# 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:

metin, yazılım, multimedya yazılımı, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturulduFirst 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: 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:

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.

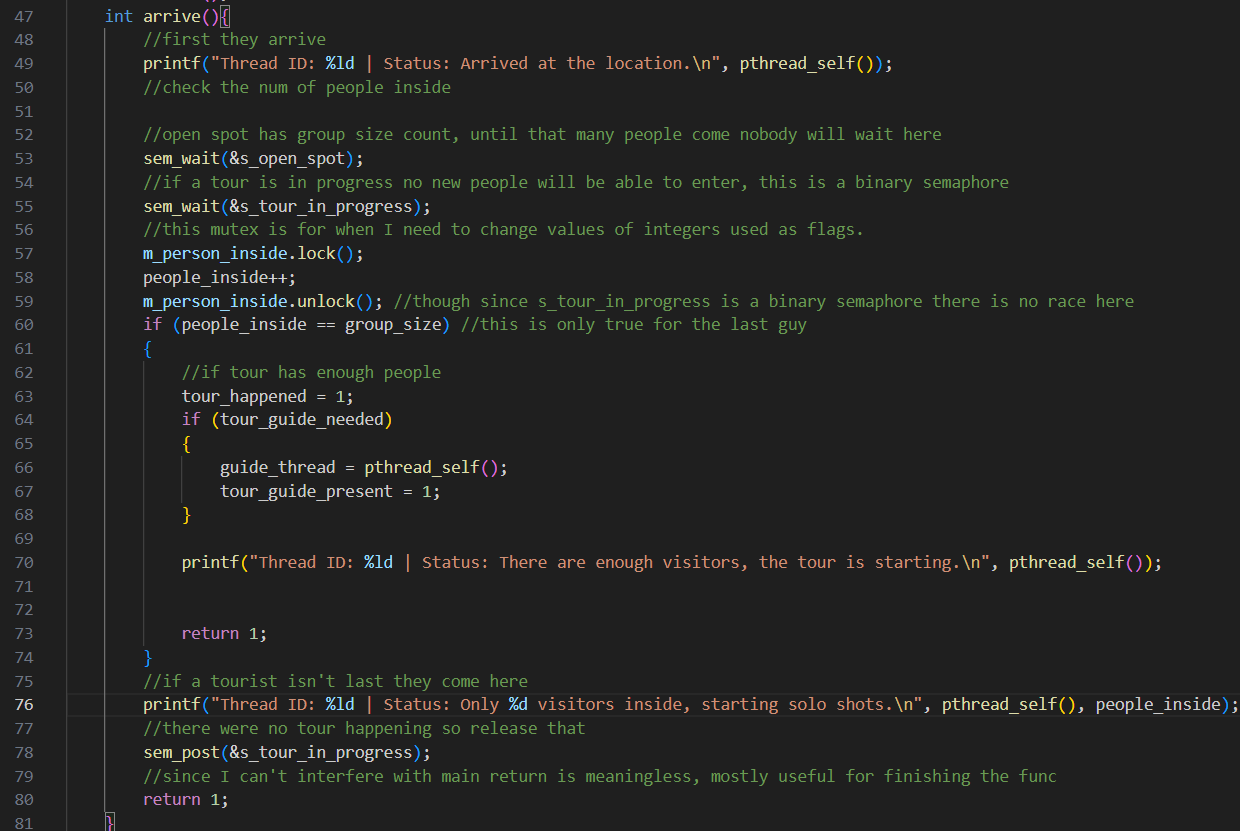
The count of open spot is the group size, so that semaphore is initialized to it.

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.

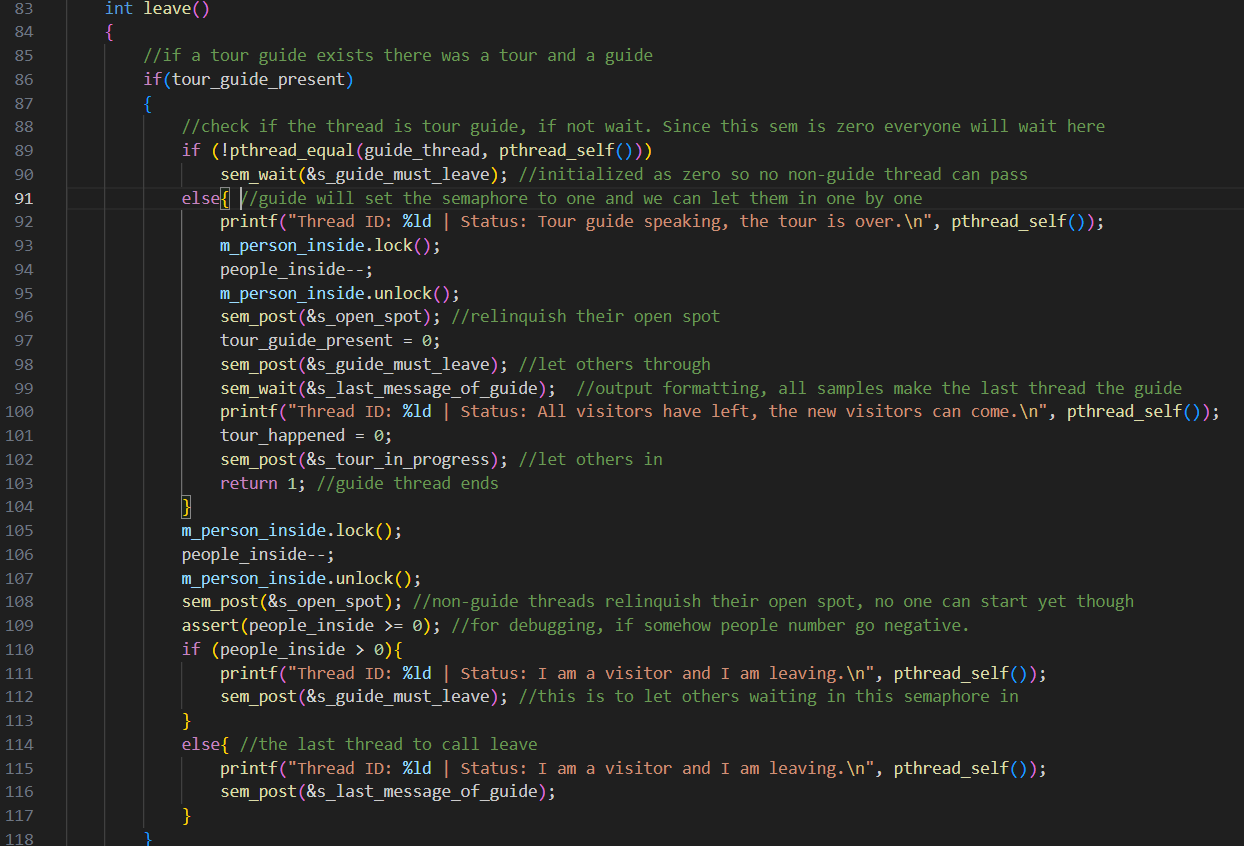
## Arrive:

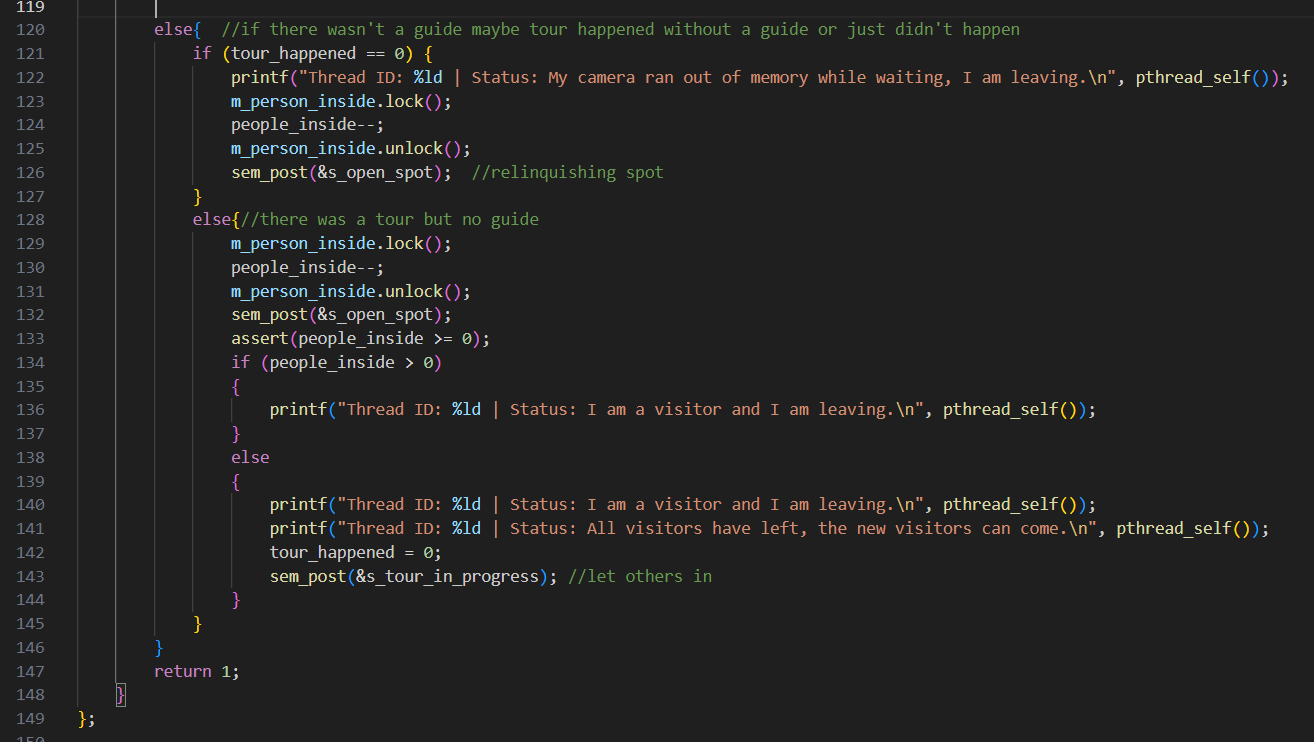


This function is the first to be called by the tourist threads.

* 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.

* 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.
* 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.