Each 6.1800 lecture will come with an outline. You can fill this in during lecture, after lecture, or not at all — it's entirely up to you how you use it. The goal of these outlines is to help you understand the main points that you should be taking away from each lecture. In some cases we will also include examples of things you should be able to do after each lecture.

*In the past, these outlines have proved to be an effective tool for studying for the exams. Note that the outlines are **not exhaustive**; there will be topics and nuances in lecture that aren't captured by the outline.*

Lecture 05: Threads

To test your understanding: after this lecture, you should be able to explain the code for yield(), yield_wait(), wait(), and notify().

- What is virtualization? *(Same first question, still! It's important)*
- What does it mean to "suspend" a thread? To "resume" it?
- What is the purpose of `yield()`? How does it work?
- What does it mean if a thread is `RUNNABLE`? `RUNNING`?
- Why does saving the current state[1] of a thread amount to saving its page-table register and stack pointer?
- What are condition variables for? What problem do they solve?
- We added calls to `wait()` and `notify()` to our bounded-buffer send (and receive) code. Why?
- What is the "lost notify" problem?
- How does `wait()` work? `notify()`?
- The ideas behind `yield_wait()` and `yield()` are similar. Why do we need `yield_wait()` (i.e., why didn't just `yield()` suffice)?
- What is preemption? What problem does it solve?

[1] By "current state" we mean memory, stack, program counters, registers, etc. We're not talking about the state of being RUNNABLE, RUNNING, etc. This is admittedly an abuse of terminology.