Gordon Lu

Professor Daniel Mosse

CS 1550

Due: 06 March 2020

CS 1550 Project 2 Report:

In this report, I will highlight the justifications to why the approach I have taken to achieve synchronization among tour guides and visitors in **museumsim.c** is fair, and deadlock and starvation-free.

I) **Fair:**

Initially, the first visitor and tour guide must wait for each other and utilize barrier synchronization before opening the museum. Once the first visitor and guide arrive, the tour guide will then open the museum, and continue to allow the remaining 9 visitors into the museum and let them tour. Any other visitors waiting outside will have to wait for another tour guide to arrive before entering the museum and will therefore be placed on a separate semaphore blocking queue. Overall, my solution is fair, as each tour guide is allotted a maximum of 10 visitors. Within **visitorArrives()** and **tourguideArrives()** I have allowed for the tour guide and visitor to wait for the initial guide and visitor to arrive. I have allowed for a maximum of 10 visitors by creating an additional semaphore that will call **up()** on a semaphore 10 times, to let a maximum of 10 visitors into the museum once the museum is open. This same semaphore will call **down()** for each of the visitors who are in the museum. This will allow for a new batch of 10 visitors and a new guide to arrive.

II) Starvation-Free

The possibility of starvation in this project is implicitly avoided. The only instance in which the processes would get starved would be in edge cases. Specifically, if more than 10 visitors arrived, while only 1 tour guide arrived, the first 10 visitors would be let into the museum with the tour guide to tour the museum, however the remaining visitors would be left waiting indefinitely for another tour guide who would never arrive, thus they would never have the chance to execute. This form of starvation will inevitably also lead to a deadlock. Additionally the case in which the number of tour guides exceeds 10 times the number of visitors is also unlikely to occur. Suppose there are 5 tour guides and 10 visitors. In this case, the first 10 visitors enter and tour the museum with the first tour guide. However, the remaining 4 tour guides will be left waiting for 10 * 4 = 40 visitors, and will never have a chance to run, so long as those visitors never arrive to the museum. Since the resources in my implementation are protected using semaphores, each process does not encode priority, they will likely not starve. Additionally, I have enforced a starvation-free policy since each tour guide will only be signaled to open and allow visitors on a museum if the previous tour guide has left and there are no visitors in the museum. Upon opening the museum in **openMuseum()**, each tour guide will give each of the waiting visitors an opportunity to tour the museum, thus the visitors will never starve.

III) Deadlock-Free:

In regards to enforcing a deadlock-free program, I have ensured that each semaphore will call **up()** and **down()** the appropriate number of times. I have enforced a strict order for the semaphores in which the processes will wait. This will effectively prevent the circular wait of deadlocks, essentially eliminating a deadlock altogether. For example, when the tour guide is awaiting a signal for the current tour guide to leave the museum, the next tour guide will call **down()** in **tourguideArrives()** and be put on that semaphore's blocked queue, this will only be signaled in **tourguideLeaves()** and will not be in a deadlock so long as there are sufficient tour guide and visitors. Additionally, race conditions have been accounted for. By utilizing locks on shared variables, this ensures that no other process can access the shared variable while the current process holds the lock. The utilization of **down()** and **up()** calls throughout my functions in museumsim are synchronized in a way such that no other process may access another's critical region, allowing for mutual exclusion, and are ensured to avoid race conditions, and are called an appropriate number of times, as to prevent a deadlock.