An Introduction

- thread
  -

    Figure 26.1: Single-Threaded And Multi-Threaded Address Spaces
- Why Use Threads?
  - parallelism
  - avoid blocking program progress due to slow I/O
- threads interact when they access shared data
  - A critical section is a piece of code that accesses a shared variable (or more generally, a shared resource) and must not be concurrently executed by more than one thread
- The Wish For Atomicity
  - synchronization primitives
- One More Problem: Waiting For Another

Thread API

- Thread Creation — pthread_create
- Thread Completion — pthread_join wait for a thread to complete
- Locks — pthread_mutex_lock, pthread_mutex_unlock
- Condition Variables — pthread_cond_wait, pthread_cond_signal
  - Condition variables are useful when some kind of signaling must take place between threads, if one thread is waiting for another to do something before it can continue.
  - To use a condition variable, one has to in addition have a lock that is associated with this condition.

Locks

- Locks: The Basic Idea — A lock is just a variable
- Pthread Locks — The name that the POSIX library uses for a lock is a **mutex**
- Building A Lock — To build a working lock, we will need some help from the hardware and the OS
- Evaluating Locks
  - whether the lock does its basic task
  - fairness
  - performance

Lock implementation

- Controlling Interrupts — disable interrupts for critical sections; this solution was invented for single-processor systems.
- A Failed Attempt: Just Using Loads/Stores
  -
    ```
    // spin-flag.mac: use of "flag" to signal
    // when waiting for another thread
    // to set flag to 0 or something, i.e. wait
    //  while(flag == 1);
    ...
    .text
    .globl thread0, thread1
    thread0:                ..@F0 ..@F0 ..@F0 
    ```