CS 45: Operating Systems

Lab 4: Threads in Xv6

DUE November 2nd at 11:59 PM

In this lab we continue working with `xv6` by building a user-space threading API & implementation. Write your answers in `README.md`. Create a `COLLAB.md` file to keep track of any outside resources you might use.

1 User-Space Threads (Non-preemptive)

Complete the `Uthread` task from the *threading lab* from MIT. Write a sentence or two about what you were able to accomplish. Chapter 8, the reading for today that describes how context switching works, is very helpful!

2 Alarm (Signals)

Complete the `Alarm` task from the *traps lab* from MIT. Write a sentence or two about what you were able to accomplish. Chapter 4 describes how trampolines and trapframes work when going between user-space and kernel-space.

3 Preemptive Threading

Building on the first two-parts, use the `alarm` to automatically schedule your threads! Make a copy of `user/uthread.c` called `user/uthread_preempt.c` that doesn't need `yield`. Do you see any problems??

4 Locks and Context

To accomplish the previous part, we need to do a little more work.

1. Create a new system call `alarm_getcontext(struct *context)` that retrieves the context of the process when the `alarm` was raised and the process was interrupted.

   You will need use the `copyout` function, similar to how `filestat` fills in a structure.

2. Create `lock(struct tlock *lock)` and `unlock(struct tlock *lock)` functions, and the associated `struct tlock` data structure.

3. Write a `thread_exit` function that threads call when they are finished. It should set `t->state = FREE` and wait to be descheduled.

4. Think about when the alarm signal should be disabled and enabled, and where you might need to employ locking in your user-space thread system.

5. OPTIONAL: Rather than using a fixed number of threads, allow an arbitrary number to be created.