Wiki »

Exercise 5 - Thread Synchronization II ¶

I denne øvelse vil der blive implementeret en parkeringsplads med et system, som til sidst kan styre biler som ønsker at køre enten ind eller ud, derudover vil det kunne holde styr om antal besøgende har nået sin maks kapacitet.

Exercise 1 Implement Park-a-Lot 2000 - Design

Til at starte med skal parkeringspladsen bare kunne håndtere en bil. Bilen skal køre op til entry-gate, blive lukket ind også vente et stykke tid inde på pladsen før den kører ud igen. Derfor laves programmet i qseudo kode, som er med til at skabe overblik og bliver nemmere at implementere. Der vil undervejs være kommentering som tydeliggør og forklare hvad de enkelte funktioner skal kunne.

Pseudo kode for Park-a-Lot 2000 med en bil:

Car skal styre bilen.

```
  Car()
{
    driveUpToPLCSEntry(); //bil kører op til entry -gate
    carEntryLock(mut); //Låser mutex'en
    carWaitingEntry = true //Ændre tilstanden til at der er en bil som venter på at
    condSignal(entry);

    while(!gateOpen); //Venter på at porten er helt åben
        condWait(entry, mut);

    driveIntoParkinglot(); //Bilen kører ind på parkeringspladsen
    carWaitingEntry = false //Ændrer tilstanden på at der ingen bil er
    condSignal(entry);
    carEntryUnlock(mut); //Låser mutex op

    // Venter i x antal tid

    driveUpToPLCSExit(); //bil kører op til exit-gate
    carExitLock(mut); //Låser mutex'en
    carWaitingExit = true //Ændre tilstanden til at der er en bil som venter på at k
    condSignal(exit);

    while(!gateOpen); //Venter på at porten er helt åben
        condWait(exit, mut);

    driveOutParkinglot(); //Bilen kører ind på parkeringspladsen
    carWaitingExit = false //Ændre tilstanden på at der ingen bil er
    condSignale(exit);
    carExitUnlock(mut); //Låser mutex op
}
```

PLCSEntry og *PLCSExit* skal repræsentere de to gates til parkeringspladsen, som skal åbne eller lukke når de får et givent signal.

```
  PLCSEntry()
{
    gateEntryLock(mut); //Entry-bummen er låst

    while(!carWaiting) //Venter på at der er en bil som holder der
        condWait(entry, mut);

    openEntryGate(); // Åben Entry-gate
    entryGateOpen = true; // Status på Entry-gate ændres
    condSignal(entry);

    while(carWaiting)
        condWait(entry, mut);

    closeGateEntry(); // Lukker Entry-gate
    entryGateOpen = false; // Status på Entry-gate ændres
    gateEntryUnlock(mut); //Mutex - Entry-bummen er låst op
}

PLCSExit()
{
    gateExitLock(mut); //Exit-bummen er låst

    while(!carWaiting) //Venter på at der er en bil som holder der
        condWait(exit, mut);

    openExitGate(); // Åbner Exit-gate
    exitGateOpen = true; // Status på Exit-gate ændres
    condSignal(exit);

    while(carWaiting)
        condWait(exit, mut);

    closeGateExit(); // Lukker Exit-gate
    exitGateOpen = false; // Status på Exit-gate ændres
    gateExitUnlock(mut); // Entry-bummen er låst op
}
```

Exercise 2 Implement Park-a-Lot 2000

Exercise 2.1 First step

Ovenstående pseudo kode implementeres. Da programmet på sigt skal kunne håndtere flere biler, får bilerne deres egen klasse, så koden bliver mere læsbar samt nemmere at udvide. I Car-klasen er funktionerne *enterPark()* og *extitPark()* nærmest identisk da bilen teoretisk skal det samme om den kører ind eller ud af parkeringspladsen. Af samme grund ligner *entryGateFunc()* og *exitGateFunc* og hinanden.

For at implementere programmet med parkeringspladsen oprettes en række conditional variables til at signalere mellem trådene, samt mutex'es til at låse de kritiske sektioner. Forneden beskrives de enkelte signalers navne og funktionalitet.
conditional variables:

- entrySig: bruges til at signalere mellem entry gate tråden og car-tråden. Disse signalerer hinanden, når de har foretaget en ændring i tilstand, som den anden skal reagere på.
- outSig: samme som entry signalet, bare for exit gate tråden.

mutexes:
- entryMut: bruges sammen med entrySig til at låse car og entry tråden i de kritiske sektioner.
- exitMut: samme som entryMut, bare for exit tråden.

Desuden testes der på en række variable, der funger som guards, når trådene modtager et conditional signal. Disse inkluderer
entryGateOpen: indikerer om entry gaten er åben.
exitGateOpen: indikerer om exit gaten er åben.
carWaitingEntry: indikerer om bilen venter på at komme ind.
carWaitingExit: indikerer om bilen venter på at komme ud.

entryGateFunc() og exitGateFunc

```
void enterPark()
 {
     //bil kører op til parking lot
     std::cout<<"Car driving to entry gate\n";
     pthread_mutex_lock(entry_);
     carWaitingEntry_=true;
     pthread_cond_signal(entry_sig_);

     while(!*entryGateOpen_)
     {
         pthread_cond_wait(entry_sig_,entry_);
     }
     //bil kører ind i parking lot
     std::cout<<"Car entering park\n";
     inLot=true;
     carWaitingEntry_=false;
     pthread_cond_signal(entry_sig_);
     pthread_mutex_unlock(entry_);
 }

 void exitPark()
 {
     //bil kører op til parking lot exit gate
     pthread_mutex_lock(exit_);
     std::cout<<"Car driving to exit gate\n";
     carWaitingExit_=true;
     pthread_cond_signal(out_sig_);

     while(!*exitGateOpen_)
     {
         pthread_cond_wait(out_sig_,exit_);
     }
     //bil kører ud af parking lot
     inLot=false;
     std::cout<<"Car exiting park\n";
     carWaitingExit_=false;
     pthread_cond_broadcast(out_sig_);
     pthread_mutex_unlock(exit_);
 }
```

enterPark() og extitPark()

```
void* entryGateFunc(void *argu)
{
    gateStruct *arg=static_cast<gateStruct*> (argu);
    while(1)
    {
        pthread_mutex_lock(arg->mut);
        while(!(arg->car->getWaitingEntry()))
        {
            pthread_cond_wait(arg->signal,arg->mut);
        }
        //åben entry gate
        std:: cout<<"Opening entry gate\n";
        *(arg->open)=true;
        pthread_cond_signal(arg->signal);
        while(arg->car->getWaitingEntry())
        {
            pthread_cond_wait(arg->signal,arg->mut);
        }
        //luk entry gate
        std:: cout<<"Closing entry gate\n";
        *(arg->open)=false;
        pthread_mutex_unlock(arg->mut);
    }
}

void* exitGateFunc(void *argu)
{
   gateStruct* arg=static_cast<gateStruct*> (argu);
    while(1)
    {
        pthread_mutex_lock(arg->mut);
        while(!(arg->car->getWaitingExit()))
        {
            pthread_cond_wait(arg->signal,arg->mut);
        }
        //åben exit gate
        std:: cout<<"Opening exit gate\n";
        *(arg->open)=true;
        pthread_cond_signal(arg->signal);
        while(arg->car->getWaitingExit())
        {
            pthread_cond_wait(arg->signal,arg->mut);
        }
        //luk exit gate
        std:: cout<<"Closing exit gate\n";
        *(arg->open)=false;
        pthread_mutex_unlock(arg->mut);
    }
}
```

I *main()* startes der med at initialisere variablerne, hvorefter de to gate-threads startes og efterfølgende joines threads.

main()

```
int main(void)
{
```

```
    //initializer variable
    pthread_cond_t entrySig;
    pthread_cond_t outSig;
    pthread_mutex_t entryMut;
    pthread_mutex_t exitMut;
    pthread_mutex_init(&entryMut,nullptr);
    pthread_mutex_init(&exitMut,nullptr);
    pthread_cond_init(&entrySig,nullptr);
    pthread_cond_init(&outSig,nullptr);
    bool entryGateOpen=false;
    bool exitGateOpen=false;
    pthread_t entry_tid;
    pthread_t exit_tid;
    Car carObj(&entryMut,&exitMut,&entrySig,&outSig,&entryGateOpen,&exitGateOpen);

    //start entry gate thread
    gateStruct entryArg;
    entryArg.car=&carObj;
    entryArg.mut=&entryMut;
    entryArg.open=&entryGateOpen;
    entryArg.signal=&entrySig;
    pthread_create(&entry_tid,nullptr,entryGateFunc,&entryArg);

    //start exit gate thread
    gateStruct exitArg;
    exitArg.car=&carObj;
    exitArg.mut=&exitMut;
    exitArg.open=&exitGateOpen;
    exitArg.signal=&outSig;
    pthread_create(&exit_tid,nullptr,exitGateFunc,&exitArg);

    //program:
    //join threads
    void* res;
    pthread_join(entry_tid,&res);
    pthread_join(exit_tid,&res);
    pthread_join(carObj.getPthread_tid(),&res);

}
```

Eksekveringen af det skrevet kode ser derfor således ud i terminalen:

```
stud@stud-virtual-machine:~/i3isu_f2020_beany_business/exe5/exe2_1$ ./bin/host/prog
Car driving to entry gate
Opening entry gate
Car entering park
Closing entry gate


Car driving to exit gate
Opening exit gate
Car exiting park
Closing exit gate


Car driving to entry gate
Opening entry gate
Car entering park
Closing entry gate


Car driving to exit gate
Opening exit gate
Car exiting park
Closing exit gate
```

Det ses her at parkerings-systemet holder styr på at bilen køre rigtigt ind og ud.

Exercise 2.2 The grandiose test

For at lave et bruger-valgt antal biler, oprettes et Car-pointer-array med det indtastede antal.

```
std::cout<<"Indtast antal biler: ";
std::cin>>n;

Car *carArray[n]

for(int i=0;i<n;i++)
{
carArray[i]=new Car(&entryMut,&exitMut,&entrySig,&outSig,&entryGateOpen,&exitGateOpe
}
```

Til at styre så alle bilernes ventetider er forskellige udnyttes det at de har et unikt index:

```
sleep(3*index_);
```

Da der ingen øvre grænse på er parkeringspladsen, vil funktionen *pthread_cond_broadcast()* anvendes, den gør at alle bilerne som holder i kø ved en gate vækkes, og herfra har mulighed for at køre ind eller ud p-pladsen. Broardcast implementeres derfor både ved *entryPark()* og *exitPark()* funktionerne.

Programmet eksekveres med 4 biler med uden overgrænse på parkeringspladsen:

```
stud@stud-virtual-machine:~/i3isu_f2020_beany_business/exe5/exe2_2$ ./bin/host/prog
Indtast antal biler: 4

Car driving 1 to entry gate
Car driving 2 to entry gate
Car driving 4 to entry gate
Car driving 3 to entry gate
Opening entry gate
Car 1 entering park
Car 4 entering park
Car 2 entering park
Closing entry gate
Opening entry gate
Car 3 entering park
Closing entry gate


Car 1 driving to exit gate
Opening exit gate
Car 1 exiting park
Closing exit gate


Car 2 driving to exit gate


Car driving 1 to entry gate
Opening exit gate
Car 2 exiting park
Closing exit gate
Opening entry gate
Car 1 entering park
Closing entry gate


Car 3 driving to exit gate
Opening exit gate


Car 1 driving to exit gate
Car 1 exiting park
Car 3 exiting park
Closing exit gate
```

Det ses at i starten når alle biler at køre op til entry gate. Hvorefter alle bliver vækket, og dem som når at få cpu tid bliver lukket ind, hvor resten så må vente. Af samme grund kan bilerne som holder i kø også godt overhale hinanden, da det er den som på det givende tidspunk der får cpu-tid, som kommer først videre. Efterfølgende eksekveres programmet ud fra deres individuelle ventetider.

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

For at styre hvor mange biler der må være på parkeringspladsen, oprettes der to nye variabler *limit* og *carsInLot*. Variablerne tjekkes der på i *entryGateFunc()*, ved at sammenligne antallet af biler som er i på p-pladsen og det maksimale antal. Så længe parkeringspladsen er fyldt, skal den ikke fortage sig noget. Men så snart en eller flere biler forlader p-pladsen, lukkes det antal ind fra dem som holder ved entry gate, så det maksimale antal er opnået igen.

```
while(*arg->carsInLot>=arg->limit)
 {}
```

Programmet eksekveres med seks biler og tre pladser på parkeringspladsen:

```
stud@stud-virtual-machine:~/i3isu_f2020_beany_business/exe5/exe2_3$ ./bin/host/prog
Indtast antal biler: 6

Indtast max biler paa plads: 3

Car driving 1 to entry gate
Car driving 2 to entry gate
Car driving 3 to entry gate
Car driving 4 to entry gate
Car driving 6 to entry gate
Car driving 5 to entry gate
Opening entry gate
Car 1 entering park
Car 6 entering park
Car 4 entering park


Car 1 driving to exit gate
Opening exit gate
Car 1 exiting park
Closing exit gate


Car driving 1 to entry gate
Car 1 entering park


Car 1 driving to exit gate
Opening exit gate
Car 1 exiting park
Closing exit gate


Car 4 driving to exit gate
Opening exit gate
Car 4 exiting park
Closing exit gate


Car driving 1 to entry gate
Car 1 entering park
Car 2 entering park
```

```
Car 1 driving to exit gate
Opening exit gate
Car 1 exiting park
Closing exit gate


Car 6 driving to exit gate
Opening exit gate
Car 6 exiting park
Closing exit gate


Car 2 driving to exit gate


Car driving 1 to entry gate
Car 1 entering park
Opening exit gate
Car 2 exiting park
Car 3 entering park
Closing exit gate
Car 5 entering park
Closing entry gate


Car 1 driving to exit gate
Opening exit gate
Car 1 exiting park
Closing exit gate


Car driving 4 to entry gate
Opening entry gate
Car 4 entering park
Closing entry gate
```

Det ses her at alle biler kører op til entry gaten, men det er kun tre som bliver lukket ind. Så snart der er en som kører helt ud, lukkes der en ny ind. Det er dog tilfældigt hvilken, da der ikke er oprettet kø som håndtere hvem der står forrest.
Derudover er det igen oftest Car 1 som laver nogle nye aktioner i programmet, det skyldes igen måden ventetiden initialiseres på, hvor Car 1 har den korteste ventetid.

- Tilføjet af Lasse Andresen for 2 måneder siden

  Læringsmål

  - Læringsmål 1 - **At designe PLCS-systemet vha. pseudokode eller flowcharts**
    Gruppen opfylder dette læringsmål, da de har skrevet relevant pseudokode, der ydermere indeholder mange og gode kommentarer. Kommentarerne forklarer forskellige funktioner, og er desuden med til at beskrive de mere komplicerede dele af systemet, som gør gruppens design nemmere at forstå. (/)

  - Læringsmål 2 - **At kunne beskrive den grundlæggende funktionalitet af conditionals**
    Gruppen er rigtig god til at beskrive de *conditionals* som de bruger i deres implementering. Dvs. hvad deres ansvar er, og hvordan de benyttes. Det kunne dog have været rigitgt godt, hvis der med forklaringer havde været demonstreret en mere teknisk forståelse af *conditionals*. Tilgengæld viser gruppen med deres implementering at de har godt styr på hvordan *conditionals* anvendes, og dette tolkes som at gruppen har tilegnet sig en grundlæggende forståelse af *conditionals*. Derfor opfylder de læringsmålet. (/)

- Læringsmål 3 - **At kunne argumentere for valget af *pthread_cond_signal* over *pthread_cond_broadcast*, eller omvendt**
  I forklarer fint hvad p_thread_broadcast gør, men I beskriver ikke hvorfor den er bedre end p_thread_signal. Der kunne i godt have kommet ind på, at den sikrer at jeres enterPark og eller exitPark bliver vækket, og hvorfor den p_thread_signal ikke sikrer det. (x)

- Læringsmål 4 - **At demonstrere indledende løsningsforslag til øvelsens 4. opgave**
  Øvelsen løses ved at der sættes et while loop ind, som kun tillader entry gaten at åbne, hvis der er plads. Hvis der ikke er plads, så ventes der på plads på parkeringspladsen. Dette while loop tjekker på to nye variabler, som også står beskrevet i wikien. Hermed er øvelsen både løst, og der er også udlagt en plan for, hvordan den blev løst. (/)

Feedback
Gruppen demonstrerer på overbevisende vis, at de har tilegnet sig fundamental viden omkring *conditionals* og deres anvendelsesmuligheder. Deres pseudokode med tilhørende kommentarer er i særdeleshed rigtigt god, og de giver indtrykket af, at gruppen inden implementeringen har reflekteret og diskuteret grundigt om det pågældende emne - *Thread Synchronization*. De løser ydermere øvelsens opgaver på fornem vis, og viser herved at de kan omsætte deres teroetiske forståelse for emnet, til konkrete løsningsmuligheder. Dog savnes der flere tekniske forklaringer i forbindelse med det aktuelle emne. Det kunne derfor havde været fordelagtigt for gruppens besvarelser, hvis der havde været øget fokus på forklaringer af relevante funktioner og datatyper. Her menes der specifikt funktioner som *pthread_cond_signal*, *pthread_cond_broadcast* og *pthread_cond_wait*. Og datatyper som f.eks. *pthread_cond_t*.

Must haves

- Relevante filer i repository (/)
- Makefiler i repository (/)
- *Mutexes & Conditionals must be used to implement the solution* - Øvelsesvejledning: Thread Synchronization II (/)

Konklusion
Der er blevet vist at gruppen igennem arbejdet med øvelserne, har gjort sig erfaringer med *thread synchronization* og specifikt med *conditionals*. Ydermere formår gruppen at demonstrere at de kan designe et thread-baseret system, beskrive grundlæggende funktionalitet af *conditionals*, og implementere komplekse sykroniserede systemer. Derfor vurderes det at denne øvelse er bestået. GG
✔

exe3_6biler_3pladser_part2.png (44,2 KB) Magnus Nygaard Lund, 2020-03-14 12:54
exe3_6biler_3pladser_part1.png (59,1 KB) Magnus Nygaard Lund, 2020-03-14 12:54
exe2_2_4biler.png (61,4 KB) Magnus Nygaard Lund, 2020-03-14 12:55
carpark1bil.png (33,5 KB) Magnus Nygaard Lund, 2020-03-17 12:27