Threads, Synchronization, and Scheduling

Eric Wu (ericwu@cs)

Topics for Today

- Project 2
  - Due tomorrow!
- Project 3
  - Due Feb. 17th!
- Threads
- Synchronization
- Scheduling

Project 2

Troubleshooting:

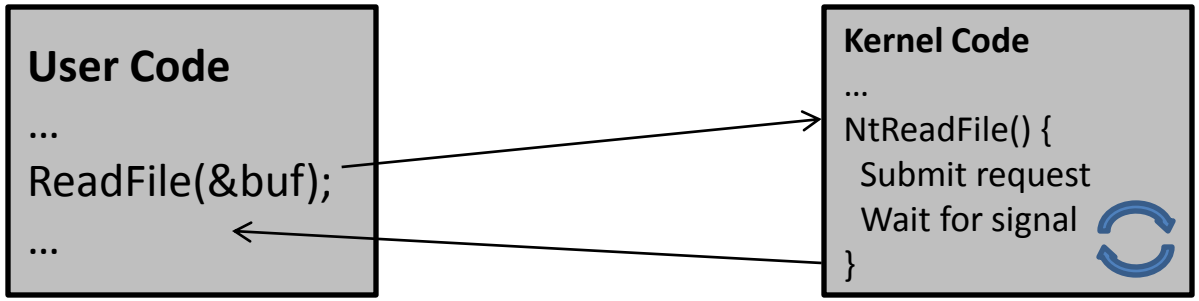
- Stock kernel doesn't run on the VM
  - Solution: do not hit reset! Use graceful shutdown.
- Variable scoping issues!
  - Solution: only declare global functions or variables **once**. All other instances of variables should be `extern`.
- Other questions?

Project 3

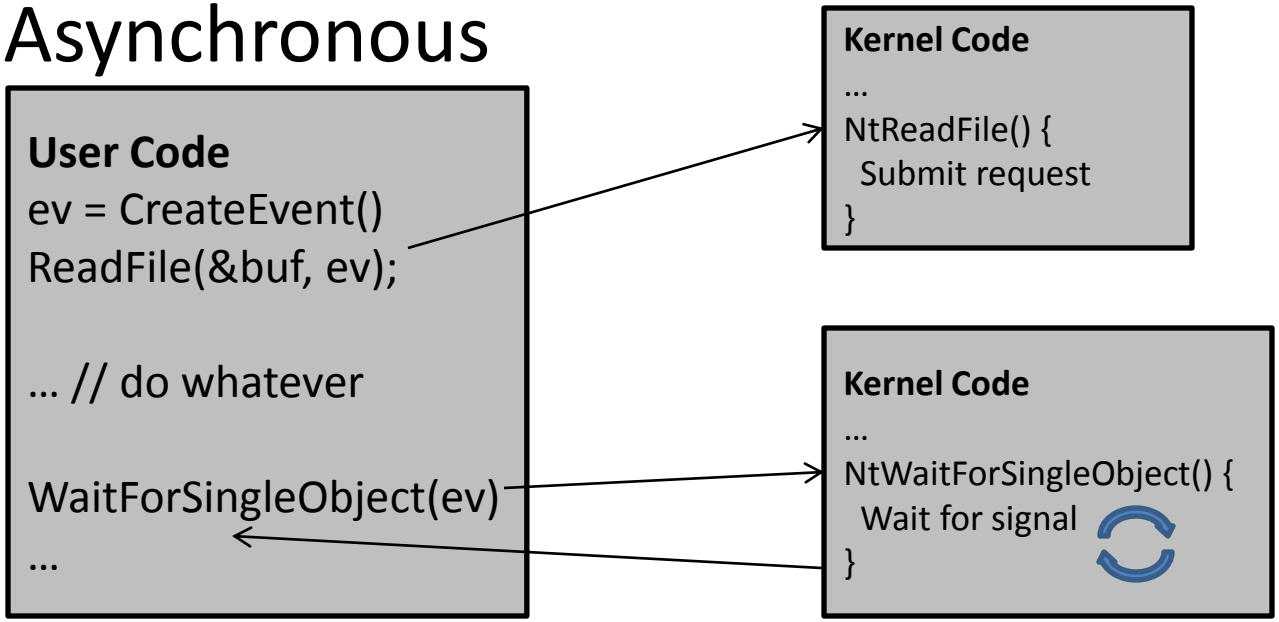
- Implement a file-copy utility
  - This is done entirely in user space. ☺ (or ☹)
- Three parts
  - Multithreaded + synchronous I/O
  - Single threaded + asynchronous I/O
  - Performance analysis of these two implementations

I/O in Windows

- Synchronous



- Asynchronous



I/O in Windows

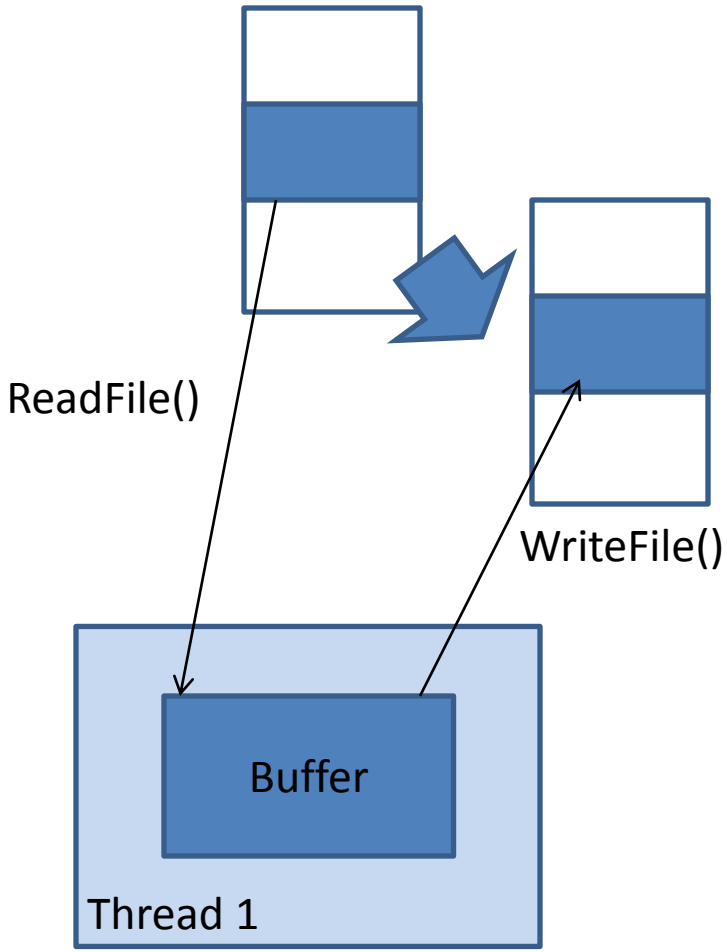
- Advantages of sync I/O?
- Advantages of async I/O?

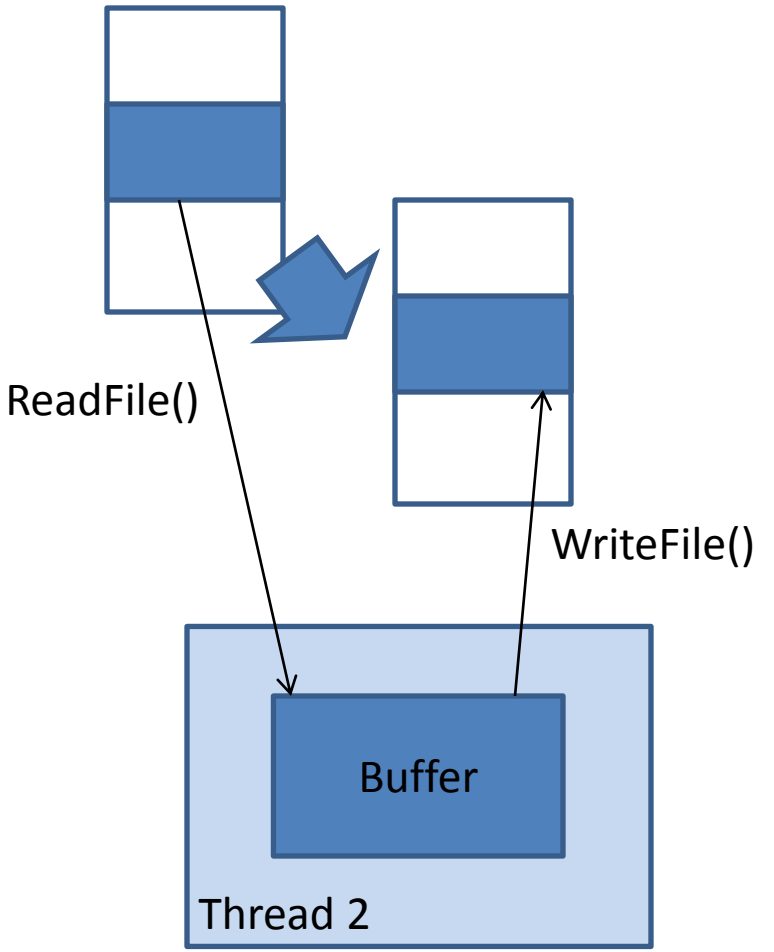
I/O in Windows

- Advantages of sync I/O?
  - Easier to program
- Advantages of async I/O?
  - Potentially more efficient
  - Can also make sync I/O more efficient with threading! How?

Project 3

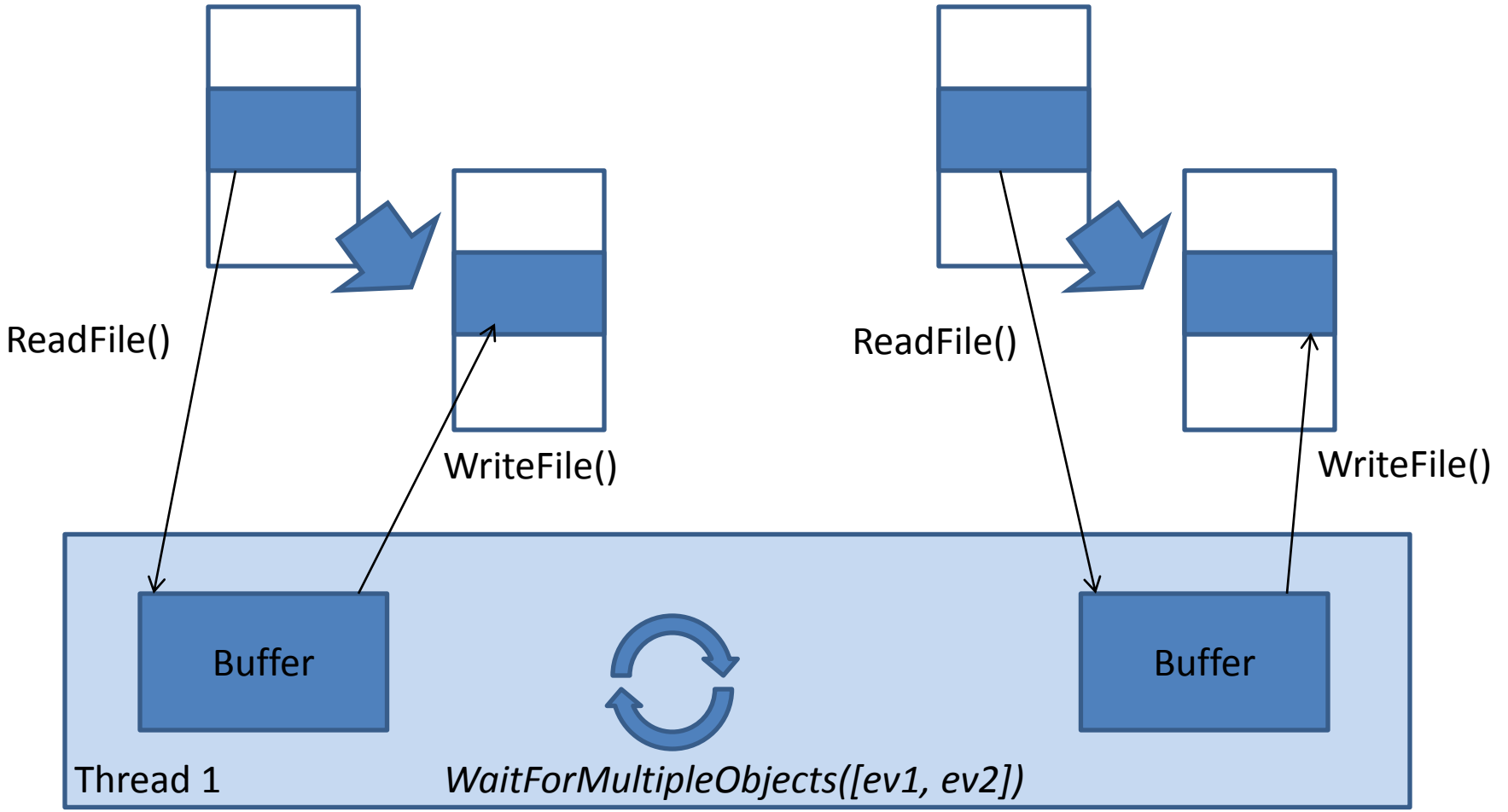
Multithreaded + Sync I/O





Project 3

Single Threaded + Async I/O



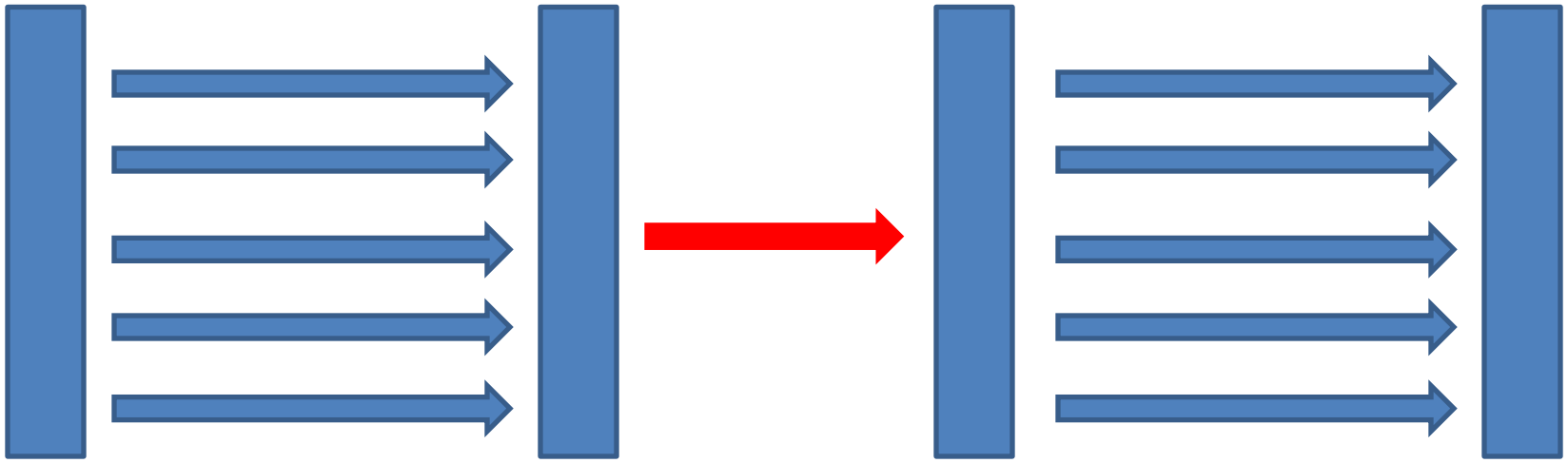
Threads

- Quick concept checks:
  - What resources in memory are shared among threads?
  - In what scenario(s) does multi-threading not perform better than single-threading?

Amdahl's Law (Abridged)

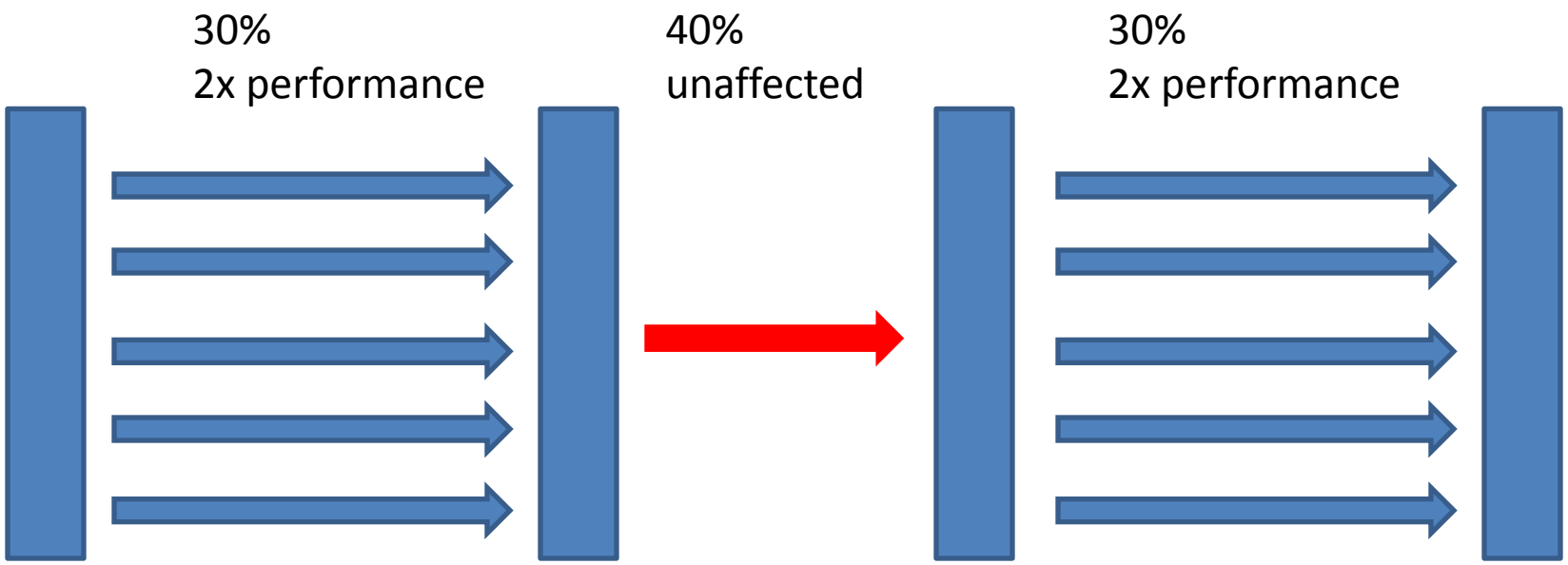
- Overall performance is given by a weighted proportion of performance increases across all segments of code.
- N_i = percent segment of the program
- P_i = performance change of that segment.

$$\sum \frac{Ni}{Pi}$$



Amdahl's Law (Abridged)

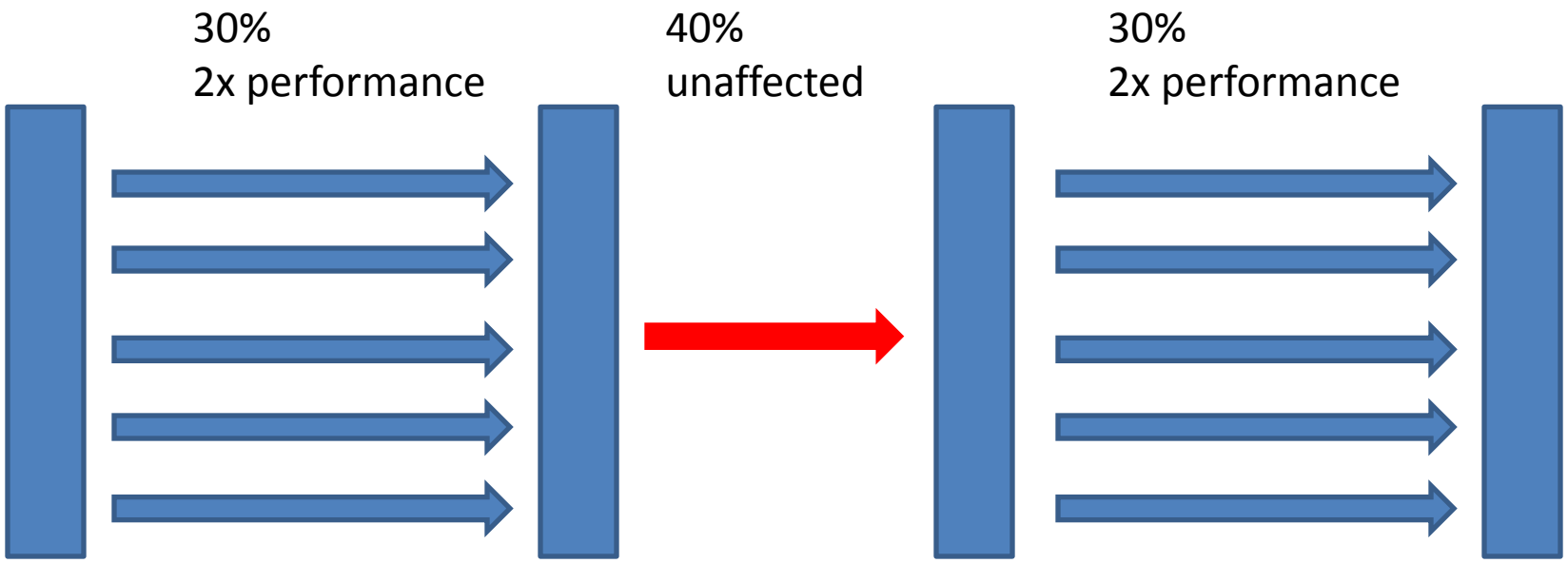
$$\sum \frac{Ni}{Pi}$$



Amdahl's Law (Abridged)

$$\frac{0.30}{2} + \frac{0.40}{1} + \frac{0.30}{2} = 0.70$$

$$\sum \frac{Ni}{Pi}$$



Synchronization

- Why do we need it?

Synchronization

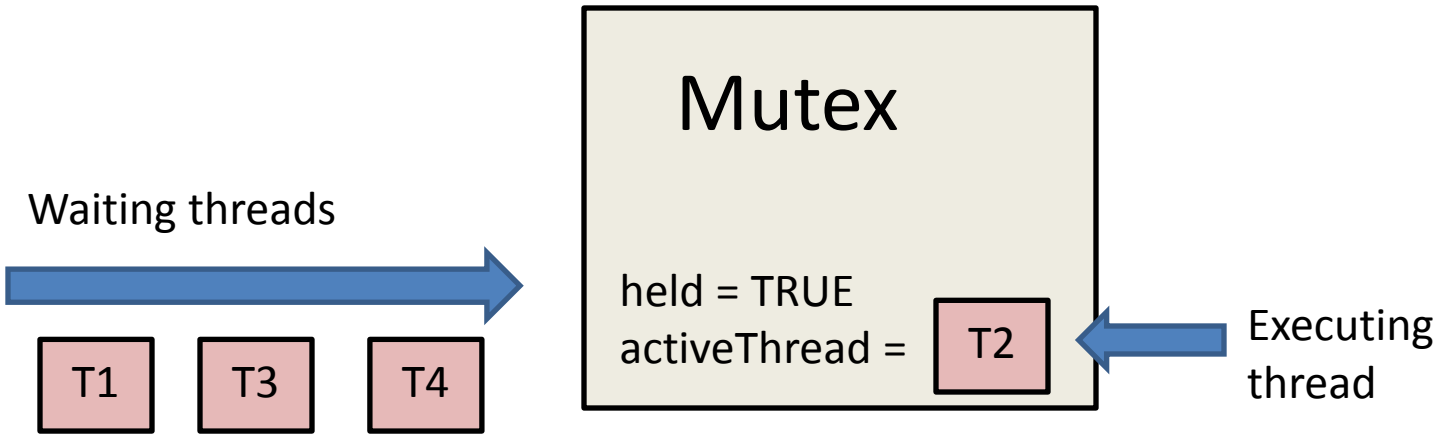
- Why do we need it?
  - Make data handling safe!
  - This was the focus of project 2

Synchronization

- Mutexes and Locks
- Semaphores
- Condition Variables
- Monitors (won’t cover in this section!)

Mutexes and Locks

- Implemented in two ways below:
  - Spinlocks
    - Busy wait (while (…) { continue; }) until lock is released.
    - Advantages and disadvantages?
  - Blocking/queueing mutexes:



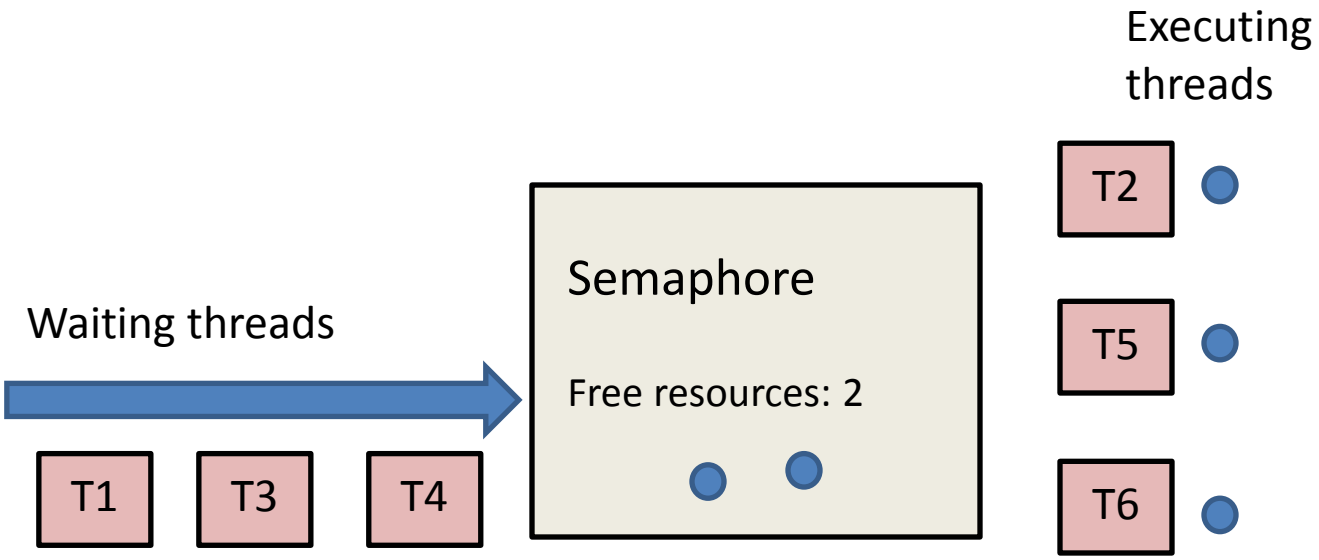
Semaphores

- Similar to locks/mutexes, but can have more than one resource.
- Operations:
  - wait (Execute thread if enough resources, else put on waiting queue.)
  - signal (Return a resource if no waiting threads, else execute a thread from waiting queue. Caller of signal also executes.)

Semaphores

- Similar to locks/mutexes, but can have more than one resource.
- Operations:
  - wait (acquire a resource)
  - signal (release a resource)

Semaphores


Executing threads
Waiting threads
Semaphore
Free resources: 2
T1
T3
T4
T2
T5
T6

Semaphores

- Benefits over mutexes and locks?
- Weaknesses?

Semaphores

- Benefits over mutexes and locks?
  - Better resource allocation!
- Weaknesses?
  - Easier to mess up
    - Forget to acquire
    - Forget to release
    - Even more difficult with non one-to-one resource to acquirer mappings

Scheduling

- Two important decisions:
  - When do I reschedule the CPU?
  - Who gets the CPU after I reschedule it?

When do I reschedule the CPU?

- Cooperative scheduling
  - Reschedule when:
    - A thread blocks on I/O
    - A thread yields()
    - A thread terminates
  - Problems?
- Preemptive scheduling
  - Reschedules at any time
  - Problems?

Who gets the CPU?

- Many algorithms for scheduling
- What are some factors to consider in scheduling?

Who gets the CPU?

- Many algorithms for scheduling
- What are some factors to consider in scheduling?
  - Not limited to: priority, waiting time, CPU utilization, average execution time, …
  - See lecture slides!