# Exercise 2

Xin Zhao
r0734810

Mutex lock is applied in the programming, since there are only three threads working simultaneously so that mutex lock is an efficient and easy way to complement synchronization. Initially, I was planning to use r/w lock in my project as two threads need to read the shared buffer and they can read it at the same time while using r/w lock. However, data are stored with two flags that indicating whether it is read by data manager or storage manager. They need to be changed in reading threads. Hence r/w lock is not applicable because the shared buffer cannot be altered in reading threads.

***1. Is deadlock avoided?***

Deadlock is avoided in the programming since the mutex lock is always unlocked when the thread finishes operation on shared buffer or log file.

***2. How can you be sure that multiple writer-threads can't access sbuffer at the same time?***

Whenever one writer-thread would like to access and change the shared buffer or log file, it must try to lock the mutex lock first. If the mutex lock is available, this thread will lock the mutex which means this thread have access to the shared buffer or log file. Otherwise, if the mutex lock is not available, the thread will be blocked until the mutex lock is free.

**3.** ***Is your synchronization solution 'fair'? For instance, if a writer or reader thread wants to access sbuffer, it should get it immediately if sbuffer is not locked by other threads. Of course, the starvation-problem should be avoided.***

The synchronization is fair for reader but not for writer. Writer unlocks the mutex lock when it complete writing one data into the shared buffer or writing one message into the log file, hence reader can access them immediately. However, reader unlock the mutex lock until it finishes reading the whole buffer. Hence, there is no starvation problem although writer and another reader can not access it immediately. One solution is locking the mutex lock within for loop but not outside the loop. But for normal speed transmission, it works well.

***4. Argue why your choice is the most optimal one. Does your choice guarantee efficient use of the CPU?***

Actually, mutex lock is capable for this problem but it would be better if conditional variable is combined together. Because if all data in the shared buffer are read by data manager, the thread still need to read the buffer throughout, leading to unnecessary loss of CPU's efficiency. By combining condition variable with mutex lock, this problem can be improved.