Assignment 7 : Deadlocks - Technical Report

The goal of this assignment was to implement the banker's algorithm. Several threads request and release resources from the bank. The banker, which is a decentralized scheduler will grants a request only if it leaves the system in a safe state. A request leaving the system in an unsafe state will be denied.

I took care of the following things while implementing the assignment :

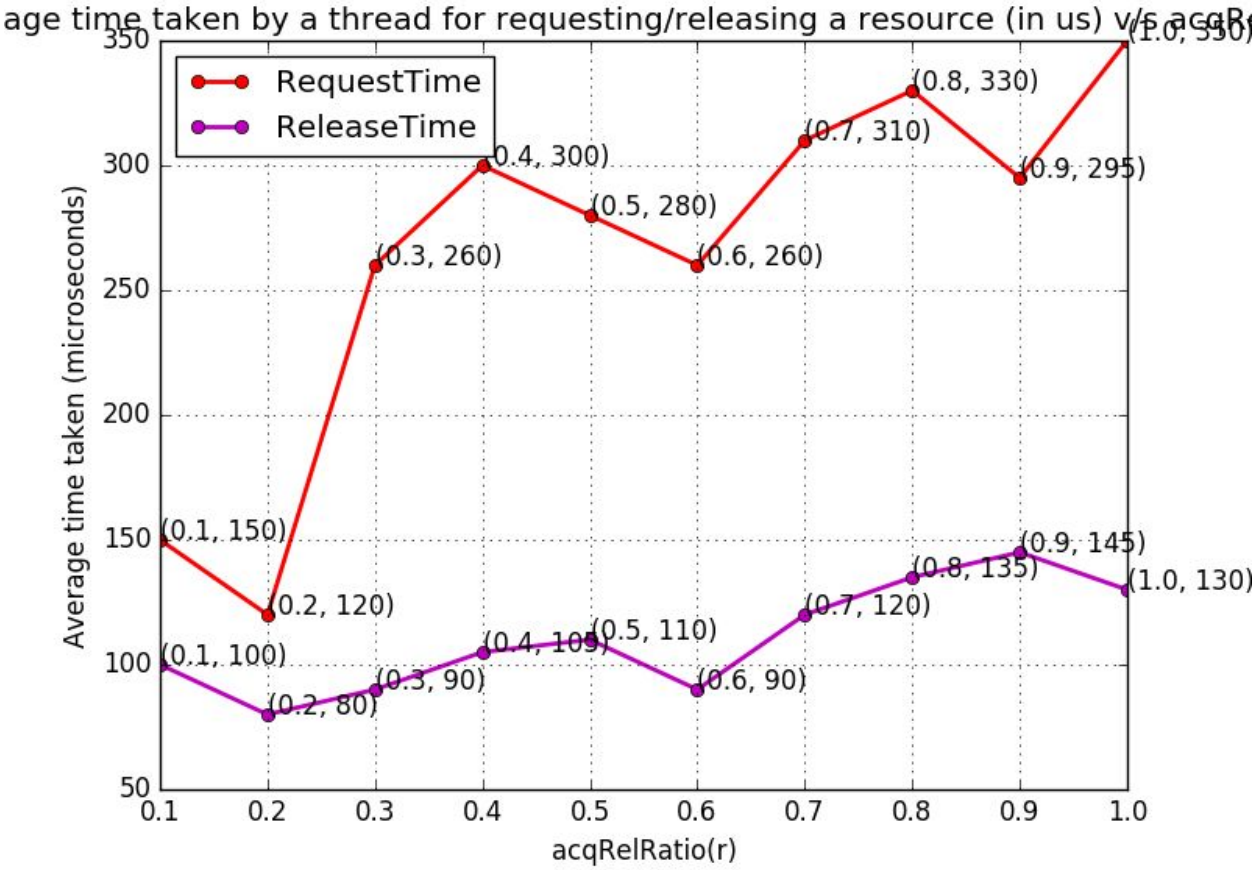
- **Deadlock avoidance** : The code, before allocating resources, calls the **is_safe_state** function to check whether after allocating the resources to the process will the leave the system in a safe state, if not, the changes made are rollbacked, and if the safe state is maintained, then the changes according to the algorithm are made.

- **Preventing race conditions** : A mutex lock is invoked every time, the functions **release** and **request** methods are called, since the data structures (Need, Max, Allocation) are continuously, and since multithreading is present, the updation operations need to be atomic.

We need to define a **state** of the system, to signify whether deadlock can occur in a system or not.

A state is considered safe if it is possible for all processes to finish executing (terminate). Since the system cannot know when a process will terminate, or how many resources it will have requested by then, the system assumes that all processes will eventually attempt to acquire their stated maximum resources and terminate soon afterward. This is a reasonable assumption in most cases since the system is not particularly concerned with how long each process runs (at least not from a deadlock avoidance perspective). Also, if a process terminates without acquiring its maximum resource it only makes it easier on the system. A safe state is considered to be the decision maker if it's going to process ready queue.

Below is the graph of the average time taken by a thread for requesting and releasing resources :



As it is clear from the graph, the average request-time for a resource is much higher than release time. Also, the request time increases with increasing Acquire-Release ratio. This is also the expected behaviour, since larger ratio implies more number of requests for resources, and thus the corresponding increase in time to serve those requests.

The only anomaly occurs some times, when the request-/release-times get lowered sometimes, which is entirely system-state dependent.