**CS 1550 Project 2 Report**

**Tianrun Gu**

I believe the highlight of this project is how to keep just one agent and at most 10 tenants every moment in the apartment. By analyzing the elements we need, obviously, we need variables for number of tenants, number of agents, and a counter to indicate how many tenants has viewed apartments and left so that we could decide whether we could end the whole program.

Let’s first consider how to keep just one agent in the apartment. Because this project rejects “busy waiting”, so we could not just use a while loop to continuously check the number of agents in the apartment. Instead, we need a semaphore to make agents “sleep” when there is already an agent inside the apartment. This is a very simple task because we just need every agent performs “down()” action before they get into apartment. And the semaphore for that is just a “binary semaphore”. Since “down()” is an atomic function, we don’t need to worry how to protect it. And every time when an agent leaves the apartment, we call “up()” so that another available agent could enter the apartment. I name this semaphore “a\_l” to represents “an agent left”. Every agent needs to call “down(a\_l)” before they could enter the apartment. In this way, we prevent more than one agent inside the apartment. Of course, when I initialize the value of this semaphore, I make it to 1 so that the first one get this semaphore could enter the apartment. And every agent inside the apartment will call “down(finish)” to wait tenants call “up(finish)” when this agent’s tenants have finished their tour.

Second, let’s talk how to make sure at most 10 tenants in the apartment. The logic is that before every tenant enter the apartment, he or she needs to know how many tenants inside the apartment. So for tracking the number of tenants, actually we need two variables: one for tracking how many tenants outside the apartment and I name it “ot” and one for tracking how many inside the apartment and I call it “inside”. Of course, every time we check such data in critical region, we need semaphores to protect it. Since only a tenant get into the apartment can update or increment the number, I call this semaphore “t\_apt”. So every time when the process need to retrieve the number of tenants inside the apartment or update this data, they need to call “down(t\_apt)” and corresponding “up”. Every time an agent enters the apartment will cal “up(door)” to indicate the door is open. The trick is before any tenant enters the apartment, when he notices there is less than 10 people (including himself), he will leave the door open for other tenants by calling “up(door)”; if there is already 10 people including himself, he won’t call “up” so that the value of the “door” semaphore is 0.

Thirdly, because of this way of implementation, whether an agent leave depend if there is any tenant inside the apartment instead of checking if any other tenant would come. And to avoid deadlock, every time a process used a semaphore, for example, if one used “down(outside)” and it wants call “down(a\_l)”, it first calls “up(outside)” to return the lock back. So no process hold a semaphore and requires another semaphore at the same time, which prevents deadlocks.

So based on these three points, I believe my implementation is fair.