# Sin #4a

## You may never take a lock in real-time code

**Normal lock:** `std::mutex::lock()`

std::mutex::lock is wrapper around `pthread_mutex_lock` (linux source code) - edited for brevity

```
while (1) {
   /* Try to acquire the lock through a CAS from 0 (not acquired) to our TID  */
   oldval = atomic_compare_and_exchange_val_acq (&mutex->__data.__lock,
                                                 tid, 0);
   if (__glibc_likely (oldval == 0))
      break;

   ...
   /* Block using the futex and reload current lock value.  */
    futex_wait ((unsigned int *) &mutex->__data.__lock, oldval,
                PTHREAD_ROBUST_MUTEX_PSHARED (mutex));
    oldval = mutex->__data.__lock;
}
return;
```

**Real-time safe**

**OS call which can block thread**

→ Lock is real-time safe as long as it's never contended

# Interposing

# Sin #4a

## You may never take a lock in real-time code

**Normal lock:** `std::mutex::lock()`

std::mutex::lock is wrapper around `pthread_mutex_lock` (linux source code) - edited for brevity

```
while (1) {
   /* Try to acquire the lock through a CAS from 0 (not acquired) to our TID  */
   oldval = atomic_compare_and_exchange_val_acq (&mutex->__data.__lock,
                                                 tid, 0);
   if (__glibc_likely (oldval == 0))
      break;

   ...
   /* Block using the futex and reload current lock value.  */
    futex_wait ((unsigned int *) &mutex->__data.__lock, oldval,
                PTHREAD_ROBUST_MUTEX_PSHARED (mutex));
    oldval = mutex->__data.__lock;
}
return;
```

**Real-time safe**

**OS call which can block thread**

→ Lock is real-time safe as long as it's never contended