# OS 1/23

*Reagan Shirk*

*January 23, 2020*

## Interrupts

- The interrupt handler is a set of all instructions that interrupts know how to do
  - Checks to see why the state is being interrupted
  - Checks the flags register to see what type of interrupt it is
- Pros and Cons of multiple interrupts
  - Pros
  - Cons
    - Interrupts are expensive
    - Interrupts happening at the same time
      - Multiple interrupts means you have to decide how to process them
      - You can block all other interrupts while one is happening, or allow other interrupts using some priority scheme
      - You can handle interrupts during fixed time periods or handle them immediately on arrival

## Memory

- You have the CPU
  - Registers are as close to the CPU as possible
  - Outside of the registers exists cache (memory) and its on the CPU
    - How should we design cache?
      - There are different layers of cache
      - Cache closest to the registers is L1 (level 1) cache
      - Multicore systems have L2 cache
    - Important part of cache heirarchy
      - We have cache available to us
      - We have memory available to us
    - Design constraints
      - How much do we have to spare?
      - How fast do we want to make this?
      - How many bytes do we need?
      - Small = fast = expensive
      - large = slow = cheap
- You have the main memory
  - Volatile
  - Larger than cache but not large enough for what we need
- You have the external memory
  - Persistent
  - Disk
- Cloud/Web
  - Offsite entirely
- Moving from small, fast, expensive memory down to large, slow, cheap memory creates:
  - Decrease in cost/bit
  - INcrease capacity
  - Increase access

  - Decrease frequency of access to memory
- Principle of Localization
  - Memory references by the processor tend to cluster
  - *Algorithm I'll upload*

# Operating System

- System calls allow the user mode of an operating system to talk to the kernel mode
- Data structures of the Kernel Mode:
  - Array: contiguous set of values
    - In C: Direct access to entries, `a[i]`
    - O(1) to access elements
    - O(n) to insert elements
    - Usually mapped to some contiguous set of memory
  - List: linked set of values
    - O(n) to access elements
    - No direct access to entries
  - Stack: Contiguous set of values with a predetermined order for insert and remove
    - Doesn't *have* to be contiguous, more like a *set* of values
    - Insertion/Removal: O(1)
    - Search: O(n)
    - We can take advantage of the predefined order, lets us retrace our steps
    - FILO (First in Last out)
  - Queue: Contiguous set of values with a predetermined order for insert and remove
    - FIFO (First in First out)
  - Trees/Dictionaries: Ordered set of values with a predetermined order for insert and remove
    - Append: O(1), O(n), O(logn)
    - Search: same
    - Insert: same
      - Order depends on the type of tree
    - Heirarchy
  - Bitmaps