Managing a Tourist Attraction with Semaphores

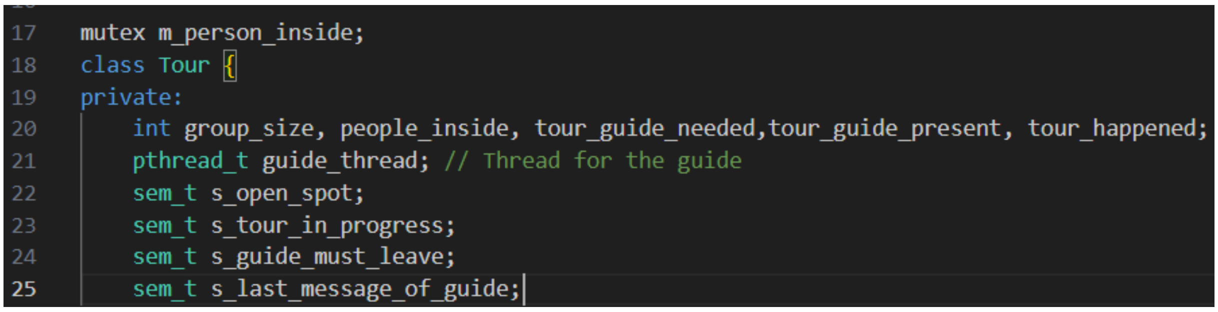
Goal

In this homework the goal is to build a Tour class that will handle tourists coming in as threads, check if there are enough of them and decide if a tour can take place. This is quite similar to the classic roller coaster problem.

There are three main problems that we are expected to solve with semaphores (some of them could be done with mutexes. However, we prefer a binary semaphore mostly for educational purposes). Firstly, no more than the specified group size of people can join the tour or get in the attraction site. Secondly when a tour starts no new people can enter the site, a result of the first rule. When threads will leave the site they must wait for the guide to announce the end of the tour. Also a non-specified tidbit I noticed is the last person to tell that there are no people left is the guide when there is one (I am telling this because this itself requires a new synchronization).

Tour Class:

First I will talk about the properties of this class to paint a background to our beautiful, to be canvas of synchronization.



```
mutex m_person_inside;
class Tour {
private:
    int group_size, people_inside, tour_guide_needed,tour_guide_present, tour_happened;
    pthread_t guide_thread; // Thread for the guide
    sem_t s_open_spot;
    sem_t s_tour_in_progress;
    sem_t s_guide_must_leave;
    sem_t s_last_message_of_guide;
```

Mutex:

m_person_inside: this is just to guarentee atomicity of an increment.

Pthread_t:

It will remember the guide thread if one thread becomes the guide.

Integers:

group_size: given to the program when the executable is called in the terminal, passed as a parameter of constructor. This tells how many people without a guide is needed for a tour to form.

people_inside: keeps track of how many people are inside, used to determine if the number is reached for a tour. Protected from race condition by mutex.

Tour_guide_needed: given to the program when the executable is called in the terminal, passed as a parameter of constructor. Tells if a guide is needed or not.

Tour_guide_present: Just a flag to tell whether there was a guide or not.

Tour_happened: A flag to tell whether the tour happened.

Semaphores(all start with s_):

s_open_spot: Starts with the count of group size. Any person going in decrements it. Once it reaches zero it will stop others from entering

s_tour_in_progress: A binary semaphore only meant to hold threads when a tour is happening:

s_guide_must_leave: An initially zero binary semaphore. It will let the guide be the first to leave.

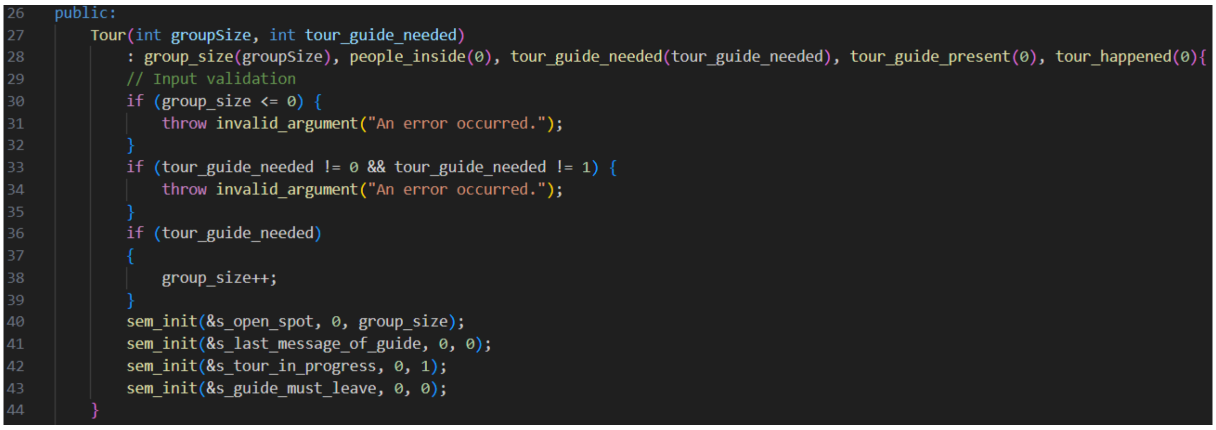
s_last_message_of_guide: An initially zero binary semaphore. Only meant to ensure the tidbit of output.

Functions:

Start:

We just declare this function. The executable fills this function.

Constructor:



```
26  public:
27      Tour(int groupSize, int tour_guide_needed)
28          : group_size(groupSize), people_inside(0), tour_guide_needed(tour_guide_needed), tour_guide_present(0), tour_happened(0){
29          // Input validation
30          if (group_size <= 0) {
31              throw invalid_argument("An error occurred.");
32          }
33          if (tour_guide_needed != 0 && tour_guide_needed != 1) {
34              throw invalid_argument("An error occurred.");
35          }
36          if (tour_guide_needed)
37          {
38              group_size++;
39          }
40          sem_init(&s_open_spot, 0, group_size);
41          sem_init(&s_last_message_of_guide, 0, 0);
42          sem_init(&s_tour_in_progress, 0, 1);
43          sem_init(&s_guide_must_leave, 0, 0);
44      }
```

At line 28 it initializes the parameters, and also initializes all other integers to zero.

Lines 30-35 checks the input.

Line 36-39 increases group size by one if a guide is needed. Since a guide requires the people excluding itself to be the number of group size. This is actually a specification of the homework, might as well could have been the inverse.

Now the initializations take 3 parameter. First is the semaphore, second is 0 if it is shared between threads of same process(our case), first count of the semaphore.

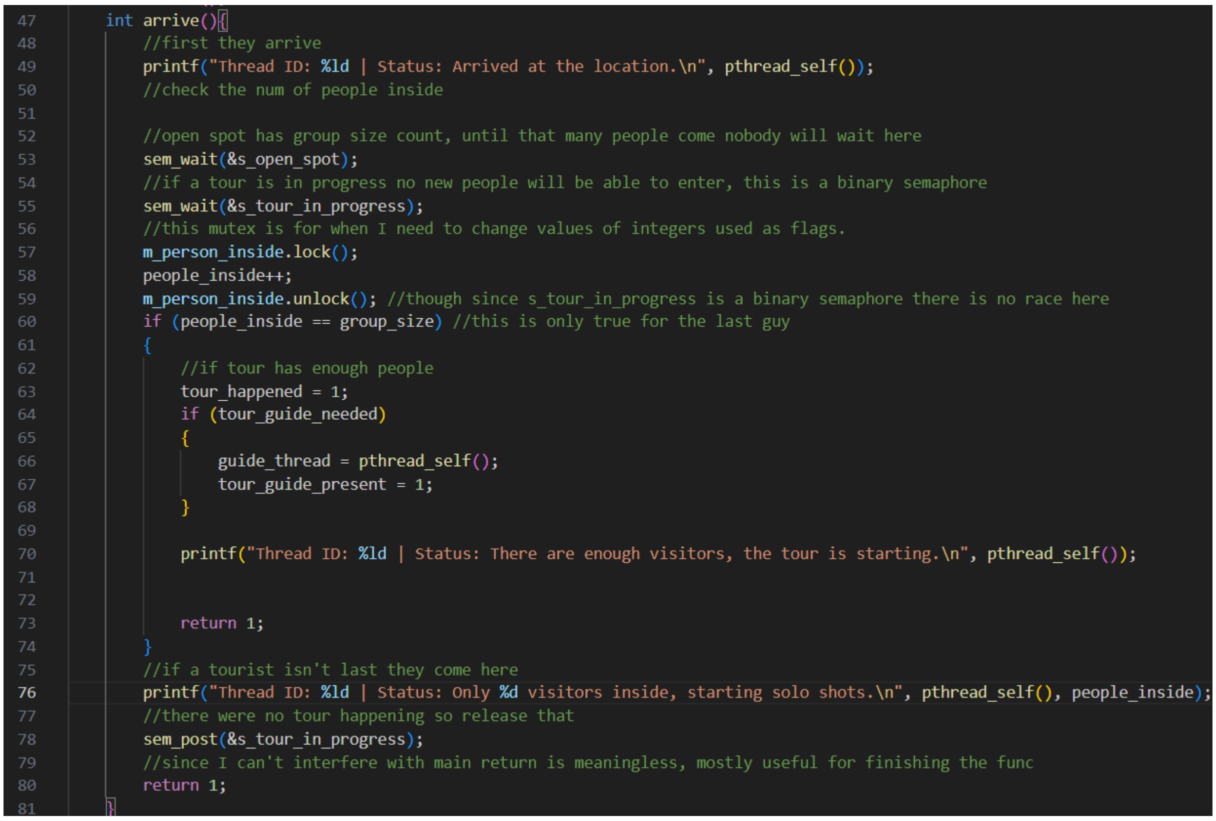
The count of open spot is the group size, so that semaphore is initialized to it.

s_last_message_of_guide is locked at first, this will make more sense in the leave function. Its purpose will be to make the guide thread wait until last guy leaves.

s_tour_in_progress is initialized as unlocked binary. This is because threads will first try to acquire it and if they don't start a tour they will relinquish this semaphore.

s_guide_must_leave(kind of funny name but it stuck with me) is a locked binary and follows the same logic with s_last_message_of_guide. Its purpose is to make the non-guide threads wait until guide outputs its announcement.

Arrive:



```
47      int arrive(){
48          //first they arrive
49          printf("Thread ID: %ld | Status: Arrived at the location.\n", pthread_self());
50          //check the num of people inside
51
52          //open spot has group size count, until that many people come nobody will wait here
53          sem_wait(&s_open_spot);
54          //if a tour is in progress no new people will be able to enter, this is a binary semaphore
55          sem_wait(&s_tour_in_progress);
56          //this mutex is for when I need to change values of integers used as flags.
57          m_person_inside.lock();
58          people_inside++;
59          m_person_inside.unlock(); //though since s_tour_in_progress is a binary semaphore there is no race here
60          if (people_inside == group_size) //this is only true for the last guy
61          {
62              //if tour has enough people
63              tour_happened = 1;
64              if (tour_guide_needed)
65              {
66                  guide_thread = pthread_self();
67                  tour_guide_present = 1;
68              }
69
70              printf("Thread ID: %ld | Status: There are enough visitors, the tour is starting.\n", pthread_self());
71
72
73              return 1;
74          }
75          //if a tourist isn't last they come here
76          printf("Thread ID: %ld | Status: Only %d visitors inside, starting solo shots.\n", pthread_self(), people_inside);
77          //there were no tour happening so release that
78          sem_post(&s_tour_in_progress);
79          //since I can't interfere with main return is meaningless, mostly useful for finishing the func
80          return 1;
81      }
```

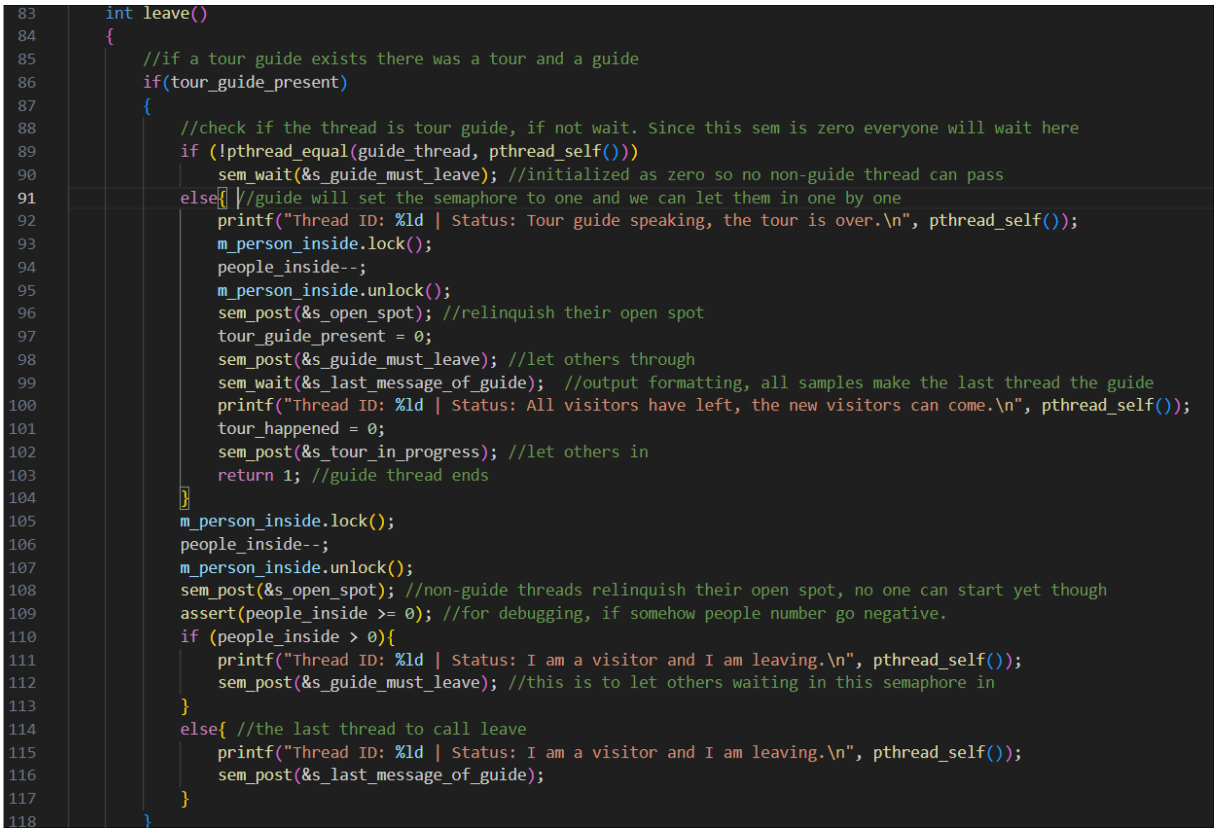
This function is the first to be called by the tourist threads.

- Line 49 is the needed output.

- Line 53, first checks if there is a spot for itself, if not it must wait. If it can enter it subtracts the count by one
- Line 55, checks if there is a tour, if there is it will wait. If not this will lock the s_tour_in_progress until this thread understands if it will cause a tour or not.
- Line57-59 just increment the people inside. Since if first semaphores are passes the person can enter the site.
- If the thread that just entered was the last needed person for a tour, it raises the flag tour_happened. And checks if a tour guide is needed. If it is, the class remembers it by assigning guide_thread to its id. And raises the flag tour_guide_present. Without relinquishing s_tour_in_progress, this thread effectively keeps it locked until everybody inside leaves and the last thread posts s_tour_in_progress.
- If there isn't enough people the thread starts solo shots and returns s_tour_in_progress count to 1 again.

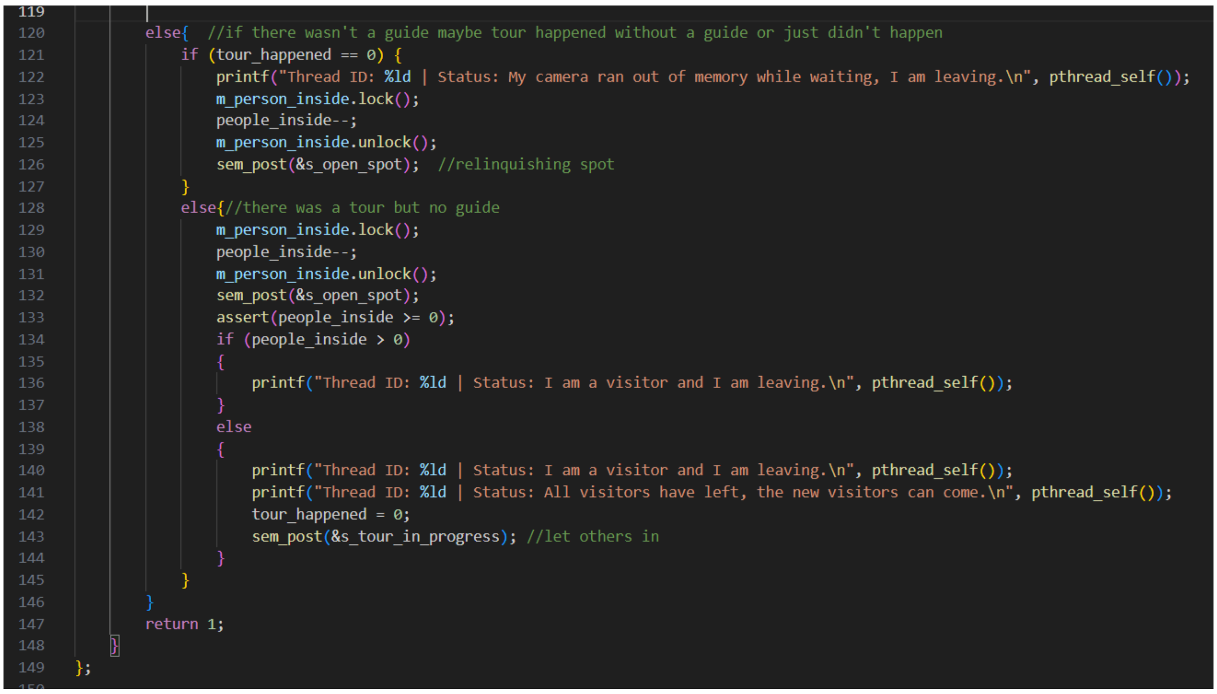
Leave:

This is the last function the threads call.



```
int leave()
{
    //if a tour guide exists there was a tour and a guide
    if(tour_guide_present)
    {
        //check if the thread is tour guide, if not wait. Since this sem is zero everyone will wait here
        if (!pthread_equal(guide_thread, pthread_self()))
            sem_wait(&s_guide_must_leave); //initialized as zero so no non-guide thread can pass
        else{ //guide will set the semaphore to one and we can let them in one by one
            printf("Thread ID: %ld | Status: Tour guide speaking, the tour is over.\n", pthread_self());
            m_person_inside.lock();
            people_inside--;
            m_person_inside.unlock();
            sem_post(&s_open_spot); //relinquish their open spot
            tour_guide_present = 0;
            sem_post(&s_guide_must_leave); //let others through
            sem_wait(&s_last_message_of_guide);  //output formatting, all samples make the last thread the guide
            printf("Thread ID: %ld | Status: All visitors have left, the new visitors can come.\n", pthread_self());
            tour_happened = 0;
            sem_post(&s_tour_in_progress); //let others in
            return 1; //guide thread ends
        }
        m_person_inside.lock();
        people_inside--;
        m_person_inside.unlock();
        sem_post(&s_open_spot); //non-guide threads relinquish their open spot, no one can start yet though
        assert(people_inside >= 0); //for debugging, if somehow people number go negative.
        if (people_inside > 0){
            printf("Thread ID: %ld | Status: I am a visitor and I am leaving.\n", pthread_self());
            sem_post(&s_guide_must_leave); //this is to let others waiting in this semaphore in
        }
        else{ //the last thread to call leave
            printf("Thread ID: %ld | Status: I am a visitor and I am leaving.\n", pthread_self());
            sem_post(&s_last_message_of_guide);
        }
    }
```

- Since this is quite a long function(due to my standards for readability, I can't make it any shorter) We will first discuss the first part of it. The case where there was a guide and hence there was a tour too. If this is the case first every thread that is not the guide will wait at line90. Since s_guide_must_leave is zero as initialized.
- The guide thread first makes the announcement. Decrements the people inside. Sets tour_guide_present flag to 0(for reusability, also it is done right after guide thread actually leaves to mimic real life). Posts s_open_spot since there is now a spot or others. Posts s_guide_must_leave and lets all other threads to enter one by one(this can of course be done so that they don't enter one by one. That however would require the semaphore to be more than binary. However, handling that would look and feel bad. Since I would have a loop dedicated to posting this semaphore.) And then waits for the last thread to finish with s_last_message_of_guide to write its last message. After writing that message we know that every thread has finished their jobs. So we post s_tour_in_progress to let a new tour begin
- If a thread isn't guide after waiting for the guide, they continue with first decrementing people inside. Then they post s_open_spot. After that if they are not the last thread they output their stuff and post s_guide_must_leave, thus letting others in.
- If the non-guide thread is the last after outputting they let the guide know everyone has left. This thread also doesn't post s_guide_must_leave because that has returned to its initial value already.



```
        
        else{  //if there wasn't a guide maybe tour happened without a guide or just didn't happen
            if (tour_happened == 0) {
                printf("Thread ID: %ld | Status: My camera ran out of memory while waiting, I am leaving.\n", pthread_self());
                m_person_inside.lock();
                people_inside--;
                m_person_inside.unlock();
                sem_post(&s_open_spot);  //relinquishing spot
            }
            else{//there was a tour but no guide
                m_person_inside.lock();
                people_inside--;
                m_person_inside.unlock();
                sem_post(&s_open_spot);
                assert(people_inside >= 0);
                if (people_inside > 0)
                {
                    printf("Thread ID: %ld | Status: I am a visitor and I am leaving.\n", pthread_self());
                }
                else
                {
                    printf("Thread ID: %ld | Status: I am a visitor and I am leaving.\n", pthread_self());
                    printf("Thread ID: %ld | Status: All visitors have left, the new visitors can come.\n", pthread_self());
                    tour_happened = 0;
                    sem_post(&s_tour_in_progress); //let others in
                }
            }
        }
        return 1;
    }
};
```

- If there wasn't a guide there are two cases. Maybe tour didn't happen, or the tour was guideless.

- Line 121 continues if the tour didn’t happen. It outputs, decrements itself from people inside and then posts an s_open_spot.
- Line 128 follows if there was a guideless tour. It decrements itself from people inside, posts s_opej_spot and then asserts people inside isn’t negative(for debugging). If the thread isn’t the last person it just outputs its leaving output. If it is the last thread it needs to output two things. Then sets tour_happened to zero. And then posts s_tour_in_progress so that a new tour can start.