Gruppe 33 - Exercise7 - # 54

OS API

"In this exercise you will get the skeleton for an OO OS API, and it will be your job to implement the missing pieces based on the knowledge you have acquired. This will enable you to compile your own OO OS Api library for use in your applications/project."

Læringsmål for øvelsen:

1. Forstå struktur og funktionalitet ift. implementeringen af OS API
2. Færdiggøre implementeringen af det udlevere OS API, samt teste og copmilere til Host og Target
3. Kunne implementere og arbejde med threads vha. inheritance metoden - run()
4. Modificere/ Udbygge PLCS vha. den færdiggjorte OS API.

Exercise 1 - Getting to know the OO OS Api

1.1 - Questions to answer:

• Inspecting the OSApi library, describe the chosen setup and how it works.
Hint: The chosen: Directory structure, define usage, platform handling (e.g. windows, linux etc.) etc.

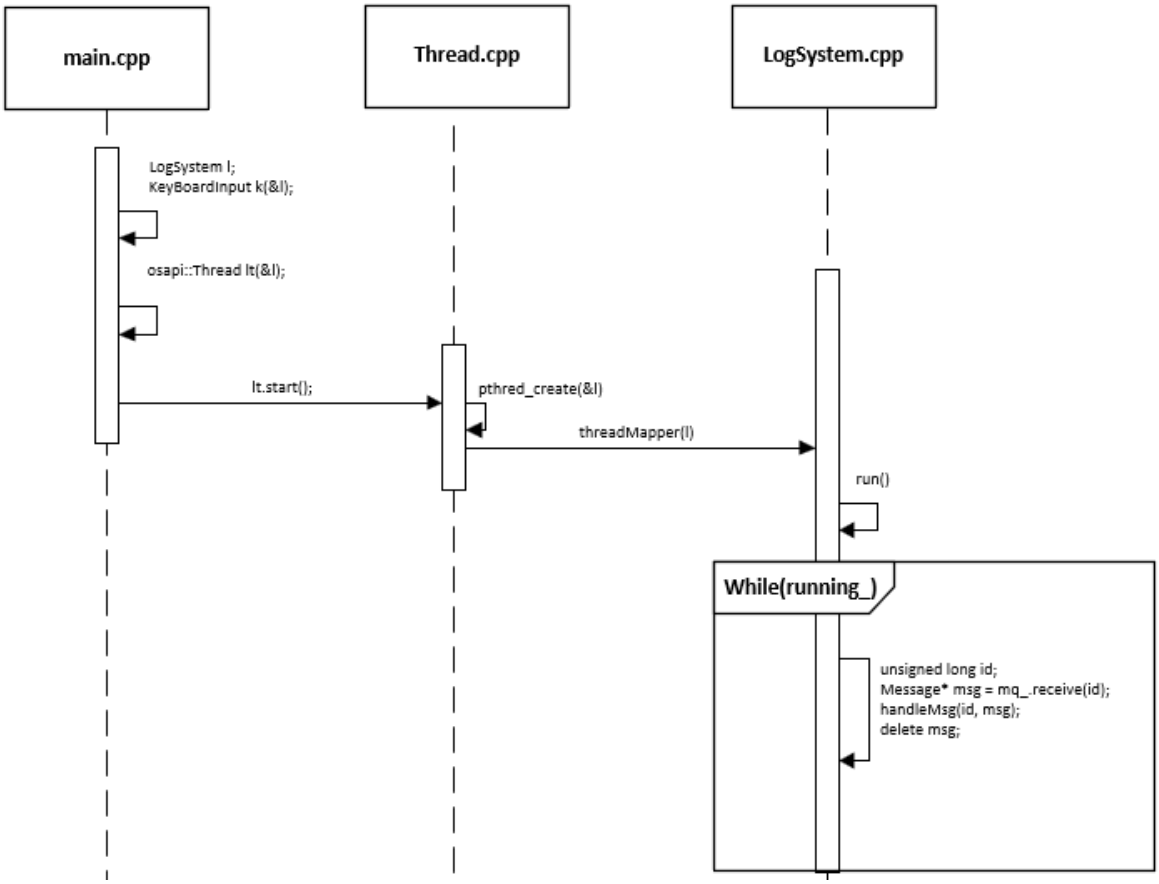
> Strukturen for den udleverede OS API "OSAPIStudent", er lave med henblik til at fungere til både windows og linux.
> Yderst er makefilen placeret samt compiler setup til target/host. Her ligger desuden .cpp filer opdelt i windows, linux og common, samt en /inc med netop OSAPI header filerne.
> I /osAPI er der header filer, som med en if sætning, linker til en specifik OS header-fil i enten en windows- eller linux folder.
> Eksmepelvis:

```
#ifndef OSAPI_COND_HPP
#define OSAPI_COND_HPP

#include <osapi/Utility.hpp>

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

#endif
```

> Der er i det udleverede materiale desuden en mappe /exampe og /test som indeholder programmer til at teste den endelige implementering.

• When using the POSIX thread API one must provide a free C function as the thread function. This means that some glue is inserted such that it is possible to have a method of a class as the thread function (run()). Describe which classes are involved, their responsibilities and how they interact. A good approach could be show a call stack or a sequence diagram.

> ThreadFunctor:ThreadMapper benyttes som fri/global variabel når der oprettes POSIX threads.
> Logsystem har nedarvet osapi::ThreadFunctor og kan derfor kalde run().
> Interaktion mellem klasser og main.cpp vises nedenfor i et sekvensdiagram.



• In the windows implementation the so-called pimpl/cheshire cat idiom is used. Explain how it is used and what is achieved in this particular situation.
See for instance the files related to the Semaphore class.

pimpl(Pointer_To_Implementation) anvendes således at hvis en klasse opdateres(her Semaphore), skal alle der arbejder med denne klasse ikke re-compile. I header filen for klassen implementeres derfor en pointer til klassen, som her er i /osapi/winx86/details

Thread.cpp | Semaphore.hpp | ThreadFunctor.cpp

stud > isu_work > exercise7 > OSApiStudent > inc > osapi > winx86 > details > Se

```
#ifndef OSAPI_WINX86_SEMAPHORE_DETAILS_HPP
#define OSAPI_WINX86_SEMAPHORE_DETAILS_HPP

#include <windows.h>
#include <osapi/Utility.hpp>

namespace osapi
{
    namespace details
    {
      class Semaphore : Notcopyable
      {
      public:
        Semaphore(unsigned int initCount);
        void wait();
        void signal();
        ~Semaphore();
      private:
        HANDLE id_;
      };
    }
}



#endif
```

Semaphore.hpp

home > stud > isu_work > exercise7 > OSApiStudent > inc > osapi > winx86 > Semaphore

```
1   #ifndef OSAPI_WINX86_SEMAPHORE_HPP
2   #define OSAPI_WINX86_SEMAPHORE_HPP
3
4   #include <osapi/Utility.hpp>
5
6   namespace osapi
7   {
8     namespace details
9     {
10      class Semaphore;
11    }
12
13    class Semaphore : Notcopyable
14    {
15    public:
16      Semaphore(unsigned int initCount);
17      void wait();
18      void signal();
19      ~Semaphore();
20    private:
21      details::Semaphore*   sem_;
22    };
23  }
24
25  #endif
26
```

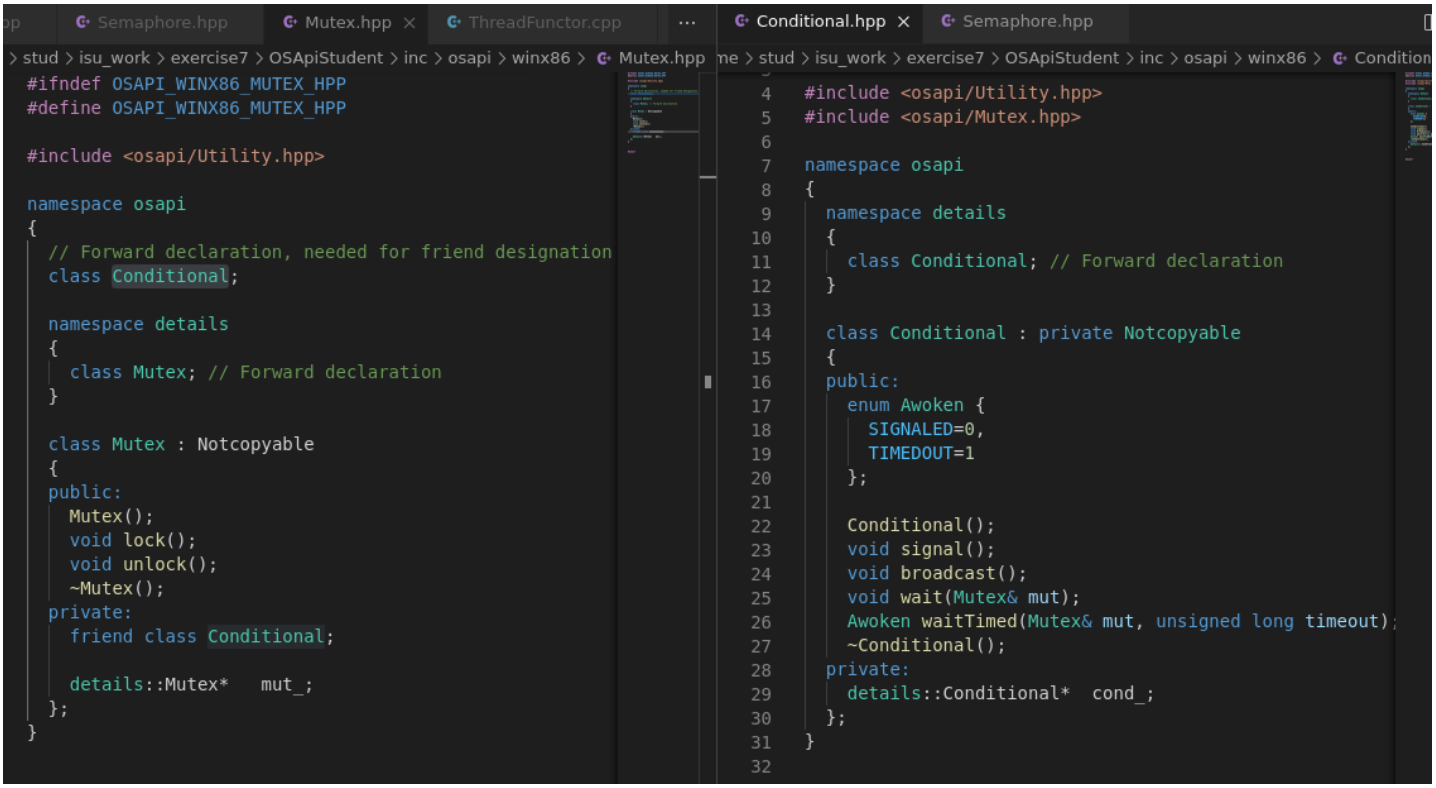
Semaphore.cpp

home > stud > isu_work > exercise7 > OSApiStudent > winx86 > Semaphore.cp

```
1   #include <osapi/Semaphore.hpp>
2   #include <osapi/winx86/details/Semaphore.hpp>
3
4   namespace osapi
5   {
6     Semaphore::Semaphore(unsigned int initCount)
7       :   sem_(new details::Semaphore(initCount))
8     {
9     }
10
11
12    void Semaphore::wait()
13    {
14      sem_->wait();
15    }
16
17
18    void Semaphore::signal()
19    {
20      sem_->signal();
21    }
22
23
24    Semaphore::~Semaphore()
25    {
26      // Needed for incomplete type - MUST be in the cpp f
27    delete sem_;
28    }
29
30  }
```

• Why is the class Conditional a friend to class Mutex? What does it mean to be a friend?
Hint: See the interface for class Mutex as coded for the windows platform.

En Friend class, kan tilgå de members som er private eller protected hos den klasse der erklæres som Friend. Implementering vises nedenfor for Mutex og Conditional:



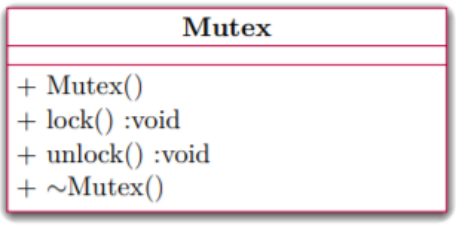
Exercise 2 - Completing the Linux skeleton of the OO OS Api

2.1 - The OSApi library

De filer som mangler, helt eller delvist, implementeres nu med udgangspunkt i: Specification_of_an_OS_Api.pdf

2.1.1 - inc/osapi/linux/Mutex.hpp

Implementeringen af Mutex.hpp er vist nedenfor.

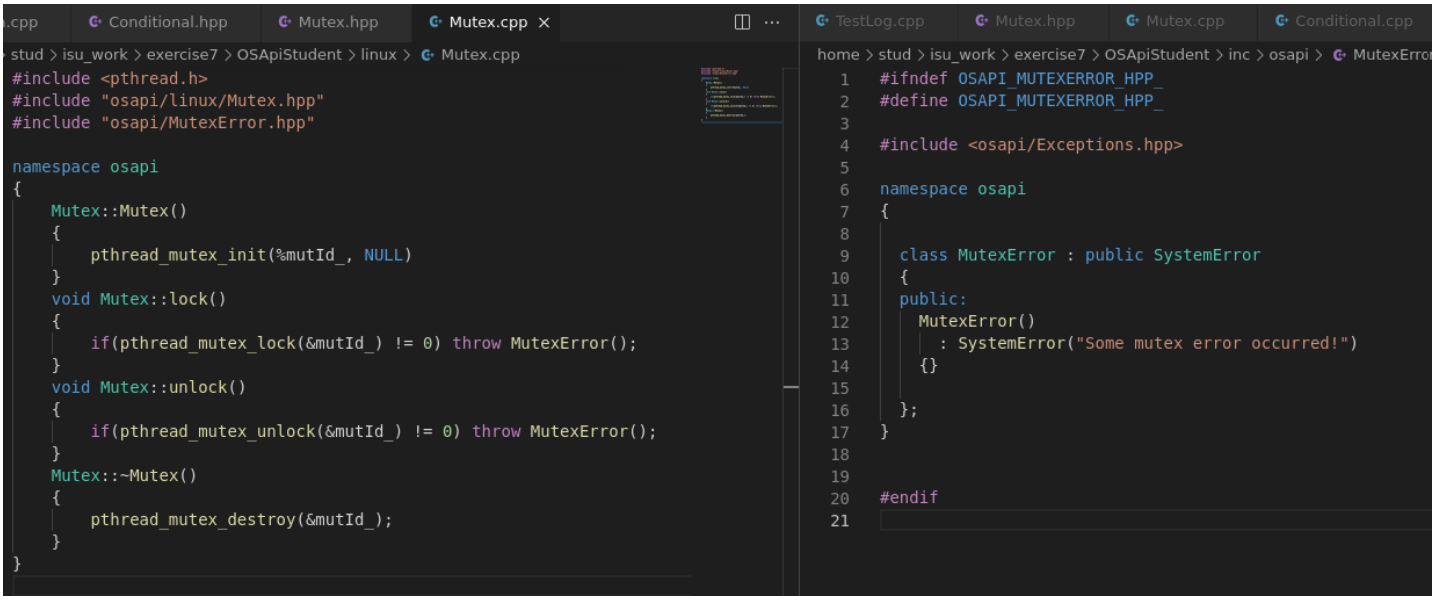


Function name	Functional description
Mutex	Constructor
lock	Locks the mutex, precondition is that it is *NOT* locked, otherwise undefined behavior.
unlock	Unlocks the mutex, precondition is that it *IS* locked, otherwise undefined behavior.
~Mutex	Destructor

```
#ifndef OSAPI_LINUX_MUTEX_HPP
#define OSAPI_LINUX_MUTEX_HPP
#include <pthread.h>
#include <osapi/Utility.hpp>
namespace osapi
{
  class Mutex : Notcopyable
  {
  public:
    Mutex();
    void lock();
    void unlock();
    ~Mutex();

  private:
    pthread_t mutId_;
  };
}

#endif
```

2.1.2 - linux/Mutex.cpp

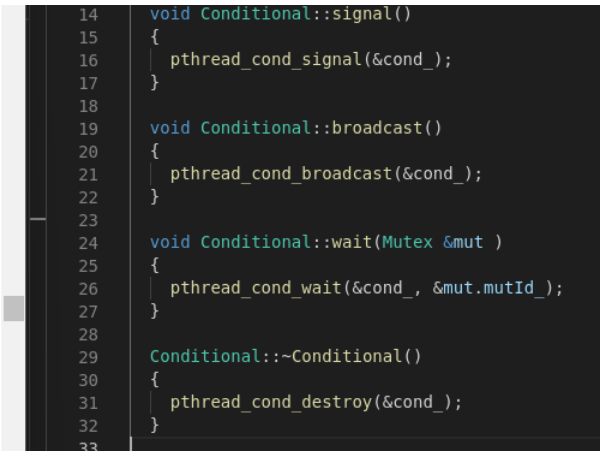
Implementering af Mutex.cpp vises nedenfor hvor MutexError.hpp er benyttet til fejlhåndteringen.



2.1.3 - linux/Conditional.cpp

Færdiggørelsen af Conditional.cpp er vist nedenfor.

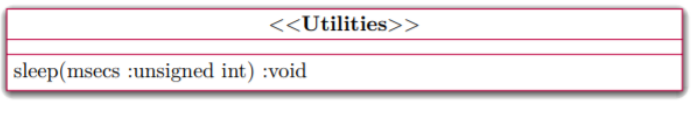
Function name	Functional description
`Awoken :enum`	Enum which is a return value for `waitTimed`, such that the reason be gauged by the calling party. • SIGNALED • TIMEDOUT
`Conditional`	Constructor
`signal`	Signals a *single* thread waiting on the conditional
`broadcast`	Signals *all* threads waiting on the conditional
`wait`	Waits on the conditional to signal it, meaning that the calling thread is suspended.
`waitTimed`	Like `wait`, but with the ability to state a timeout on the length which the calling thread is prepared to wait.
`~Conditional`	Destructor



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

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

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

Conditional::~Conditional()
{
  pthread_cond_destroy(&cond_);
}
```

2.1.4 - linux/Utility.cpp

Implementeringen af Utility.cpp er vist nedenfor. Da der i linux versionen bennyttes usleep istedet for sleep multipliceres msecs.



Function name	Functional description
`sleep`	Suspends the current thread of execution for that amount of seconds. *Do note that even though the delay is in millisecond, you are not guaranteed that kind of precision. It highly depends on the kernel/OS you are using.*

Table 2.2: Elaboration on Timer utilities.

```
#include <unistd.h>
#include <osapi/Utility.hpp>
namespace osapi
{
void sleep (unsigned long msecs);
{
    usleep(msecs*1000);

}
}
```

2.1.5 - linux/Thread.cpp

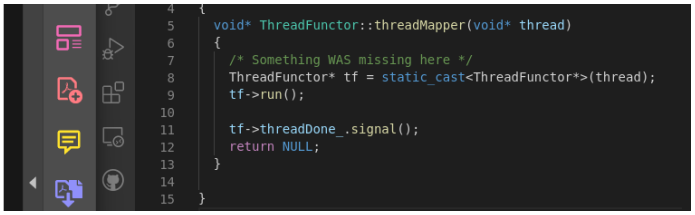
Koden til færdiggørelsen af Thread.cpp er vist nedenfor. I programmet er der tilføjet thread creation og getName():
De ønskede default værdier fra "Specification_of_an_OS_Api" er allerede implementeret i header filen.

```
.
.
/* Thread creation */
      if(pthread_create(&threadId_, &attr, ThreadFunctor::threadMapper, tf_) != 0) throw ThreadErr
or();

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

2.1.6 - linux/ThreadFunctor.cpp

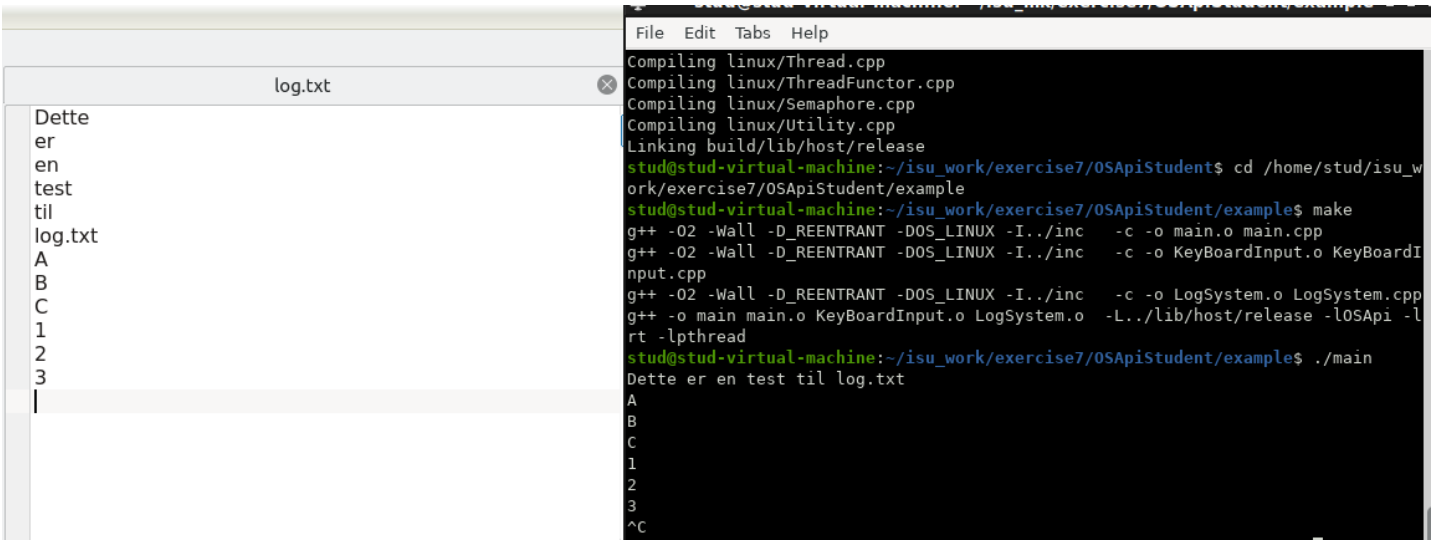
Færdiggørelsen af ThreadFunctor.cpp er vist nedenfor og den manglende kode er netop færdiggørelsen af det "workaround", så metoden run() kan bennytes som thread funktionen, når der laves threads. Kode udsnittet til venstre er fra undervisningsmaterialet til øvelsen.

```
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 }
```



2.2 - Test af implementering og færdiggørelsen af OSApi library

2.2.1 - Test af skrivning til Log.txt



2.2.2 - Test af Timer

```
stud@stud-virtual-machine:~/isu_work/exercise7/OSApiStudent/test$ ./TestTimer
2021-11-24 13:35:57.694 DBG (TestTimer.cpp:104 - run) Creating and arming timer...
2021-11-24 13:35:57.694 DBG (TestTimer.cpp:114 - run) Starting event loop
2021-11-24 13:35:58.695 DBG (TestTimer.cpp:57 - handleTimeOut1) Got timeout1, rearming...
2021-11-24 13:35:58.695 DBG (TestTimer.cpp:63 - handleTimeOut2) Got timeout 2, rearming...
2021-11-24 13:35:59.695 DBG (TestTimer.cpp:57 - handleTimeOut1) Got timeout1, rearming...
2021-11-24 13:35:59.695 DBG (TestTimer.cpp:63 - handleTimeOut2) Got timeout 2, rearming...
2021-11-24 13:36:00.195 DBG (TestTimer.cpp:69 - handleTimeOut3) Got timeout 3, rearming...
2021-11-24 13:36:00.695 DBG (TestTimer.cpp:57 - handleTimeOut1) Got timeout1, rearming...
2021-11-24 13:36:00.695 DBG (TestTimer.cpp:63 - handleTimeOut2) Got timeout 2, rearming...
2021-11-24 13:36:01.695 DBG (TestTimer.cpp:91 - handler) Got termination signal
2021-11-24 13:36:01.695 DBG (TestTimer.cpp:128 - run) Thread terminating...
```

2.2.3 - Test af Threads

```
stud@stud-virtual-machine:~/isu_work/exercise7/OSApiStudent/test$ ./TestThread
Iteration : 0
Iteration : 1
Iteration : 2
Iteration : 3
Iteration : 4
Iteration : 5
```

2.2.4 - Test af Time

```
stud@stud-virtual-machine:~/isu_work/exercise7/OSApiStudent/test$ ./TestTime
Running 1 test case...

*** No errors detected
```

2.2.5 - Test af Log

```
stud@stud-virtual-machine:~/isu_work/exercise7/OSApiStudent/test$ ./TestLog
2021-11-24 13:48:15.215 DBG (TestLog.cpp:100 - main) Hello
```

2.2.6 - Resultater

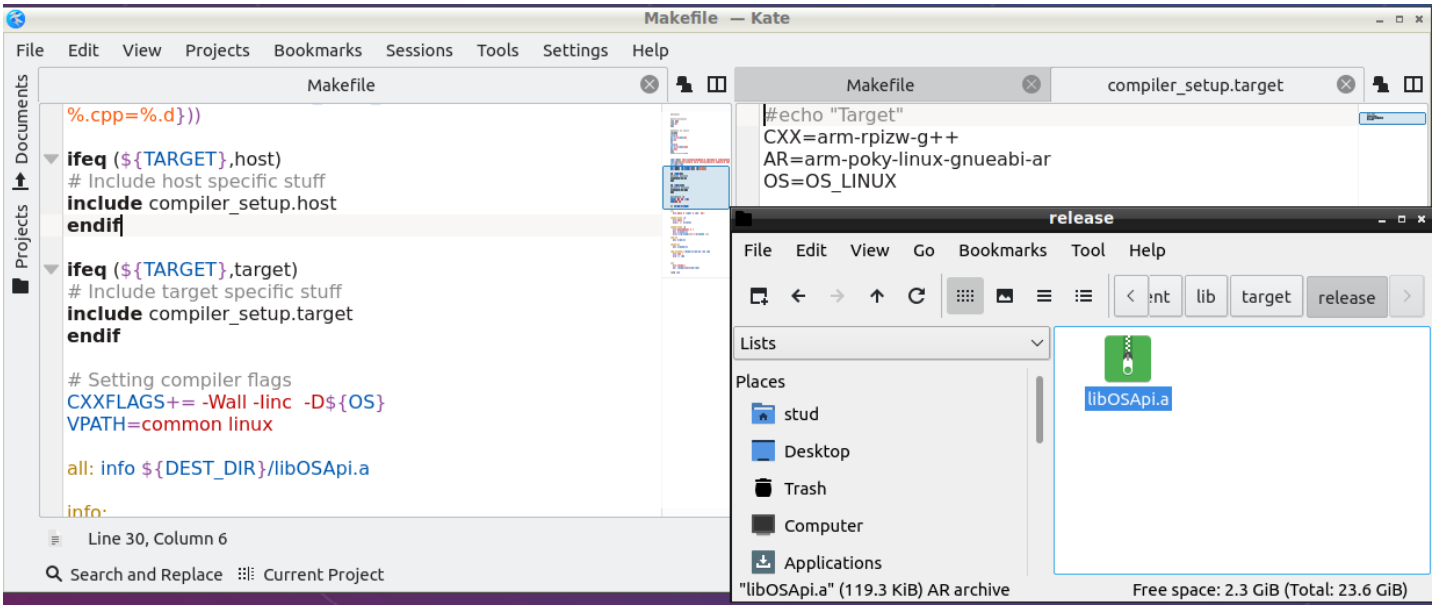
Resultaterne for test er tilfredstillende hvorfor implementeringen vurderes at være lykkedes.
Til testene er der benyttet udleverede filer under hhv. /examples og /tests - som er kompileret ved en mindre modificering af makefilen i den respektive mappe.

Exercise 3 - On target

3.1 - OSAPI compiled til target

Først verificeres at OSAPI libary, kan compile til target(rpi). Dette gøres ved brug af den udlevered makefile i /osApiStudent med kommandoen: make TARGET=target

```
stud@stud-virtual-machine:~/isu_work/exercise7/OSApiStudent$ make TARGET=target all
Generating dependency for linux/Conditional.cpp
Compiling for 'target' in 'release' mode...
Compiling linux/Conditional.cpp
Linking build/lib/target/release
```



3.2 - OSAPI examples compiled til target

Makefilen modificeres således at libpath sættes til lib/target/release og koden fra compiler_setup_target kopieres ind.

Der er taget udgangspunkt i materialet "Using_the_new_OSApi_in_a_Makefile.pdf"

```
# Quick and dirty (does not handle changes in h-file)
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/taget/debug
else
CXXFLAGS=-O2
LIBPATH=$(BASEPATH)/lib/target/release
endif
# Setup the CFLAGS to ensure that the relevant warnings,
includes and defines.
CXXFLAGS+=-Wall -D_REENTRANT -DOS_LINUX -I$(BASEPATH)/
inc

ifeq (${TARGET},target)
# Include target specific stuff
CXX=arm-rpizw-g++
AR=arm-poky-linux-gnueabi-ar
OS=OS_LINUX
endif
```

Og Kopieres der til og eksekveres på target:

```
root@raspberrypi0-wifi:~# ./example
Tester skrivning til log.txt
^C
root@raspberrypi0-wifi:~# cat log.txt
Tester
skrivning
til
log.txt
root@raspberrypi0-wifi:~#
```

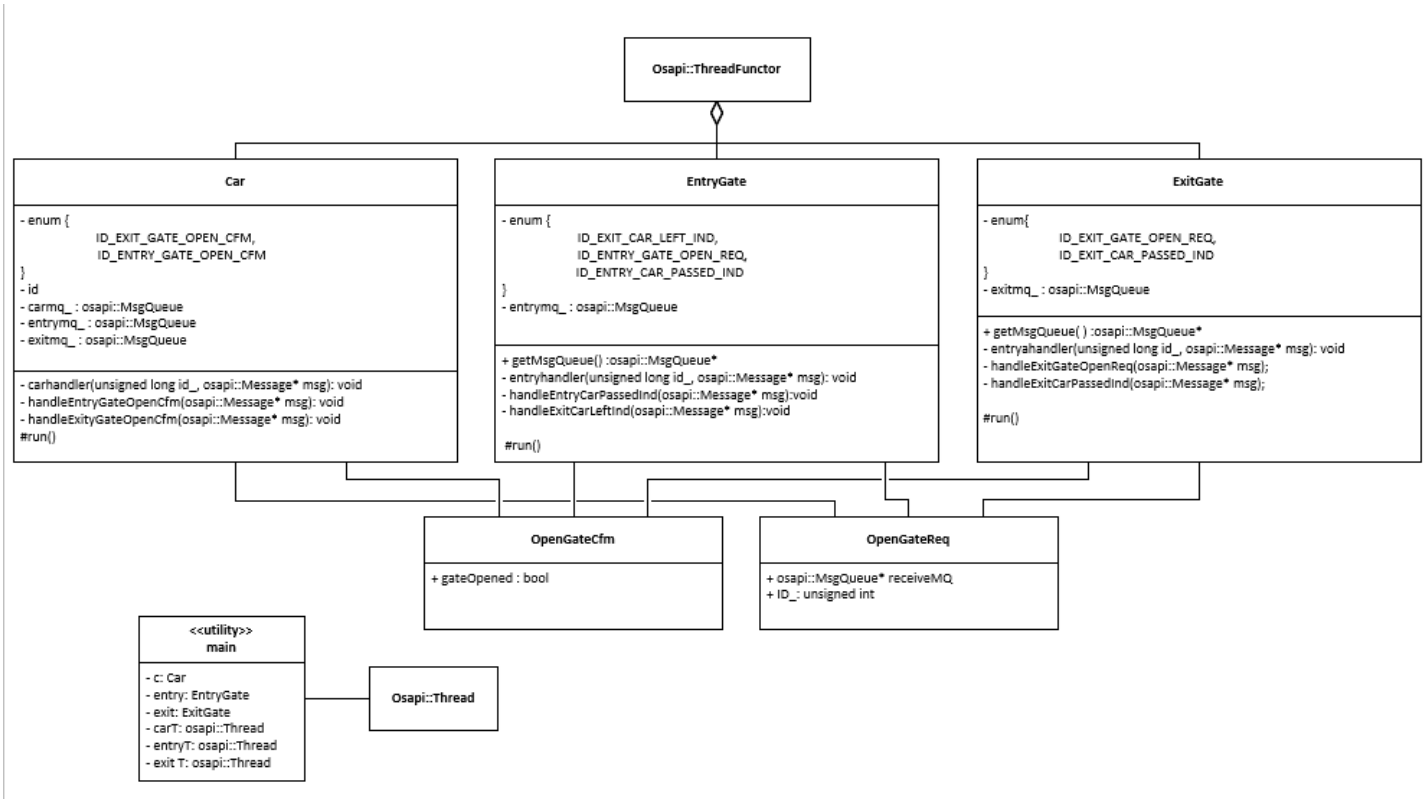
Exercise 4 - PLCS now the OS Api

4.1 - Design

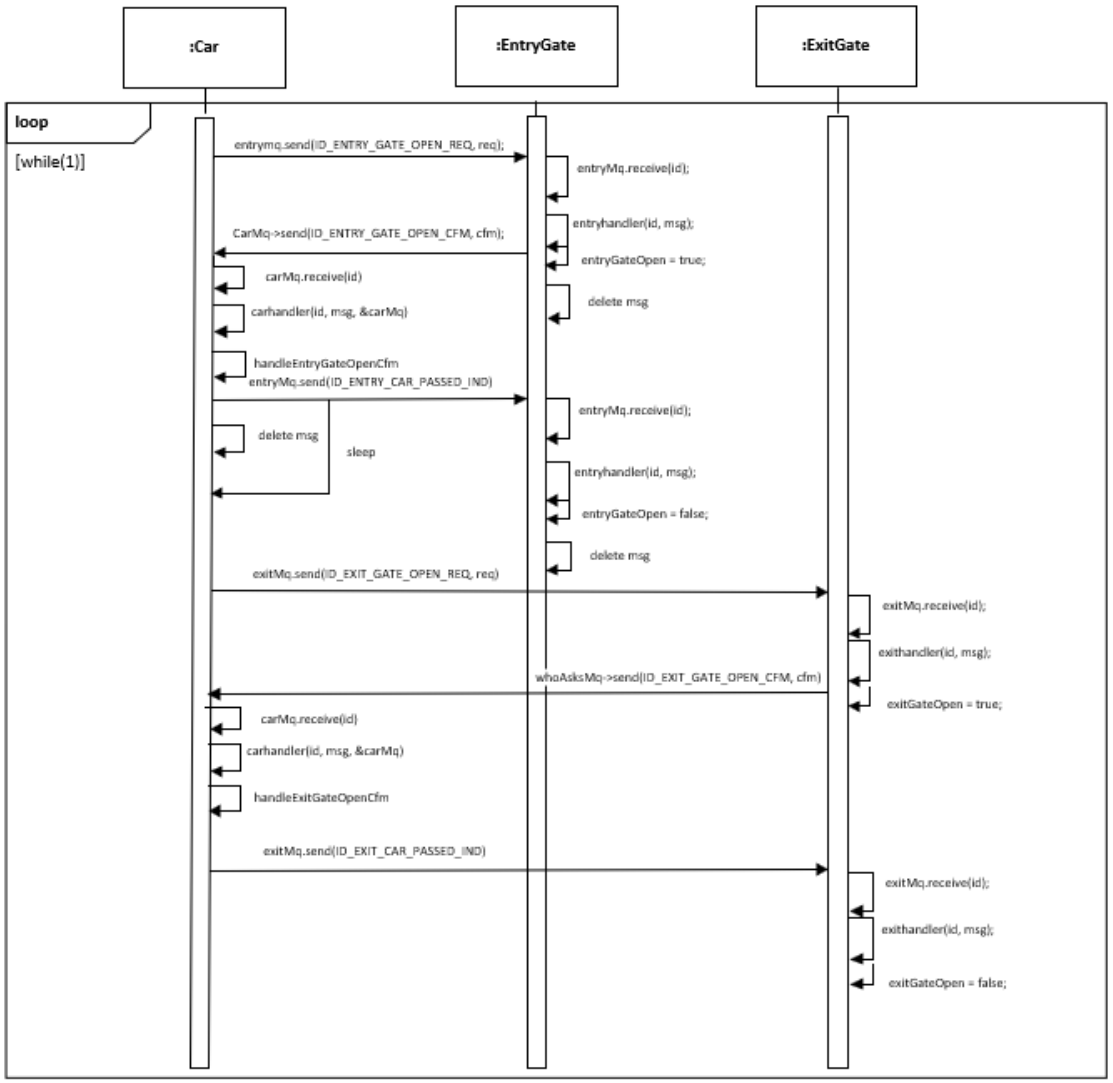
Den første tanke var at lave 3 klasser (Car, Entry og Exit), men 2 yderligere klassere (GateReq og GateCfm) blev sidenhed tilføjet.
De tre threads (Car, Entry, Exit) deles op i hver sin klasse og med hver sin MsgQueue. I hver klasses run(){..} palceres eventLoopet med recieve() og egen handler() som i sidste øvelse.

Nedenfor vises et Klassediagram og et Sekvensdiagram, som implementeringen vil laves ud fra.

4.1.1 - Klassediagram

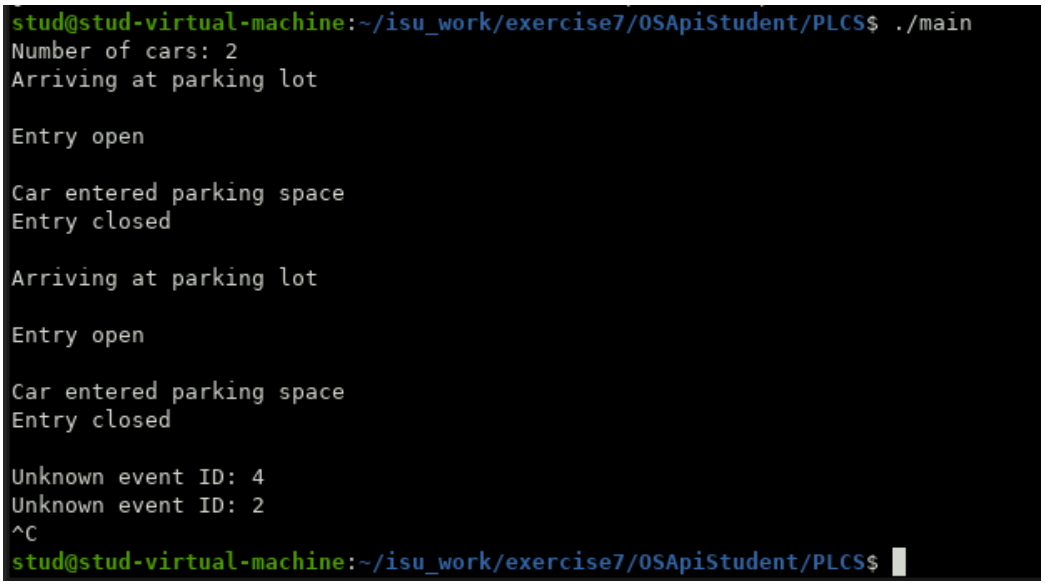


4.1.2 - Sekvensdiagram



4.3 - Restultat af Implementeringen

Implementeringen er delvist lykkedes, men der er problemer med exitGaten, på trods af næsten 1:1 forhold til entryGaten, som det ikke er lykkedes at løse.
Car og Entry threads fungere efter hensigten og resultat vises nedenfor:



4.3 - Questions to answer:

• Which changes did you make and why?

Den største forskel har været opdeling af threads i egne klasser istedet for at det hele samles, og en række gloable variabler udnyttes til at opnå den ønske funtionalitet.

Der vises her nogle eksempler fra Car klassen:

```
class Car : public osapi::ThreadFunctor

osapi::MsgQueue carMq_;
osapi::MsgQueue* entryMq_;
osapi::MsgQueue* exitMq_;

void Car::run()
{
std::cout << "Arriving at parking lot\n" << std::endl;
OpenGateReq* msg = new OpenGateReq(&carMq_, ID_ENTRY_GATE_OPEN_CFM);
entryMq_->send(EntryGate::ID_ENTRY_GATE_OPEN_REQ, msg);

for(;;)
{
     unsigned long id;
     osapi::Message* msg = carMq_.receive(id);
     carHandler(id, msg);
     delete msg;
}
```

PLCS er modificeres til at benytte OSAPI, dette inkludere Thread, ThreadFunc og run() samt Message og MsgQueue

• Does this modification whereby you utilize the OSApi library improve your program
or not. Substantiate you answer with reasoning.

> Det er nu endelig lykkedes at undgå globale variabler hvilket i sig selv mindsker risikoen for fejl og uhensigtsmæssig adfærd.
> Denne udvidelse med OS API, er naturligvis også en smartere måde såfremt der skal laves applikationer som skal benyttes at forskellige OS.
> Måden osapi::threads implementeres giver flere muligheder for ThreadFunc, Hvilket kan hjælpe til eventuelle udvidelse af applikationen.

Filer

sem.png	141 KB	24.11.2021	Rasmus Baunvig Aagaard
friend.png	124 KB	24.11.2021	Rasmus Baunvig Aagaard
mutexHpp.png	114 KB	24.11.2021	Rasmus Baunvig Aagaard
mutexCpp.png	90,6 KB	24.11.2021	Rasmus Baunvig Aagaard
util.png	108 KB	24.11.2021	Rasmus Baunvig Aagaard
tFunc.png	53,2 KB	24.11.2021	Rasmus Baunvig Aagaard
sekvensdiagram.png	17,9 KB	24.11.2021	Rasmus Baunvig Aagaard
logTest.png	65,8 KB	24.11.2021	Rasmus Baunvig Aagaard
timerTest.png	54,2 KB	24.11.2021	Rasmus Baunvig Aagaard
threadTest.png	12,9 KB	24.11.2021	Rasmus Baunvig Aagaard
timeTest.png	8,68 KB	24.11.2021	Rasmus Baunvig Aagaard
loggerTest.png	9,78 KB	24.11.2021	Rasmus Baunvig Aagaard
condCPp.png	170 KB	24.11.2021	Rasmus Baunvig Aagaard
apiTarget.png	110 KB	25.11.2021	Rasmus Baunvig Aagaard
makeExample.png	69,3 KB	25.11.2021	Rasmus Baunvig Aagaard
klasse.png	34,8 KB	25.11.2021	Rasmus Baunvig Aagaard
sekvens.png	35,3 KB	25.11.2021	Rasmus Baunvig Aagaard
testImpl.png	28,9 KB	25.11.2021	Rasmus Baunvig Aagaard