CS 1550 - Introduction to Operating Systems
Project 2 - Process Synchronization

Dr. Sherif Khattab

Safe Apartment Inspection

Abbas Ahmed
aza16@pitt.edu

<u>Report</u>

For my project, I used 6 Semaphores, out of which <u>four</u> are **binary semaphores** and <u>two</u> are **counting semaphores**. I also use <u>four</u> integer variables for counters that keep track of tenants inside the apartment, tenants waiting to get in the apartment, tenants the agent has seen, and a boolean flag which is 0 or 1 depending on whether there's an agent inside. One of the semaphores checks whether if there's an agent in an apartment by using a binary semaphore so that only one agent can be in the apartment. After the agent acquires this semaphore, he/she waits on the 'tenants' binary semaphore, which is released by the first tenant that arrives in the apartment. When a tenant arrives, and after signaling the agent if it's a first tenant to arrive, it waits on the semaphore to enter the apartment. The counter that keeps tracks of whether there are tenants waiting to get in the apartment gets incremented every time a new tenant arrives and decremented when the tenant enters the apartment. When the agent arrives, he/she also updates a boolean flag to let the tenant know whether if there's an agent inside already. When the agent first arrives, it moves on to letting the waiting tenants in one by one until it runs out of capacity(10 in this case) or until there are no tenants waiting anymore. After letting the tenants inside the apartment, the agent goes to sleep by using down on 'leave' semaphore. If a new tenant arrives after the agent goes to sleep, he/she checks whether if there's space available by checking the number of the tenants the agent has seen, which is stored as an integer counter. If the agent has seen less than 10 tenants, the new tenant updates the counter variables and proceeds directly to the apartment, otherwise, it waits on the 'enter' semaphore until a new agent releases it. Once the tenants are let in, they sleep for 2 seconds, as if they were inspecting an apartment, then proceed to exit. Upon exiting, the tenants decrement the counter of tenants inside the apartment, and if the tenant leaving the apartment is the last tenant, it checks whether if there are tenants waiting or if the agent has seen a total of 10 tenants, then it releases the ‘tenants’ semaphore so that the next agent can proceed directly to let the waiting tenants enter the apartment. Then the tenant finally releases the ‘leave’ binary semaphore so that the current agent can wake up and leave the apartment. Before an agent leaves, he/she updates the counters so that the next agent can use them fresh and then releases the ‘agent’ binary semaphore, to let another agent arrive in the apartment.

The tenants and agents are created from the main method by forking processes and the delaying of the processes takes place within the main method. At every point, if an agent arrives, or a tenant arrives, or if an agent opens up the apartment for inspection, or if the tenant starts inspecting, or if the agent or tenant leaves the apartment, there are print statements that output this progress to the console. The print statements are protected by a binary semaphore called ‘printex’. The counter updates are also protected at every point by a binary semaphore called ‘mutex’. From my thought processing, as I have shown above, I think that this a fair and starvation free solution. At every point, I check the right counters, update the semaphores only when they’re needed by letting in the agent and tenants the right amount and rightfully handling every scenario there might be a race condition.