Next: condition variables, reader/writer locks, and more joy

We're going to do this one live, but I don't have the pthreads library memorized, so here goes:

pthread_cond_t = PTHREAD_COND_INITIALIZER

pthread_cond_wait(cv, mutex) //remember: releases the lock.

pthread_cond_signal(cv)

The point here: if one thead needs to do something, and another thread needs to wait for the first thread to finish doing that thing, then we need a notification framework!!

Talk about the spurious-wakeup thing. Imagine with the class how we can re-use the basics from futex or park/unpark to implement condition variables.

Do a bit of livecoding to figure out how to use these in an example; ask the class for situations where we might want condition variables, hypothetical pieces of code, and then *code live* and live dangerously

(I cannot find the code that we did in class!! Sorry all. I'm looking :))

Semaphores

The "wait" one is P(), the "post" one is V(). Call them sem_wait and sem_post.

You call V() to say "a new thread should be allowed to enter" and you call P() to say "I am a thread who wishes to enter".

Concretely: the semaphore is initialized to a certain number. If that number is negative *after calling P*, then the thread that called P waits for the semaphore to become positive. V() just increments the semaphore.

(do a couple of examples of calling / releasing patterns with thread schedules on the board, as you usually do with this sort of thing)

Let's take a look at one of the examples from before, perhaps including the one we just came up with a few minutes ago, and see how we might be able to implement it with semaphores instead of CVs. ***A question: what value should we set the semaphore to?***

(Note: you *can* use semaphores directly in your code---in fact, in POSIX they're still the preferred way to synchronize across *processes* and not just across threads---but usually you won't. Condition Variables plus Mutexes, maybe with a sprinkling of reader-writer locks, is really all you need.)

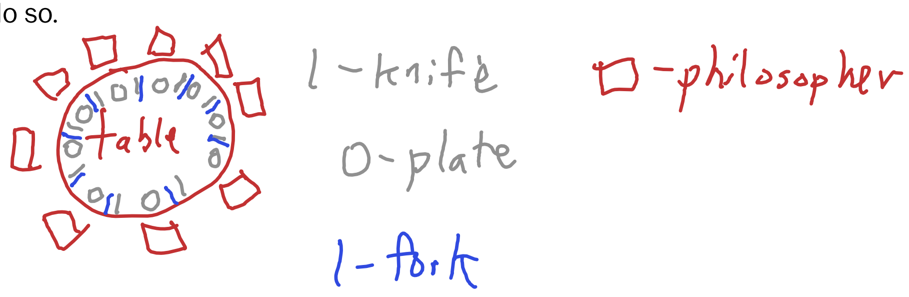
Ok! It turns out, semaphores are a general primitive that can cover *all* sorts of locking cases! In particular, we can use semaphores to *implement* a mutex. How might we do this? Any ideas? (it's just about the initial value)

Ok, reader-writer locks! You can explain how reader-writer locks work, basically. Draw on the board the reader lock, and the writer lock; illustrate a schedule of several threads getting reader locks, and then *blocking those threads* until we acquire a writer-lock.

Let's ask the class for an example of a scenario where we might want to have a reader-writer lock, just like we did with the condition variables. Hopefully they can come up with one.

It turns out, semaphores are general enough to *also* implement these! Let's see if we can come up wit *how* we can use semaphores to implement reader-writer locks. Hint: an easy way to do this involves two semaphores, but that's not the only way. Another hint: unlike a mutex, the thread that "releases" a binary semaphore doesn't *need* to be the same thread that "acquires" it! Final hint: if we care about fairness, we might want *three* semaphores.

(If time allows): let's talk about Deadlock. In particular, let's use the dining philosophers to do so.



What happens if two philosophers try to grab the same fork?