Wiki »

Exercise 7 - OS Api

Exercise 1 - Getting to know the Api

Library setup

Api'et har følgende struktur:
Include filerne ligger i "OSApiStudent/inc/osapi" Disse er header filer der peger på en headerfil med samme navn, der vælges afhængigt af compiler setup. Forneden ses include-filen "Mutex.hpp":

```
#ifndef OSAPI_MUTEX_HPP_
#define OSAPI_MUTEX_HPP_

#include <osapi/MutexError.hpp>

#if defined(OS_WINX86)
#include "osapi/winx86/Mutex.hpp"
#elif defined(OS_LINUX)
#include "osapi/linux/Mutex.hpp"
#elif defined(OS_FREERTOS)
#include "osapi/freertos/Mutex.hpp"
#else
#error "No known OS defined"
#endif



#endif
```

Ovenstående billede viser, hvordan den headerfil der inkluderes, peger videre på den headerfil der passer til det operativsystem der defineres i filen compiler_setup. Hvis operativsystemet var defineret til OS=OS_WINX86 ville "Mutex.hpp" include "osapi/winx86/Mutex.hpp" Denne fil ses forneden:

```
4   #include <osapi/Utility.hpp>
5
6   namespace osapi
7   {
8     // Forward declaration, needed for friend designation
9     class Conditional;
10
11    namespace details
12    {
13      class Mutex; // Forward declaration
14    }
15
16    class Mutex : Notcopyable
17    {
18    public:
19      Mutex();
20      void lock();
21      void unlock();
22      ~Mutex();
23    private:
24      friend class Conditional;
25
26      details::Mutex*   mut_;
27    };
28  }
29
30
31
32  #endif
```

På udklippet af denne fil ses en header-fil med funktions-prototyper, der har en tilhørende implementations fil, som der kigges på om lidt. Det er værd at bemærke namespacet details. Der deklarerer en klasse Mutex. Dette bliver der set nærmere på i afsnittet om pimpl-idiomet. Der bliver også kigget på hvorfor klassen “Conditional” erklæres som en friend til “Mutex” klassen. Ovenstående header-fil viser de funktioner der kan kaldes i OS Api’et. Der findes selvfølgelig en tilhørende implementation, og denne ligger i stien “OSApiStudent/winx86/Mutex.cpp”

```
namespace osapi
{
  Mutex::Mutex()
    : mut_(new details::Mutex)
  {
  }

  void Mutex::lock()
  {
    mut_->lock();
  }

  void Mutex::unlock()
  {
    mut_->unlock();
  }

  Mutex::~Mutex()
  {
    // Needed for incomplete type - MUST be in the cpp file where the pimpl is known!!!
    delete mut_;
  }

}
```

Ovenstående fil viser implementationen af mutex-klassen. Dette ser umiddelbart ikke ud af meget, men skyldes at alle funktionerne kalder en tilsvarende funktion vha. pointeren til Mutex klassen i details namespace. Selve implementationen er altså skjult for brugeren af Api'et, der typisk blot vil have adgang til header-filen for Mutex-klassen, og ikke kende noget til Mutex-klassen i details namespace's header eller implementations fil, hvor al funktionaliteten ligger.

Pthread_create: Free c function -> C++ class function - implementation

Der er to klasser i Api'et der er involveret i oprettelsen af en tråd. Klasserne *Thread* og *ThreadFunctor*. *Thread* står for at oprette tråden, mens *ThreadFunctor* er en enkelt instans af en tråd. Dette fungerer således, at *Thread* i sin constructor tager en pointer til et *ThreadFunctor* objekt, samt en række andre parametre der kan benyttes i oprettelsen af en tråd. i *Thread* funktionen start() kaldes *pthread_create(), det interessante i dette kald af pthread_create(), er at funktionspointeren er en pointer til den statiske funktion threadMapper i klassen _ThreadFunctor*. Denne funktion tager en void pointer som parameter, hvilket jo er et krav til den funktionspointer der gives med som parameter i pthread_create(). Som argument til trådfunktionen gives i kaldet af pthread_create() en pointer til det *ThreadFunctor* objekt som *Thread* blev instansieret med.
Altså: *Thread* kalder pthread_create() med funktionen threadMapper() og en pointer til et *ThreadFunctor* objekt.

Nu kigges der nærmere på funktionen threadMapper(). Først og fremmest er threadMapper() en statisk funktion, dvs. ikke knyttet til nogen instans af *ThreadFunctor*, hvilket er vigtigt, da pthread_create() kræver en fri funktion som trådfunktion. Desuden tager threadMapper() en void pointer som parameter og returnerer en void pointer, hvilket også er et krav til trådfunktionen.

Nu kigges der på hvad der sker i selve threadMapper() funktionen:

```
01 void* ThreadFunctor::threadMapper(void* thread)
02 {
03   ThreadFunctor* tf = static_cast<ThreadFunctor*>(thread);
04   tf->run();
05
06   tf->threadDone_.signal();
07   return NULL;
08 }
```

Ovenstående billeder viser implementationen af threadMapper() funktionen, og er taget fra slides fra undervisningen.

Det kan ses at threadMapper caster argumentet til en *ThreadFunctor* pointer. Via. denne pointer kaldes *ThreadFunctor* funktionen run(). Det er denne funktion der indeholder den funktionalitet der skal udføres i tråden. I forbindelse med event-driven programmering er det denne der indeholder event loopet. Det er værd at bemærke at da run() er en member funktion i klassen *ThreadFunctor* har den adgang til dennes attributter, som f.eks. kunne være en besked-kø.

OS Api *pimpl* idiom usage

Pimpl opnås ved at tage oprette en forward declaration til sine private member variabler, som står i en anden cpp-fil, frem for inde i hpp-filen. I klassen definitionen sættes forward declaration som en pointer under private member variabler.

Man opnår herved, at sin implementation af en klasses member variabler gemmes væk, og kun er tilgængelig via pointeren. Man holder derfor andre i at kunne se selve koden fx. kunder eller mulige hacker.
Derudover hvis man ændrer på de private member variabler af klassen behøver man nu ikke at recompile det hele, da den ikke afhænger af dem, men af pointeren. Man vil derfor kunne oprette nyere versioner, hvor klassen vil kunne håndtere de nye samt gamle versioner.

Conditional friend of Mutex - why?

En friend class kan få adgang til private og protected dele af den klasse den er 'venner' med. Dette giver mulighed for at holde data og funktioner skjult, mens udvalgte klasser kan få adgang.
Conditional klassen er ven til mutex klassen fordi den skal bruge adgang til mutex klassen details, som er

private. Hvis man undersøger det nærmere er det specifikt mutex funktionen nativeHandle, som bruges at Conditional klassen.

Eksempel på at nativeHandle bruges af Conditional klassen:

```
void Conditional::wait(Mutex& mut)
{
  if(!SleepConditionVariableCS (&cond_, &mut.nativeHandle(), INFINITE)) throw ConditionalError();
}
```

Exercise 2 - Completing the Linux skeleton of the OO OS Api

Denne delopgave handler det om enten at oprette eller færdiggøre udleverede filer, så OSApi biblioteket er komplet og er klar til at kompilere. Alle filer er i linux delen af API'en.

Mutex-headeren skrives så den lever op til specifikationerne for Api'et.
Desuden gøres Conditional til friend af Mutex klassen. Dette er for at Conditional kan tilgå Mutex klassen private Mutex variabel.

inc/osapi/linux/Mutex.hpp

```
#ifndef OSAPI_LINUX_MUTEX_HPP
#define OSAPI_LINUX_MUTEX_HPP
#include <osapi/Utility.hpp>
#include <pthread.h>

namespace osapi
{
    class Conditional;

    class Mutex : Notcopyable
    {
    public:
        Mutex();
        void lock();
        void unlock();
        ~Mutex();

    private:
        friend class Conditional;
        pthread_mutex_t mut_;
    };

}
#endif
```

Mutex.cpp implementeres ved brug af de kendte pthread funktioner til manipulation af mutexes.

linux/Mutex.cpp

```
#include <osapi/Mutex.hpp>

namespace osapi
{
    Mutex::Mutex()
```

```
    {
        pthread_mutex_init(&mut_,NULL);

    }

    void Mutex::lock()
    {
        pthread_mutex_lock(&mut_);
    }

    void Mutex::unlock()
    {
        pthread_mutex_unlock(&mut_);
    }

    Mutex::~Mutex()
    {
        pthread_mutex_destroy(&mut_);

    }

}
```

Conditional.cpp filen færdiggøres, dvs. metoderne signal, broadcast og wait implementeres. Til dette bruges pthread_cond funktionerne _wait,_signal og _broadcast.

linux/Conditional.cpp

```
  void Conditional::signal()
  {
      pthread_cond_signal(&cond_);
  }

  void Conditional::broadcast()
  {
      pthread_cond_broadcast(&cond_);
  }

  void Conditional::wait(Mutex& mut)
  {
      pthread_cond_wait(&cond_,&mut.mut_);
  }
```

Da Utility.cpp ikke var en udleveret fil skulle den oprettes, formålet med denne funktion er at sleepe i et bestemt stykke tid. Derfor er det eneste som er implementeret er en void sleep. Grunden til at parameteren gannges med 1000, er fordi Utility-sleep tager imod millisekunder, hvor usleep bruger microsekunder, og ved at gange op fås det i samme forhold.

linux/Utility.cpp

```
#include <osapi/Utility.hpp>

namespace osapi
{
    void sleep(unsigned long msecs) {
```

```
        usleep(msecs * 1000);  // Ganges med 1000 da Utility bruger millisekunder og
    }
}
```

I Thread klassen manglede der er blive implementere Thread creation og metoder getName(). I thread creation sørges der for error handling med funktionen ThreadError(). Det sendes threadId, attr og threadMapper fra threadfunctor samt tf_(threafunctor) med som parametre.
getName() er en simpel get funktion der returnerer navnet på tråden.

linux/Thread.cpp

```
// Thread creation implementation

if(pthreadcreate(&threadId, &attr, ThreadFunctor::threadMapper, tf_) != 0) throw Thr

// Missing method imlpementation

std::string Thread::getName() const
  {
    return name_;
  }
```

Denne file eksisterede allerede og skulle derfor bare skrives færdig. Det som der er blevet tilføjet er *static_cast* og *run()* og den ser derfor således ud:

linux/ThreadFunctor.cpp

```
#include <osapi/Thread.hpp>

namespace osapi
{
  void* ThreadFunctor::threadMapper(void* thread)
  {
    /* Something is missing here - Determine what! */
    ThreadFunctor* tf = static_cast<ThreadFunctor*> (thread);
    tf->run();

    tf->threadDone_.signal();
    return NULL;
  }

}
```

Efterfølgende er det nu muligt at kompilere API'en ved hjælp ad den vedlagte makefile og gøres ved følgende kommando:

```
stud@stud-virtual-machine:~/Desktop/Courses/i3isu_f2020_beany_business/exe7/OSApiStudent$ make DEBUG=1 TARGET=host all
Generating dependency for linux/Thread.cpp
Compiling for 'host' in 'debug' mode...
Compiling linux/Thread.cpp
Linking build/lib/host/debug
stud@stud-virtual-machine:~/Desktop/Courses/i3isu_f2020_beany_business/exe7/OSApiStudent$
```

For at tjekke om programmet er blevet kompileret efterhensigten tjekkes om *libOSApi.a* er blevet oprette og ligger det ønskede sted.

```
stud@stud-virtual-machine:~/Desktop/Courses/i3isu_f2020_beany_business/exe7/OSApiStudent/lib/host/debug$ ls
libOSApi.a
```

Example

Ved hjælp af det vedlagte directory */example* testes de enkelte tråde i forhold to threadfunctor og deres messagequeue. For at komme så langt tjekkes om biblioteket virker efter hensigten

```
stud@stud-virtual-machine:~/i3isu_f2020_beany_business/exe7/OSApiStudent/example$ make DEBUG=1 TARGET=host all
g++ -ggdb -O0 -Wall -D_REENTRANT -DOS_LINUX -I../inc    -c -o main.o main.cpp
g++ -ggdb -O0 -Wall -D_REENTRANT -DOS_LINUX -I../inc    -c -o KeyBoardInput.o KeyBoardInput.cpp
g++ -ggdb -O0 -Wall -D_REENTRANT -DOS_LINUX -I../inc    -c -o LogSystem.o LogSystem.cpp
g++ -o main main.o KeyBoardInput.o LogSystem.o  -L../lib/host/debug -lOSApi -lrt -lpthread
```

Efterfølgende eksekveres programmet som ligger i *main* og fik følgende resultat:

```
stud@stud-virtual-machine:~/Desktop/Courses/i3isu_f2020_beany_business/exe7/OSApiStudent/example$ ./main
Lasse sucks
A lot
Of
Lollipops
^C
stud@stud-virtual-machine:~/Desktop/Courses/i3isu_f2020_beany_business/exe7/OSApiStudent/example$ ls
KeyBoardInput.cpp  KeyBoardInput.o  LogSystem.cpp  LogSystem.o  log.txt  main  main.cpp  main.o  Makefile
stud@stud-virtual-machine:~/Desktop/Courses/i3isu_f2020_beany_business/exe7/OSApiStudent/example$ cat log.txt
Lasse
sucks
A
lot
Of
Lollipops
stud@stud-virtual-machine:~/Desktop/Courses/i3isu_f2020_beany_business/exe7/OSApiStudent/example$
```

Igennem denne example-folder er det hermed osapi'ets threadfunctor og messagequeue virker. Programmet fungerer ved at en klasse indlæser keyboard inputs, og sender disse som beskeder til en log. Der gemmer de skrevne beskeder i en fil. På udklippet ses det at den skrevne tekst er gemt i filen "log.txt". Ergo må api'et også fungere til at sende beskeder mellem to tråde.

Test

For at teste de forskellige funktioner mere individuelt bruges directoryet */test*, den tilhørende make-file er dog mangelfuld og skal derfor også implementeres, der tages dog udgangspunkt fra den som var i example-folderen. Her vil loggen, threads og timer i forhold til sleep-funktionen.

/test Makefile

```
//Quick and dirty (does not handle changes in h-file)
SRCS=TestThread.cpp
OBJS=$(SRCS:.cpp=.o)
BASEPATH=..

//Determine whether this is a debug build or not
ifdef DEBUG
CXXFLAGS=-ggdb -O0
LIBPATH=$(BASEPATH)/lib/host/debug
else
CXXFLAGS=-O2
LIBPATH=$(BASEPATH)/lib/host/release
endif

//Setup the CFLAGS to ensure that the relevant warnings, includes and defines.
CXXFLAGS+=-Wall -D_REENTRANT -DOS_LINUX -I$(BASEPATH)/inc

%.o : %cpp
```

```
    g++ $(CXXFLAGS) -c -o $@ $^
//Then again, note how the flags are NOT part of the linking process
main: $(OBJS)
    g++ -o main $(OBJS)  -L$(LIBPATH) -lOSApi -lrt -lpthread

all: main

clean:$(OBJS)
    rm -f *.o main
```

TestLog

```
stud@stud-virtual-machine:~/i3isu_f2020_beany_business/exe7/OSApiStudent/test$ ./main
2020-04-20 18:36:40.962 DBG (TestLog.cpp:100 - main) Hello
^C
stud@stud-virtual-machine:~/i3isu_f2020_beany_business/exe7/OSApiStudent/test$
```

Ved dette testprogram oprettes en log med det specifikke tidspunkt med teksten hello, hvorefter den venter et stykke tid og lukker igen.

TestTimer

```
stud@stud-virtual-machine:~/i3isu_f2020_beany_business/exe7/OSApiStudent/test$ ./main
Running 1 test case...

*** No errors detected
stud@stud-virtual-machine:~/i3isu_f2020_beany_business/exe7/OSApiStudent/test$
```

Testthread

```
stud@stud-virtual-machine:~/i3isu_f2020_beany_business/exe7/OSApiStudent/test$ ./main
Iteration : 0
Iteration : 1
Iteration : 2
Iteration : 3
Iteration : 4
Iteration : 5
stud@stud-virtual-machine:~/i3isu_f2020_beany_business/exe7/OSApiStudent/test$
```

Navnet på testprogrammet fortæller allerede hvad den vil teste, theads. Den opretter og starter en thread, får den til at vente, hvorefter dens prioritet ændres, og denne process gentages 5 gange. Tessprogrammets output er for hver iteration.

Alle tre testprogrammer har heldigvis de forventede udfald, og det er hermed vist at de individuelle funktioner virker som de skal.

Exercise 3 - On target

I denne opgave skal OSApi testes om det også fungere på RPI, det kræver dog først at det bliver rekompileret til ARM-arkitektur som RPI'en kører med. Derfor sættes kompileren i makefile til at være *arm-rpizw-g++* istedet for, samt øverst tilføjes *CXX=arm-rpizw-g++* så det er sikkert den bruger ARM-kompileren. Derudover laves bibliotekstien også om fra *host* til det nyoprettede *target*. Makefilen ender derfor med at se således ud:

/example Makefile - Target

```
# Quick and dirty (does not handle changes in h-file)
CXX=arm-rpizw-g++                                                    //<- tilføje
```

```
SRCS=main.cpp KeyBoardInput.cpp LogSystem.cpp
OBJS=$(SRCS:.cpp=.o)
BASEPATH=..
# Determine whether this is a debug build or not
ifdef DEBUG
CXXFLAGS=-ggdb -O0
LIBPATH=$(BASEPATH)/lib/target/debug                    //<- ændring
else
CXXFLAGS=-O2
LIBPATH=$(BASEPATH)/lib/target/release                  //<- ændring
endif
# Setup the CFLAGS to ensure that the relevant warnings, includes and defines.
CXXFLAGS+=-Wall -D_REENTRANT -DOS_LINUX -I$(BASEPATH)/inc
#%.o : %cpp
#    arm-rpizw-g++ $(CXXFLAGS) -c -o $@ $^          //<- ændring
# Then again, note how the flags are NOT part of the linking process
main: $(OBJS)
    arm-rpizw-g++ -o main $(OBJS)  -L$(LIBPATH) -lOSApi -lrt -lpthread
all: main
clean:$(OBJS)
    rm -f *.o main
```

Efterfølgende kopiers programmet til til target og eksekveres, hvilket gav følgende resultat:

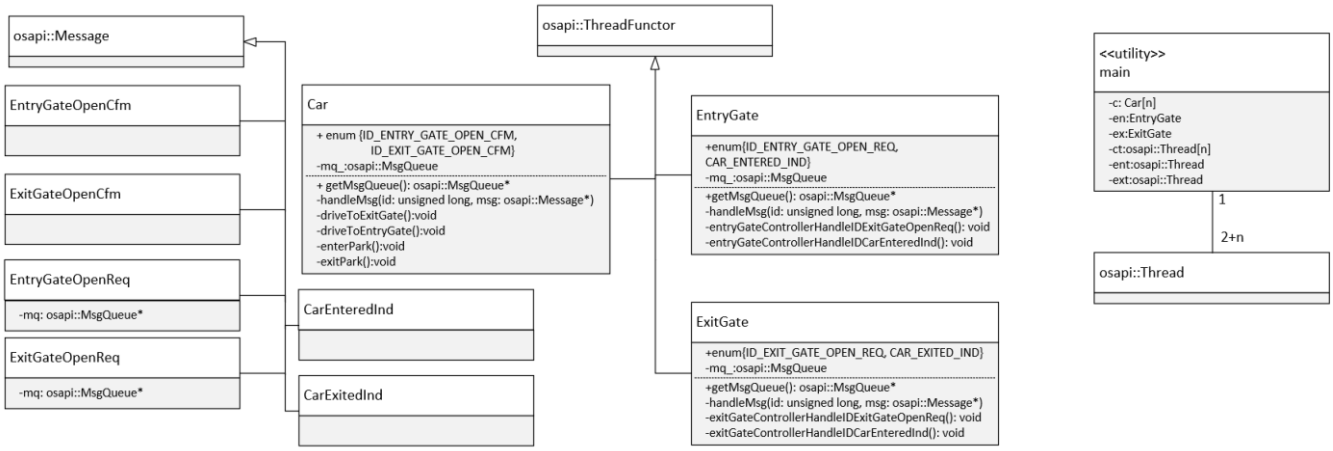
```
root@raspberrypi0-wifi:~/dest# ./main
Iteration : 0
Iteration : 1
Iteration : 2
Iteration : 3
Iteration : 4
Iteration : 5
root@raspberrypi0-wifi:~/dest#
```

Herved er det vist at det virker at bruge vores OSApi til et target.

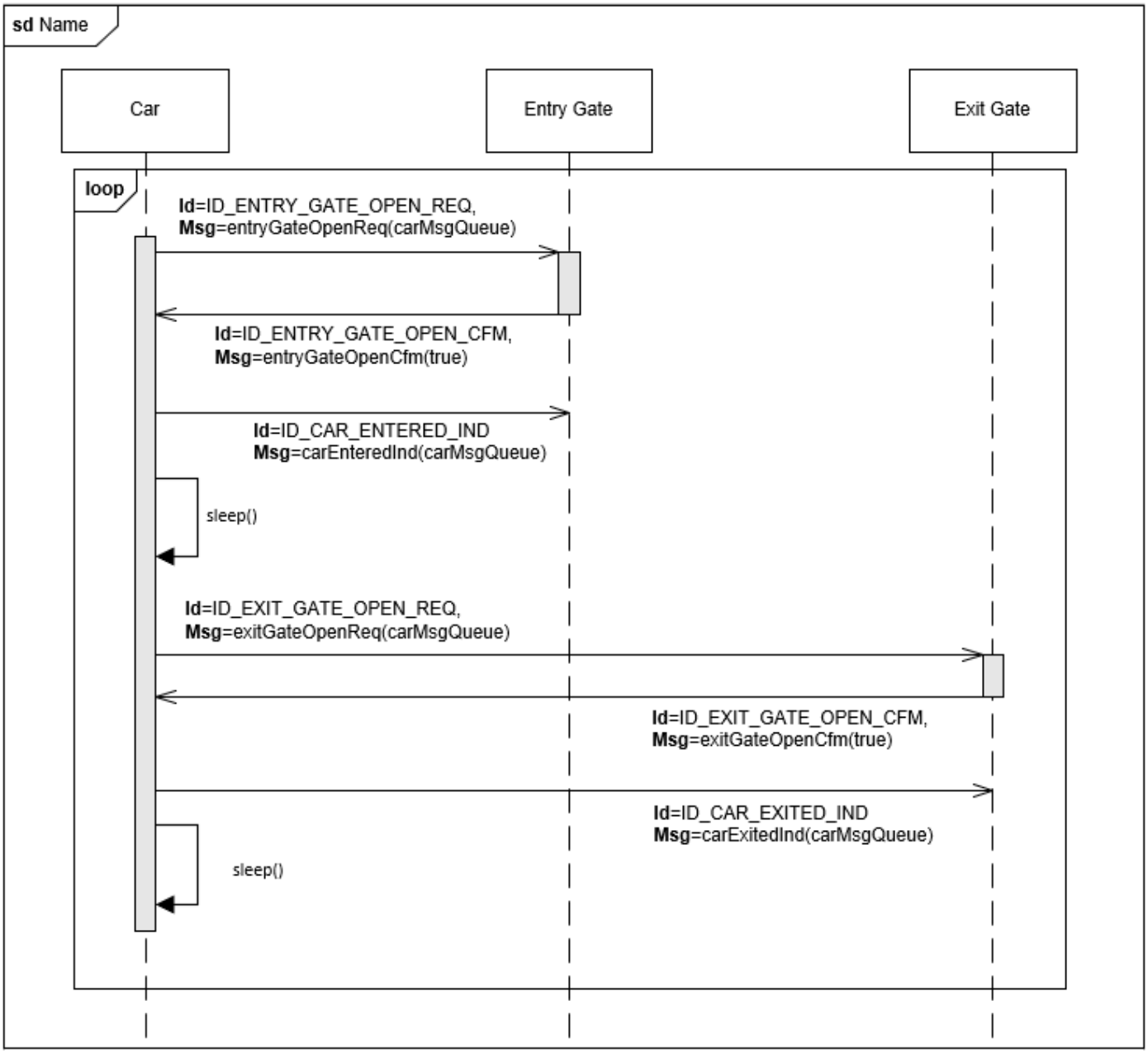
Exercise 4 - PLCS now the OS Api

Design

Denne delopgave handler om at udvide det eller velfungerende PLCS program. Så i stedet for at bruge de nuværende Message og Message queues, skal applikationen nu køre som et OS Api.
For ikke at miste overblikket laves der et klasse-diagram samt et sekvens-diagram. Klasse-diagrammet for dette program kommer derfor til at se således ud:



Der burges det samme sekvens-diagram fra Exercise 6 - Thread Communication, da programmet vil fungere på samme måde, og der er derfor ikke her ændringerne vil forekomme.



Implementing

Car.hpp

```
#ifndef CAR_HPP_
#define CAR_HPP_

#include <osapi/MsgQueue.hpp>
#include <osapi/ThreadFunctor.hpp>
#include "EntryGate.hpp"
#include "ExitGate.hpp"

struct EntryGateOpenCfm : public osapi::Message
{
};
struct ExitGateOpenCfm : public osapi::Message
{
};

class EntryGate;
class ExitGate;

class Car : public osapi::ThreadFunctor
{
public:
    enum{
        EXIT_GATE_OPEN_CFM,
        ENTRY_GATE_OPEN_CFM
    };
    static const int MAX_QUEUE_SIZE=1;
    Car(int number, EntryGate* entrygate, ExitGate* exitgate);

private:
    void driveToEntryGate();
    void enterPark();
    void driveToExitGate();
    void exitPark();
    void handleMsg(unsigned long id, osapi::Message*);

protected:
    void run();
    osapi::MsgQueue mq_;
    EntryGate* entryGate_;
    ExitGate* exitGate_;
    int number_;
};

#endif
```

Car.cpp

```
#include "Car.hpp"
#include <iostream>
#include <unistd.h>
Car::Car(int number, EntryGate* entrygate, ExitGate* exitgate)
: mq_(MAX_QUEUE_SIZE)
{
    number_=number;
    entryGate_=entrygate;
    exitGate_=exitgate;
}
```

```
void Car::driveToEntryGate()
{
    entryGate_->mut_.lock();
    entryGate_->cond_.broadcast();
    while(entryGate_->carAtGate)
    {

        entryGate_->cond_.wait(entryGate_->mut_);
    }
    entryGate_->carAtGate=true;
    std::cout<<"Car nr. "<<number_<<" driving up to entry gate."<<std::endl;
    EntryGateOpenReq* openReq=new EntryGateOpenReq;
    openReq->mq=&mq_;
    entryGate_->getMsgQueue()->send(EntryGate::ENTRY_GATE_OPEN_REQ,openReq);
    entryGate_->mut_.unlock();

}

void Car::enterPark()
{
    entryGate_->mut_.lock();
    while(!entryGate_->open)
    {
    entryGate_->cond_.wait(entryGate_->mut_);
    }
    CarEnteredInd* enteredInd= new CarEnteredInd;
    std::cout<<"Car nr. "<<number_<<" entered park."<<std::endl;
    entryGate_->getMsgQueue()->send(EntryGate::CAR_ENTERED_IND,enteredInd);
    entryGate_->mut_.unlock();

    sleep(number_);
    driveToExitGate();

}

void Car::driveToExitGate()
{
    exitGate_->mut_.lock();
    exitGate_->cond_.broadcast();
    while(exitGate_->carAtGate)
    {

        exitGate_->cond_.wait(exitGate_->mut_);
    }
    exitGate_->carAtGate=true;
    std::cout<<"Car nr. "<<number_<<" driving up to exit gate."<<std::endl;
    ExitGateOpenReq* openReq=new ExitGateOpenReq;
    openReq->mq=&mq_;
    exitGate_->getMsgQueue()->send(ExitGate::EXIT_GATE_OPEN_REQ,openReq);
    exitGate_->mut_.unlock();
}

void Car::exitPark()
{
    exitGate_->mut_.lock();
    while(!exitGate_->open)
    {
```

```
    exitGate_->cond_.wait(exitGate_->mut_);
    }
    CarExitedInd* exitedInd= new CarExitedInd;
    std::cout<<"Car nr. "<<number_<<" exited park."<<std::endl;
    exitGate_->getMsgQueue()->send(ExitGate::CAR_EXITED_IND,exitedInd);
    exitGate_->mut_.unlock();
    sleep(number_*2);
    driveToEntryGate();
}

void Car::handleMsg(unsigned long id, osapi::Message*)
{
    switch(id)
    {
    case EXIT_GATE_OPEN_CFM:
        exitPark();
        break;
    case ENTRY_GATE_OPEN_CFM:
        enterPark();
        break;
    default:
        std::cout<<"Unknown event..."<<std::endl;
        break;
    }
}

void Car::run()
{
    driveToEntryGate();
    for(;;)//ever
    {
        unsigned long id;
        osapi::Message* msg=mq_.receive(id);
        handleMsg(id,msg);
        delete msg;
    }
}
```

EntryGate.hpp

```
#ifndef ENTRY_GATE_HPP_
#define ENTRY_GATE_HPP_

#include <osapi/MsgQueue.hpp>
#include <osapi/ThreadFunctor.hpp>
#include <osapi/Mutex.hpp>
#include <osapi/Conditional.hpp>
#include "Car.hpp"

struct EntryGateOpenReq: public osapi::Message
{
    osapi::MsgQueue* mq;
};
struct CarEnteredInd: public osapi::Message
{
};
```

```
class EntryGate: public osapi::ThreadFunctor
{
public:
    enum{
        CAR_ENTERED_IND,
        ENTRY_GATE_OPEN_REQ
    };
    static const int MAX_QUEUE_SIZE=10;
    EntryGate();
    osapi::MsgQueue* getMsgQueue();
    osapi::Mutex mut_;
    osapi::Conditional cond_;
    bool open;
    bool carAtGate;

private:
    osapi::MsgQueue mq_;
    void handleMsg(unsigned long id, osapi::Message*);
    void handleIDEntryGateOpenReq(EntryGateOpenReq *req);
    void handleIDCarEnteredInd();

protected:
    void run();
};

#endif
```

EntryGate.cpp

```
#include "EntryGate.hpp"
#include <iostream>

    EntryGate::EntryGate()
    : mq_(MAX_QUEUE_SIZE)
    {
        open=false;
        carAtGate=false;
    }

    void EntryGate::handleMsg(unsigned long id, osapi::Message* msg)
    {
        switch(id)
        {
        case CAR_ENTERED_IND:
            handleIDCarEnteredInd();
            break;
        case ENTRY_GATE_OPEN_REQ:
            handleIDEntryGateOpenReq(static_cast<EntryGateOpenReq*>(msg));
            break;
        default:
            std::cout<<"Unknown event..."<<std::endl;
            break;
        }
    }

    void EntryGate::handleIDEntryGateOpenReq(EntryGateOpenReq *req)
    {
```

```
        osapi::MsgQueue* mq=req->mq;
        std::cout<<"Opening entry gate."<<std::endl;
        open=true;
        EntryGateOpenCfm* cfm=new EntryGateOpenCfm;
        mq->send(Car::ENTRY_GATE_OPEN_CFM,cfm);
        cond_.broadcast();

    }

    void EntryGate::handleIDCarEnteredInd()
    {

        std::cout<<"Closing entry gate."<<std::endl;
        open=false;
        carAtGate=false;
        cond_.broadcast();

    }

    void EntryGate::run()
    {
        for(;;)//ever
        {
            unsigned long id;
            osapi::Message* msg=mq_.receive(id);
            handleMsg(id,msg);
            delete msg;
        }
    }

    osapi::MsgQueue* EntryGate::getMsgQueue()
    {
        return &mq_;
    }
```

ExitGate.hpp og ExitGate kan findes i Repository'et

main.cpp

```
#include "Car.hpp"
#include "EntryGate.hpp"
#include "ExitGate.hpp"
#include <osapi/Thread.hpp>

int main()
{
    EntryGate en;
    ExitGate ex;
    Car *carArray[5];
    osapi::Thread *tArray[5];
    for(int i=1;i<=5;++i)
    {
        carArray[i-1]=new Car(i,&en,&ex);
        tArray[i-1]= new osapi::Thread(carArray[i-1]);
    }
```

```
    osapi::Thread ent(&en);

    osapi::Thread ext(&ex);

    ext.start();
    ent.start();
    for(int i=0;i<5;++i)
    {
        tArray[i]->start();
    }

    ent.join();
    ext.join();
    for(int i=0;i<5;++i)
    {
        tArray[i]->join();
    }

    delete[] tArray;
    delete[] carArray;
}
```

Makefile

```
# Quick and dirty (does not handle changes in h-file)
SRCS=main.cpp Car.cpp EntryGate.cpp ExitGate.cpp
OBJS=$(SRCS:.cpp=.o)
BASEPATH=..
# Determine whether this is a debug build or not
ifdef DEBUG
CXXFLAGS=-ggdb -O0
LIBPATH=$(BASEPATH)/lib/host/debug
else
CXXFLAGS=-O2
LIBPATH=$(BASEPATH)/lib/host/release
endif
# Setup the CFLAGS to ensure that the relevant warnings, includes and defines.
CXXFLAGS+=-Wall -D_REENTRANT -DOS_LINUX -I$(BASEPATH)/inc
#%.o : %cpp
#    g++ $(CXXFLAGS) -c -o $@ $^
# Then again, note how the flags are NOT part of the linking process
main: $(OBJS)
    g++ -o main $(OBJS)  -L$(LIBPATH) -lOSApi -lrt -lpthread
all: main
clean:$(OBJS)
    rm -f *.o main
```

Resultater

PLCS med en bil:

```
stud@stud-virtual-machine:~/Desktop/Courses/i3isu_f2020_beany_business/exe7/OSApiStudent/carpark_try$ make DEBUG=1 all
g++ -ggdb -O0 -Wall -D_REENTRANT -DOS_LINUX -I../inc   -c -o Car.o Car.cpp
g++ -o main main.o Car.o EntryGate.o ExitGate.o  -L../lib/host/debug -lOSApi -lrt -lpthread
stud@stud-virtual-machine:~/Desktop/Courses/i3isu_f2020_beany_business/exe7/OSApiStudent/carpark_try$ ./main
Car nr. 1 driving up to entry gate.
Opening entry gate.
Car nr. 1 entered park.
Closing entry gate.
Car nr. 1 driving up to exit gate.
Opening exit gate.
Car nr. 1 exited park.
Closing exit gate.
Car nr. 1 driving up to entry gate.
Opening entry gate.
Car nr. 1 entered park.
Closing entry gate.
Car nr. 1 driving up to exit gate.
Opening exit gate.
Car nr. 1 exited park.
Closing exit gate.
^C
```

PLCS med flere biler:

```
stud@stud-virtual-machine:~/Desktop/Courses/i3isu_f2020_beany_business/exe7/OSApiStudent/carpark_try$ ./main
Car nr. 1 driving up to entry gate.
Opening entry gate.
Car nr. 1 entered park.
Closing entry gate.
Car nr. 4 driving up to entry gate.
Opening entry gate.
Car nr. 4 entered park.
Closing entry gate.
Car nr. 2 driving up to entry gate.
Opening entry gate.
Car nr. 2 entered park.
Closing entry gate.
Car nr. 5 driving up to entry gate.
Opening entry gate.
Car nr. 5 entered park.
Closing entry gate.
Car nr. 3 driving up to entry gate.
Opening entry gate.
Car nr. 3 entered park.
Closing entry gate.
Car nr. 1 driving up to exit gate.
Opening exit gate.
Car nr. 1 exited park.
Closing exit gate.
Car nr. 2 driving up to exit gate.
Opening exit gate.
Car nr. 2 exited park.
Closing exit gate.
Car nr. 1 driving up to entry gate.
Car nr. 3 driving up to exit gate.
Opening entry gate.
Opening exit gate.
Car nr. 3 exited park.
Closing exit gate.
Car nr. 1 entered park.
Closing entry gate.
^C
```

Diskussion

Kort sagt er denne opgave en objekt-orienteret løsning på samme problem, som blev løst i Exercise 6. Dvs. at i stedet for frie; funktioner til handling af beskeder, enums med beskedID, beskedkø, trådfunktion, mutex osv. er der oprettet en klasse til hver tråd, der indkapsulerer den nødvendige funktionalitet for at have et EDP-baseret tråd-program.
Denne indkapsulering gør programmet mere fejlsikkert, dette skyldes at man ved at implementere hver tråds attributter i en klasse, får en compilerfejl hvis man forsøger at benytte f.eks. en forkert beskedkø i sin tråd. Denne beskyttelse havde man ikke i den forrige version af programmet(exercise 6).

Altså er de mest essentielle ændringer der er foretaget, at de enkelte trådes attributter nu på compilerniveau er knyttet til de enkelte tråde, mens det tidligere blot var en "kontrakt" som programmøren opstillede.

Derved kan man altså også sige at ændringen forbedrer programmet. Desuden er indkapslingen af kode markant større, da det eneste der skal foretages i main er at oprette objekter af de tre klasser, samt trådobjekter til disse.

Konklusion

Denne øvelse har givet en større og mere grund forståelse af hvordan OO OS Api biblioteker er forbundet, som yderligere gjorde det muligt at færdigøre OSApi biblioteket, som så både virkede på host og target. Efter dette var vi rystet så godt på at det var muligt at forbedre PLCS til at køre på OS Api.

- **Tilføjet af Asger Holm Brændgaard for cirka en måned siden**

  Learning objective #1 Lave et afsnit der forklarer strukturen og den basale funktionalitet af OO OS API

  - Gruppen formår at forklare den basale struktur for biblioteket. Der pointeres f.eks hvordan operativ systemet defineres i *compiler_setup*, og hvad dette betyder når biblioteket bliver oprettet, godt. Der kunne med fordel sættes screendumps ind af mappestrukturen. Pimpl idiomet bliver også beskrevet. **OK**

  Learning objective #2 Være i stand til at færdiggøre de manglende dele af OS API library

  - Gruppen formår at færdiggøre biblioteket med gode kommentarer til hvilke dele der er blevet opdateret. **OK**

  Learning objective #3 Make and test the OS API

  - Skærmbilleder er lavet, som fremviser en succesfuld indsættelse af biblioteket. Dertil kommer en række test af biblioteket. Disse test viser korrekt funktionalitet af biblioteket og denne funtktioner. **OK**

  Learning objective #4 Implementer PLCS ved brug af OS API

  - Klasse - og sekvensdiagram for den nye iteration af PLCS er lavet. Her i OS API'en tilføjet. Dernæst er alle klasser blevet opdateret med *osapi*. Til slut er programmet bliver kørt, og ud fra de kørte tests, kan det konkluderes, at gruppen har formået af udnytte biblioteket til forbedre deres program. **OK**

  Feedback:
  Alt i alt en god opgave. Der bliver givet gode forklaringer på hvilke skridt der bliver taget, og derefter konkluderet tilstrækkeligt på disse. Gruppen formår yderemere at sætte biblioteket op til at kunne fungerer på RPI, ved at ændre for hvilken arkitektur denne skal bygges på.
  Der er lidt lemfældig omgang med brug af screendumps samt (pre/code) strukturen. Den eneste rettelse skulle være, at man skulle holde den samme linje igennem hele wiki'en. Dette er dog en mere æstetisk rettelse, som gør det "rarere" for læseren.
  Overvej at ædnre strukturen for jeres repository, så det er klart hvilke makefiles der hører til hvilke exercises, og de forskellige iterationer af biblioteket.
  Must have:

  - Relevant files in repository **OK**
  - Makefiles in repository **OK**

  Conclusion:

  En god wiki, hvor der er blevet lagt arbejde i at skrive forklarende tekst udover de indsatte billeder. Super godt arbejde drenge.
  4/4 Læringsmål er nået. Bedre end godt. 🙂

include_mutexhpp.png (16,6 KB) Magnus Nygaard Lund, 2020-04-20 09:28

OS_mutexhpp.png (21,9 KB) Magnus Nygaard Lund, 2020-04-20 09:29

osapi_mutexhpp.png (35,2 KB) Magnus Nygaard Lund, 2020-04-20 10:25

threadfunctor.png (29,3 KB) Magnus Nygaard Lund, 2020-04-20 10:27

eksempel_nativeahndle.png (16,2 KB) Magnus Nygaard Lund, 2020-04-20 10:29

biblo.png (27 KB) Magnus Nygaard Lund, 2020-04-20 19:51
lasse_sucker.png (45,7 KB) Magnus Nygaard Lund, 2020-04-20 19:51
libOSApi.png (9,47 KB) Magnus Nygaard Lund, 2020-04-20 19:51
exampel_biblo.png (25,2 KB) Magnus Nygaard Lund, 2020-04-20 20:16
testthread.png (19,1 KB) Magnus Nygaard Lund, 2020-04-20 20:44
testtime.png (14,9 KB) Magnus Nygaard Lund, 2020-04-20 20:44
hello_testlog.png (16,2 KB) Magnus Nygaard Lund, 2020-04-20 20:45
target.png (9,5 KB) Magnus Nygaard Lund, 2020-04-21 10:46
opg_3_sekvens.png (25,4 KB) Magnus Nygaard Lund, 2020-04-23 08:52
klassediagram.png (108 KB) Magnus Nygaard Lund, 2020-04-23 09:32
PLCS_OSApi.png (59,3 KB) Magnus Nygaard Lund, 2020-04-23 11:30
PLCS_OSApi_flerebiler.png (83,3 KB) Magnus Nygaard Lund, 2020-04-23 12:44