Types of Locks

- Spinlock: spins on the CPU while the lock is busy
  - wastes the CPU
  - why use this at all on a single core system?
  - sometimes you can't block! (interrupt handler context)
  - still good for short critical section

- Sleeplock: blocks/sleeps while the lock is busy
  - context switch overhead
  - locks used by also disables interrupts
Lock Granularity

- How much shared data should a lock protect?
- A single lock for an entire array (coarse grained)
  - Does provide safe access, simple, easy to perform multi-entries ops.
  - No concurrent access to the array
- One lock per array entry (fine grained)
  - Allows for concurrent independent ops on each entry
  - Higher locking overheads, easier to get deadlocks

- Can go finer (protect part of a struct)

- PCB
  - PID
  - Only accessed by the process

- Scheduling state
  - Accessed by processes (exit, wait)
  - Scheduler

- Disable interrupts
  - Avoids preemption
  - Provides mutual exclusion on a single core
  - Privileged instr. not general
*k inode case study*

```c
struct {
    struct spinlock lock;
    struct inode inode[NINODE];
    struct inode inodelfile;
} icache;
```

```c
// in-memory copy of an inode
struct inode {
    uint dev;  // Device number
    uint inum; // Inode number
    int ref;   // Reference count
    int valid; // Flag for if node is valid
    struct sleeplock lock;
}

// copy of disk inode (see fs.h for details)
short type;
short devid;
uint size;
struct extent data;
```

```c
struct inode *idup(struct inode *ip) {
    acquire(&icache.lock);
    ip->ref++;
    release(&icache.lock);
    return ip;
}
```

Spinlock protects ref field of every single inode struct.

```c
int concurrent_readi(struct inode *ip, char *dst, uint int retval;
locki(ip);
retval = readi(ip, dst, off, n);
unlocki(ip);
```
Monitors

- Design pattern & synchronization construct that coordinate threads based on events

- A monitor = a lock + resource state(s) + condition variables

  - Manages waiters of a condition
  - Wait \[XK: \text{sleep}\]
    - Put the calling thread to waiter list
    - Blocks the thread & releases the lock atomically
    - Reacquires the lock & then returns upon unlock
  - Signal
    - Wake up a waiter (Blocking \(\rightarrow\) Ready)
    - Remove from the waiter list
  - Broadcast \[XK: \text{wake}\]
    - Wake up all waiters, used when the condition may enable multiple waiters (e.g., \(N\) threads need to wake up at time \(x\))
Basic Pattern

MESA Monitor: has this semantics, a new thread may acquire
the lock before the lock-up occurs.

Heavy monitor: the make-up reader is guaranteed to lock next (other signals are
interrupted).