Uniprocessor Lock Implementation

```
struct lock {
    int locked;
    struct queue q;
};

void lock_acquire(
        struct lock *l) {
    intr_disable();
    if (!l->locked) {
        l->locked = 1;
    } else {
        queue_add(&l->q,
            thread_current());
        thread_block();
    }
    intr_enable();
}
```

```
void lock_release(
        struct lock *l) {
    intr_disable();
    if (queue_empty(&l->q) {
        l->locked = 0;
    } else {
        thread_unblock(
            queue_remove(&l->q));
    }
    intr_enable();
}
```

Locks for Multi-Core, v1

```
struct lock {
    int locked;
};

void lock_acquire(
        struct lock *l) {
    while (swap(&l->locked, 1)) {
        /* Do nothing */
    }
}
```

```
void lock_release(
        struct lock *l) {
    l->locked = 0;
}
```

Locks for Multi-Core, v2

```
struct lock {
    int locked;
    struct queue q;
};

void lock_acquire(
        struct lock *l) {
    if (swap(&l->locked, 1)) {
        queue_add(&l->q,
            thread_current());
        thread_block();
    }
}
```

```
void lock_release(
        struct lock *l) {
    if (queue_empty(&l->q) {
        l->locked = 0;
    } else {
        thread_unblock(
            queue_remove(&l->q));
    }
}
```

Locks for Multi-Core, v3

```
struct lock {
    int locked;
    struct queue q;
    int sync;
};

void lock_acquire(
        struct lock *l) {
    while (swap(&l->sync, 1)) {
        /* Do nothing */
    }
    if (!l->locked) {
        l->locked = 1;
        l->sync = 0;
    } else {
        queue_add(&l->q,
            thread_current());
        l->sync = 0;
        thread_block();
    }
}
```

```
void lock_release(
        struct lock *l) {
    while (swap(&l->sync, 1)) {
        /* Do nothing */
    }
    if (queue_empty(&l->q) {
        l->locked = 0;
    } else {
        thread_unblock(
            queue_remove(&l->q));
    }
    l->sync = 0;
}
```

Locks for Multi-Core, v4

```
struct lock {
    int locked;
    struct queue q;
    int sync;
};

void lock_acquire(
        struct lock *l) {
    while (swap(&l->sync, 1)) {
        /* Do nothing */
    }
    if (!l->locked) {
        l->locked = 1;
        l->sync = 0;
    } else {
        queue_add(&l->q,
            thread_current());
        thread_current()->state = BLOCKED;
        l->sync = 0;
        reschedule();
    }
}
```

```
void lock_release(
        struct lock *l) {
    while (swap(&l->sync, 1)) {
        /* Do nothing */
    }
    if (queue_empty(&l->q) {
        l->locked = 0;
    } else {
        thread_unblock(
            queue_remove(&l->q));
    }
    l->sync = 0;
}
```

Locks for Multi-Core, v5

```
struct lock {
    int locked;
    struct queue q;
    int sync;
};
void lock_acquire(
        struct lock *l) {
    intr_disable();
    while (swap(&l->sync, 1)) {
        /* Do nothing */
    }
    if (!l->locked) {
        l->locked = 1;
        l->sync = 0;
    } else {
        queue_add(&l->q,
            thread_current());
        thread_current()->state = BLOCKED;
        l->sync = 0;
        reschedule();
    }
    intr_enable();
}
```

```
void lock_release(
        struct lock *l) {
    intr_disable();
    while (swap(&l->sync, 1)) {
        /* Do nothing */
    }
    if (queue_empty(&l->q) {
        l->locked = 0;
    } else {
        thread_unblock(
            queue_remove(&l->q));
    }
    l->sync = 0;
    intr_enable();
}
```