Allen Adams, Evan Minor

3-25-20

CS 440

Project 3

**a)**

Our solution takes advantage of several buffers, locks, and other Booleans to properly control the flow of bathroom traffic. Initially we call a single thread that then spawns our variable number of bathroom participants. These threads are then assigned an id based on order of creation, a gender (randomly assigned), as well as a time interval (randomly assigned). Once the thread has been created its properties are then passed into a method called OnePerson() that establishes the routine each thread (person) should follow. The first step being to announce their creation and information in the form of an arrival message. Then the threads are told to execute the arrive() method, this is where we implement most of our scheduling algorithm. Here we create a series of if statements that consider 6 different outcomes (or 3 repeated twice for each gender). The first/second establishes a set of rules if there is no one in the bathroom. For instance, here we assume that the first person to arrive can claim the lock and enter the bathroom. Once that given person has obtained the lock, they then place a “boys (girls)” sign on the door to indicate who occupies the bathroom. The 3rd/4th if statements considers what a person should do in the case that the bathroom has at least one occupant of the same gender. Here more people can enter and even cut in line of others if the sign on the door matches their gender. We do this by checking that the buffer size is less than 3, the lock is true, and if the sign matches the gender. Assuming all three checks are passed the person can enter. Statements 1-4 are the only ways for the threads to break out of the queue, we assume that all threads that do not initially meet these conditions are grouped in the queue. The final set of if (else) statements 5/6 manage how the threads behave If the other gender occupies the bathroom or the buffer is full. Here we simply allocate the threads to a queue where they stand by until they meet conditions 1-4. From here the threads are sent to the UseFacilities() method where they “use the bathroom” for a given period. Once they finish their business, they pass onward to the Depart() method where the threads made a departure statement and the buffer is decreased. In the case that the buffer is decreased back to 0 we unlock the bathroom for the other gender to then occupy.

**b)**

Arrive(){

if (girl == true)

if (buffer ===0 && lock == false)

lock = true;

buffer++;

sign == girl;

enter();

if (buffer < 3 && lock == true && sign == girl)

buffer++;

enter();

else

wait();

if (boy == true)

if (buffer ===0 && lock == false)

lock =true;

buffer++;

sign == boy;

enter();

if (buffer < 3 && lock == true && sign == boy)

buffer++;

enter();

else

wait(); }

Depart(){

buffer--;

if( buffer == 0 )

lock = false; }

**c)**

Although our solution was not perfect, we felt that it was about as close as it was going to get. It worked well! Firstcomers were able to claim the lock for their gender with no problem. Threads of the same gender were able to enter the bathroom without exceeding the buffer limit. And once the bathroom became empty, threads were able to properly trade off lock ownership to the other gender smoothly. It’s worth mentioning that no thread was stuck in purgatory or died as a result of our implementation.

Problems did arise in other areas however, such as tracking current time, efficiently filling the buffer, and execution order. One (small) issue we ran into with tracking the order of thread execution was found in updating current time. This variable would often act up on us because it was being accessed by so many threads concurrently. We found that oddly enough, changing the seed value would produce incorrect values for the current time (consistently). For example, threads would give notifications stating at 0 seconds, then 4 second, then back to 0 seconds. Although it did not compromise thread execution, it was a minor annoyance we struggled with. We found one way to fix this bug was to keep the default seed. The other (major) problem we found in our implementation was getting the threads to efficiently fill the buffer. We observed that the typical course of execution for our threads involved one thread claiming the bathroom at the start then immediately revoking ownership to the other gender. In this instance the buffer only reaches one, then is set back to 0 before other threads (of the same gender) have time to occupy the bathroom. This problem is most common at the start of execution but has occurred sporadically throughout execution. However, after one or two swaps the buffer will typically fill as expected. Possible causes of this problem might involve how threads are created or how our arrival() method is implemented. The final (major) problem we found was that threads would give a notification that they exited bathroom in the wrong order. An example of this is sometimes when departing the bathroom thread, A will take 3 seconds and Thread B will take 6 yet thread B prints its exit message first. At this point in time it is unknown weather the problem cause by an ill placed print statement or a flawed depart method. Either way this problem will occur periodically.

**Implemented in JAVA using Eclipse**

<https://github.com/Adam1400/MultiThread440>