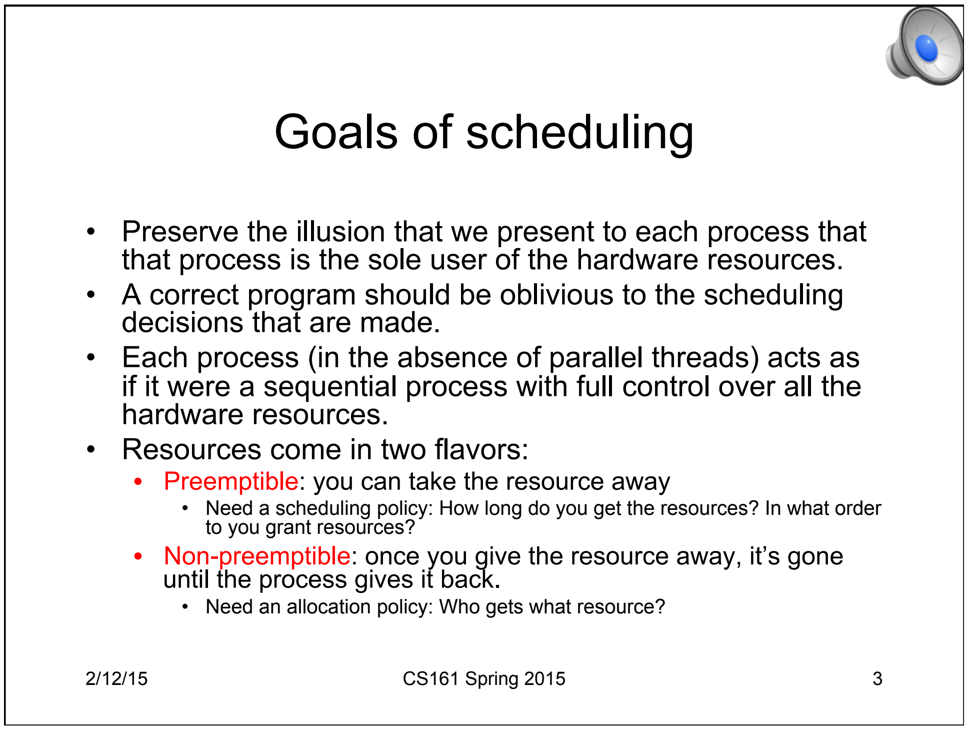
Scheduling

- Topics
  - The scheduling problem
  - Mechanism and policy
  - Approaches to scheduling
- Learning Objectives:
  - Define the scheduling problem that an operating system must solve.
  - Discuss the trade-offs between fairness and efficiency.
  - Describe several different scheduling algorithms.

The Scheduling Problem

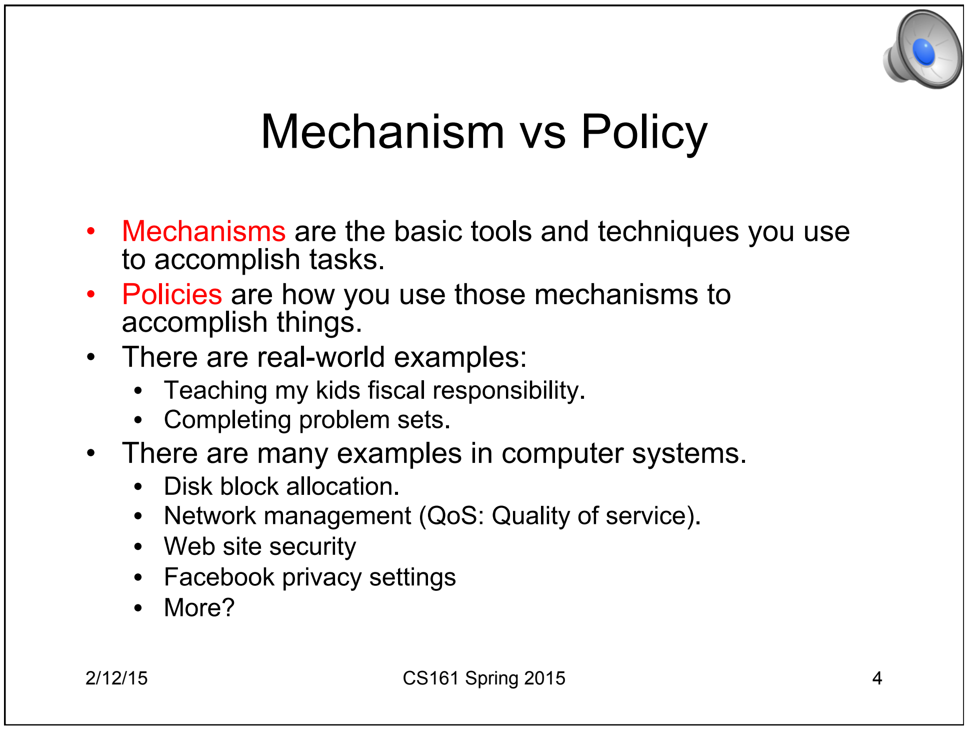
- The OS must make sure that processes do not interfere with one another. This means it must:
  - Guarantee that all processes get to run: fair scheduling or multiplexing.
  - Make sure they do not modify each other's state: protection.
- At the heart of an operating system, its functionality is simple: it is a dispatcher:
  - Let the current process run.
  - Save the current process state.
  - Load the state of another process.
  - Run the new process.
- The act of selecting a process to run is called scheduling.



Goals of scheduling

- Preserve the illusion that we present to each process that that process is the sole user of the hardware resources.
- A correct program should be oblivious to the scheduling decisions that are made.
- Each process (in the absence of parallel threads) acts as if it were a sequential process with full control over all the hardware resources.
- Resources come in two flavors:
  - Preemptible: you can take the resource away
    - Need a scheduling policy: How long do you get the resources? In what order to you grant resources?
  - Non-preemptible: once you give the resource away, it's gone until the process gives it back.
    - Need an allocation policy: Who gets what resource?

2/12/15 CS161 Spring 2015 3



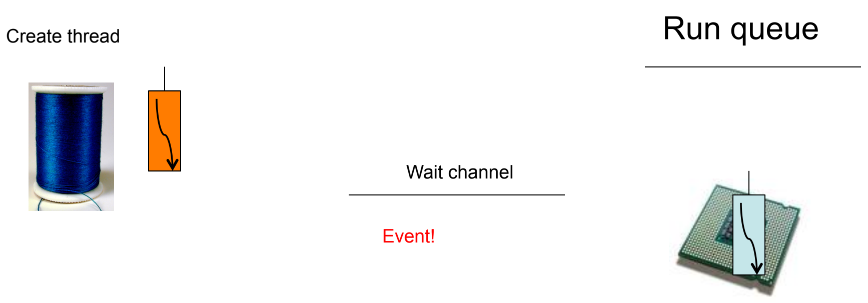
Mechanism vs Policy

- Mechanisms are the basic tools and techniques you use to accomplish tasks.
- Policies are how you use those mechanisms to accomplish things.
- There are real-world examples:
  - Teaching my kids fiscal responsibility.
  - Completing problem sets.
- There are many examples in computer systems.
  - Disk block allocation.
  - Network management (QoS: Quality of service).
  - Web site security
  - Facebook privacy settings
  - More?

2/12/15 CS161 Spring 2015 4

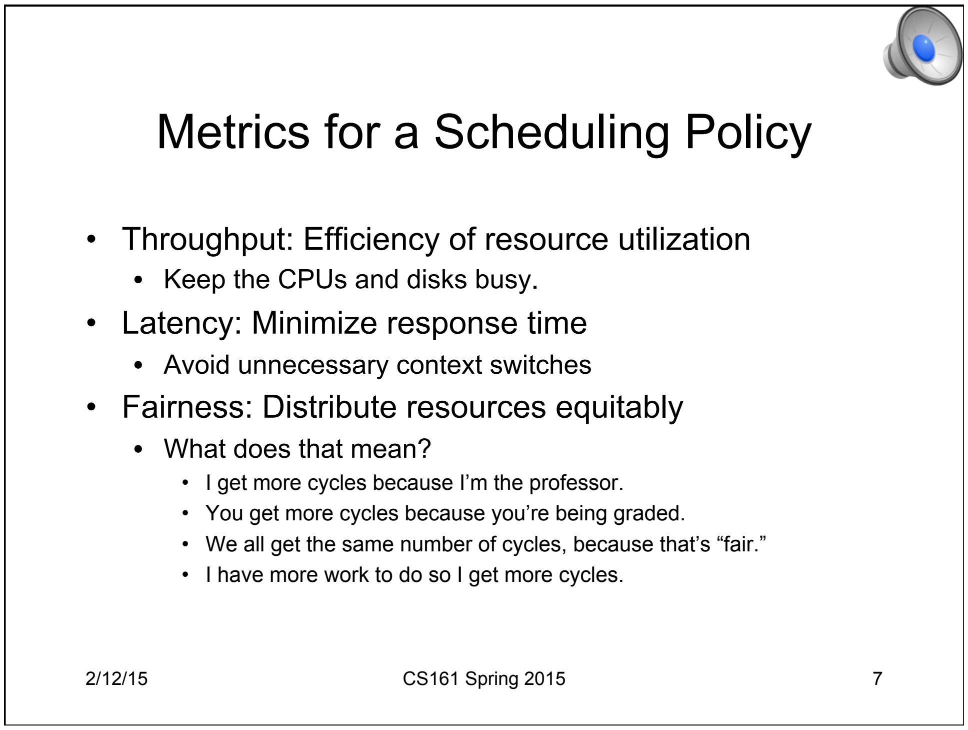
Scheduling Mechanisms

- Create thread: a way to create items to schedule.
- Run queue: a list of threads that are runnable.
- Wait channel: a list of threads that are blocked on some particular event.
- Timer interrupts: can set a CPU timer that will generate an interrupt at a particular time.



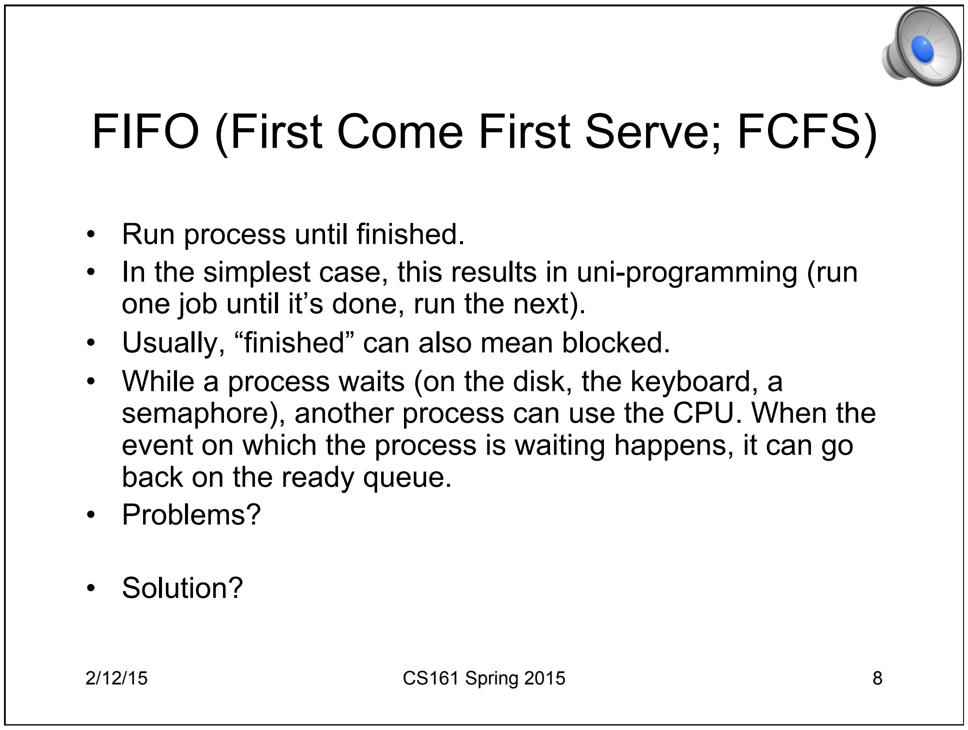
Enhancing the Mechanism

- What are some of the limitations of our existing mechanism?
  - The run queue is FIFO.
  - Threads are only de-scheduled when they voluntarily relinquish the processor.
- So, what can we add to our mechanism?
  - A more sophisticated data structure for the run queue.
  - Timers: allow us to generate an interrupt so that we can de-schedule threads.
  - Perhaps multiple run queues:
    - Perhaps have different queues for different priorities.
    - Perhaps have different queues for different processors.



Metrics for a Scheduling Policy

- Throughput: Efficiency of resource utilization
  - Keep the CPUs and disks busy.
- Latency: Minimize response time
  - Avoid unnecessary context switches
- Fairness: Distribute resources equitably
  - What does that mean?
    - I get more cycles because I'm the professor.
    - You get more cycles because you're being graded.
    - We all get the same number of cycles, because that's "fair."
    - I have more work to do so I get more cycles.



FIFO (First Come First Serve; FCFS)

- Run process until finished.
- In the simplest case, this results in uni-programming (run one job until it's done, run the next).
- Usually, "finished" can also mean blocked.
- While a process waits (on the disk, the keyboard, a semaphore), another process can use the CPU. When the event on which the process is waiting happens, it can go back on the ready queue.
- Problems?

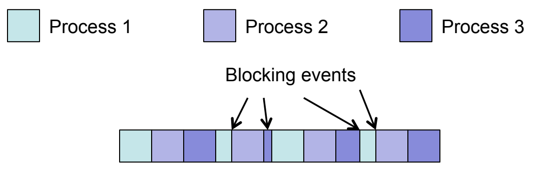
- Solution?

FIFO (First Come First Serve; FCFS)

- Run process until finished.
- Inthe simplest case, this results in uni-programming (run one job until it's done, run the next).
- Usually, "finished" can also mean blocked.
- While a process waits (on the disk, the keyboard, a semaphore), another process can use the CPU. When the event on which the process is waiting happens, it can go back on the ready queue.
- Problems?
  - One process can monopolize CPU.
- Solution?
  - Limit maximum amount of time a process can run. Call this unit a time slice.

Round Robin

- Run a process for one time slice (or until it blocks), then let another process run.
- If process blocked voluntarily, it goes into a blocked state.
- If not, process is put at the end of the ready queue.
- Each process gets approximately equal fraction of the CPU.



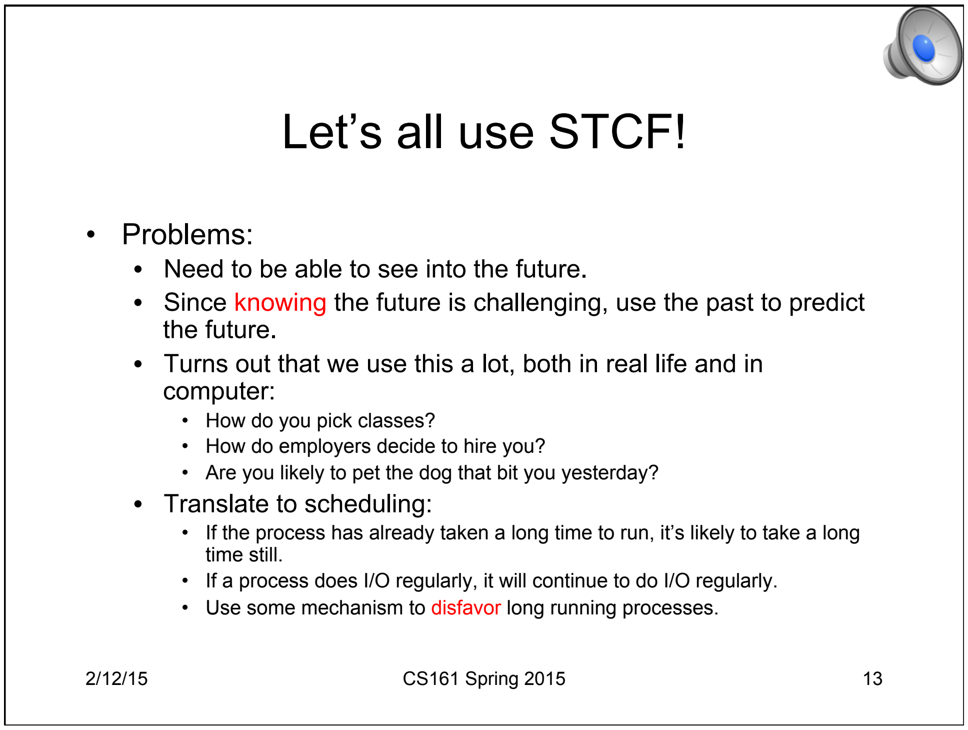
- What happens if the time slice isn't chosen properly?
  - Too short?
  - Too long?

Priority-based Round Robin

- Run highest priority first.
- Use round robin within a priority.
- When de-scheduling; put at end of queue of appropriate priority.
- Problems?
  - Round Robin sometimes produces weird results.
  - How long will it take 10 processes of 100 time slices each to complete?
  - What is the average time to completion?
  - What would it be with FCFS?
- What is the absolute best we can do?

Shortest Time to Completion First

- Assume we have total information.
  - We know the future, in particular, when a process will either exit or voluntarily block.
  - We can see all the processes all the time.
- Run the job with the shortest time to completion first (STCF).
- This will minimize the average response time.



Let’s all use STCF!

- Problems:
  - Need to be able to see into the future.
  - Since knowing the future is challenging, use the past to predict the future.
  - Turns out that we use this a lot, both in real life and in computer:
    - How do you pick classes?
    - How do employers decide to hire you?
    - Are you likely to pet the dog that bit you yesterday?
  - Translate to scheduling:
    - If the process has already taken a long time to run, it’s likely to take a long time still.
    - If a process does I/O regularly, it will continue to do I/O regularly.
    - Use some mechanism to disfavor long running processes.

2/12/15 CS161 Spring 2015 13