**Homework 4**

**Jacob Hedén Malm**
**9804051499**
**jacmalm@kth.se**
**Concurrent Programming**

For this assignment I chose the unisex bathroom problem. I had successfully finished the same problem in the previous task using semaphores, so I knew how to structure the logic around controlling access to bathroom in a fair, deadlock-free way.

I moved the synchronization measures to a separate class called Monitor.cpp and made a header file called Monitor.h. This was my first time doing OOP programming with C++, so I found it useful to learn how to do this.

The Monitor consists of 4 global variables, men/womenUsingBathroom and men/womenWaiting. For synchronization, I substituted the previous 2 signalling/wait semaphores with two condition variables, and the binary locking semaphore for a mutex lock. The lock fills exactly the same function as in the previous assignment. The wait semaphores were initialized to zero, and when a process wished to enter the bathroom, they tried to decrement these semaphores until they were signalled by another process that they were free to use the bathroom. This operation was substituted with waiting on a condition variable. The incrementing of the semaphore was substituted with another process signalling the condition variable, letting a process waiting on it through.

The monitor consists of 4 public methods controlling access to the bathroom. When a man/womanEnter/Exit. When a process wants to enter, they call their genders Enter method. Here, the monitor checks the variables in the same manner as in the previous assignment, and allows the monitor to pass the condition variable when it is allowed to enter the bathroom.

The process of using the bathroom is not part of the monitor class. A process using the bathroom generates a random time to sleep, and returns. When a process exits the bathroom, it calls its genders Exit method. This method updates the variables, and if necessary signals to the other gender that they are free to start using the bathroom.