Today

Conditional variables!
Why use them?

• Useful when you only want to enter a critical section under certain conditions.
• Avoid “busy waiting”, where a single thread wastes time repeatedly checking for a condition while waiting for it to become true. (“Are we there yet? Are we there yet? Are we there yet? Are we there yet? Are we there yet? Are we there yet? Are we there yet? Are we there yet? Are we there yet? Are we there yet? Are we there yet? Are we there yet?..
API

• To create a conditional variable:
  
  ```
  pthread_cond_t condvar = PTHREAD_COND_INITIALIZER;
  ```

• To wait for a conditional variable:
  
  ```
  pthread_cond_wait(&condvar, &mutex);
  ```

  This will unlock the `mutex`, wait for a signal `condvar`, then try to lock the `mutex` when the signal arrives.
API

To signal a conditional variable:

```c
int pthread_cond_broadcast(pthread_cond_t *cond);
```
Unblock all threads blocks on `cond`.; one is able to lock `mutex` again.

```c
int pthread_cond_signal(pthread_cond_t *cond);
```
Unblock at least 1 thread blocked on `cond`.; it locks `mutex`
Example: Producer-Consumer Problem

• Consumer has to wait while the buffer is empty, producer has to wait while buffer is full.
• busywait.c contains a busy wait:
  Consumer locks buffer, consumes if item, unlocks buffer
  Producer locks buffer, produces if space, unlocks buffer
Example: Producer-Consumer Problem

condvar.c shows how to fix this with conditional variables:

**Consumer** locks buffer, if empty unlock and wait to hear buffer_full, then locks and consumes.

**Producer** locks buffer, if full unlock and wait to hear buffer_empty, then locks and produces.
Try it

Can you modify condvar.c for:
- Multiple consumers
- Multiple producers
- Broadcast vs. Signal on the conditional variables?
Questions?