Mutex Locks

Too Much Milk with Locks

Thread A

```
1   lock.acquire();
2   if (noMilk) {
3       buy milk;
4   }
5   lock.release();
```

Thread B

```
1   lock.acquire();
2   if (noMilk) {
3       buy milk;
4   }
5   lock.release();
```

Implementing Locks: Interrupts (version 1)

```
class Lock {
  public:
    void acquire();
    void release();
}
```

```
Lock::acquire() {
  disable interrupts;
}
```

```
Lock::release() {
  enable interrupts;
}
```

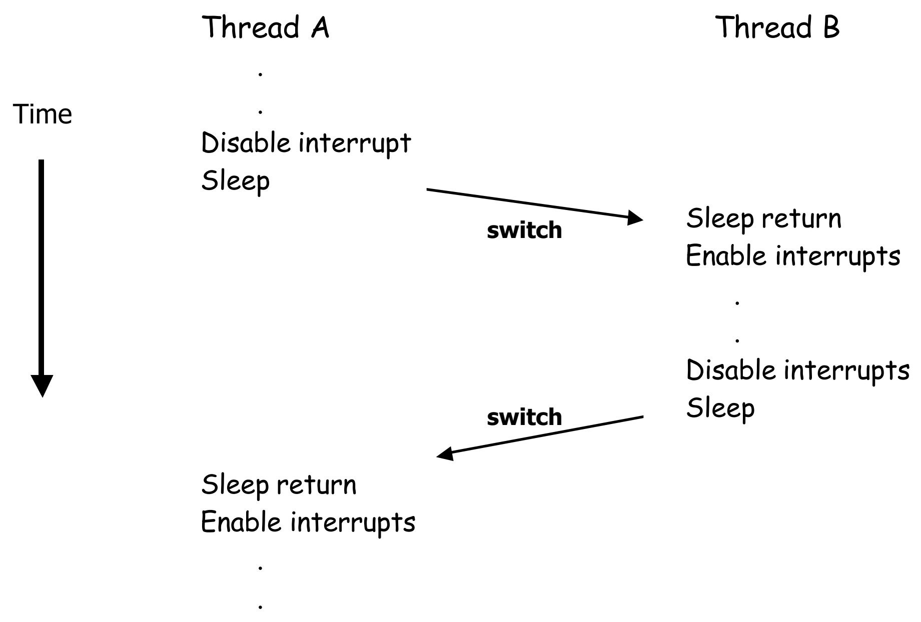
Implementing Locks: Interrupts (version 2)

```
class Lock {
  public:
    void acquire();
    void release();
  private:
    int value = FREE;
    Queue Q = empty;
}
```

```
Lock::acquire() {
  disable interrupts;
  if (value == BUSY) {
     add curThread to Q;
     put curThread to sleep;
  } else {
     value = BUSY;
  }
  enable interrupts;
}
```

```
Lock::release() {
  disable interrupts;
  if queue not empty {
     take thread T off Q;
     put T on ready queue;
  } else {
     value = FREE;
  }
  enable interrupts;
}
```

Interrupt Disable/Enable Pattern



Implementing Locks: Atomic Test&Set

```
class Lock {
  public:
    void acquire();
    void release();
  private:
    int value = FREE; // FREE = 0
                      // BUSY = 1
}
```

```
Lock::acquire() {
  while (test&set(value) == BUSY) {
    // do nothing
  }
}
```

```
Lock::release() {
  value = FREE;
}
```

Minimizing Busy-Waiting

```
class Lock {
  public:
    void acquire();
    void release();
  private:
    int value = FREE;
    int guard = 0;
    Queue Q = empty;
}
```

```
Lock::acquire() {
  while (test&set(guard) == 1) {
     // do nothing
  }
  if (value == BUSY) {
     put curThread on Q;
     put curThread to sleep & guard = 0;
  } else {
     value = BUSY;
     guard = 0;
  }
}
```

```
Lock::release() {
  while (test&set(guard) == 1) {
     // do nothing
  }
  if Q is not empty {
     take T off Q;
     put T on ready queue;
  } else {
     value = FREE;
  }
  guard = 0;
}
```