第五，六次作业

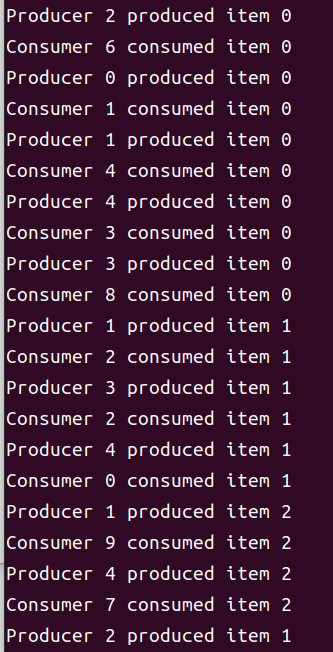
1. 使用Condition Variables编写生产者消费者问题(假设缓冲区大小为10,系统中有5个生产者，10个消费者)。并回答以下问题： 1. 在生产者和消费者线程中修改条件时为什么要加mutex? 2. 消费者线程中判断条件为什么要放在while而不是if中?

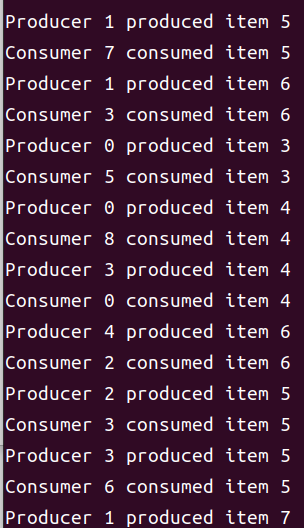
代码：





运行该程序：





问题：1. 在生产者和消费者线程中修改条件时为什么要加mutex?

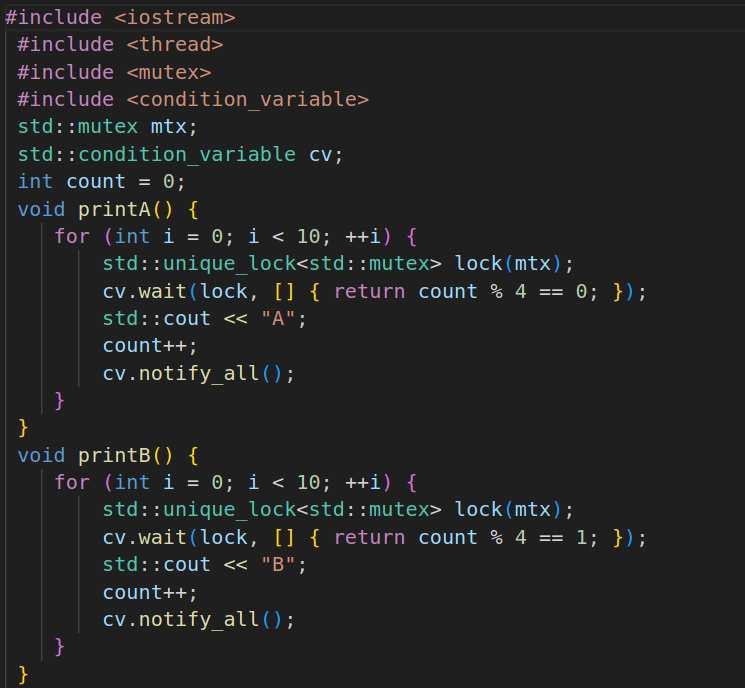
答：确保线程之间的互斥访问。如果没有互斥锁的保护，可能会出现竞争条件（Race Condition），导致 数据不一致或不正确的结果。互斥锁的作用是保证在访问共享资源（如缓冲区）之前，线程会先获取 锁，保证只有一个线程能够修改共享资源，其他线程需要等待。

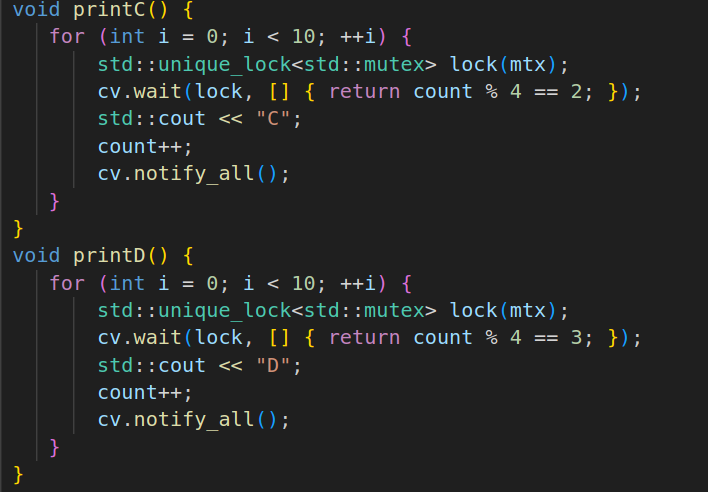
2. 消费者线程中判断条件为什么要放在while而不是if中?

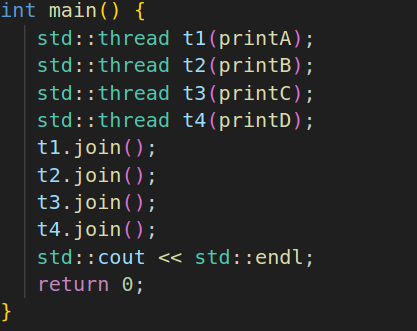
答：防止虚假唤醒（Spurious Wakeup）。虚假唤醒指的是在没有收到显式的通知或信号的情况下，等待 条件的线程被唤醒。如果使用 if 语句来判断条件，当线程被虚假唤醒时，它将继续执行后续代码，可能 会导致程序逻辑错误。使用 while 循环来判断条件可以在虚假唤醒时再次检查条件，确保条件满足才继 续执行后续代码，避免了逻辑错误。

2.4个线程，线程1循环打印A，线程2循环打印B，线程3循环打印 C，线程4循环打印D。完成下面两个问题：1. 输出 ABCDABCDABCD…2. 输出 DCBADCBADCBA…

答：

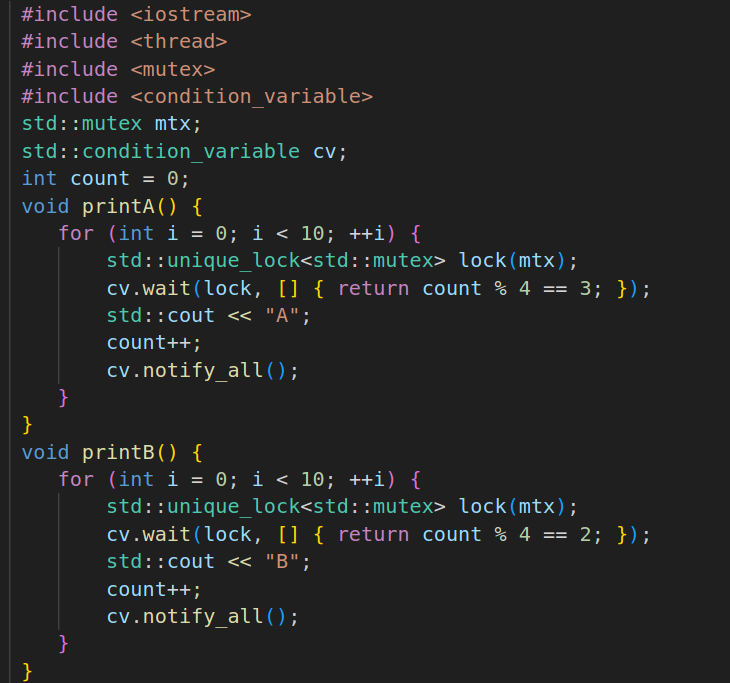


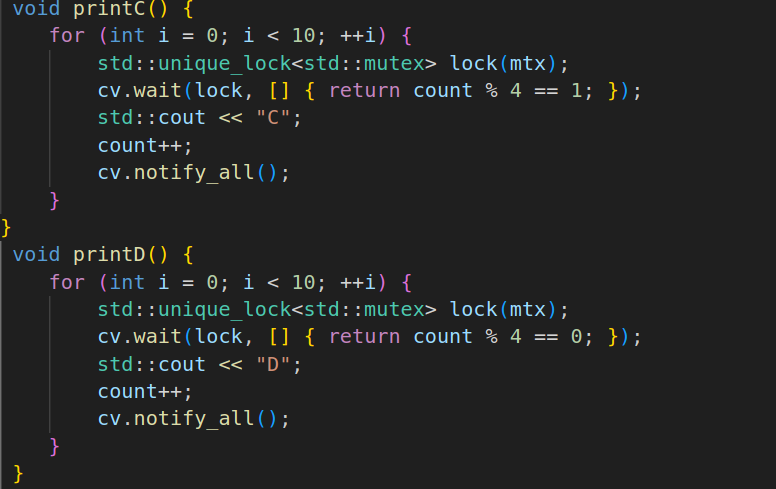


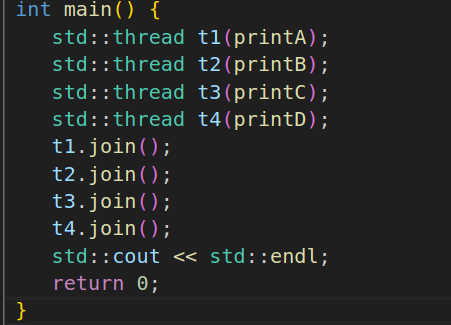


运行代码。得到结果：

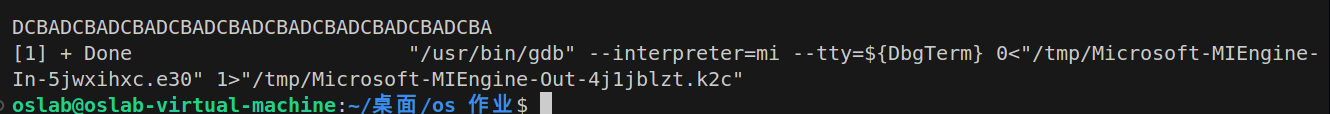






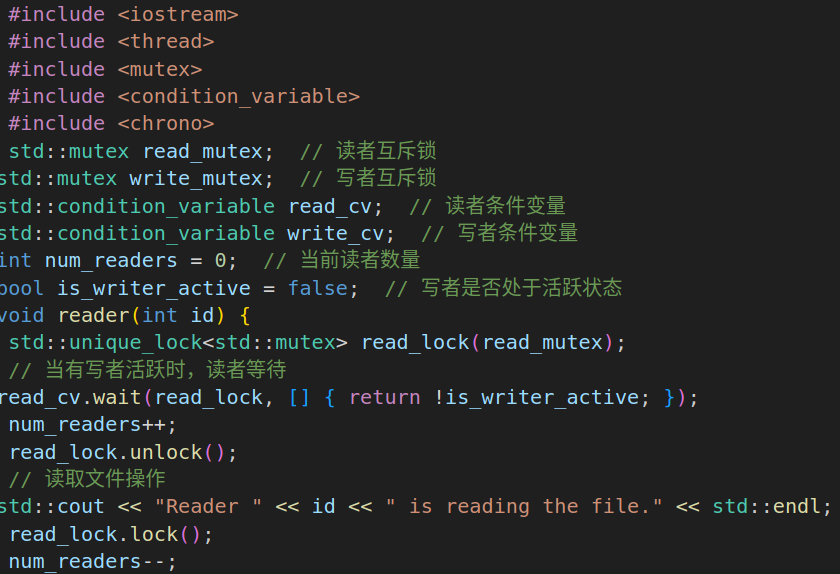


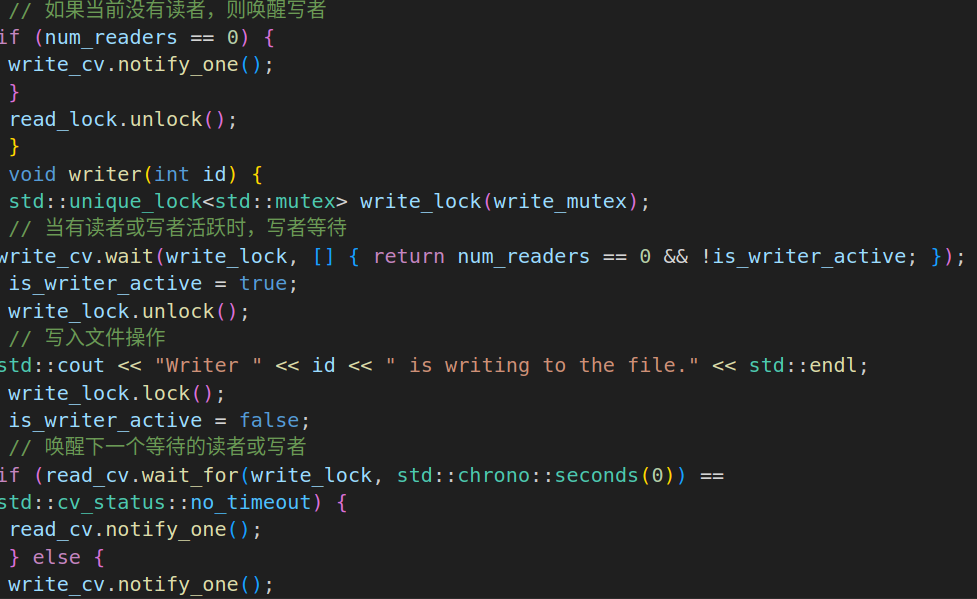
运行程序得到结果：

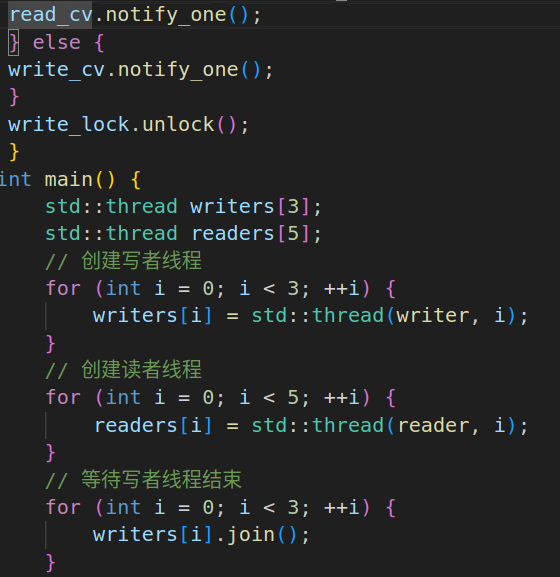


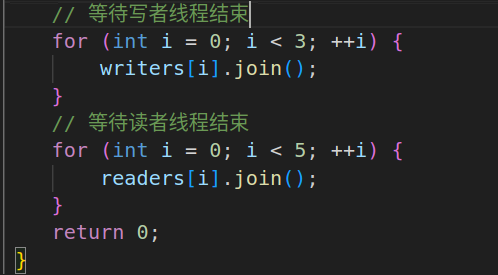
写者优先作业: 1. 写者线程的优先级高于读者线程。 2. 当写者到来 时，只有那些已经获得授权的读进程才被允许完成 它们的操 作，写者之后到来的读者将被推迟，直到写者完成。 3. 当没有写者进程时读者进程应该能够同时读取文件。

要实现写者优先的读者写者问题，可以使用互斥锁和条件变量来实现同步。下面是一个示例代码：

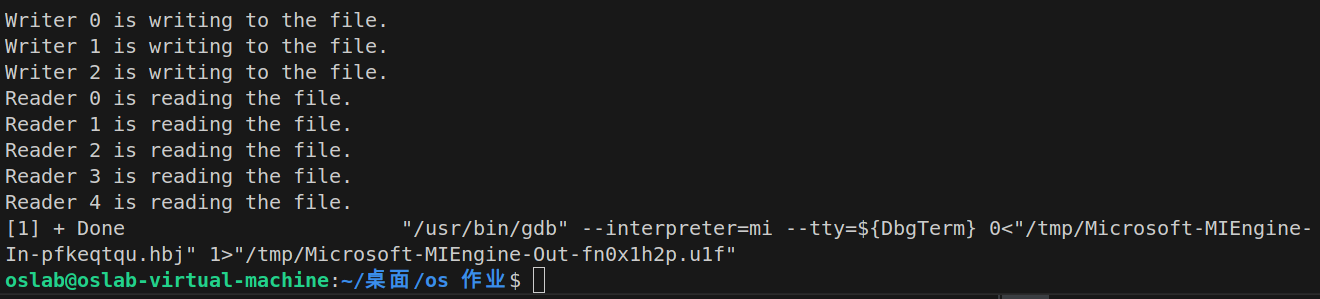








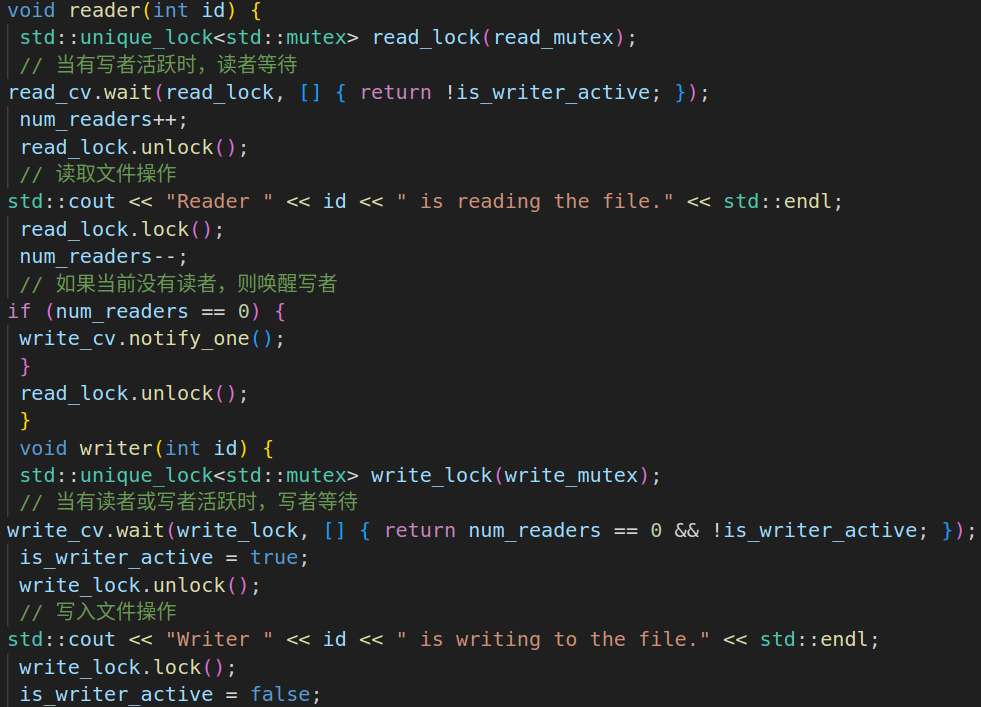
运行程序得到结果：

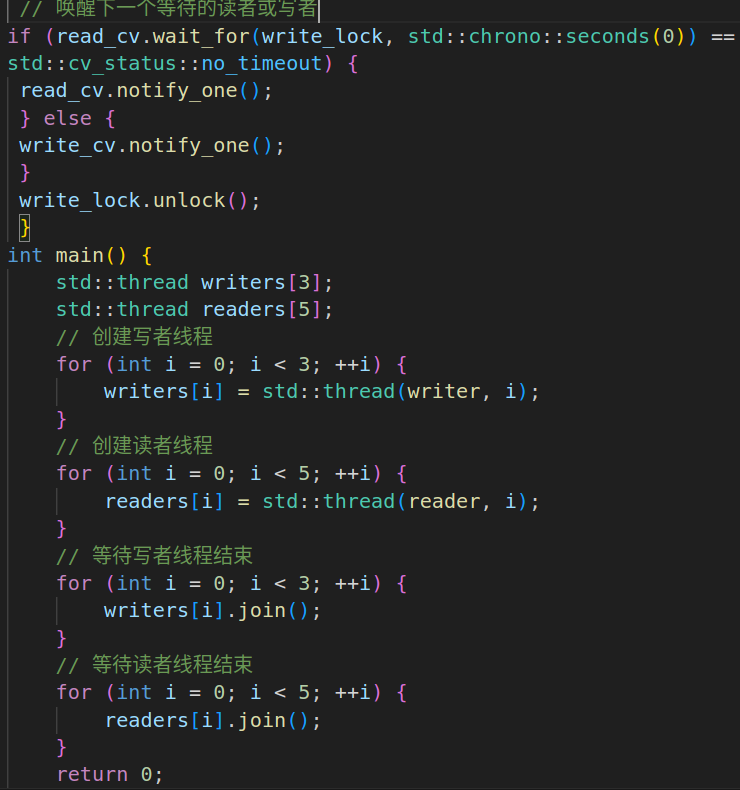


在上述代码中，读者线程和写者线程通过互斥锁（read\_mutex和write\_mutex）和条件变量 （read\_cv和write\_cv）来实现同步。读者在执行读取文件操作前，会先检查是否有活跃的写者，如 果有则等待条件变量read\_cv的通知。写者在执行写入文件操作前，会先检查是否有活跃的读者或写 者，如果有则等待条件变量write\_cv的通知。这样就实现了写者优先的效果。 当没有写者进程时，读者进程可以同时读取文件。读者在执行读取操作前，会先检查是否有活跃的写 者，如果没有则直接进行读取操作，而不需要等待。这是通过在读者线程中使用条件变量的等待函数 read\_cv.wait(read\_lock, [] { return !is\_writer\_active; });来实现的。只有当没有活跃的写者 时，读者线程才会被唤醒执行读取操作。

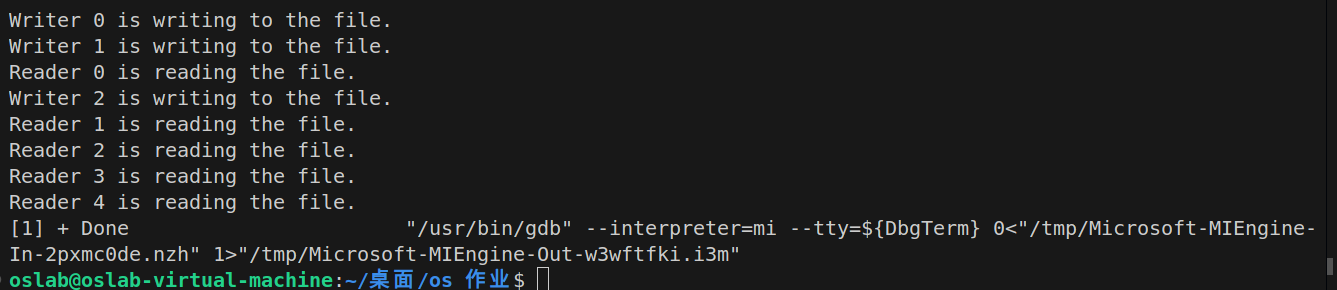
公平竞争：1. 优先级相同。 2. 写者、读者互斥访问。 3. 只能有 一个写者访问临界区。 4. 可以有多个读者同时访问临界资源。

要实现公平竞争的读者写者问题，可以使用互斥锁和条件变量来实现同步。代码：





运行程序得到结果：



在上述代码中，读者和写者线程使用互斥锁（read\_mutex和write\_mutex）和条件变量（read\_cv 和write\_cv）来实现同步。读者在执行读取文件操作前，会先检查是否有活跃的写者，如果有则等待 条件变量read\_cv的通知。写者在执行写入文件操作前，会先检查是否有活跃的读者或写者，如果有则 等待条件变量write\_cv的通知。 公平竞争的要点是，在互斥锁和条件变量中使用适当的等待和唤醒机制，以确保读者和写者能够按照公 平的顺序访问临界区。