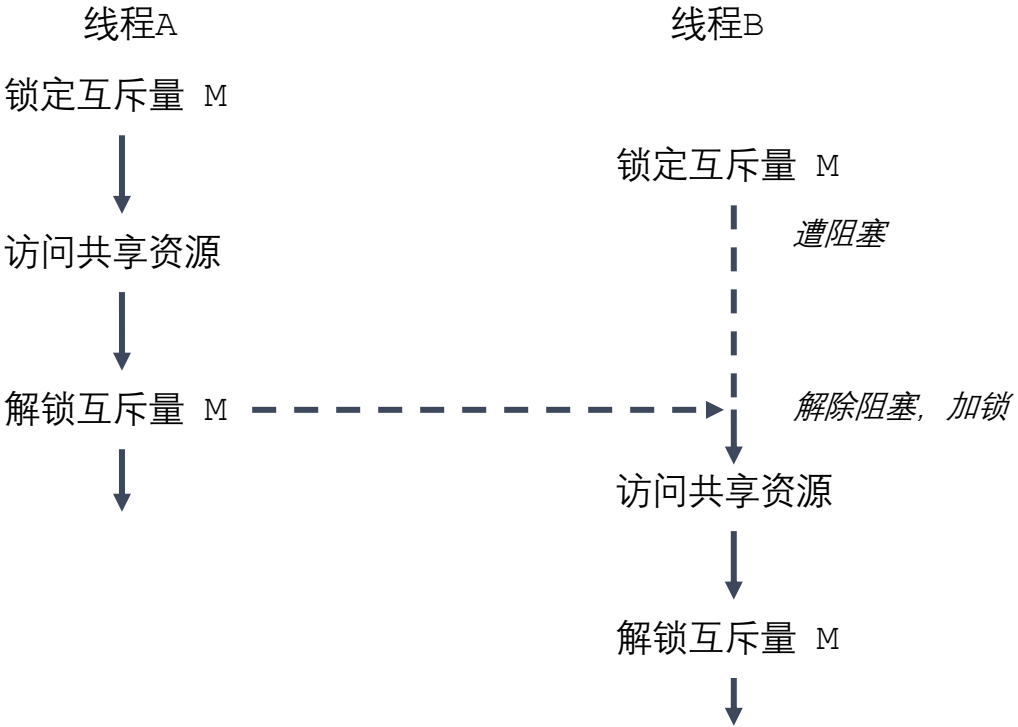
线程同步

- 线程的主要优势在于，能够通过全局变量来共享信息。不过，这种便捷的共享是有代价的：必须确保多个线程不会同时修改同一变量，或者某一线程不会读取正在由其他线程修改的变量。
- 临界区是指访问某一共享资源的代码片段，并且这段代码的执行应为原子操作，也就是同时访问同一共享资源的其他线程不应终端该片段的执行。
- 线程同步：即当有一个线程在对内存进行操作时，其他线程都不可以对这个内存地址进行操作，直到该线程完成操作，其他线程才能对该内存地址进行操作，而其他线程则处于等待状态。

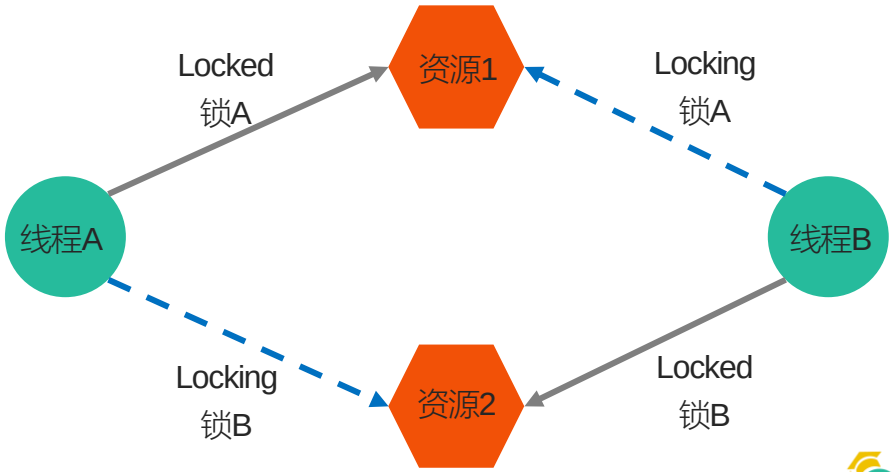
- 为避免线程更新共享变量时出现问题，可以使用互斥量（mutex 是 mutual exclusion 的缩写）来确保同时仅有一个线程可以访问某项共享资源。可以使用互斥量来保证对任意共享资源的原子访问。
- 互斥量有两种状态：已锁定（locked）和未锁定（unlocked）。任何时候，至多只有一个线程可以锁定该互斥量。试图对已经锁定的某一互斥量再次加锁，将可能阻塞线程或者报错失败，具体取决于加锁时使用的方法。
- 一旦线程锁定互斥量，随即成为该互斥量的所有者，只有所有者才能给互斥量解锁。一般情况下，对每一共享资源（可能由多个相关变量组成）会使用不同的互斥量，每一线程在访问同一资源时将采用如下协议：
  - 针对共享资源锁定互斥量
  - 访问共享资源
  - 对互斥量解锁

- 如果多个线程试图执行这一块代码（一个临界区），事实上只有一个线程能够持有该互斥量（其他线程将遭到阻塞），即同时只有一个线程能够进入这段代码区域，如下图所示：



- 互斥量的类型 pthread_mutex_t
- int pthread_mutex_init(pthread_mutex_t *restrict mutex,
  const pthread_mutexattr_t *restrict attr);
- int pthread_mutex_destroy(pthread_mutex_t *mutex);
- int pthread_mutex_lock(pthread_mutex_t *mutex);
- int pthread_mutex_trylock(pthread_mutex_t *mutex);
- int pthread_mutex_unlock(pthread_mutex_t *mutex);

- 有时，一个线程需要同时访问两个或更多不同的共享资源，而每个资源又都由不同的互斥量管理。当超过一个线程加锁同一组互斥量时，就有可能发生死锁。
- 两个或两个以上的进程在执行过程中，因争夺共享资源而造成的一种互相等待的现象，若无外力作用，它们都将无法推进下去。此时称系统处于死锁状态或系统产生了死锁。
- 死锁的几种场景：
  - 忘记释放锁
  - 重复加锁
  - 多线程多锁，抢占锁资源



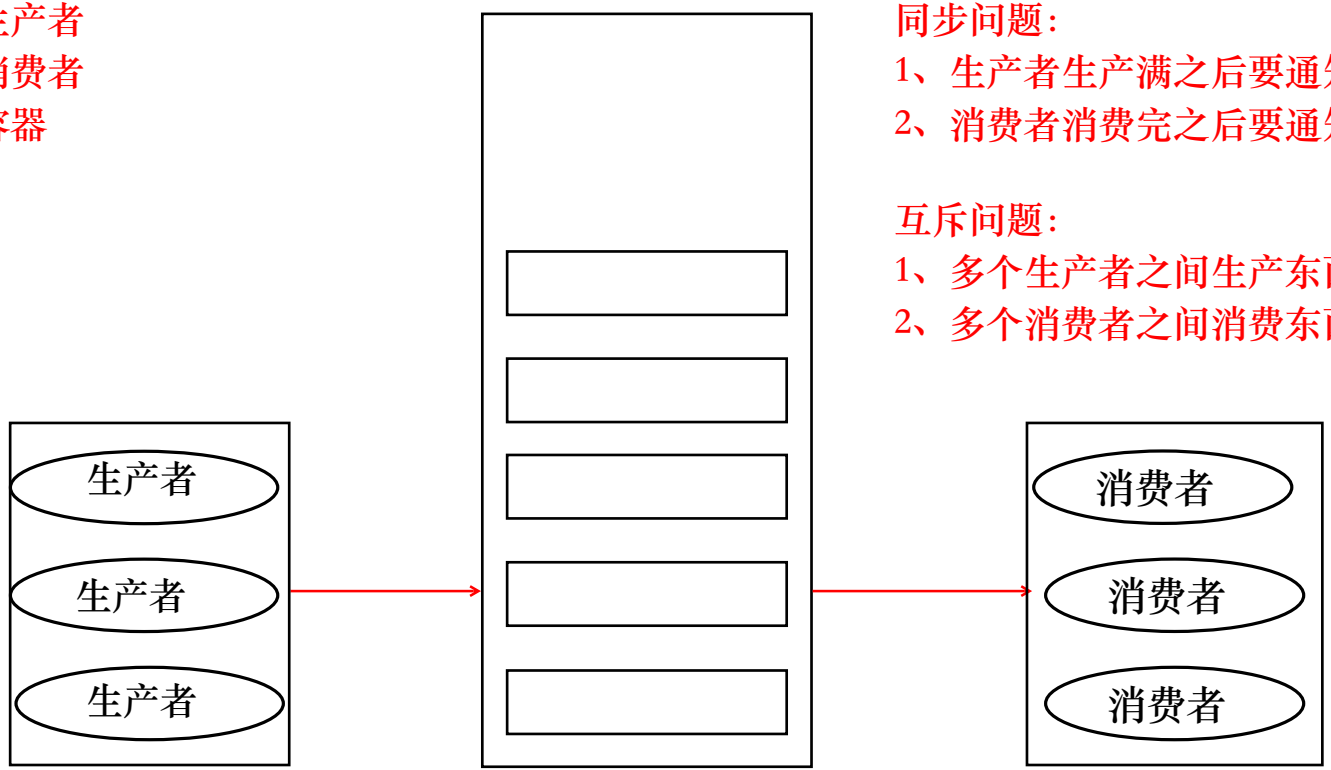
- 当有一个线程已经持有互斥锁时，互斥锁将所有试图进入临界区的线程都阻塞住。但是考虑一种情形，当前持有互斥锁的线程只是要读访问共享资源，而同时有其它几个线程也想读取这个共享资源，但是由于互斥锁的排它性，所有其它线程都无法获取锁，也就无法读访问共享资源了，但是实际上多个线程同时读访问共享资源并不会导致问题。
- 在对数据的读写操作中，更多的是读操作，写操作较少，例如对数据库数据的读写应用。为了满足当前能够允许多个读出，但只允许一个写入的需求，线程提供了读写锁来实现。
- 读写锁的特点：
  - 如果有其它线程读数据，则允许其它线程执行读操作，但不允许写操作。
  - 如果有其它线程写数据，则其它线程都不允许读、写操作。
  - 写是独占的，写的优先级高。

- 读写锁的类型 pthread_rwlock_t
- int pthread_rwlock_init(pthread_rwlock_t *restrict rwlock,
  const pthread_rwlockattr_t *restrict attr);
- int pthread_rwlock_destroy(pthread_rwlock_t *rwlock);
- int pthread_rwlock_rdlock(pthread_rwlock_t *rwlock);
- int pthread_rwlock_tryrdlock(pthread_rwlock_t *rwlock);
- int pthread_rwlock_wrlock(pthread_rwlock_t *rwlock);
- int pthread_rwlock_trywrlock(pthread_rwlock_t *rwlock);
- int pthread_rwlock_unlock(pthread_rwlock_t *rwlock);

1. 生产者
2. 消费者
3. 容器

同步问题：
1、生产者生产满之后要通知消费者消费
2、消费者消费完之后要通知生产者生产

互斥问题：
1、多个生产者之间生产东西要互斥
2、多个消费者之间消费东西要互斥



- 条件变量的类型 pthread_cond_t
- int pthread_cond_init(pthread_cond_t *restrict cond, const pthread_condattr_t *restrict attr);
- int pthread_cond_destroy(pthread_cond_t *cond);
- int pthread_cond_wait(pthread_cond_t *restrict cond, pthread_mutex_t *restrict mutex);
- int pthread_cond_timedwait(pthread_cond_t *restrict cond, pthread_mutex_t *restrict mutex, const struct timespec *restrict abstime);
- int pthread_cond_signal(pthread_cond_t *cond);
- int pthread_cond_broadcast(pthread_cond_t *cond);

- 信号量的类型 sem_t
- int sem_init(sem_t *sem, int pshared, unsigned int value);
- int sem_destroy(sem_t *sem);
- int sem_wait(sem_t *sem);
- int sem_trywait(sem_t *sem);
- int sem_timedwait(sem_t *sem, const struct timespec *abs_timeout);
- int sem_post(sem_t *sem);
- int sem_getvalue(sem_t *sem, int *sval);

牛客大学
-专业求职辅导-

THANKS

关注【牛客大学】公众号
回复“牛客大学”获取更多求职资料