Operating Systems Laboratory (CS39002)
Spring Semester 2021-2022

Members:
Utsav Basu (20CS30057)
Swarup Padhi (20CS30062)
Rounak Saha (20CS30043)
Anamitra Mukhopadhyay (20CS30064)

Assignment 5: Usage of Semaphores to synchronize between threads

DESIGN DOC

MACROS

1. **REQTIME**
   Definition: #define REQTIME 10

2. **STAYTIME:**
   Definition: #define STAYTIME 20

GLOBAL VARIABLES:

1. **constexpr int ROOM_SIZE{2};**

   Description: Maximum occupants in a room before cleaning is required

2. **int32_t numRooms;**

   Description: N, number of rooms

3. **int32_t numGuests;**

   Description: Y, number of guests

4. **int32_t numCleaners;**

   Description: X, number of cleaners

5. **vector<pthread_cond_t> guest_cond;**

Description: vector of conditional variables for each guest thread, size: numGuests

6. **vector<pthread_mutex_t> guest_mutex;**

   Description: vector of mutex locks associated with the conditional variables of each guest thread, size: numGuests

7. **vector<int32_t> pr_guests;**

   Description: vector of priority values of guests, size: numGuests

8. **Hotel *hotel;**

   Description: pointer to a global `Hotel` datastructure used throughout the program (Hotel datastructure detailed in next section)

DATA STRUCTURES USED:

1. **Room**

   Fields:

Name	Type	Purpose
occupancy	int32_t	state of a room: 0 means clean, 1 means about to get dirty, 2 means dirty (equal to number of guests who have completed their stay in this room after last cleaning)
totalTime	int32_t	time elapsed in an occupied state since last cleaning
guest	int32_t	index of guest currently staying in this room, -1 if empty
guestPriority	int32_t	priority of guest currently staying in this room, -1 if

		empty
room_mutex	pthread_mutex_t	mutex_lock to grant exclusive access to fields of the structure to any thread

Methods:

1. **void cleanRoom();**

   Description: intended to be called by the cleaner thread, sets the fields to a clean state (`occupancy` and `totalTime` to 0, `guest` and `guestPriority` to -1), no need to lock since only one cleaner thread can access a room at a time

2. **bool allotGuest(int32_t guest, int32_t priority);**

   Description: checks for allotment of the room to a guest of index **guest** and which has priority **priority**, room is denied only in the following cases: either the room is dirty OR the `guestPriority` is >= **priority**. In case the request is successful and the currently occupying guest is being replaced due to lower priority, the corresponding guest thread (who would be performing a pthread_cond_timedwait for a time equal to the period of its requested stay) is signaled to wake up. The fields of the room are set to denote the new occupancy of **guest**. The room is kept locked with `room_mutex` throughout the time its fields are being checked or modified

   Returns:
   True if the room could be alloted to guest, False otherwise

3. **void updateTotalTime(int32_t time);**

   Description: locks the room and increments the `totalTime` field by **time**

4. **void checkoutGuest(int32_t time);**

Description: updates `totalTime` field using a call to updateTotalTime(**time**), then locks the room and sets the other fields to denote an empty room, intended to be called when a guest completes its stay successfullly

2. Hotel

Fields:

Name	Type	Purpose
rooms	vector<Room> (size: numRooms)	an array of rooms maintaining current information of each room accessible to the underlying data structures and methods
requestLeft	sem_t	a semaphore, initialized as ROOM_SIZE * numRooms, at any instant the semaphore counts the ROOM_SIZE - total number of guest requests which have been accepted
roomAllot_mutex	pthread_mutex_t	mutex lock to grant excessive access to the entire array of rooms for allotting to a guest thread
cleaner_cond	pthread_cond_t	a conditional variable to signal the cleaner threads to start working
cleaner_mutex	pthread_mutex_t	mutex lock associated with the cleaner conditional variable
cleaners	vector<pthread_t> (size: numCleaners)	cleaner threads
CleanerSem	sem_t	semaphore, counts upto numCleaners, cleaner threads wait on it, signaled by guest threads

		when more than permitted requests accepted and rooms need to be cleaned
roomToClean	int32_t	only relevant when rooms are being cleaned by cleaner threads, denotes the immediate next dirty room not yet cleaned, value is -1 rest of the time
roomsCleaned	int32_t	only relevant when rooms are being cleaned by cleaner threads , denotes the number of rooms whose cleaning is complete (including sleep time), value is -1 rest of the time

Methods:

1. **void startCleaners();**

   Description: spawns the cleaner threads

2. **bool allotRoom(int32_t guest, int32_t priority);**

   Description: checks for allotment of any room to a guest of index **guest** and which has priority **priority**, if possible. The room array is locked with the roomAllot_mutex, and then traversed to find a most suitable room where the **guest** may be allotted, the most suitable room is defined as the room(s) that are clean and have `guestPriority` < **priority** (includes the case where the room is empty where `guestPriority` = -1). It then checks if the most suitable room could be actually allotted to the guest by calling allotGuest() for rooms[most_suitalbe], clearly if this room could not be given to **guest**, none can be given and the request has to be declined. It then unlocks the roomAllot_mutex.

   Returns:
   Room number (number of the most_suitable room as we defined) if the room could be alloted to **guest**, -1 otherwise

3. **void checkoutGuest(int32_t roomNumber, int32_t time);**

   Description: calls Room::checkoutGuest(**time**) on the room indexed by **roomNumber**

4. **bool checkGuestInHotel(int32_t roomNumber, int32_t gid);**

   Description: checks if the guest indexed by **gid** is currently occupying room indexed by **roomNumber**, useful when the guest thread waits for its duration of stay on a pthread_cond_timedwait to detect spurious wakeups

   Returns: True if **gid** is occupying **roomNumber**, False otherwise

5. **void updateTotalTime(int32_t roomNumber, int32_t time);**

   Description: calls Room::updateTotalTime(**time)** on room[**roomNumber**]

WORKFLOW

Guest thread:
void *guestThread(void *arg); (defined in guest.cpp)

Description:
- Receives as argument the index of the guest to which this thread belongs to.
- Runs a while(true) loop with the following actions:
- Sleep for a random time between 10-20 seconds
- Acquires `cleaner_mutex` lock
- Proceeds to make a request for a room, checks the value of `requestLeft` semaphore,
  - if found 0 it means the number of accepted room requests >= numRooms * ROOM_SIZE(which in our case is 2), this means all rooms are dirty and the cleaner threads are to be called immediately:
  - checks the value of hotel->roomToClean,
    - if found -1 then the cleaner threads have not been called by any guest yet, so sets hotel->roomToClean and hotel->roomsCleaned to 0 and signals `CleanerSem` numRooms number of times (`CleanerSem`

is decremented by 1 each time a room is cleaned, so it should be initialized with the value numRooms before cleaning starts). Before signalling the cleaners, any guests who might be sleeping (as a part of their stay) are signalled and made to checkout their room
            - if not found -1, it means some other guest thread has already done the necessary stuff to start the process of cleaning (setting the required cleaner fields, signalled the sleeping guest threads and the cleaners)
        - starts wait on semaphore `hotel->requestLeft` till the next valid request can be made i.e. after all rooms are cleaned
    - if `hotel->requestLeft` is not yet 0, the guest goes ahead to make a request for a room to the hotel (call to hotel->allotRoom)
    - if it returns -1 (no room could be allotted to this guest, it loops back to the start of the while loop to start a fresh random sleep)
    - if the request was successful the semaphore `hotel->requestLeft` is decremented by 1 using a call to sem_wait
    - generates a random stay duration and starts a pthread_cond_timedwait for that duration, may be woken up for being replaced by some guest of higher priority or the cleaning process starting
    - a return with error value ETIMEDOUT from pthread_cond_timedwait signifies successful completion of its stay in the room for the entire duration of time it requested the room for, and then checkouts the room
    - checks if broken from the pthread_cond_timedwait loop due to being replaced by another high priority guest
    - loops back to the start of the while(true) loop to start a random sleep

Cleaner thread:
void *cleanerThread(void *arg); (defined in cleaner.cpp)

Description:
- Receives the index of the cleaner to which this thread belongs to
- Runs a while(true) loop with the following actions:

- Wait on the semaphore `hotel->CleanerSem`, which will be signalled only when cleaning is required
- finds out which room to clean from `hotel->roomToClean` and increments its value by 1
- cleans (sleeps) for time equal to `totalTime` of that room and then sets the fields of the selected room to that of clean room by a call to Room::cleanRoom()
- increments `hotel->roomsCleaned`
- if `hotel->roomsCleaned` reaches numRooms this means the entire cleaning process is over, reset `hotel->roomToClean` and `hotel->roomsCleaned` to -1 and signal `hotel->requestLeft` ROOM_SIZE * numRooms numbers of times to let the guest threads make valid requests again. This marks the end of the cleaning process and this part of code is only executed by the cleaner thread that cleans the last room.

Main thread:
int main(int argc, char const *argv[]) (defined in main.cpp)

Description:
- Uses arguments of the main function to initialize numRooms, numCleaners, numGuests and resizes the vectors accordingly
- Initializes the global mutexes and conditional variables
- Creates a Hotel object and spawns the guest and the cleaner threads
- Flushes stdout every 5 seconds

To run: $ ./main <cleaner_count> <guest_count> <room_count>