**Description of Implementation**

*rwLock.c*

In this file, we used the lecture implementation of the read-write locks. This file is included in main3.c and main4.c

*main1.c*

For main1.c, we used a single mutex for the entire string array. A server thread first accepts and parses the message sent by the client. The thread then locks the mutex before doing any reads or writes to the array and then unlocks it right after. We contained only the setContent() and getContent() function calls in the mutex to minimize the time a thread locks it such that other threads could proceed processing their requests quicker.

*main2.c*

For main2.c, we used a separate mutex for each element in the string array. We use an array of mutexes of the same length as the string array, where the index of a mutex refers to the index of the element in the string array it corresponds to, to store the separate mutexes. A server thread accepts and parses the message sent by the client first, then locks the mutex in the same index as the position of the string in the request. The thread is then safe to read or write the string which the mutex corresponds to (the one at the same index as the mutex the thread has locked). Similar to the main1.c, we contain only the setContent() and getContent() function calls in the mutex to minimize the time a thread spends in it.

*main3.c*

main3.c is implemented in a similar way as main1.c; we use a single read-write lock for the entire string array. After a server thread parses the message it accepts, it locks only the operation it is performing. Therefore, a write-lock is run only when a thread is writing, and it only contains the setContent() function call and a read-lock is run only when a thread is reading, and it only contains the getContent() function call. This is different from a mutex which does not distinguish between reads and writes and, therefore, contains both the setContent() and getContent() functions calls in the same mutex.

*main4.c*

For main4.c, we used a separate read-write lock for each element in the string array. Like main2.c, we used an array of read-write locks the same size as the string array where the index of the lock is the same as the string it corresponded to in the array. The critical section is handled like main3.c where the server thread performs a write-lock only when using the setContent() function and the read-lock only when using the getContent() function. Additionally, it only locks the read-write lock of the string the request it is handling is made to.

**Performance Discussion**