Akshay Thejaswi
2/14/14

Project 3 Readme/Writeup

Files

- bathroom.h – header file for the bathroom module
- bathroom.c – implementation of bathroom module
- bathroomSim.c – bathroom test program
- Makefile
  - make : make all components
  - make all : make all components
  - make bathroomSim : compiles the simulation program to test the bathroom module
  - make bathroom.o : compile just the bathroom module
  - make clean
- Test output
  - test10 : output from first recommended arguements
  - test100 : output from second recommended arguements
  - test350 : output from third recommended arguements

Running

./bathroomSim [nUsers] [meanLoopCount] [meanArrival] [meanStay]

Recommended argument lists:

- 10 3 1 1 (short readable output)
- 100 20 10 10
- 350 100 10 10 (upper limit in my VM with 3 physical cores)

Controlling Access

The bathroom module of this project synchronized by using mutexes with condition variables. The mutexes are used in both the Enter() and Leave() functions so that only one thread may by entering or exiting the simulated bathroom at a time.

In Enter(), the thread attempts to lock the mutex, count_mutex. If the opposite gender is in the bathroom, the thread calls pthread_cond_wait with the condiotion empty. This causes the thread to wait until the bathroom is empty. Once the thread is gives access, it increments the occupancy count, statistics and sets the gender if the bathroom was previously empty.

In Leave() the thread attempt to lock the count_mutex at the beginning. Once inside the function body, it updates the count (demcrements) and if it was the last one out, it notifies any threads waiting on the empty condition by calling pthread_cond_broadcast. The thread then releases the count_mutex, and returns.

Invariants - There are a few invariants in the bathroom module.

- Only one thread is executing in Enter() or Leave() at a time.
- Count is never less than 0

- The mutex guarantees that the curernt thread is the only one editing the state of the bathroom

Multithreaded Test Progran

The test program start *nUsers* pthreads to test the bathroom module effectivley. After dispatching the threads, the main thread enters a loop waiting for the thread to finish. As each thread completes execution, it increments a global variable once. The main thread can be sure that all thread have finished running when the variable, finishCount is equal to the number of threads. It then proceeds to join() each thread that it spawned.

The Individual() threads test the bathroom module by calling Enter() Leave() and sleeping in the correct order and for the given number or loops. When the thread has finished its loop count, it locks a global mutex, and then prints out its statistics. All thread only print when they have locked the mutex, so it is guaranteed that no two thread will try to print to stdout at the same time. While locking the mutex, the thread also increments the global variable finish count to indicate to the main thread that it has completed excecution.