugly aspects of the machine
  - Must protect investment in existing software & applications
  - Users care about applications, not the OS
  - A good OS is the one that is most transparent
- Contrast Windows, MacOS & UNIX

CSSE 2310 7231

The System Administrator View

- An OS is a program that allows the efficient and equitable usage of resources:
  - How can it track usage for accounting?
  - How easy is it to install new software?
  - Security
  - Fairness
- Contrast Windows, MacOS, UNIX, and mainframe systems

History of Operating Systems

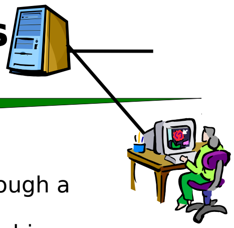


- Earliest computers had no OS
  - programmed directly
- Initially, OS was just a run-time library
  - You linked your application with the OS, loaded the whole program into memory, and ran it
  - How do you get it into the computer? Through the control panel!
- Simple batch systems (mid 1950s – mid 1960s)
  - Permanently resident OS in primary memory
  - Loaded a single job from card reader, ran it, loaded next job...
  - Control cards in the input file told the OS what to do
  - Spooling allowed jobs to be read in advance onto tape/disk

Multiprogramming Systems

- Multiprogramming systems increased utilization
  - Developed in the 1960s
  - Keeps multiple runnable jobs loaded in memory
  - Overlaps I/O processing of a job with computation of another
  - Benefits from I/O devices that can operate asynchronously
  - Requires the use of interrupts and DMA
  - Optimizes for throughput at the cost of response time

Time Sharing Systems



- Timesharing (1970s) allows interactive computer use
  - Users connect to a central machine through a terminal
  - User feels as if they have the entire machine
  - Based on time-slicing: divides CPU equally among the users
  - Allows active viewing, editing, debugging, executing process
  - Security mechanisms needed to isolate users
  - Requires memory protection hardware for isolation
  - Optimizes for response time at the cost of throughput

Personal Operating Systems

- PC OS's, 1974+
  - Apple II, and others
- MSDOS 1980+
  - The PC revolution
- Windowing OS 1983+
  - Apple (MacOS) & Xerox (Pilot OS)
  - Windows 3.1, OS/2
- Computers are cheap everyone has a computer
- Initially, the OS was a library
- Advanced features were added back
  - Multiprogramming, memory protection, etc

Distributed Operating Systems

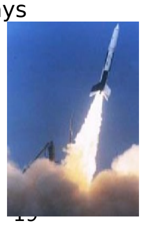
- Cluster of individual machines
  - Over a LAN or WAN or fast interconnect
  - No shared memory or clock
- Asymmetric vs. symmetric clustering
- Sharing of distributed resources, hardware and software
  - Resource utilization, high availability
- Permits some parallelism, but speedup is not the issue
- SANs, Oracle Parallel Server

Parallel Operating Systems

- Multiprocessor or tightly coupled systems
- Many advantages:
  - Increased throughput
  - Cheaper
  - More reliable
- Asymmetric vs. symmetric multiprocessing
  - Master/slave vs. peer relationships

Real Time Operating Systems

- Goal: To cope with rigid time constraints
- Hard real-time
  - OS guarantees that applications will meet their deadlines
  - Examples: TCAS, health monitors, factory control
- Soft real-time
  - OS provides prioritization, on a best-effort basis
  - No deadline guarantees, but bounded delays
  - Examples: most electronic appliances
- Real-time means “predictable”
  - NOT fast



OS Types & Examples

- **Desktop**: MSDOS, Windows 95/98/ME/NT/ 2000/XP/Vista/7, MacOS, Linux
- **Workstation / Server**: HPUX, AIX, Solaris, Linux, BSD (many variants), Windows Server, Novell Netware
- **Minicomputers**: OS/400, VMS
- **Mainframes**: CMS/MVS (now z/OS)
- **Embedded**: OS-9, VxWorks, Lynx, PalmOS, Windows CE, Symbian OS, uCLinux, IOS
  - Some of these are real-time

Short Break

Stand up and stretch

Hardware Support

- Operating system needs to:
  - control I/O devices
  - control access to the hardware

  all while denying these privileges to user programs:
  - for protection
  - for abstraction/ease of use
- Hardware supports two modes of operation (or more):
  - access to hardware & I/O devices is done through privileged instructions, these are only available in **“supervisor” mode**
  - privileged instructions cannot be executed in **“user” mode**

Implementation

- Using a bit in the processor (i.e. 0 or 1)
- Operating system code runs in supervisor mode, while user program code runs in user mode
- Switching from user to supervisor mode occurs on:
  - **interrupts**: hardware devices needing service
  - **exceptions**: user program acts silly (divide by 0, bus error, etc)
  - **trap** instructions: user program requires OS service (**system call**)
- Switching back occurs by an **RTI** instruction

On Interrupts

- Hardware calls the operating system at a pre-specified location
- Operating system saves state of the user program
- Operating system identifies the device and cause of interrupt
- Responds to the interrupt (possibly killing program, <CTRL-C>)
- Operating system restores state of the user program (if applicable) or some other user program
- Execute an RTI instruction to return to the user program
- User program continues exactly at the same point it was interrupted.

Key Fact: None of this is visible to the user program

On Exceptions

- Hardware calls the operating system at a pre-specified location
- Operating system identifies the cause of the exception (e.g. divide by 0)
- If user program has exception handling specified, then OS adjust the user program state so that it calls its handler
- Execute an RTI instruction to return to the user program
- If user program did not have a specified handler, then OS kills it and runs some other user program, as available

Key Fact: Effects of exceptions are visible to user programs and cause abnormal execution flow

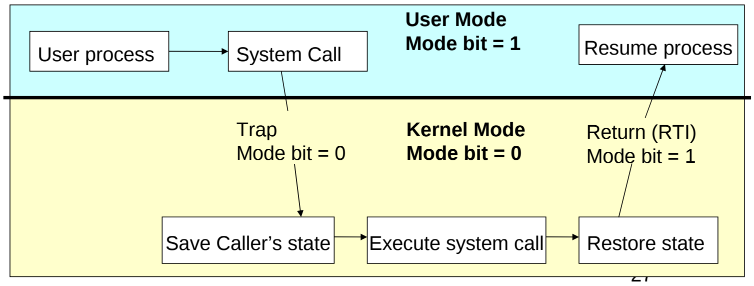
On System Calls

- User program executes a trap instruction (system call)
- Hardware calls the operating system at a pre-specified location
- Operating system identifies the required service and parameters, e.g. **`open(filename, O_RDONLY);`**
- Operating system executes the required service
- Operating system sets a register to contain the result of call
- Execute an RTI instruction to return to the user program
- User program receives the result and continues

Key Fact: To the user program, it appears as a function call executed under program control

Crossing Protection Boundaries

- User calls OS procedure for “privileged” operations
- Calling a kernel mode service from user mode program:
  - Using System Calls
  - System Calls switches execution to kernel mode



System Calls

- Programming interface to services provided by the OS
- Typically written in a high-level language (C or C++)
- Mostly accessed by programs using APIs
- Three common APIs:
  - Win32 API for Windows
  - **POSIX** API for POSIX-based systems (UNIX, Linux, Mac OS X)
    - **This is the emphasis for COMP2303**
  - Java API for the Java virtual machine (JVM)

Reducing System Call Overhead

- Problem: The user-kernel mode distinction poses a performance barrier
  - Crossing this hardware barrier is costly.
  - System calls take 10x-1000x more time than a procedure call
- Solution: Perform some system functionality in user mode
  - Libraries (DLLs) can reduce number of system calls,
    - by caching results (getpid) or
    - buffering (open/read/write vs. fopen/fread/fwrite).

OS Structure

- An OS is just a program:
  - It has a main() function, which gets called only once (during boot)
  - Like any program, it consumes resources (such as memory), can do silly things (like generating an exception), etc.
- But it is a very strange program:
  - It is “entered” from different locations in response to external events
  - It does not have a single thread of control, it can be invoked simultaneously by two different events (e.g. sys call & an interrupt)
  - It is not supposed to terminate
  - It can execute any instruction in the machine

Booting the System

- CPU loads boot program from ROM (e.g. BIOS in PC's)
- Boot program:
  - Examines/checks machine configuration (number of CPU's, how much memory, number & type of hardware devices, etc.)
  - Builds a configuration structure describing the hardware
  - Loads the operating system, and gives it the configuration structure
- Operating system initialization:
  - Initialize kernel data structures
  - Initialize the state of all hardware devices
  - Creates a number of processes to start operation (e.g. getty in UNIX, the Windowing system in Windows)

Operating System in Action

- After basic processes have started, the OS runs user programs, if available, otherwise enters the **idle loop**
- In the idle loop:
  - OS executes an infinite loop (UNIX)
  - OS performs some system management & profiling
  - OS halts the processor and enter in low-power mode (notebooks)
  - OS computes some function (DEC's VMS on VAX computed pi)
- OS wakes up on:
  - interrupts from hardware devices
  - traps from user programs
  - exceptions from user programs