Welcome back^3 to CS439!

Quiz everyone say WHEEEEEE!

```
while (true) {
  check_feedback();
}
```

How was the quiz?

A. easy
B. mostly fine
C. mostly fine, but not enough time
D. too hard, but finished mostly in time
E. too hard and not enough time
F. too hard regardless of time

Stress

- 439H is **not an easy class**
  - Lots of new material
  - Unfamiliar programming environments
  - Fast, often relentless pace
- Struggling in this course is normal
  - There will be times you won't know the answer or solution
  - This is expected - we want everyone to succeed, but the only way we can help is if you ask for it
- If you find yourself overwhelmed or spending more time on this class than you think you should be, **please reach out** to Dr. Gheith or the TAs
  - We can help out as far as the class goes
  - We can provide other resources if we are not able to help

Mental health resources available at UT

P3

```
check_feedback([]
    (auto feedback) {
        ASSERT(
            feedback.max() != 'A'
        );
    }
}
```

How is p3 going?

A. that’s a thing?
B. Cloned the project.
C. Looked through the starter code.
D. Started planning/writing code
E. Done with at least one part of the project
F. Done with the whole project but still failing a couple test cases
G. p3 speedrun glitchless
H. passing t0

Why so many synchronization primitives?

- Imagine that we want to implement the core synchronization part as few times as possible
    - i.e. scheduling callbacks properly, queueing callbacks for later, etc.
- What fundamental primitives could we use to achieve this?
- Given synchronization primitive x, could you use it to easily implement y?

Semaphores!

- What is a semaphore?

Semaphores!

- What is a semaphore?
  - An example of a **universal synchronization primitive**
  - All the things you made in p2 can be done in terms of this!
    - (this is p3)
  - Contains a single counter representing how many people can use the semaphore before being forced to wait
  - Initialization: the counter is set to some integer value
  - `down(work)`:
    - When the counter is greater than 0, decrement the counter and schedule `work`
    - Does not schedule `work` or do anything else until the counter is positive
  - `up()`:
    - Increments the counter

How can we use a semaphore?

Let’s build a simple lock:

```
Semaphore sem{1};
lock(Work work) {
    sem.down(work);
}
unlock() {
    sem.up();
}
```

How can we use a semaphore?

How can I change this lock to allow 2 people to run at once?

```
Semaphore sem{1};
lock(Work work) {
    sem.down(work);
}
unlock() {
    sem.up();
}
```

How can we use a semaphore?

How can I change this lock to allow 2 people to run at once?

```
Semaphore sem{2};
lock(Work work) {
    sem.down(work);
}
unlock() {
    sem.up();
}
```

A note on throughput

Which one of these locks is better?

```
Semaphore sem{1};
lock(Work work) {
    sem.down(work);
}
unlock() {
    sem.up();
}
```

```
Atomic<bool> taken{false};
lock() {
    while (taken.exchange(true)) {}
}
unlock() {
    taken.store(false);
}
```

A note on throughput

Which one of these locks is better?

- A **spinlock** (right), well, *spins*/burns CPU cycles while waiting for the lock to be available
    - Useful if we expect the critical section to be really short - the overhead of switching to another task (and back later) might be higher than simply waiting for a bit
- A **blocking lock*** (left) will block* the task from running until the critical section is ready for it
    - Useful for longer critical sections where burning milliseconds of CPU time is just a waste

*p2/p3 doesn't have blocking in the traditional sense where a thread's execution is suspended and fully context switched out of. Instead we just don't let the task associated with the critical section run.

Bonus: Monitors

- Monitors are **mutexes** (locks) + **condition variables**
- Condition variables support two main operations:
  - `wait`: Waits for the condition variable to be `signal`led
  - `signal/notify`: Schedules any tasks that are `wait`ing
- Is this as powerful as a semaphore?

Questions?

credit to Meyer Zinn for the meme