Lecture 4 - Exercises

////////////////////////// Kommentar fra d.31/10 //////////////////////////
Brian har oploaded kode til repo igen! Jubii

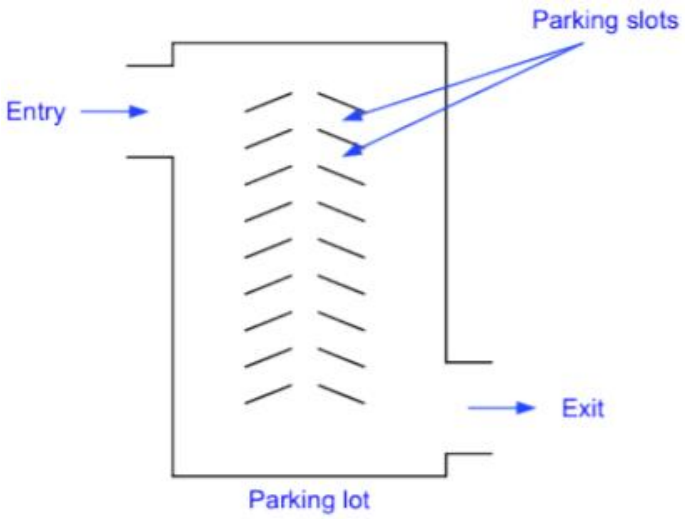
////////////////////////// Kommentar fra d.30/10 //////////////////////////
Brian har ødelagt repo, så den tilrettede kode er pt ikke tilgængelig, da han har slettet det

////////////////////////// Denne opgave er blevet delvist rettet til d.29/10 //////////////////////////
Følgende ændringer er fortaget:
- Opgave 1 er opdateret så bilen venter inde på parkeringspladsen, som der er krævet i opgaveformuleringen.
- Opgave 1 er ændret så programmet ikke crasher efter en enkelt gennemkørsel. Dette er gjort ved at lave én mutex i stedet for at have to. Nyt billede af konsollen er indsat for at vise dette.
- Opgave 2 er ændret så indgang og udgang holdes åbne, så længe at der er biler i kø, fremfor kun at tillade én bil ind og ud af gangen. Dette er gjort fordi det andet var for svært at implementerer. Det var dog forsøgt implementeret ved indsættelse af flere while-løkker, men dette resulterede i at programmet crashede når der var mere end tre biler.
- Opgave 2 er ændret så programmet ikke crasher når der var mere end tre biler. Dette er gjort ved at lave én mutex i stedet for at have to, som i opgave 1.
- Spørgsmålene og svarene under Opgave 2 er omformuleret samt opsat anderledes, for at give en bedre forståelse af vores svaret.
- Opgave 3 er ændres så programmet benytter samme kode-standart som Opgave 1 og Opgave 2.
- Opgave 3 er ændret så programmet ikke umiddelbart crasher på tilfældige tidspunker.

//

I denne øvelse skal der implementeres et "Parking Lot Control System" (PLCS) der overvåger en parkeringsplads og giver adgang til biler, der ønsker at komme ind og ud af parkeringspladsen.
Til dette skal der benyttes Mutex'er og Conditionals; Altså IKKE semaphores.

Der er givet følgende sketch:



A simple sketch of the PLCS

PLCS har følgende krav:

- Biler kører ind på parkeringspladsen, bliver der et stykke tid, kører ud af parkeringspladsen, venter et stykke tid, og kører ind på parkeringspladsen igen.
- En bil der ønsker at komme ind på parkeringspladsen skal bede om adgang fra "PLCS entry guard". Når adgang er givet, kan bilen køre ind.
- En bil der ønsker at køre ud af parkeringspladsen skal bede om lov fra "PLCS exit guard". Når adgang er givet, kan bilen køre ud.

- Hver bil skal repræsenteres som en enkelt thread. Dette gælder også "PLCS entry guard" og "PLCS exit guard". Altså, hvis programmet køres med én bil, skal der være tre threads.

- Hver "Guard" skal styrer en "dør". Når der ikke er nogen biler der ønsker adgang, skal døren være lukket.

Implement Park-a-lot 2000

First step

Implementer PLCS og test at programmet virker med én bil.
Før implementeringen blev der udarbejdet et flow-diagram. Flowdiagrammet (herunder) viser hvordan mutexen venter på på signaler fra hhv. car, entry og exit.

Flow-diagramFlow-diagram

Car_flow.png

Udfra flow-diagrammet blev koden implementeret for car, entry og exit, som det ses herunder. Koden er implementeret således, det stemmer overens med flow-diagrammet.

Kode til implementering af carKode til implementering af car

```
void* car(void* arg)
{
    int carNumber = *((int*)arg);

    while(1)
        {
        cout << "Car " << carNumber << " requests entry" << endl;

        pthread_mutex_lock(&Mutx);

        carWantIn = true;

        pthread_cond_signal(&entryCond);

        while(!gateInOpen)
            {
            cout << "Entry is closed" << endl;
            pthread_cond_wait(&entryCond, &Mutx);
            }

        cout << "Car " << carNumber << " enters" << endl;

        carWantIn = false;

        pthread_cond_signal(&entryCond);
        pthread_mutex_unlock(&Mutx);

        sleep(2); // Car waits here for a while

        pthread_mutex_lock(&Mutx);

        carWantOut = true;

        pthread_cond_signal(&exitCond);
            cout << "Car " << carNumber << " requests exit" << endl;

        while(!gateOutOpen)
            {
                pthread_cond_wait(&exitCond, &Mutx);
            }
            cout << "Car " << carNumber << " exits" << endl;

        carWantOut = false;
```

```
        pthread_cond_signal(&exitCond);

        pthread_mutex_unlock(&Mutx);

        //wait until repeat
        sleep(2);
     }
     return nullptr;
}
```

Kode til implementering af guardEntryKode til implementering af guardEntry

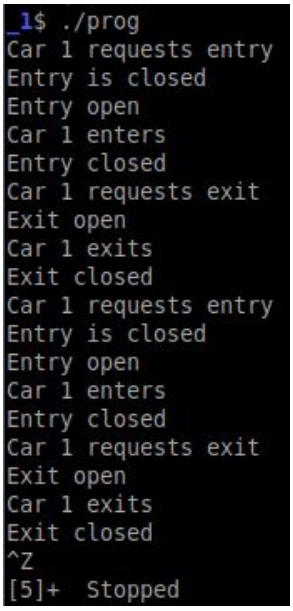
```
void* guardEntry(void* arg)
{
    while(1)
    {
        pthread_mutex_lock(&Mutx);

        //No car is waiting
        while(!carWantIn)
            {
            pthread_cond_wait(&entryCond, &Mutx);
            }

        gateInOpen = true;
        pthread_cond_broadcast(&entryCond);
        cout << "Entry open" << endl;

        //A car is waiting
        while(carWantIn)
            {
            pthread_cond_wait(&entryCond, &Mutx);
            }
        gateInOpen = false;
        pthread_cond_broadcast(&entryCond);
        cout << "Entry closed" << endl;

        pthread_mutex_unlock(&Mutx);
    }
    return nullptr;
}
```

Kode til implementering af guardExitKode til implementering af guardExit

```
void* guardExit(void* arg)
{
    while(1)
    {
        pthread_mutex_lock(&Mutx);

        while(!carWantOut)
            {
            pthread_cond_wait(&exitCond, &Mutx);
            }
        gateOutOpen = true;
        pthread_cond_broadcast(&exitCond);
        cout << "Exit open" << endl;

        while(carWantOut)
            {
            pthread_cond_wait(&exitCond, &Mutx);
```

```
            }
        gateOutOpen = false;
        pthread_cond_broadcast(&exitCond);
        cout << "Exit closed" << endl;

        pthread_mutex_unlock(&Mutx);
    }
    return nullptr;
}
```

Nedenfor ses et billede af terminalen med en enkelt bil på parkeringspladsen.



Se alt kode samt makefile i repository:
https://redmine.ase.au.dk/courses/projects/i3isu_e2018_hal9000/repository/revisions/master/show/Lecture4_exercises/Exercise1_1

The grandiose test

Test implementeringen af PLCS med flere biler. Bilerne skal vente på parkeringspladsen i forskellige mængder af tid, før de køre ud igen. Der er ikke noget maksimum for antallet af biler.

For at implementerer flere biler, oprettes der flere tråde. Disse laves så hver tråd får deres egne ID. På denne måde kan der holdes øje med om bilerne køre ind på og ud af parkeringspladsen.

Implementering af bilernes ID'erImplementering af bilernes ID'er

```
    //Make an amount of IDs
    int id[numberOfCars];
    for (int i = 0; i<numberOfCars;i++)
    {
        id[i] = i;
    }

    //Make the cars!
    for (int i = 0; i < numberOfCars; i++)
    {
        pthread_create(&carThread[i], NULL, car, &id[i]);
    }

    //Make the rest!
    pthread_create(&entryThread, NULL, guardEntry, entry_);
    pthread_create(&exitThread,  NULL, guardExit,  exit_ );
```

```
    //Join everything
    for (int i = 0; i < numberOfCars; i++)
    {
        pthread_join(carThread[i], NULL);
    }

    pthread_join(entryThread, NULL);
    pthread_join(exitThread,  NULL);
```

Funktionen rand() blev brugt til at give hver bil en tilfældig mængde af ventetid på parkeringspladsen, før de kørte ud igen:

Implementering af tilfældig opholdstid for bilerne på parkeringspladsenImplementering af tilfældig opholdstid for bilerne på parkeringspladsen

```
        // Random amount of pause, before trying to request exit
        randWaitTime=rand()%5+1;
        sleep(randWaitTime);
```

Det blev endvidere implementeret at indgang og udgang holder døren åben, så længe at der er biler der gerne vil ind og ud.
Der blev testet med fem "biler":

Terminalvindue ved test af fem bilerTerminalvindue ved test af fem biler

5_CARS.png

Der blev også testet med op til 100 biler. Der er ikke inkluderet billeder af dette, da det var relativt svært at holde styr på. Dog crashede programmet ikke biggrin.png

Se alt kode samt makefile i repository:
https://redmine.ase.au.dk/courses/projects/i3isu_e2018_hal9000/repository/revisions/master/show/Lecture4_exercises/Exercise1_2

Forklar hvad pthread_cond_broadcast() gør. Argumenter for hvorfor den blev brugt / ikke brugt.

> I denne opgave blev broadcast-funktionen benyttet.
> Da der er flere / tråde i opgaven, kunne det have været et probleme med at pthread_cond_signal-funktionen ikke vækker den rigtige mutex og derved "unblocker" de rigtige tråd, da den blot vækker minimum én mutex, og derfor ikke nødvendigvist den/de ønskede. Med pthread_cond_broadcast-funktionen "unblocker" man alle tråde (altså, vække alle mutexes) der har samme conditional variable, på en gang. I dette tilfælde var broadcast nødvendig, ved mange biler, for at programmet ikke skulle crashe. Når der blev testet ved få biler, var det intet problem at benytte signal da det virkede til at den alligevel vækker alle trådene, og derved tillader at biler kan kører ind og ud af parkeringspladsen.

Det er ikke en del af opgaven at sørge for at de biler der er i kø til at komme ind på parkeringspladsen, kommer ind i den rigtige rækkefølge, men forklar hvorfor bilerne ikke venter i kø. Hvad kan man gøre for at fixe dette problem?

> Bilerne venter ikke i en kø, så når de venter på deres tur til at køre ind så er det bare tilfældigt.

Alle bilerne er som tråde låst under samme condition. Når der låses op, er det mest bare de heldige biler der har de rigtige cpu-tider der får lov at køre. Dette kunne evt. løses ved at benytte bilernes id til at give adgang i en bestemt rækkefølge.

Extending PLCS, now with a limit on the number of cars

I denne opgave skal der benyttes "time-boxing". Dette blev dog ikke gjort, fordi at Brian glemte det.
Implementer at "PLCS entry guard" KUN lukker indgangen op, når parkeringspladsen ikke er fuld. Der skal altså implementeres et maksimum. Dette gøres ved to nye variabler:

VisHide

```
//New int to define the amount of cars allowed
int maxCapacity  = 4;
int theAmountOfCarsThatAreInTheParkingLot = 0;
```

Der er ændret i indgangs-while-løkken i car så det nu også kræver at der er plads på parkeringspladsen, før døren åbner:

VisHide

```
        //wait here until door is open or if theres no more room in the parkinglot
        while(!gateInOpen || theAmountOfCarsThatAreInTheParkingLot == maxCapacity)
            {
            pthread_cond_wait(&entryCond, &Mutx);
            }

        theAmountOfCarsThatAreInTheParkingLot++;
```

Antallet af biler på parkeringspladsen dekrementeres når en bil forlader parkeringspladsen, med:

```
theAmountOfCarsThatAreInTheParkingLot--; //theres one less car
```

Se alt kode samt makefile i repository:
https://redmine.ase.au.dk/courses/projects/i3isu_e2018_hal9000/repository/revisions/master/show/Lecture4_exercises/Exercise2_2

Der blev testet med fem biler, men med en maxCapacitet på 2.

Terrminalvindue ved test af 5 bilermen kun 2 pladser

2_2test.png

Det ses på billedet at Car 1 er den første bil der kommer ind på parkeringspladsen, hvorefter Car 4 kommer ind. Begge disse biler når dog at, forlade parkeringspladsen inden inden Car 0 og Car 4(igen) kommer ind. Herefter ses det at der først kommer en bil ind på parkeringspladsen, når mængden af biler på parkeringspladsen er 1 ud af 2.
Der blev i denne del også forsøgt med 1000 biler og 47 parkeringspladser (en typisk situation i Aahus).

Terrminalvindue ved test for sjovTerrminalvindue ved test for sjov

spass_test.png

Filer

Sketch_PLCS.png	23,5 KB	2018-10-05	Brian Nymann
Terminal1_1.png	15,1 KB	2018-10-08	Mie Nielsen
three_cars.png	26,4 KB	2018-10-08	Brian Nymann
capacity_test.png	34,6 KB	2018-10-08	Brian Nymann
Car_flow.png	94,3 KB	2018-10-09	Martin Lundberg
1_CAR.png	12,5 KB	2018-10-29	Brian Nymann
5_CARS.png	18,5 KB	2018-10-29	Brian Nymann
2_2test.png	36,6 KB	2018-10-29	Brian Nymann
spass_test.png	44,2 KB	2018-10-29	Brian Nymann