**EE360P Assignment 3**

Alex Ma (am74874)

Danny Vo (dpv292)

1. The proposed solution to the problem only modifies Lamport’s mutex algorithm for read applications while leaving it untouched for writing applications. Read accesses are allowed to run alongside one another, and to allow this we have read requests only check that they have timestamps less than timestamps belonging to write requests (while still receiving acks from all other queued process). Writing accesses are left untouched which means they still fall under the algorithm’s original rules, i.e. cannot execute while any other processes/accesses (whether read or write) are occurring.
   1. We can extend Lamport’s mutex algorithm to solve *k*-mutual exclusion problems where only at most *k* processes are allowed in the critical section concurrently. We modify Lamport’s algorithm such that we only allow any given process *P* with its associated timestamp *T* into the critical section if and only if there exists < *k* processes in the critical section, all of which have request timestamps < *T*. In other words, if the number of lower-timestamped processes or unreceived acks < *k*.  
      Assume that this modification doesn’t solve the prompt, i.e. there can be more than *k* processes in the critical section at any given time. However, if a process *P* is within the first *k* in a queue, the critical section will not contain anymore than the first *k* processes which is not possible due to the imposed restriction =><=.
   2. We can also extend Ricart and Agrawala’s mutex algorithm to solve the same *k*-mutual exclusion problem. We modify the existing algorithm such that any process that requests access to the critical section is now forced to wait for at the very least *N*-*k okay* messages where *N* is the total number of processes associated with the target system. With this change we can ensure that less than *k* processes are in the critical section currently.  
      Assume that this modification does not solve the *k*-mutual exclusion problem, i.e. there can be more than *k* processes in the critical section at any given time. However for a given process *P*, for it to enter the critical section it must receive *N*-*k* *okay* messages so it could never enter the critical section if *k* processes are in it already =><=.