1. User-level threads and swtch()

We'll study this in the context of user-level threads.

Per-thread state in thread control block:

```c
typedef struct tcb {
    unsigned long saved_rsp;    /* Stack pointer of thread */
    char *t_stack;         /* Bottom of thread's stack */
    /* ... */
};
```

Machine-dependent thread initialization function:

```c
void thread_init(tcb **t, void (*fn) (void *), void *arg);
```

Machine-dependent thread-switch function:

```c
void swtch(tcb *current, tcb *next);
```

Implementation of swtch(current, next):

```c
# gcc x86-64 calling convention:
# on entering swtch():
#  register %rdi holds first argument to the function ("current")
#  register %rsi holds second argument to the function ("next")

# Save call-preserved (aka "callee-saved") regs of 'current'
pushq %rbp
pushq %rbx
pushq %r12
pushq %r13
pushq %r14
pushq %r15

# Store old stack pointer, for when we swtch() back to "current" later
movq (%rdi), %rsp
# %rsp = %rdi->saved_rsp

# Restore call-preserved (aka "callee-saved") regs of 'next'
popq %r15
popq %r14
popq %r13
popq %r12
popq %rbx
popq %rbp

# Resume execution, from where "next" was when it last entered swtch()
ret
```

2. Example use of swtch(): the yield() call.

A thread is going about its business and decides that it’s executed for long enough. So it calls yield(). Conceptually, the overall system needs to now choose another thread, and run it:

```c
void yield() {
    tcb* next = pick_next_thread(); /* get a runnable thread */
    tcb* current = get_current_thread();
    swtch(current, next);
    /* when 'current' is later rescheduled, it starts from here */
}
```

3. How do context switches interact with I/O calls?

This assumes a user-level threading package.

The thread calls something like "fake_blocking_read()": This looks to the _thread_ as though the call blocks, but in reality, the call is not blocking:

```c
#define EAGAIN -1

int fake_blocking_read(int fd, char* buf, int num) {
    int nread = -1;
    while (nread == -1) {
        /* this is a non-blocking read syscall */
        nread = read(fd, buf, num);
        if (nread == -1 && errno == EAGAIN) {
            /* read would block, so let another thread run */
            /* and try again later (next time through the loop). */
            yield();
        }
    }
    return nread;
}
```