Author: Przemek Gardias (pmgardias)

Running:

To compile the program, run "make clean all" (clean is only used if re-compiling) in the extracted directory. In order to run the program, run "./sim 100 10 15 25" (these are example parameters, used in Test #1). The output of the program will print in the shell.

Tests:

The three tests for which output is included for are as follows:

Test #1 - ./sim 100 10 15 25

Test #2 - ./sim 250 30 5 5

Test #3 - ./sim 1000 15 15 15

Each of these tests have example output provided in the form of an ouputX.txt (X equals the test number) file bundled in the zip. Results will not be identical between runs due to the indeterminism of the program since it relies on randomly generated numbers over a distribution centered around the provided mean (provided through the command line arguments).

Contents:

bathroom.c – Contains the bathroom module, consisting of the primary functions Enter(), Leave(), Initialize(), and Finalize() as well as helper functions for calculating elapsed time, changing state of the bathroom, and adding/removing occupants from the bathroom.

bathroom.h – Interface for the bathroom module, contains the structure for the bathroom object and prototypes for the functions that the simulator requires.

bathroomSim.c – Contains main() and Individual(), which control the behavior of the primary thread and the individual threads respectively.

Makefile – Contains the scripts to build the program given the source and header files.

The rest of the files are used to provide an explanation of the program through examples or write ups.

General:

The simulation program, at its core, simply creates the bathroom instance and the number of threads required. Each of the threads which the simulation creates execute Individual, which uses helper functions to generate the random numbers for number of loops, arrival time, and stay time. Each thread then tries to access the bathroom and then leave by calling the respective functions Enter() and Leave(), provided by the interface bathroom.h. Each of the threads individually handle printing out the statistics of their execution. Printing the thread’s statistics requires a lock so that each thread completes their printing without being interrupted.

Invariant:

The global bathroom instance has three states - vacant, male occupied, and female occupied which are checked after the lock is given. The state of the bathroom stays constant until two conditions are met: the bathroom is vacant and a thread attempts to enter it, in which case the state of the bathroom is changed to (occupied by) that gender. In the other case, the bathroom is emptied after being occupied by a gender, which causes threads which are waiting on the condition variable to then wake up and enter the bathroom. The invariance of the state allows checking against the state in order to ascertain whether a thread is able to enter the bathroom, or wait for the bathroom. Additionally, because of the way the queue functions, there should not be any threads waiting for the bathroom that are the same gender as the state of the bathroom. The queue should always consist solely of threads that are of the opposite gender.

Queue Behavior:

A queue is built up for the bathroom by the opposite gender when one gender is occupying the bathroom. The queue size is increased when a thread attempts to enter the bathroom while it is occupied by the opposite gender and the thread then waits on the condition variable to be able to enter the bathroom. Although an actual queue structure is not implemented, the program follows a similar functionality which is simulated by the condition variable. When the waiting threads are awoken by pthread\_cond\_broadcast, the scheduler determines which threads then proceed to run, rather than running in the order in which they were placed in a queue structure. This allows the "jumping the queue" functionality which is required that means waiting threads are able to access the bathroom if the same gender is currently occupying the bathroom, even if there are other threads waiting for the bathroom.

Notes:

Running tests with larger numbers of users (such as Test #3) will take a long time. In order to show functionality without wasting time, using a smaller number of users (<300) may be ideal.

Testing of this program was done on a virtual machine which was utilizing 6 cores.

The program was able to run with 20,000 users, although performance with such a large number of users was inconsistent. It seems that the machine which it was tested on was able to support ~23,000 maximum threads before throwing errors that creating threads failed.