Gruppe 33 - Exercise5 - # 31

Thread Synchronization II - Parking Lot Control system

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.

Læringsmål for øvelsen:

1. Designe systemet(PLCS) med pseudo kode eller flowchart
2. Implementere designet ved brug af mutex og threads
3. Forståelse for brugen Guards, signalering og blocking.
4. Benytte condSignal og condWait til implementeringen

Exercise 1 Implement Park-a-Lot 2000 - Design

Der er nedenfor i pseudo code beskrevet hvordan de tre threads, car-, entry- og exitthread, designes for PLCS.
Der er taget udgangspunkt i eksemplet "CASE - PARK-A-LOT 2000" fra undervisningsmaterialet.

Car Thread

```
carDriverThread()
{
    driveUpToEntry();
    lock(entryMut);
    carWaitingEntry = true;
    condSignal(entry);

    while(!garageEntryOpen)
        condWait(entry,entryMut);

    driveIntoGarage();
    carWaitingEntry = false;
    condSignal(entry);
    unlock(entryMut);

    sleep(parkTime);

    driveUpToExit();
    lock(exitMut);
    carWaitingExit = true;
    condSignal(exit);

    while(!garageExitOpen)
        condWait(exit,exitMut);

    driveOutOfGarage();
    carWaitingExit = false;
    condSignal(exit);
    unlock(exitMut);

    sleep(waitTime);

}
```

Entry Guard Thread

```
garageEntryControllerThread()
{
    lock(entryMut);
    while(!carWaitingEntry)
        condWait(entry,entryMut);

    openGarageEntryDoor();
    garageEntryOpen = true;
    condSignal(entry);

    while(carWaitingEntry)
        condWait(entry,entryMut);

    closeGarageEntryDoor();
    garageEntryOpen = false;
    unlock(entryMut);
}
```

Entry Guard Thread

```
garageExitControllerThread()
{
    lock(exitMut);
    while(!carWaitingExit)
        condWait(exit,exitMut);

    openGarageExitDoor();
    garageExitOpen = true;
    condSignal(exit);

    while(carWaitingExit)
        condWait(exit,exitMut);

    closeGarageExitDoor();
    garageExitOpen = false;
    unlock(exitMut);
}
```

Exercise 2 Implement Park-a-Lot 2000

Exercise 2.1 First step

Designet af PLCS fra exercise 1 implementeres nu med en enkelt bil, som efter at have forladt Parkerings garagen, vender tilbage og processen gentages.

Resultat af implementering

```
stud@stud-virtual-machine:~/isu_work/exercise5/bin/x86-64$ ./prog
Car drives up to Garrage Entry ...
Garrage Entry now Open ...
Car drives into Garrage Parking Lot ...
Car is parked...
Garrage Entry now Closed ...
Car drives up to Garrage Exit ...
Garrage Exit now Open ...
Car drives out of Garrage Parking Lot ...
Garrage Exit now Closed ...
Car drives up to Garrage Entry ...
Garrage Entry now Open ...
Car drives into Garrage Parking Lot ...
Car is parked...
Garrage Entry now Closed ...
Car drives up to Garrage Exit ...
Garrage Exit now Open ...
Car drives out of Garrage Parking Lot ...
Garrage Exit now Closed ...
```

Diskussion / Konklusion

I denne implementering er bilen parkeret i garagen(parkTime) i 5 sekunder og processen gentages(waitTime) efer 10 sekunder. condition signals var i første test sat til "entry" og "exit", hvilket er navne allerede brugt, hvorfor der blev ændret til "entryCon" og exitCon".

Exercise 2.2 The grandiose test

System udbygges med flere biler, og med forskellige parkeringstider(parkTime). PLCS har endnu ikke et loft for et maksimum antal antal biler/parkeringsplader.

Resultat

```
stud@stud-virtual-machine:~/isu_work/exercise5/exercise_2_2/bin/x86-64$ ./prog
New Car thread: 1
Car 1 -> drives up to Garrage Entry ...
New Car thread: 2
Car 2 -> drives up to Garrage Entry ...
New Car thread: 3
Car 3 -> drives up to Garrage Entry ...
New Car thread: 4
Car 4 -> drives up to Garrage Entry ...
Garrage Entry now Open ...
Car 3 -> drives into Garrage Parking Lot ...
Car 3 -> is parked...
Car 1 -> drives into Garrage Parking Lot ...
Car 1 -> is parked...
Car 4 -> drives into Garrage Parking Lot ...
Car 4 -> is parked...
Car 2 -> drives into Garrage Parking Lot ...
Car 2 -> is parked...
Garrage Entry now Closed ...
Car 4 -> drives up to Garrage Exit ...
Garrage Exit now Open ...
Car 4 -> drives out of Garrage Parking Lot ...
Garrage Exit now Closed ...
Car 3 -> drives up to Garrage Exit ...
Garrage Exit now Open ...
Car 3 -> drives out of Garrage Parking Lot ...
Garrage Exit now Closed ...
Car 2 -> drives up to Garrage Exit ...
Garrage Exit now Open ...
Car 2 -> drives out of Garrage Parking Lot ...
Garrage Exit now Closed ...
Car 1 -> drives up to Garrage Exit ...
Garrage Exit now Open ...
Car 1 -> drives out of Garrage Parking Lot ...
Garrage Exit now Closed ...
```

Diskussion / Konklusion

Der laves en thread til hver car og de forskellige parkeringstider(parkTime) sættes ved 5 - carId.
I ovenstående resultat ses at carEntry ikke når at lukke, da de 4 threads/cars laves/ankommer kort efter hinanden, dette vil eventuelt løses ved længere sleep imellem pthread_create

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

pthread_cond_broadcast() benyttes til at "unblock" alle threads, som er "blocked" af en specifik condition.
https://pubs.opengroup.org/onlinepubs/007904975/functions/pthread_cond_broadcast.html

```
int pthread_cond_broadcast(pthread_cond_t *cond);
```

I denne øvelse er der benyttet pthread_cond_wait() til at blokere. Dette sker ved at release, en i forvejen "locked" mutex, tjekke for et specifikt conditional signal og herefter lock igen.
Dette er implementeret i eksempelvis entry guarden:

```
pthread_mutex_lock(&entryMut);
while(!carWaitingEntry){
    pthread_cond_wait(&entryCon, &entryMut);
    }
```

Hvor der i car thread, bliver benyttet conditional signal "entryCon" til as passere:

```
pthread_mutex_lock(&entryMut);
        carWaitingEntry = true;
        pthread_cond_signal(&entryCon);
```

Problemet, hvor pthread_cond_broadcast() kunne benyttes, opstår når der oprettes flere car threads, som alle vil blive blokeret, og afvente at mutex´n bliver released.

- *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?*

Problemet opstår når en locked mutex bliver released, da sheduleren her bestemmer hvilken af de førhen blocked threads, som for lov.
Ved at implementer prioritering eller Circular wait condition til at bestemme rækkefølgen, bør ovenstående problem kunne løses.

Exercise 3 Extending PLCS, now with a limit on the number of cars

System udbygges med et maksimum for antallet af biler/parkeringsplader i parkeringsgaragen. PLCS entry guarden skal derfor tjekke om maksimum er nået inden flere biler får lov til at køre ind.

Design og implementering

Som med entry og exit control implementeres garageCapacityCheck med egen mutex og signal condition.(capacityMut og capacityCond)

```
void *garageCapacityCheck(void* arg)
{
    while(1){
        pthread_mutex_lock(&capacityMut);
        while(!carWaitingCapacity){
            pthread_cond_wait(&capacityCond, &capacityMut);
        }
        if(numOfCars == capacity){
            garageFull = true;
        }
        else{
            garageFull = false;
        }
        pthread_mutex_unlock(&capacityMut);
    }
}
```

Car thread udbygges, og et codesnippet vises nedenfor:

```
pthread_mutex_lock(&capacityMut);
carWaitingCapacity = true;
pthread_cond_signal(&capacityCond);

while(garageFull){
    pthread_cond_wait(&capacityCond, &capacityMut);
    }

numOfCars = numOfCars +1;
carWaitingEntry = false;
pthread_mutex_unlock(&capacityMut);
```

Resultat med 2 parkeringsplader

```
Car 1 -> drives up to Garrage  ...
Garrage has 2 parking lots left...
Car 1 -> Got a free parking spot  ...
Garrage Entry now Open ...
Car 1 -> drives into Garrage Parking Lot ...
```

```
Garrage Entry now Closed ...
Car 1 -> is parked...
Car 2 -> drives up to Garrage  ...
Garrage has 1 parking lots left...
Car 2 -> Got a free parking spot  ...
Garrage Entry now Open ...
Car 2 -> drives into Garrage Parking Lot ...
Garrage Entry now Closed ...
Car 2 -> is parked...
Car 1 -> drives up to Garrage Exit ...
Car 3 -> drives up to Garrage  ...
Garrage has 0 parking lots left...
Garrage Exit now Open ...
Car 1 -> drives out of Garrage Parking Lot ...
Car 3 -> Got a free parking spot  ...
Garrage Exit now Closed ...
Garrage Entry now Open ...
Car 3 -> drives into Garrage Parking Lot ...
Car 3 -> is parked...
Garrage Entry now Closed ...
Car 2 -> drives up to Garrage Exit ...
Garrage Exit now Open ...
Car 2 -> drives out of Garrage Parking Lot ...
Garrage Exit now Closed ...
Car 3 -> drives up to Garrage Exit ...
Garrage Exit now Open ...
Car 4 -> drives up to Garrage  ...
Garrage has 1 parking lots left...
Car 4 -> Got a free parking spot  ...
Car 3 -> drives out of Garrage Parking Lot ...
Garrage Exit now Closed ...
```

Resultat med 1 parkeringsplader

```
Car 1 -> drives up to Garrage  ...
Garrage has 1 parking lots left...
Car 1 -> Got a free parking spot  ...
Garrage Entry now Open ...
Car 1 -> drives into Garrage Parking Lot ...
Car 1 -> is parked...
Garrage Entry now Closed ...
Car 2 -> drives up to Garrage  ...
Garrage has 0 parking lots left...
Car 1 -> drives up to Garrage Exit ...
Garrage Exit now Open ...
Car 1 -> drives out of Garrage Parking Lot ...
Garrage Exit now Closed ...
Car 3 -> drives up to Garrage  ...
Car 2 -> Got a free parking spot  ...
Garrage Entry now Open ...
Car 2 -> drives into Garrage Parking Lot ...
Car 2 -> is parked...
Garrage Entry now Closed ...
Garrage has 0 parking lots left...
Car 4 -> drives up to Garrage  ...
Garrage has 0 parking lots left...
Car 2 -> drives up to Garrage Exit ...
Garrage Exit now Open ...
Car 2 -> drives out of Garrage Parking Lot ...
Garrage Exit now Closed ...
```

Diskussion / Konklusion

Det er lykkedes at implementer en løsning, hvor der er en maksimum capacity, som forhindrer car threads i at komme forbi entry gaten, når garagen er fuld.
Der er fortsat ingen "kø" i form af hvis både car2 og car3 venter på en plads, er car2 ikke sikret at få næste ledige.

Implementering af løsning til 2.3 blev lidt rodet, ustruktureret og ukommenteret. Her kunne med fordel været benyttet pseudo code eller flowcharts som anbefalet.
Ligeledes burde der fra start været tænkt scoped locking ind, for at sikre lock/unlock, hvilket heller ikke er blevet gjort.