CS 61C:
Great Ideas in Computer Architecture

Lecture 22:
Operating Systems

Krze Asanovic & Randy H. Katz
http://inst.eecs.berkeley.edu/~cs61c/fa17

CS61C so far...

So How is a Laptop Any Different?

Raspberry Pi (Less than $40 on Amazon in 2017)

Adding I/O

It's a Real Computer!
But Wait...

• That’s not the same! When we run VENUS, it only executes one program and then stops.
• When I switch on my computer, I get this:

Yes, but that’s just software! The Operating System (OS)

Well, “Just Software”

• The biggest piece of software on your machine?
• How many lines of code? These are guesstimates:

Operating System

What Does the OS do?

• OS is first thing that runs when computer starts
• Finds and controls all devices in the machine in a general way
  − Relying on hardware specific “device drivers”
• Starts services (100+)
  − File system,
  − Network stack (Ethernet, WiFi, Bluetooth,...),
  − TTY (keyboard),
  − ...
• Loads, runs and manages programs:
  − Multiple programs at the same time (time-sharing)
  − Isolate programs from each other (isolation)
  − Multiplex resources between applications (e.g., devices)

Agenda

• Devices and I/O
• Polling
• Interrupts
• OS Boot Sequence
• Multiprogramming/time-sharing

Agenda

• Devices and I/O
• Polling
• Interrupts
• OS Boot Sequence
• Multiprogramming/time-sharing
How to Interact with Devices?

• Assume a program running on a CPU. How does it interact with the outside world?
• Need I/O interface for Keyboards, Network, Mouse, Screen, etc.
  – Connect to many types of devices
  – Control these devices, respond to them, and transfer data
  – Present them to user programs so they are useful

Instruction Set Architecture for I/O

• What must the processor do for I/O?
  – Input: read a sequence of bytes
  – Output: write a sequence of bytes
• Interface options
  a) Special input/output instructions & hardware
  b) Memory mapped I/O
    § Portion of address space dedicated to I/O
    § I/O device registers there (no memory)
    § Use normal load/store instructions, e.g. lw/sw
    § Very common, used by RISC-V

Memory Mapped I/O

• Certain addresses are not regular memory
• Instead, they correspond to registers in I/O devices

Processor-I/O Speed Mismatch

• 1 GHz microprocessor I/O throughput:
  – 4 Gb/s (lw/sw)
  – Typical I/O data rates:
    § 10 B/s (keyboard)
    § 100 M B/s (bluetooth)
    § 50 M-60 M B/s (USB 2)
    § 100 M-1 G B/s (WiFi, depends on standard)
    § 125 M B/s (Gigabit Ethernet)
    § 550 M-1 G B/s (cutting edge SSD)
    § 3.2 G B/s (USB 3.1 Gen 2)
    § 5.4 G B/s (DDR4 DRAM)
  – These are peak rates – actual throughput is lower
• Common I/O devices neither deliver nor accept data matching processor speed

Agenda

• Devices and I/O
  • Polling
  • Interrupts
  • OS Boot Sequence
  • Multiprogramming/time-sharing
Processor Checks Status before Acting

• Device registers generally serve two functions:
  • Control Register, says it’s OK to read/write (I/O ready)
  [think of a flagman on a road]
  • Data Register, contains data
• Processor reads from Control Register in loop
  – Waiting for device to set Ready bit in Control reg (0 → 1)
  – Indicates “data available” or “ready to accept data”
• Processor then loads from (input) or writes to (output) data register
• I/O device resets control register bit (1 → 0)
• Procedure called “Polling”

I/O Example (Polling)

• Input: Read from keyboard into a
  • lui 0,t0    #7ffff0000 (io addr)
  • waitloop: lw $t1,$0    #read control
  •andi $t1,$t1,0x1    #ready bit
  • beq $t1,zero,waitloop
  • lw $a0,4($t0)    #data
• Output: Write to display from a
  • lui 0,t0    #7ffff0000
  • waitloop: lw $t1,8($t0)    #write control
  •andi $t1,$t1,0x1    #ready bit
  • beq $t1,zero,waitloop
  • sw $a1,12($t0)    #data

“Ready” bit is from processor’s point of view!

Cost of Polling?

• Assume for a processor with
  – 1 GHz clock rate
  – Taking 400 clock cycles for a polling operation
  • Call polling routine
  • Check device (e.g., keyboard or window input available)
  • Return
  – What’s the percentage of processor time spent polling?
• Example:
  – Mouse
  – Poll 30 times per second
  • Set by requirement not to miss any mouse motion
    [which would lead to choppy motion of the cursor on the screen]

Peer Instruction

Hard disk: transfers data in 16-byte chunks and can transfer at 16
MB/second. No transfer can be missed. What percentage of processor

Cost of Polling?

• Assume for a processor with
  – 1 GHz clock rate
  – Taking 400 clock cycles for a polling operation
  • Call polling routine
  • Check device (e.g., keyboard or window input available)
  • Return
  – What’s the percentage of processor time spent polling?
• Example:
  – Mouse
  – Poll 30 times per second
  • Set by requirement not to miss any mouse motion
    [which would lead to choppy motion of the cursor on the screen]

Peer Instruction

Hard disk: transfers data in 16-byte chunks and can transfer at 16
MB/second. No transfer can be missed. What percentage of processor
time is spent in polling (assume 1 GHz clock)?

• 2%
• 4%
• 10%
• 20%

What is the Alternative to Polling?

• Polling wastes processor resources
• Akin to waiting at the door for guests to show up
  – What about a bell?
• Computer lingo for bell:
  – Interrupt
  – Occurs when I/O is ready or needs attention
  • Interrupt current program
  • Transfer control to special code “interrupt handler”

Agenda

• Devices and I/O
• Polling
• Interrupts
• OS Boot Sequence
• Multiprogramming/time-sharing
In CS61C (other definitions in use elsewhere):

- **Interrupt** — caused by an event *external* to current running program
  - E.g., key press, disk I/O
  - Asynchronous to current program
  - Can handle interrupts on any convenient instruction
    - "Whenever it's convenient, just don't wait too long"
  - Exception — caused by some event *during* execution of one instruction of current running program
    - E.g., divide by zero, bus error, illegal instruction
    - Synchronous
    - Must handle exception precisely on instruction that causes exception
      - "Drop whatever you are doing and act now"
  - Trap — action of servicing interrupt or exception by hardware jump to interrupt or trap handler* code

### Terminology

An external or internal event that needs to be processed — by another program — the OS. The event is often unexpected from original program’s point of view.

#### Traps/Interrupts/Exceptions:

- **Program**
- **Trap handler**
- **Handler Execution**
- **Stack Frame**
- **Incoming interrupt suspends instruction stream**
- **Manual handling**
- **Exception**
- **Interrupt**
- **Decision**
- **Execution**
- **Synchronous**
- **Asynchronous**
- **Spike**
- **Register**
- **PC**
- **CPU**
- **PC address**
- **Exception cause**
- **Status**
- **Trap**
- **Hardware jump**
- **Machine state**
- **Instruction**
- **Exception**
- **Instruction following offending ADD**
- **Virtual memory to function properly** (see next lecture)
- **Transfer execution to trap handler**
- **Machine Exception Program Counter**
- **Incoming interrupt suspends instruction stream**
- **Manually handling**
- **Exception**
- **Interrupt**
- **Decision**
- **Execution**
- **Synchronous**
- **Asynchronous**
- **Spike**
- **Register**
- **PC**
- **CPU**
- **PC address**
- **Exception cause**
- **Status**
- **Interrupt handler**
- **Hardware jump**
- **Machine state**
- **Instruction**
- **Exception**
- **Instruction following offending ADD**
- **Virtual memory to function properly** (see next lecture)
**Administivia**

- Homework 5 is due tomorrow!
- Project 3: Performance Programming
  - Due Monday, Nov 20 (right before Thanksgiving break)
  - Project Party tomorrow 7-9pm in Cory 293
- Guerrilla Session tonight in 7-9pm in Cory 293
  - Topics covered: Parallelism & MapReduce (Floating Point if time permits)
  - Second to last guerrilla session—come and get valuable exam practice!
- Project 4 will be released by next Monday at the latest
  - You’ll get at least 2 weeks to complete it
  - Designed to be straightforward!

---

**Agenda**

- Devices and I/O
- Polling
- Interrupts
  - OS Boot Sequence
  - Multiprogramming/time-sharing

---

**What Happens at Boot?**

- When the computer switches on, it does the same as VENUS: the CPU executes instructions from some start address (stored in Flash ROM)

  *BIOS:* Find a storage device and load first sector (block of data)

  2. **Bootloader** (stored on, e.g., USB drive): Load the OS kernel from disk into a location in memory and jump into it

  3. **OS Boot:** Initialize services, drivers, etc.

  4. **init:** Launch an application that waits for input in loop (e.g., terminal/bootloop)

---

**Launching Applications**

- Applications are called “processes” in most OSs
  - Thread: shared memory
  - Process: separate memory
  - Both threads and processes run (pseud)simultaneously
- Apps are started by another process (e.g., shell) calling an OS routine (using a “syscall!”)
  - Depends on OS, but Linux uses `fork` to create a new process, and `execve` (execute file command) to load application
- Loads executable file from disk (using the file system service) and puts instructions & data into memory (.text, .data sections), prepares stack and heap
- Set `argv` and `argc`, jump to start of main
- Shell waits for main to return (`join`)

---

*BIOS: Basic Input Output System*
Supervisor Mode

• If something goes wrong in an application, it could crash the entire machine. And what about malware, etc.?
• The OS enforces resource constraints to applications (e.g., access to memory, devices)
• To help protect the OS from the application, CPUs have a supervisor mode (e.g., set by a status bit in a special register)
  – A process can only access a subset of instructions and (physical) memory when not in supervisor mode (user mode)
  – Process can change out of supervisor mode using a special instruction, but not into it directly – only using an interrupt
  – Supervisory mode is a bit like “superuser”
  – But used much more sparingly (most of OS code does not run in supervisory mode)
• Errors in supervisory mode often catastrophic (blue “screen of death”, or “I just corrupted your disk”)

Syscalls

• What if we want to call an OS routine? E.g.,
  – to read a file,
  – launch a new process,
  – ask for more memory (malloc),
  – send data, etc.
• Need to perform a syscall:
  – Set up function arguments in registers,
  – Raise software interrupt (with special assembly instruction)
• OS will perform the operation and return to user mode
• This way, the OS can mediate access to all resources, and devices

Agenda

• Devices and I/O
• Polling
• Interrupts
• OS Boot Sequence
• Multiprogramming/time-sharing

Multiprogramming

• The OS runs multiple applications at the same time
• But not really (unless you have a core per process)
• Switches between processes very quickly (on human time scale) – this is called a “context switch”
• When jumping into process, set timer interrupt
  – When it expires, store PC, registers, etc. (process state)
  – Pick a different process to run and load its state
  – Set timer, change to user mode, jump to the new PC
• Deciding what process to run is called scheduling

Protection, Translation, Paging

• Supervisor mode alone is not sufficient to fully isolate applications from each other or from the OS
  – Application could overwrite another application’s memory.
  – Typically programs start at some fixed address, e.g. 0xFFFFFF
• Solution: Virtual Memory
  – Gives each process the illusion of a full memory address space that it has completely for itself
  – But used much more sparingly (most of OS code does not run in supervisory mode)
  – Errors in supervisory mode often catastrophic (blue “screen of death”, or “I just corrupted your disk”)

And, in Conclusion, ...

• Basic machine (datapath, memory, I/O devices) are application agnostic
• Some concepts / processor architecture apply to large variety of applications, e.g.,
  – OS with command line and graphical interface (Linux, …)
  – Embedded processor in network switch, car engine control, …
• Input / output (I/O)
  – Memory mapped: appears like “special kind of memory”
  – Access with usual load/store instructions (e.g., lw, sw)
• Exceptions
  – Notify processor of special events, e.g. divide by 0, page fault (next lecture)
  – “Precise” handling immediately at offending instruction
• Interrupts
  – Notification of external events, e.g., keyboard input, disk or Ethernet traffic
• Multiprogramming and supervisory mode
  – Enables and isolates multiple programs
• Take CS162 to learn more about operating systems