Project 2 Writeup

In this project, I implement a program, that generates multiple processes to simulate real-life situation of inspecting apartments, using the semaphores and syscalls I built myself in the kernel.

Besides the semaphore of agent_info_lock, the program has five more semaphores to ensure it is deadlock free and starvation free: tenant, agent, apt_door, tenant_action, agent_waits.

First, the program starts with agent(s) and tenants arriving at the apartment. When an agent arrives, he calls down(tenant) to wait until a tenant arrives and calls up(tenant), telling the agent that there is a tenant waiting, so that the agent can open the door. As for the first tenant, he also needs to call down(agent) to wait for an agent. This ensures that an agent and tenant depend on each other to enter the apartment.

Second, for the agents, when an agent arrives, he calls down(apt_door) to lock the apartment. When he leaves, calls up(apt_door) to unlock the apartment. This ensures that the other agents cannot enter the apartment when there is an agent in it.

Third, for the tenants, every tenant has to call down(tenant_action) when he arrives, then call up(tenant_action) when the critical section of tenantArrives() finishes. This ensures that only one tenant can arrive at a time (or only one tenant can be in the critical section).

At last, for the agents, he calls down(agent_waits) after opening the door, waiting for the tenants finishing their tours. When the last tenant leaves the apartment, tenantLeaves() function calls up(agent_waits), so that the agent now can leave the apartment. This ensures that the agent will leave the apartment when there is no tenant in it.

In a conclusion, the solution is fair and it is deadlock and starvation free.