Wiki »

Exercise 5 - Thread Synchronization II - Parking Lot System ¶

Introduction

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

Exercise 1: Implement Park-a-Lot 2000 - Design(%20)

Here we show the pseudo code which we used to design the Park-a-Lot 2000 solution:

View Code

```
#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>

using namespace std;

void entryGuard(void *entry);
void exitGuard(void *exit);
void vehiclesThread();

int main()
{
    Init Mutex and so on.

    pthread_t t1, t2, t3;

    pthread_create(creating t1, call vehiclesThread);
    pthread_create(creating t2, call entryGuard);
    pthread_create(creating t3, call exitGuard);

    return 0;
}

void entryGuard(void *entry)
{
    lock(mutex);
    while (!vehicleWaiting)
        condWait(entry, mutex);

    openEntryGate();
    entryGateOpen = true;
    condSignal(entry);

    while (vehicleWaiting)
```

```
        condWait(entry, mutex);

    closeEntryGate();
    entryGateOpen = false;
    condSignal(entry);
    unlock(mutex);

}

void exitGuard(void *exit)
{
    lock(mutex);
    while (!waitExit)
        condWait(exit, mutex);

    openExitGate();
    openExitGate = true;
    condSignal(exit);

    while (waitExit)
        condWait(exit, mutex);

    closeExitGate();
    openExitGate = false;
    unlock(mutex);

}

void vehiclesThread()
{
    driveToEntryGate();
    lock(mutex);
    vehicleWaiting = true;
    condSignal(entry);

    while (!entryGateOpen)
        condWait(entry, mutex);

    driveIntoPLCS();
    vehicleWaiting = false;
    condSignal(entry);
    parkInPLCS(randomTime);

    waitExit = true;
    exitCond(exit);

    while (!exitGateOpen)
        condWait(exit, mutex);

    exitPLCS();
    waitExit = false;
    exitCond(exit);
    unlock(mutex);
}
```

Exercise 2: Implement Park-a-Lot 2000 - Implementation(%40)

In this part of the exercise, we will implement the Park-a-Lot 2000 system using threads in Linux:

Exercise 2.1: First step

Based on the first exercise, we will implement the PLCS and verify that it works with just a single car that enters PLCS, waits there for som time and then exits. This behaviour should continue until the program is terminated.

Here is the code for the PLCS.cpp file:

View PLCS.cpp

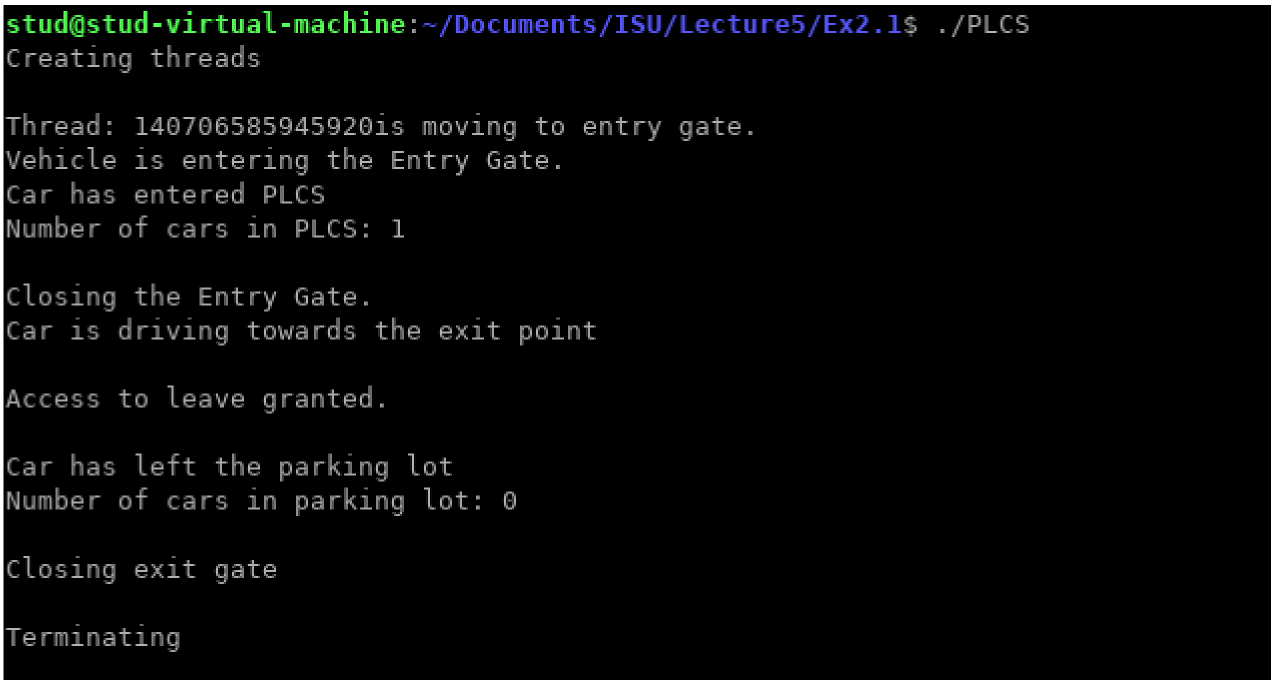
```
#include <pthread.h>
#include <iostream>
#include <stdlib.h>
#include <string>
#include <unistd.h>

using namespace std;

void *entryGuard(void *arg);
void *exitGuard(void *arg);
void *vehiclesThread(void *arg);

bool vehicleWaiting, exitWait, entryAccess, exitAccess = false;

pthread_mutex_t mut = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t entry;
pthread_cond_t exitLot;

int vehicleCount = 0;

int main()
{
    cout << "Creating threads\n" << endl;

    pthread_mutex_init(&mut, nullptr);

    pthread_t t1, t2, t3;

    pthread_create(&t1, nullptr, vehiclesThread, (void*)
    (intptr_t)pthread_self());

    pthread_create(&t2, nullptr, entryGuard, (void*)
    (intptr_t)pthread_self());

    pthread_create(&t3, nullptr, exitGuard, (void*)
        (intptr_t)pthread_self());

    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
    pthread_join(t3, NULL);

    cout << "Terminating\n" << endl;
    return 0;
}

void *entryGuard(void *arg)
{
    pthread_mutex_lock(&mut);

    while(!vehicleWaiting)
        pthread_cond_wait(&entry, &mut);

    entryAccess = true;
        cout << "Vehicle is entering the Entry Gate." << endl;
    pthread_cond_signal(&entry);

    while(vehicleWaiting)
        pthread_cond_wait(&entry, &mut);

    entryAccess = false;
    cout << "Closing the Entry Gate." << endl;
    pthread_mutex_unlock(&mut);
    pthread_cond_signal(&entry);

    return NULL;
}

void *exitGuard(void *arg)
{
    pthread_mutex_lock(&mut);

    while(!exitWait)
        pthread_cond_wait(&exitLot, &mut);

    exitAccess = true;
    cout << "Access to leave granted.\n" << endl;
    pthread_cond_signal(&exitLot);

    while(exitWait)
        pthread_cond_wait(&exitLot, &mut);

    exitAccess = false;
    cout << "Closing exit gate\n" << endl;
    exitAccess = false;
    pthread_mutex_unlock(&mut);

    return NULL;

}

void *vehiclesThread(void *arg)
{
    pthread_mutex_lock(&mut);
    cout << "Thread: " << long(arg) << "is moving to entry gate." << endl;
    vehicleWaiting = true;
    pthread_cond_signal(&entry);

    while(!entryAccess)
        pthread_cond_wait(&entry, &mut);
    vehicleCount++;
    cout << "Car has entered PLCS\n" << "Number of cars in PLCS: " << vehicleCount <
    pthread_cond_signal(&entry);

    vehicleWaiting = false;
    while(entryAccess)
        pthread_cond_wait(&entry, &mut);
    sleep(1);

    cout << "Car is driving towards the exit point\n" << endl;

    exitWait = true;
    pthread_cond_signal(&exitLot);
    while(!exitAccess)
        pthread_cond_wait(&exitLot, &mut);

    vehicleCount--;
    cout << "Car has left the parking lot\n" << "Number of cars in parking lot: " <<
    vehicleCount << "\n" << endl;

    exitWait = false;

    pthread_cond_signal(&exitLot);
    pthread_mutex_unlock(&mut);

    return NULL;
}
```

Og herunder kan vores Makefile ses:

View Makefile

```
SOURCES=PLCS.cpp
OBJECTS=$(SOURCES:.cpp=.o)
CXX=g++
EXE=PLCS
CXXFLAGS=-ggdb -I.

$(EXE): $(OBJECTS)
    $(CXX) -o $@ $< -pthread

%.o: %.cpp
    $(CXX) -c $< $(CXXFLAGS)

.PHONY: clean
clean:
    rm -f $(EXE) $(OBJECTS)
```

Her kan det ses at vi kører programmet på vores host:



```
stud@stud-virtual-machine:~/Documents/ISU/Lecture5/Ex2.1$ ./PLCS
Creating threads

Thread: 140706585945920is moving to entry gate.
Vehicle is entering the Entry Gate.
Car has entered PLCS
Number of cars in PLCS: 1

Closing the Entry Gate.
Car is driving towards the exit point

Access to leave granted.

Car has left the parking lot
Number of cars in parking lot: 0

Closing exit gate

Terminating
```

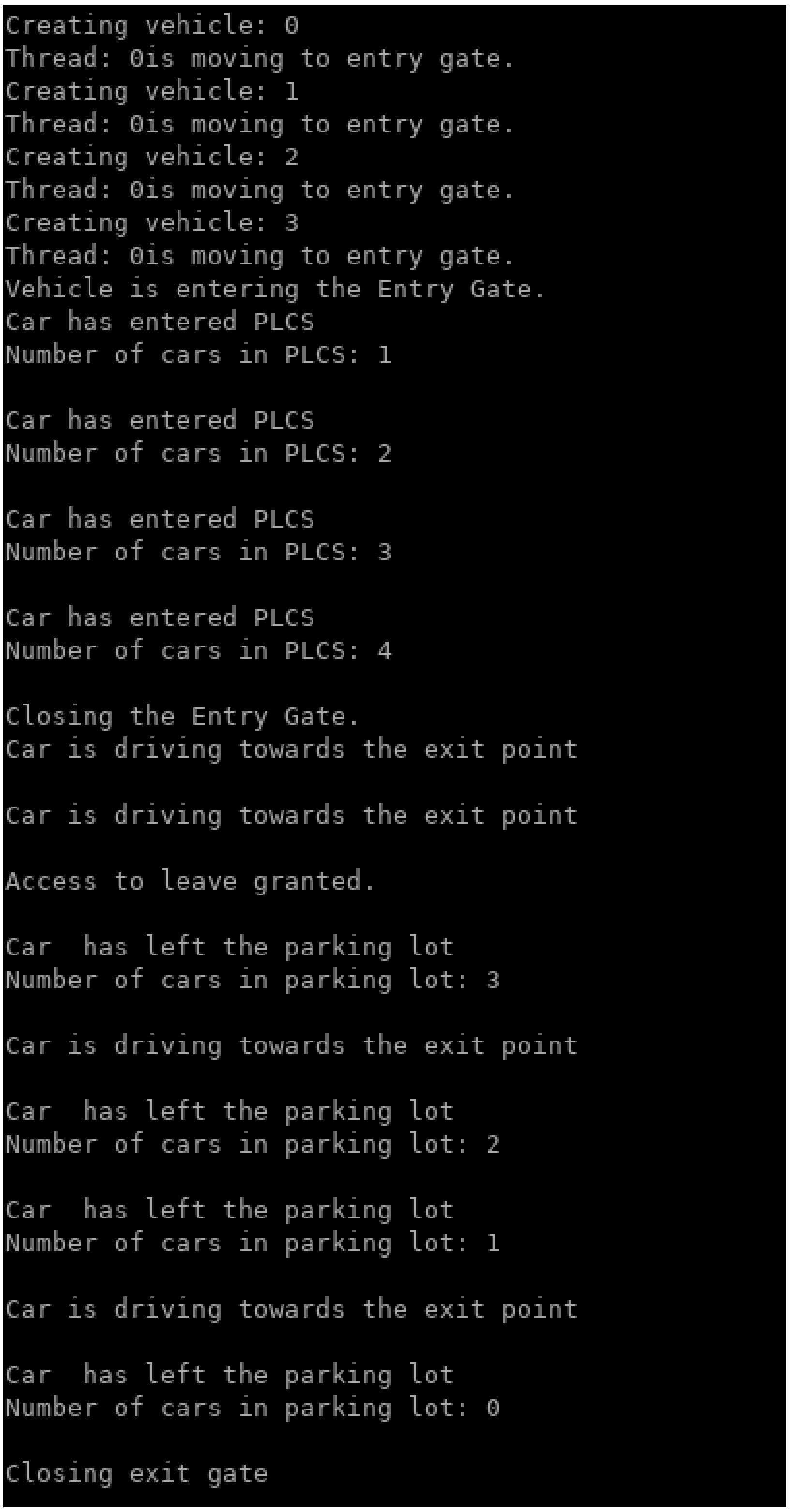
Exercise 2.2. The grandiose test

Now we must be able to handle multiple cars, with no upper bound to as how many can enter the parking lot. Here is the code for this exercise:

View PLCS2.cpp

```
#include <pthread.h>
#include <iostream>
#include <stdlib.h>
#include <string>
#include <unistd.h>
using namespace std;
void *entryGuard(void *arg);
void *exitGuard(void *arg);
void *vehiclesThread(void *arg);
bool entryAccess, exitAccess = false;
int entryWait, exitWait = 0;
pthread_mutex_t mut1 = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mut2 = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t entry;
pthread_cond_t exitLot;
int vehicleCount = 0;
int main()
{
    cout << "Creating threads\n" << endl;
        pthread_mutex_init(&mut1, nullptr);
    pthread_mutex_init(&mut2, nullptr);
    int numVehicles = 0;
    cout << "Enter number of vehicles: " <<endl;
    cin >> numVehicles;

        pthread_t t2, t3;
    pthread_t t1[numVehicles];

    for(int i=0;i<numVehicles;i++)
    {
        pthread_create(&t1[i], nullptr, vehiclesThread, nullptr);
        cout << "Creating vehicle: " << i << endl;
    }

        pthread_create(&t2, nullptr, entryGuard, nullptr);

        pthread_create(&t3, nullptr, exitGuard, nullptr);

    //pthread_join(t1[input], NULL);
    pthread_join(t2, NULL);
    pthread_join(t3, NULL);

    cout << "Terminating\n" << endl;

        return 0;
}
void *entryGuard(void *arg)
{
    while(1)
    {
    pthread_mutex_lock(&mut1);

    while(entryWait == 0)
        pthread_cond_wait(&entry, &mut1);

    entryAccess = true;
        cout << "Vehicle is entering the Entry Gate." << endl;
    pthread_cond_broadcast(&entry);

    while(entryWait != 0)
        pthread_cond_wait(&entry, &mut1);

    cout << "Closing the Entry Gate." << endl;
    entryAccess = false;

    pthread_mutex_unlock(&mut1);
    pthread_cond_broadcast(&entry);
    }
    return NULL;
}
void *exitGuard(void *arg)
{
    while(1)
    {
    pthread_mutex_lock(&mut2);

    while(exitWait == 0)
        pthread_cond_wait(&exitLot, &mut2);

    exitAccess = true;
    cout << "Access to leave granted.\n" << endl;
    pthread_cond_broadcast(&exitLot);

    while(exitWait != 0)
        pthread_cond_wait(&exitLot, &mut2);

    cout << "Closing exit gate\n" << endl;
    exitAccess = false;
    pthread_mutex_unlock(&mut2);
    pthread_cond_broadcast(&exitLot);
    }
    return NULL;
}
void *vehiclesThread(void *arg)
{
    //while(1)
    {
    pthread_mutex_lock(&mut1);
    cout << "Thread: " << long(arg) << "is moving to entry gate." << endl;
    entryWait++;
    pthread_cond_signal(&entry);

    while(!entryAccess)
        pthread_cond_wait(&entry, &mut1);

    vehicleCount++;
    cout << "Car has entered PLCS\n" << "Number of cars in PLCS: " << vehicleCount <
    pthread_cond_signal(&entry);

    entryWait--;
//      while(entryAccess)
//          pthread_cond_wait(&entry, &mut1);
    pthread_mutex_unlock(&mut1);

    sleep(1);

    pthread_mutex_lock(&mut2);
    cout << "Car is driving towards the exit point\n" << endl;

    exitWait++;
    pthread_cond_signal(&exitLot);

    while(!exitAccess)
        pthread_cond_wait(&exitLot, &mut2);

    vehicleCount--;
    cout << "Car " /*<< pthread_self()*/ << " has left the parking lot\n" << "Number
    vehicleCount << "\n" << endl;

    exitWait--;
    pthread_cond_signal(&exitLot);
    pthread_mutex_unlock(&mut2);
    }
    return NULL;
}
```

And here we see it running in therminal:



```
Creating vehicle: 0
Thread: 0is moving to entry gate.
Creating vehicle: 1
Thread: 0is moving to entry gate.
Creating vehicle: 2
Thread: 0is moving to entry gate.
Creating vehicle: 3
Thread: 0is moving to entry gate.
Vehicle is entering the Entry Gate.
Car has entered PLCS
Number of cars in PLCS: 1

Car has entered PLCS
Number of cars in PLCS: 2

Car has entered PLCS
Number of cars in PLCS: 3

Car has entered PLCS
Number of cars in PLCS: 4

Closing the Entry Gate.
Car is driving towards the exit point

Car is driving towards the exit point

Access to leave granted.

Car  has left the parking lot
Number of cars in parking lot: 3

Car is driving towards the exit point

Car  has left the parking lot
Number of cars in parking lot: 2

Car  has left the parking lot
Number of cars in parking lot: 1

Car is driving towards the exit point

Car  has left the parking lot
Number of cars in parking lot: 0

Closing exit gate
```

Explain what pthread_cond_broadcas() does:

pthread_cond_broadcast() unblocks all threads, which were locked by a given conditional variable. Unlike pthread_cond_wait(), which unblocks at leat one thread, who is waiting on a given conditional variable.

Why does the cars not wait in line but overtake each other? How can it be fixed?

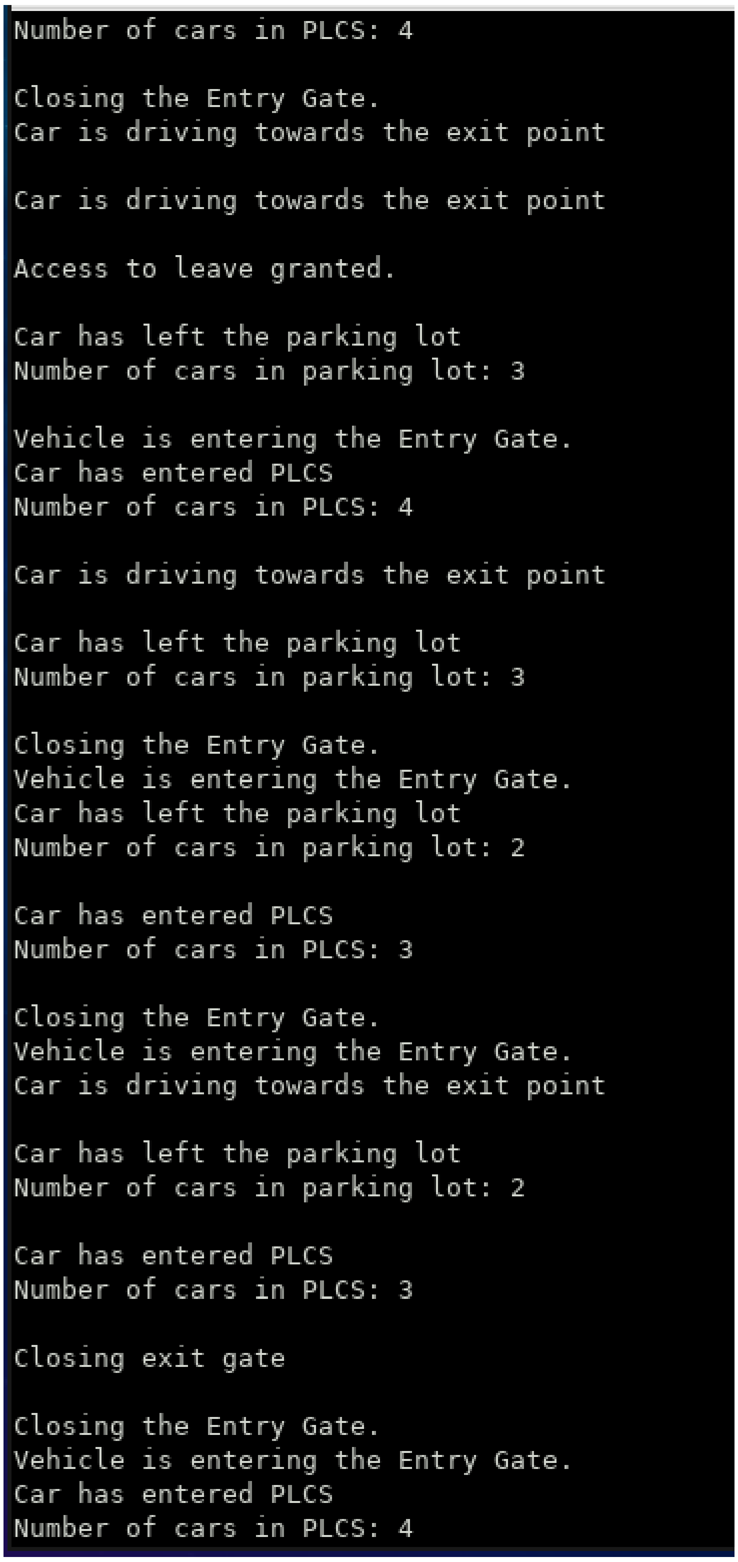
All the cars / vehicles are not created at the same time, which is why some will start before others. To fix it you could try creating them using a counter, which would only increment when the car before has entered.

Exercise 3: Extending PLCS, now with a limit on the number of cars (40%)

To solve the problem regarding a finite number of parking spaces, we made the variable parkingSpaces, which is then compared with the variable vehicleCount in the entry guard in an empty while loop. We had to make some changes to the entry guard and vehicles thread to ensure that only one vehicle could enter at the same time which apparently was a problem in the earlier versions. The problem was solved by making ad extra signal "entrying" so the vehicle threads couldn't signal each other and all in all fixing the signaling system.

```
#include <pthread.h>
#include <iostream>
#include <stdlib.h>
#include <string>
#include <unistd.h>

using namespace std;

void *entryGuard(void *arg);
void *exitGuard(void *arg);
void *vehiclesThread(void *arg);

bool entryAccess, exitAccess = false;
int entryWait, exitWait = 0;

pthread_mutex_t mut1 = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mut2 = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t entry;
pthread_cond_t entrying;
pthread_cond_t exitLot;

int vehicleCount = 0;
int parkingSpaces = 4;

int main()
{
    cout << "Creating threads\n" << endl;

    pthread_mutex_init(&mut1, nullptr);
    pthread_mutex_init(&mut2, nullptr);
    int numVehicles = 0;
    cout << "Enter number of vehicles: " <<endl;
    cin >> numVehicles;

    pthread_t t2, t3;
    pthread_t t1[numVehicles];

    for(int i=0;i<numVehicles;i++)
    {
        pthread_create(&t1[i], nullptr, vehiclesThread, nullptr);
        cout << "Creating vehicle: " << i << endl;
    }

    pthread_create(&t2, nullptr, entryGuard, nullptr);

    pthread_create(&t3, nullptr, exitGuard, nullptr);

    pthread_join(t2, NULL);
    pthread_join(t3, NULL);

    cout << "Terminating\n" << endl;
    return 0;
}

void *entryGuard(void *arg)
{
    while(1)
    {
        while( vehicleCount >= parkingSpaces)
        {}
        pthread_mutex_lock(&mut1);

        while(entryWait == 0)
            pthread_cond_wait(&entrying, &mut1);

        entryAccess = true;
        cout << "Vehicle is entering the Entry Gate." << endl;
        pthread_cond_signal(&entry);

        pthread_cond_wait(&entrying, &mut1);

        cout << "Closing the Entry Gate." << endl;

        entryAccess = false;

        pthread_mutex_unlock(&mut1);
    }
    return NULL;
}

void *exitGuard(void *arg)
{
    while(1)
    {
    pthread_mutex_lock(&mut2);

    while(exitWait == 0)
        pthread_cond_wait(&exitLot, &mut2);

    exitAccess = true;
    cout << "Access to leave granted.\n" << endl;
    pthread_cond_broadcast(&exitLot);

    while(exitWait != 0)
        pthread_cond_wait(&exitLot, &mut2);

    cout << "Closing exit gate\n" << endl;
    exitAccess = false;
    pthread_mutex_unlock(&mut2);
    pthread_cond_broadcast(&exitLot);
    }
    return NULL;

}

void *vehiclesThread(void *arg)
{
    //while(1)
    {
    pthread_mutex_lock(&mut1);
    cout << "Thread: " << long(arg) << "is moving to entry gate." << endl;
    entryWait++;
    pthread_cond_signal(&entrying);

    while(!entryAccess)
    {
    pthread_cond_wait(&entry, &mut1);
    }

    vehicleCount++;
    cout << "Car has entered PLCS\n" << "Number of cars in PLCS: " << vehicleCount <

    pthread_cond_signal(&entrying);

    entryWait--;
    pthread_mutex_unlock(&mut1);

    sleep(1);

    pthread_mutex_lock(&mut2);
    cout << "Car is driving towards the exit point\n" << endl;

    exitWait++;
    pthread_cond_signal(&exitLot);

    while(!exitAccess)
        pthread_cond_wait(&exitLot, &mut2);

    vehicleCount--;
    cout << "Car has left the parking lot"  << endl;
    cout << "Number of cars in parking lot: " << vehicleCount << endl << endl;

    exitWait--;

    pthread_cond_signal(&exitLot);
    pthread_mutex_unlock(&mut2);
    }

    return NULL;
}
```

On the picture below is it shown how the vehicle count doesn't rise above 4, which is the amount of parking spaces chosen for this test. This means that the code works.



```
Number of cars in PLCS: 4

Closing the Entry Gate.
Car is driving towards the exit point

Car is driving towards the exit point

Access to leave granted.

Car has left the parking lot
Number of cars in parking lot: 3

Vehicle is entering the Entry Gate.
Car has entered PLCS
Number of cars in PLCS: 4

Car is driving towards the exit point

Car has left the parking lot
Number of cars in parking lot: 3

Closing the Entry Gate.
Vehicle is entering the Entry Gate.
Car has left the parking lot
Number of cars in parking lot: 2

Car has entered PLCS
Number of cars in PLCS: 3

Closing the Entry Gate.
Vehicle is entering the Entry Gate.
Car is driving towards the exit point

Car has left the parking lot
Number of cars in parking lot: 2

Car has entered PLCS
Number of cars in PLCS: 3

Closing exit gate

Closing the Entry Gate.
Vehicle is entering the Entry Gate.
Car has entered PLCS
Number of cars in PLCS: 4
```

- Tilføjet af Sonny Vagn Andersen for 6 dage siden

Læringsmål 1: Benyt pseudokode til at designe systemet

- Gruppen har lavet et fint design med pseudokode. Dog ligger noget af det lidt for tæt op af rigtig kode til min smag. Der bruges eksempelvis pointere og rigtige funktionsnavne (fx pthread_create). Begge dele hører ikke hjemme i pseudokode efter min mening. Alt i alt er det dog stadig rigtig fint, og læringsmålet er godkendt.

Læringsmål 2: Implementer PLCS til én bil med mutex og conditionals på baggrund af pseudokoden

- Gruppen har lykkedes med at implementere deres design. Der gøres fint brug af både mutex og conditionals. PLCS håndterer fint at lukke én bil ind og ud. Læringsmålet er godkendt.

Læringsmål 3: Udvid PLCS til at kunne håndtere flere biler

- Gruppen har ændret i koden, så der er mulighed for håndtering af flere biler. Det ser ud til, at det er lykkedes rigtig fint. Desværre har gruppen ikke nummereret bilerne, så man kan ikke se, hvilken rækkefølge, de kører ind og ud i. Dette var heller ikke et krav, men det kunne have været sjovt at kunne se. Ellers godt arbejde. Læringsmålet er godkendt.

Læringsmål 4: Sæt et loft på antallet af biler, PLCS kan håndtere og få det til at fungere

- Gruppen har lavet en grænse, der hedder 4 biler, og så testes der ud fra det. Gruppen har fået det til at lykkedes. Man kan stadig ikke se rækkefølgen på bilerne, men antallet af biler i PLCS overstiger aldrig de 4. Rigtig fint løst. Læringsmålet er godkendt.

Feedback
Alt i alt en rigtig fin besvarelse. Dog kan der godt skrues ned for mængden af kode, der vises i besvarelsen. Nøjes med nogle udsnit af væsentlige dele. Resten kan ses i repository. Det samme gælder makefiler. Der er ingen grund til at vise koden for en makefile i besvarelsen. Med det sagt, så er alt lykkedes rigtig godt, og gruppen er nået længere med exercise 3 end forventet. Godt arbejde.

Must haves

- Relevant files in repository ✓
- Makefiles in repository ✓
- Mutex og conditionals skal bruges; semaphore forbudt (/)

Konklusion
Gruppen har fået godkendt alle 4 læringsmål, og alle must haves er opfyldt. Besvarelsen vurderes derfor til ok.

Ex2_1_terminal.PNG (23 KB) Mikkel Mahler, 2019-03-13 09:52

Ex2_2.PNG (38,4 KB) Mikkel Mahler, 2019-03-19 20:21

OPG_3.PNG (42,4 KB) Jonas Jesper Tække, 2019-03-19 22:18