Thread Synchronization II
Parking Lot Control system

Søren Hansen <sha@ase.au.dk>
V1.7

Introduction

In this exercise you will implement a Parking Lot Control System (PLCS) which monitors a parking lot and grants access to cars that wishes to enter and exit the parking lot.

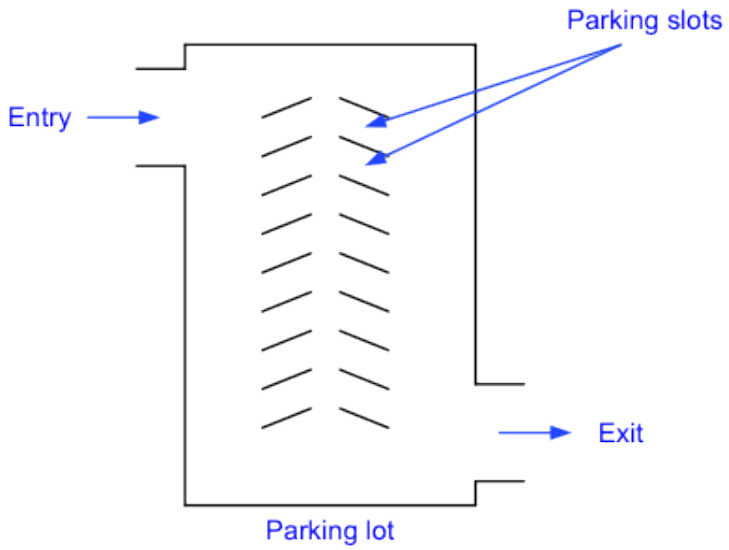
Prerequisites

Finished Exercise *Thread Synchronization I*.

Absolute requirements (Must haves for approval)

- Mutexes & Conditionals must be used to implemented the solution. E.g. using semaphores is not allowed.

This is a sketch of the PLCS:



A simple sketch of the PLCS

These are the requirements for the PLCS:

- Cars enter the parking lot, stay there for a while, and then exit the parking lot again. Then they wait a while before they re-enter the parking lot. This happens continuously.
- An arriving car must request permission to enter the parking lot from the *PLCS entry guard*. When permission is granted, the car may enter the parking lot.
- An exiting car must request permission to exit the parking lot from the *PLCS exit guard*. When access is granted, the car may exit the parking lot.
- Each car must be represented by a single thread, the same goes for the *PLCS entry guard* and the *PLCS exit guard*. Running the program with one car would therefore result in *3* threads.
- Each guard controls a *door* and when no car is waiting to enter, the *door* must be closed.

Thread Synchronization II
Parking Lot Control system

Søren Hansen <sha@ase.au.dk>
V1.7

Use the *Park-a-Lot 2000* example of the recent lecture as an example of how a car may interact with the *PLCS entry (or exit) guard* to request access to enter (or exit) the parking lot and receive permission to do so.

These exercises *must* be implemented using `mutex/conditional` constructs. Do note that if you want to use global variables then do it by all means.

Finally if design information is left out from you point of view... then consider it by design. Determine what you think should be done in that particular situation and make a note.

Exercise 1 Implement Park-a-Lot 2000 (%60)

Exercise 1.1 First step

Implement the PLCS and verify that it works with a single car that enters the parking lot, waits there for some time and exits the parking lot again. This behavior should continue until the program is terminated.

Before you implement anything write out your solution in pseudo code or flowcharts. This applies to all exercises...

Exercise 1.2 The grandiose test

Repeat the above test this time with multiple cars. Furthermore every car should wait a different amount of time in the parking lot before exiting. In this scenario the car park does not have an upper bound on the number of cars it has space for.

Verify that all cars are able to enter and exit as would be expected.

Explain what `pthread_cond_broadcast()` does and argue as to why you needed or didnt need it.

During the process of figuring out the solution, you will discover that the cars will seemingly *not* wait in line, but overtake each other on the way in or out. This is *not* part of the assignment to ensure an order. Never the less why does this happen, and can you think of an approach that would fix it?

Exercise 2 Extending PLCS, now with a limit on the number of cars (40%)

Before commencing on the exercise
This exercise is deliberately specified to award 40% despite the fact that it is extremely difficult. It is required that you start the exercise but not that you complete it. Use *Time-boxing*[1] when you strive to complete the exercise. By using this approach you ensure that you do not use an excessive amount of time on it.

It is not everyone that will complete it! In other words consider this a challenge :)

We now add an additional requirement to the PLCS:

[1] See `http://en.wikipedia.org/wiki/Timeboxing` for an explanation

- The entry guard must ensure that entry is not granted to a car if the parking lot is full. In that case cars wanting to enter must wait. This is implies that the car park has a maximum allowable capacity which must not be exceeded.

Extend your PLCS to handle this situation and verify that it actually does as you expect. Remember to test the scenario where a car leaves a full parking lot, enabling a waiting car to enter.

This last exercise is quite difficult, which is why careful planning is necessary before you code. Otherwise you will simply fail.

Hint: *The challenge here is to determine the signaling condition for waking on conditional variables.*