Lab 6 - Uthread

User level threads

- Threads can be implemented entirely at the user level
- Your job for this lab: complete user level threads in xv6
  - Lots of code already written (scheduler, thread struct)
- Need to implement:
  - Thread creation
  - Context switching [& associated thread metadata]

Thread creation

```
void
thread_create(void (*func)())
```

Need to somehow save the pointer to the function passed in (run when thread is scheduled)

Set up correct metadata for thread

Context switch

Context switch: Switching from one thread of execution to another

- Need to consider saving registers (only callee---why?)
- Adjusting stack pointer (threads have their own stacks, see thread struct)
- Going back to where the thread was previously executing

Callee saved registers

s0-s11; should be saved in a context switch

Context switches can be treated as another function call --- so, it is the callee's responsibility to *restore* these registers once you "return" from the context switch (i.e., when the thread being switched away from is scheduled again).

ra register

- RISC-V register used for storing return address
- ret is equivalent to jumping to what’s stored in ra

It is caller saved, but it might be useful to look into using it to manage pointing threads to the correct instruction.

ra register continued

Threads call thread_yield to give themselves up to the scheduler, which will then call thread_schedule (which should then call your context switch routine eventually).

When calling your thread_switch code, ra will point to the code after thread_switch in thread_schedule. Should you jump there in a context switch? Is that old ra value meaningful to you/will it be meaningful later on?

Assembly instructions/registers to keep in mind

- sd reg, addr
  - stores register in memory (all 64 bits)
  - Example: sd a0, (8)sp -> puts contents of a0 register into sp + 8
- ld reg, addr
  - loads 64 bit word at addr into register
- a0,a1
  - First two argument registers (useful for context switch)