Tània Pazos, u185115
Júlia López, u214964

Report Lab 3

1. Imagine that you have a data source which allows N simultaneous reads. What would you change, in order to implement it?

In the original assignment we allow 4 threads to read concurrently. In order to allow N threads, we would need to initialize the semaphore with N resources i.e. changing the third argument to N sem_init(&semaphore, 0, N).
Of course, the already implemented synchronization tools would still play a crucial role in preventing race conditions. Indeed, this change would not affect the synchronisation tools implemented in the code.

2. Why there is no need for synchronisation in the function unreserveFile? Would it be in the case of the previous question?

In our original case, where there is one thread per file, there is no need for syncronisation in the function unreserveFile since we implement a semaphore and a mutex lock to prevent two threads from getting the same file. Because of that, it is not possible that two threads access the same file at the same time and thus, will never be able to unreserve it simultaneously.

In the case where we allow N simultaneous reads, that is, several threads can read from the same file at the same time, we would need synchronization since there could be a race condition in the function unreserveFile since two threads could try to update a specific value of the fileAvailable array at the same time which would lead to data corruption.
This could be prevented by using a mutex lock before accessing the array so that only one thread could unreserve it at a time.

3. Compare the execution time between a version of 1 thread, and 4 threads, using the timer seen in P1. Create a table when processing a number of files ranging from 1 to 10 (you can make copies of large file.txt).

In order to allow only 1 thread instead of 4 threads to execute concurrently, we simply change the third parameter of the sem_init() function from nFilesTotal (4) to 1.

Number of files	**Execution time in single thread (in microseconds)**	**Execution time in multi-thread (in microseconds)**
1	124	135
2	127	125
3	129	127
4	149	147

5	164	163
6	170	179
7	185	172
8	193	187
9	227	206
10	249	225

As the table shows, the multi-threaded environment allows for smaller execution times compared to the single-threaded environment, especially when the number of input files is large.