Architectural Support for Operating Systems

Today
- Computer system overview

Next time
- OS components & structure
Announcements and reminders

- In case you have missed anything ...
- Project 1 is out!
Computer architecture and OS

- OS is intimately tied to the hardware it runs on
  - OS design is impacted by it
  - OS needs result on new architectural features

- Abstract model of a simple computer
The brain with a basic operation cycle
- Fetch next instruction
- Decode it to determine type & operands
- Execute it

... and a specific set of instructions
- E.g. combine ops (ADD), control flow, data movement
- Architecture specific - Pentium != SPARC

Since memory access is slow ... registers
- General regs to hold variables & temp. results
- Special regs such as Program Counter, Stack Pointer
This model is overly simplistic: pipeline architectures, superscalar, ...

Multithreading/Hyperthreading and multicore
Memory

- **Ideal** – fast, large, cheap and persistent
- **Real** – storage hierarchy
  - **Registers**
    - Internal to the CPU & just as fast
    - 64x64 in a 64 bit machine
  - **Cache**
    - Split into cache lines
    - If word needs is in cache, get in ~2 cycles
  - **Main memory**
  - **Hard disk**
  - **Magnetic tape**
  - **Coherency?**

First core-based memory: IBM 405 Alphabetical Accounting Machine
Architectural trends impact OS design ...

- **Processing power**
  - Doubling every 18 months (100x per decade)
  - but power is a serious issue

*Normalized (to i486) power versus normalized scalar performance for multiple generations of Intel microprocessors*

*http://www.intel.com/pressroom/kits/core2duo/pdf/epi-trends-final2.pdf*
Architectural trends impact OS design ...

- **Primary memory capacity**
  - Same and for the same reason
  
<table>
<thead>
<tr>
<th>Year</th>
<th>Capacity</th>
<th>Price</th>
<th>Price per MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>1980</td>
<td>64KB</td>
<td>$405.00</td>
<td>$6,480/MB</td>
</tr>
<tr>
<td>1990</td>
<td>8MB</td>
<td>$851.00</td>
<td>$106/MB</td>
</tr>
<tr>
<td>2000</td>
<td>64MB</td>
<td>$99.89</td>
<td>$1.56/MB</td>
</tr>
<tr>
<td>2009</td>
<td>4GB</td>
<td>$39.99</td>
<td>$0.010/MB</td>
</tr>
<tr>
<td>2014</td>
<td>4GB</td>
<td>$29.99</td>
<td>$0.007/MB</td>
</tr>
</tbody>
</table>

*http://www.jcmit.com/memoryprice.htm*
Architectural trends impact OS design ...

- **Disk capacity**
  - Double every 12 months (1000x per decade)

1961 IBM 1301
~26MB ~$115,500 (~890k in 2013)

Disk pack
Weight: ~10lbs
Capacity: 2MB

2014 WD 3TB My Book Essential ~ $120

1961 ~ $4,440/MB
2014 ~ $0.00004/MB
Architectural trends impact OS design

- **Solid state storage (SSD)**
  - 10-100k random IOs per second
  - 800 MB/s transfer rates
  - Costly, but quickly riding Moore’s law
  
  
  2011 Crucial 512GB SSD $750
  2012 … $400
  2014 … $210

**Hard to imagine with hard drives** – “The SSD revolution”, arstechnica
June 25, 2012
Architectural trends impact OS design ...

- Gap between CPU and I/O speeds

Relative speed of key components – an unbalanced system
FAWN project @ CMU

*http://www.cmu.com/~dga
Architectural trends impact OS design

- Optical bandwidth today
  - Doubling every 9 months (Butter’s law)
  - 50% improvement each year for home users (Nielsen’s law)
  - Factor of 10,000 every decade
  - 10x as fast as disk capacity!
  - 100x as fast as processor performance!

- What are some of the implications of these trends?
  - E.g.: from mainframes to desktops to cloud computing
And now a short break …

Loop

STARE BLANKLY AT SCREEN

ABSENTLY CHECK SMALLER DEVICE

OPEN NEWS SITE

GET BORED

START READING

xkcd
... and OS needs shape the architecture

- Arch support can simplify/complicate OS tasks
  - E.g., early PC OS (DOS, MacOS) lacked support for virtual memory, partly because HW lacked key features

- Features built primarily to support OS’s:
  - Protected modes of execution (kernel vs. user)
  - Protected instructions
  - System calls (and software interrupts)
  - Memory protection
  - I/O control operations
  - Timer (clock) operation
  - Interrupts and exceptions
  - Synchronization instructions
Consider timesharing

- Multiprogramming & timesharing are useful
  - Multiprogramming – “using different parts of the hardware at the same time for different tasks”
  - Timesharing – “several persons making use of the computer at the same time”

- but
  - How to protect programs from each other & kernel from all?
  - How to handle relocation? OS may need to run a particular program at different times from several locations
OS protection

- Some instructions are restricted to the OS
  - e.g. Directly access I/O devices, manipulate memory state management

- How does the CPU know if a protected instructions should be executed?
  - Architecture must support 2+ mode of operation
  - Mode is set by status bit in a protected register (PSW)
    - User programs execute in user mode, OS in kernel mode

- Protected instructions can only execute in kernel mode
Crossing protection boundaries

- How can apps do something privileged?
  - e.g. How do you save a file if you can't do I/O?

- User programs must call an OS procedure
  - Ask the OS to do it
  - OS defines a set of **system calls**
  - User-mode program makes a system call
  - How does the user to kernel-mode transition happen?
Crossing protection boundaries

- The system call …
  - Causes an exception which vector to a kernel handler
  - Passes a parameter indicating which syscall is
  - Saves caller's state so it can be restored
    - *What would happen if the kernel didn’t save state?*
  - OS must verify caller's parameters
    - *Why should it do that?*
  - Must be a way to go back to user once done
    - A special instruction sets PC to the return address and the execution mode to user

- *A bit like a regular subroutine call, right?*
Crossing protection boundaries

A system call

read(int fd, void *buffer, int numbytes)

User mode
Kernel mode

trap handler

sysread() kernel routine

ERET instruction

Save user PC
PC = trap handler address
Enter kernel mode

Save application state
Verify system call number
Find sys_read handler

Verify args
Initiate read
Choose next proc to run
Setup return values
Restore application state

Restore PC
Enter user mode
Exception handling and protection

- All entries to the OS use the same mechanism
  - Acquiring privileged mode and branching to trap handler are inseparable

- Interrupts, exceptions and traps
  - Interrupt: asynchronous, caused by an external hw event
  - Exception: synchronous; unexpected, automatically generated trap (coerced rather than requested); divide by zero
  - Trap: synchronous; programmer initiated, expected transfer of control to a special handler

- Privileged instructions and resources are the basis for most everything: Memory protection, I/O …
Next issue – Memory relocation

- Simplest model – base + limit
  - Base (start) of program + limit registers
  - Used by both CDC 6600 (first supercomputer) and Intel 8088, hear of the IBM PC
  - Changing program means changing base+limit
  - Solves relocation and protection (check on the CDC 6600)
  - Cost 2 registers + cycle time incr

- More sophisticated alternatives
  - 2 base and 2 limit registers for text & data; allow sharing program text
  - Paging, segmentation, virtual memory

If \( \geq \) base and \( < \) base + limit, OK else trap to OS with error
OS needs shape the architecture – I/O

- **I/O Device**
  - Device + Controller (simpler I/F to OS; think SCSI)
    - Read sector x from disk y → (disk, cylinder, sector, head), …

- **How does the kernel start an I/O?**
  - Special I/O instructions
  - Memory-mapped I/O

- **How does it notice when the I/O is done?**
  - Polling – *are we done yet?*
  - Interrupts – *let me know when you are done?*

- **How does it exchange data with the I/O device?**
  - Programmed I/O
  - Direct Memory Access (DMA)
OS control flow

- OSs are event driven
  - Once booted, all entry to kernel happens as result of an event (e.g. signal by an interrupt), which
    - Immediately stops current execution
    - Changes to kernel mode, event handler is called

- Kernel defines handlers per event type
  - Specific types are defined by the architecture
    - e.g. timer event, I/O interrupt, system call trap
Interrupts and exceptions

Three main types of events: interrupts & exceptions

- Exceptions/traps caused by SW executing instructions
  - E.g., a page fault
  - E.g., an attempted write to a read-only page
  - An expected exception is a “trap”, unexpected is a “fault”

- Interrupts caused by HW devices
  - E.g., device finishes I/O
  - E.g., timer fires
How can the OS retains control when a program gets stuck in an infinite loop?
– Use a HW timer that generates a periodic interrupt
– Before it transfers to a user program, the OS loads the timer with a time to interrupt (how long?)
– When time's up, interrupt transfers control back to OS
  • OS pick a program to schedule next (which one?)

Should the timer be privileged?
– For reading or for writing?
Synchronization

• Issues with interrupts
  – Many at a time, executing code can interfere with interrupted code
  – OS must be able to synchronize concurrent processes

• Synchronization
  – Guarantee that short instruction sequences (e.g. read-modify-write) execute atomically
  – Two methods
    • Turn off interrupts, execute sequence, re-enable interrupts
    • Have special, complex atomic instructions – test-and-set

Management of concurrency & asynchronous events is the biggest difference between systems-level & traditional app programming
Summary

- This is far from over – new architectural features are still being introduced
  - Support for virtual machine monitors
  - Hardware transaction support
  - Support for security
  - …

- Transistors are free so Intel/AMD/… need to find applications that require new hardware that you would want to buy …