Uthreads
Overview

- Write your own user-level threads package
- A simple version of pthreads
- Write a scheduler - which threads run when
- Write some synchronization devices - mutexes, condition variables
Context Switching

A thread has decided to yield to another thread or is preempted

1. Saves current state to a context
2. Saves the context to memory
3. Swaps out its context for the new thread’s context

A context includes args, eip/ebp, saved registers and local variables
Context Switching
Scheduler

- Uses priority scheduling
- Threads run until they either:
  - yield* (placed back in the runnable queue)
  - block (put on a waiting queue)
- In uthread_switch, determine which thread should run next and finally call support code method uthread_swapcontext which actually switches the two threads' contexts
- *threads may yield voluntarily, or yield when their time slice runs out
Mutexes

- uthread_mtx_lock - lock an unlocked mutex
  - If locked, put current thread on the mutex wait queue
- uthread_mtx_trylock - try to lock a locked mutex
  - Should not block
- uthread_mtx_unlock - wake up next thread in the mutex wait queue and schedule it by moving it to the runnable queue
Condition Variables

- Waiting on a condition variable
  - Lock/unlock mutex
  - Enqueue thread
- Broadcasting
  - Wake and dequeue all threads blocked on condition
- Signaling
  - Wake and dequeue top thread blocked on condition
Reaper

- We create the reaper for you
- Reaper removes any threads that have either:
  - detached
  - have already been joined with after they finish execution
- Make sure to make threads reapable where appropriate
Testing

- We've given you a starting point for your tests in test.c
- Creates 10 threads; each thread locks a mutex and waits on a condition variable
- Each other thread signals the previous
- Test is done when all threads have exited the loop and printed out their status
- main joins with all threads
Testing cont.

- Make sure to test priority-based scheduling - threads with different priorities where they are not added in order
- Test with > 10 threads
- Write some simple multi-threaded programs with shared state
Tips

- For error handling, look at analogous pthreads_* functions man pages for examples of possible errors
  - Also if you don’t exactly remember how pthreads works
- `printf(3)` is not thread-safe!
- GDB will not show each uthread as a separate thread!
- Core dump debugging -
  - `uname -c unlimited` will cause programs to dump core when they segfault; then you can do
Tips

● Name of the game: thinking concurrently
● Don’t have to write a lot of code
● **Always** need to be thinking about
  ○ Am I modifying shared state, or only state on my stack?
  ○ **Could I be preempted here?**
  ○ **What bad things could go wrong?**
● What are some examples of shared state?

“Shared mutable state is the root of all evil.” — somebody smart

(lots of shared state in this course)
Plan of Attack

- uthread.c: resource allocation code, basic functions - start here
- uthread_sched.c: scheduling functions
- uthread_mtx.c, uthread_cond.c: finally, implement mutexes and condition variables