Q1. Describe Function(功能) of pthread_create:
int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine) (void *), void *arg);
在调用进程中创建一个新线程，该线程唤醒 start_routine()进行执行。

Q2. Describe Function(功能) of pthread_join:
int pthread_join(pthread_t thread, void **retval);
等待参数中的线程 thread 执行结束，如果 retval 非空，则将线程 thread 的退出状态存于 retval 指向的地址。如果线程 thread 被取消，PTHREAD_CANCELED 将会被存在 retval 指向的地址。
通常主线程对所有创建的线程使用该函数,避免主线程在其他线程之前退出。

Q3. Describe Function(功能) of pthread_mutex_lock:
int pthread_mutex_lock(pthread_mutex_t *mutex);
锁住 mutex 所指向的互斥锁，成功则返回 0. 互斥锁被锁定线程释放后才能被另一线程获取。

补充：mutex 互斥锁

在编程中，引入了对象互斥锁的概念，来保证共享数据操作的完整性。每个对象都对应于一个可称为" 互斥锁" 的标记，这个标记用来保证在任一时刻，只能有一个线程访问该对象。

Q4. Describe Function(功能) of pthread_cond_wait:
int pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t *mutex);
释放锁 mutex,阻塞线程。当该阻塞的线程被其他线程唤醒时,重新获取锁 mutex.
条件变量可利用线程间共享的全局变量进行同步，通常和一个互斥锁结合在一起使用。

Q5. Describe Function(功能) of pthread_cond_signal:
给另外一个正在处于阻塞状态的线程发送信号 ,使其脱离阻塞状态,继续执行。如果没有线程处在阻塞等待状态，该函数也会成功返回。阻塞线程通常指的是由于 pthread_cond_wait 而阻塞的线程。若有多个线程被阻塞，根据调度原则选择一个进程发送信号。
在单核处理器上，最多给一个线程发送信号，而在多核处理器上可以给多个阻塞线程发送信号。

Q6. Describe Function(功能) of pthread_mutex_unlock:
int pthread_mutex_unlock(pthread_mutex_t *mutex);
解除 mutex 所指向的互斥锁的锁定状态。

Q7. Describe Function(功能) of sem_open:
sem_t *sem_open(const char *name, int oflag);
sem_t *sem_open(const char *name, int oflag, mode_t mode, unsigned int value);
创建新的或者打开已存的名为 name 的信号量。 参数 oflag 控制函数调用时的操作类型。权限 mode 以及初始值 value 只在新建名为 name 的信号量时被传入。

Q8. Describe Function(功能) of sem_wait:

int sem_wait(sem_t *sem);

将 sem 指向的信号里量值减一。

具体而言，如果此时信号量的值大于 0，则立马执行减一并返回，否则进入阻塞状态直到调用被中断或者信号量的值大于 0。

补充：

int sem_trywait(sem_t *sem);

int sem_timedwait(sem_t *sem, const struct timespec *abs_timeout);

Q9. Describe Function(功能) of sem_post：

int sem_post(sem_t *sem);

将 sem 指向的信号量的值加一。此后，如果信号量的值大于 0，则另一阻塞于 sem_wait 调用的进程或者线程将会被唤醒来获取该信号量。

Q10. Describe Function(功能) of sem_close：

int sem_close(sem_t *sem);

关闭 sem 指向的信号量，释放信号量分配给调用进程的所有资源。

关闭不是删除，删除用 unlink 来实现。

Q11. Producer-Consumer Problem（understand producer_consumer.c）: Are the data that consumers read from the buffer are produced by the same producer? producer_consumer.c

在样例程序中，消费者读取的是同一个生产者生产的内容因为此时只有一个生产者。

但通常而言，如果存在多个生产者轮流工作，消费者可以读取多个生产者的内容。

Q12. Producer-Consumer Problem（understand producer_consumer.c）: What is the order of the consumer's read operations and the producer's write operations, and their relationship

生产者先进行写操作，然后消费者进行读操作，因为消费者无法从一个空的 buffer 里读取内容。当 buffer 为满后，生产者无法再进行写操作了，此时只有消费者进行读操作直到 buffer 不再为满。同样，buffer 为空时，消费者无法进行读操作，只能让生产者执行写操作。具体说明见下。 总体而言，读写操作在时间上有着先后关系。

Q13. Producer-Consumer Problem（understand producer_consumer.c）: Briefly describe the result of the program

首先，程序通过 pthread_create 创建了 producer 以及 consumer 这两个线程，轮流对 buffer 进行写、读操作，生产者每次写入一个随机字母，消费者每次读出一个字母显示出来。同时通过一个互斥锁，在进行任一操作前获取锁，操作后释放锁，保证了在一个时刻读、写中只能有一个对 buffer 进行操作。

当 IS_FULL 条件成立，buffer 为满，producer 通过 pthread_cond_wait 释放锁并进入阻塞状态。consumer 获取锁，通过 pthread_cond_signal 以及 full 变量唤醒 producer，再从 buffer 中读取内容，最后释放锁。producer 与 consumer 又开始轮流对 buffer 进行读写。

同理， 当 IS_EMPTY 条件成立，buffer 为空，consumer 通过 pthread_cond_wait 释放锁并进入阻塞状态。producer 获取锁，通过 pthread_cond_signal 以及 empty 变量唤醒 consumer，再向 buffer 中写内容，最后释放锁。然后 consumer 继续执行。

Q14. Producer-Consumer Problem（understand producer_consumer.c）: What queue is used in this program, and its characteristics?
环形队列。
特点是占用线性空间，首尾相连，先进先出，使得存读快速，空、满状态的判定也十分简单。

Q15. Producer-Consumer Problem（understand producer_consumer.c）: Briefly describe the mutual exclusion mechanism of this program
生产者与消费者通过一个互斥锁，在进行任一操作前获取锁，操作后释放锁，保证了在一个时刻读、写中只能有一个对 buffer 进行操作。

即使当 IS_FULL 条件成立，buffer 为满，producer 释放锁进入阻塞状态，consumer 获取锁并唤醒 producer，读取内容后释放锁。producer 又获取了锁。同理， 当 IS_EMPTY 条件成立，buffer 为空，consumer 释放锁并进入阻塞状态。producer 获取锁并唤醒 consumer，再向 buffer 中写内容后释放锁。然后 consumer 又获取锁。