CS31202 - OPERATING SYSTEMS LABORATORY

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Assignment 5 : Semaphores

Authors:
Burra Nithish (20CS10018)
Kartik Pontula (20CS10031)
Astitva (20CS30007)
Rahul Mandal (20CS30039)

Date: March 22, 2023

Data Structures

We have modelled the entire `Hotel` as a struct containing the total occupancy (number of occupants since last clean-up) and a vector of `Room` structs. Each `Room` struct contains information on `pthread_t` of resident guest (if present), priority of that guest (if present), occupancy since last cleanup and total stay time since last cleanup.

In addition to the above, we have utilized the following synchronization data structures:

- An N-valued semaphore to ensure the number of resident guests doesn't exceed N
- A binary semaphore to enforce mutual exclusion whenever the `Hotel` struct is being read/written
- A binary semaphore to enforce mutual exclusion while printing to standard output

We have also utilized mutex locks and condition variables (one for each `Room`, each guest thread and each cleaner thread), and two `pthread_barrier` objects to implement a rendezvous point for cleaners.

Thread Behaviour

Main Thread

- Reads X, N, Y from command-line input, initializes necessary data structures.
- Assigns distinct random priorities to the guest threads.
- Calls `pthread_create()` to spawn Y guest and X cleaner threads, sending them necessary arguments.
- Waits for threads via `pthread_join()`.
- Destroys data and synchronization structures.

Guest Thread

- Sleeps for a random time (10-20 s), then calls `sem_trywait()` (non-blocking semaphore wait call) on the N-valued semaphore. There can be two outcomes:
  - Semaphore decremented: Guest proceeds to find an empty room and occupy it.
  - Semaphore is already 0: Guest attempts to find a lower priority guest in the hotel to evict. There can be two outcomes here:

    * No such guest exists: Guest has no choice but to call `sem_wait()` again on the primary semaphore and stay blocked until another guest vacates.
    * A target room found: Guest simply updates the target `Room` object with new credentials and signals the previous occupant to evict using a conditional broadcast on the condition variable corresponding to that `Room`. The primary semaphore is unchanged here.

- After a guest has acquired a room, it performs a `pthread_cond_timedwait()` where the timer here is the duration of stay. It keeps waiting for itself to be signalled by another guest to evict (if any), else it times out and vacates the room normally.
- Whether a guest finishes its stay normally or gets evicted, it updates the `Room`'s stay time attribute. For normal completion it also marks the `Room` as empty.
- When the guest is unable to book a room (because there exists no unoccupied room with occupancy i 2) then it waits on a barrier until all guests also wait on that barrier. Once every guest reaches the barrier, this triggers the cleaners to proceed with clean-up (since total occupancy of the `Hotel` is 2N and every room is vacated).
- In the meantime the guests perform a conditional wait for the total occupancy to be restored to 0. (i.e. they wait for clean-up).
- After cleanup the guests repeat the aforementioned behaviours.

Cleaner Thread

- Waits for `Hotel`'s occupancy to reach 2N **AND** for all guests to finish vacating. After that happens, the cleaners proceed to perform cleanup. This synchronization is ensured with a barrier that breaks when X cleaner threads + Y guest threads reach the `pthread_barrier_wait()` call.
- Each cleaner acquires a random uncleaned `Room`.
- It marks the room as being cleaned (by setting its occupancy to -1) and ”cleans” it for a duration of k times its stay time.
- After waking up it resets the `Room` and tries to find another `Room` to clean.
- This process is repeated until all `Room`s are cleaned. After this they just broadcast the guests to wake up and start booking again.