Lecture 6 - Exercises

Exercise 1 - Completing the Linux skeleton of the OO OS Api

In the first part of this exercise the missing parts of the given OS Api will be added. Since we only added a few things in each file, we won't elaborate on the specifics.

Implementation of Mutex.hpp

Everything was missing

Mutex.hppMutex.hpp

```
#ifndef OSAPI_LINUX_MUTEX_HPP
#define OSAPI_LINUX_MUTEX_HPP

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

namespace osapi
{
    class Conditional; //Make friend class

    class Mutex : private Notcopyable
    {
    public:
        Mutex();
        ~Mutex();
        void lock();
        void unlock();
    private:
        friend class Conditional;
        pthread_mutex_t mut_;
    };
}

#endif
```

Implementation of Mutex.cpp

Everything was missing

Mutex.cppMutex.cpp

```
#include "osapi/linux/Mutex.hpp" 

namespace osapi
{
        Mutex::Mutex()
        {
            mut_ = PTHREAD_MUTEX_INITIALIZER;
        }

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

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

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

Implementation of Conditional.cpp

Already started in the given material

Changes in Conditional.cppChanges in Conditional.cpp

```
#include <errno.h>
#include <osapi/ConditionalError.hpp>
#include "osapi/linux/Mutex.hpp"
#include <osapi/linux/Conditional.hpp>

namespace osapi
{
  Conditional::Conditional()
  {
    if(pthread_condattr_init(&condattr_) != 0) throw ConditionalError();
    if(pthread_condattr_setclock(&condattr_, CLOCK_MONOTONIC) != 0) throw ConditionalError();
    if(pthread_cond_init(&cond_, &condattr_) != 0) throw ConditionalError();
  }

    Conditional::~Conditional()
    {
        pthread_cond_destroy(&cond_);
        pthread_condattr_destroy(&condattr_);
//conditional attribute is used in the constructor, so we're gonna destroy it too
    }

    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_);
    }

  Conditional::Awoken Conditional::waitTimed(Mutex& mut, unsigned long timeout)
  {
    struct timespec ts;

    // Get monotonic clock - current time
    clock_gettime(CLOCK_MONOTONIC, &ts);

    // Calculate new absolute time by getting current monotonic time and offsetting it
    size_t secs = timeout/1000;
    size_t msecs = timeout - secs*1000;

    ts.tv_sec += secs;
    ts.tv_nsec += msecs*1000000;
    size_t overflow = ts.tv_nsec/1000000000;

    if(overflow)
    {
      ts.tv_sec += overflow;
      ts.tv_nsec -= overflow*1000000000;
    }

    int res = pthread_cond_timedwait(&cond_, &mut.mut_, &ts);

    switch(res)
    {
      case ETIMEDOUT:
        return TIMEDOUT;
        break;

      case 0:
        return SIGNALED;
        break;

      default:
        throw ConditionalError();
    }
  }
}
```

Implementation of Utility.cpp

Everything was missing

Utility.cppUtility.cpp

```
#ifndef OSAPI_LINUX_UTILITY_CPP
#define OSAPI_LINUX_UTILITY_CPP

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

namespace osapi
{
    void sleep(unsigned long msecs)
    {
        usleep(msecs*1000); //Convert from microsec to milisec
    }
}

#endif
```

Implementation of Thread.cpp

Already started in the given material

Changes in Thread.cppChanges in Thread.cpp

```
#include <unistd.h>
#include <errno.h>
#include <iostream>
#include <osapi/Thread.hpp>
```

```
namespace osapi
{
  Thread::Thread(ThreadFunctor* tf,
                 Thread::ThreadPriority priority,
                 const std::string& name,
                 bool autoStart)
    : tf_(tf), priority_(priority), name_(name), attached_(true)
  {
    if(autoStart)
      start();
  }

  Thread::~Thread()
  {
    detach();
  }

  void Thread::start()
  {

    if(getuid() == 0) // Check to see if we are root
    {
      /* Steps to go through */
      pthread_attr_t  attr;
      if(pthread_attr_init(&attr) != 0) throw ThreadError();
// Init attr
      sched_param sched_p;
      if(pthread_attr_setschedpolicy(&attr, SCHED_RR) != 0) throw ThreadError();
// Set RR scheduling (RT, timesliced)
      if(pthread_attr_setinheritsched(&attr, PTHREAD_EXPLICIT_SCHED) != 0) throw
 ThreadError();
// Create thread with explicit (non-inherited) scheduling - setting priority will not work otherwi
se!
      sched_p.sched_priority = static_cast<int
>(priority_);                                        // Set priority
      if(pthread_attr_setschedparam(&attr, &sched_p) != 0) throw
 ThreadError();                    // Use the priority

      /* Creation of thread */
      if(pthread_create(&threadId_, &attr, ThreadFunctor::threadMapper, tf_) != 0) throw
 ThreadError();

      pthread_attr_destroy(&attr);
    }
    else
    {
      /* Creation of thread if we are NOT root*/
      if(pthread_create(&threadId_, NULL, ThreadFunctor::threadMapper, tf_) != 0) throw
 ThreadError();
    }

    // If ok no exception was thrown thus we have a valid thread and its attached!
    attached_ = true;
  }

  void Thread::setPriority(Thread::ThreadPriority priority)
  {
    if(!attached_) throw ThreadError();
    if(getuid() == 0) // Check to see if we are root
    {
      if(pthread_setschedprio(threadId_, static_cast<int>(priority)) != 0) throw ThreadError();
      priority_ = priority;
    }
    else
    {
      // Do nothing
    }
  }

  Thread::ThreadPriority Thread::getPriority() const
  {
    return priority_;
  }

  //Now implementet

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

  void Thread::join()
  {
    if(!attached_) throw ThreadError();
    tf_->threadDone_.wait();
  }

  void Thread::detach()
  {
    if(!attached_) throw ThreadError();
    attached_ = false;
    tf_ = NULL;
    pthread_detach(threadId_);
  }

}
```

Implementation of ThreadFunctor.cpp

Already started in the given material

Changes in ThreadFunctor.cppChanges in ThreadFunctor.cpp

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

namespace osapi
{
  void* ThreadFunctor::threadMapper(void* thread)
  {
    /* Something is missing here - Determine what! */

    /* This is stolen from the slides from class (page 26) */
    ThreadFunctor* tf = static_cast<ThreadFunctor*>(thread);
    tf->run();

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

Testing and makefiles

To test the program, we have used the given source files found in the example and test directory. To be able to use these. however, changes in the Makefile were needed. The new finished Makefile kan be found in our repo.

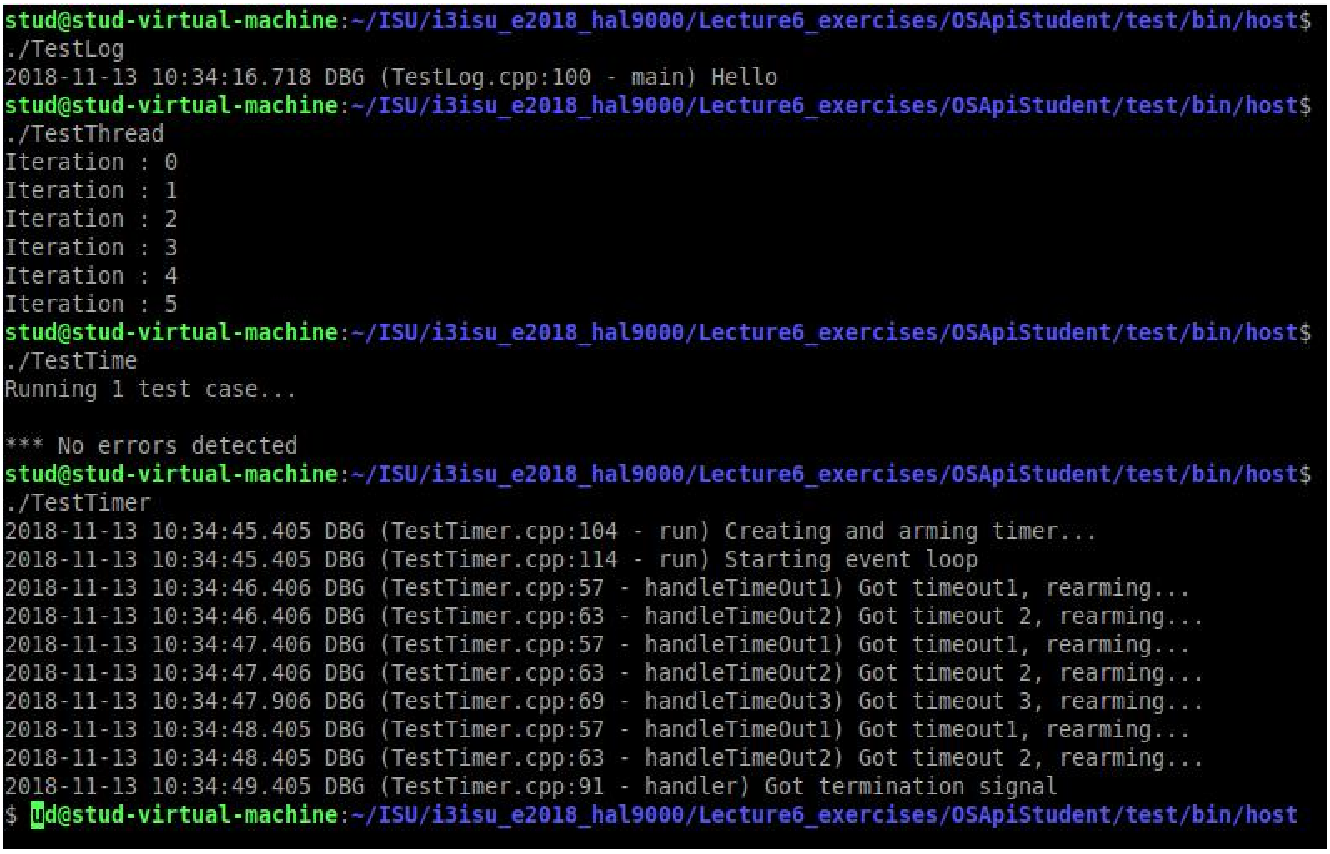
Firstly the makefile is made in the root directory, then:

Command	Function
make ARCH=THREAD TARGET=host	Building TestThread
make ARCH=LOG TARGET=host	Building TestLog
make ARCH=TIME TARGET=host	Building TestTime
make ARCH=TIMER TARGET=host	Building TestTimer

Making the test filesMaking the test files

makeTestFiles.png

Tests
After running the program in the Linux terminal, we were given the following outputs for all the tests:



```
stud@stud-virtual-machine:~/ISU/i3isu_e2018_hal9000/Lecture6_exercises/OSApiStudent/test/bin/host$
./TestLog
2018-11-13 10:34:16.718 DBG (TestLog.cpp:100 - main) Hello
stud@stud-virtual-machine:~/ISU/i3isu_e2018_hal9000/Lecture6_exercises/OSApiStudent/test/bin/host$
./TestThread
Iteration : 0
Iteration : 1
Iteration : 2
Iteration : 3
Iteration : 4
Iteration : 5
stud@stud-virtual-machine:~/ISU/i3isu_e2018_hal9000/Lecture6_exercises/OSApiStudent/test/bin/host$
./TestTime
Running 1 test case...

*** No errors detected
stud@stud-virtual-machine:~/ISU/i3isu_e2018_hal9000/Lecture6_exercises/OSApiStudent/test/bin/host$
./TestTimer
2018-11-13 10:34:45.405 DBG (TestTimer.cpp:104 - run) Creating and arming timer...
2018-11-13 10:34:45.405 DBG (TestTimer.cpp:114 - run) Starting event loop
2018-11-13 10:34:46.406 DBG (TestTimer.cpp:57 - handleTimeOut1) Got timeout1, rearming...
2018-11-13 10:34:46.406 DBG (TestTimer.cpp:63 - handleTimeOut2) Got timeout 2, rearming...
2018-11-13 10:34:47.406 DBG (TestTimer.cpp:57 - handleTimeOut1) Got timeout1, rearming...
2018-11-13 10:34:47.406 DBG (TestTimer.cpp:63 - handleTimeOut2) Got timeout 2, rearming...
2018-11-13 10:34:47.906 DBG (TestTimer.cpp:69 - handleTimeOut3) Got timeout 3, rearming...
2018-11-13 10:34:48.405 DBG (TestTimer.cpp:57 - handleTimeOut1) Got timeout1, rearming...
2018-11-13 10:34:48.405 DBG (TestTimer.cpp:63 - handleTimeOut2) Got timeout 2, rearming...
2018-11-13 10:34:49.405 DBG (TestTimer.cpp:91 - handler) Got termination signal
$ ud@stud-virtual-machine:~/ISU/i3isu_e2018_hal9000/Lecture6_exercises/OSApiStudent/test/bin/host
```

Terminal output after running the test programs

Questions

Descripe the chosen setup of the OSApi and how it works

The API consists of a root directory, OSApiStuden, with the following directories:
*__inc__ in which another folder is found; osapi. This folder contains header files. The header files in the osapi-folder's common, while the **linux** and **winx86** specific files are divided into to folders by that name.

- **common** - Holds the implementation files (.cpp) for the .hpp files ind the osapi-folder.
- **linux** - Holds the **linux** specific implementation files.
- **winx86** - Holds the **winx86** specific implementation files.

In addition to this, the root directory contains an example and a test folder for Lecture 6.

When using POSIX thread API it's important to provide a free C function as the thread function. Describe which classes are involved, their responsibilities and how they interact.

> The class Thread take reference to the class ThreadFunctor. In the ThreadFunctor the pure virtual function void run() is found, which means the class must be inherited and implemented before it can be used. This makes ThreadFunctor a base class, which kan be referenced to the class Thread.
> The class Thread is responsible of calling the pthread_create function, which refers to ThreadFunctor 's function threadMapper(). This gives pthread_create an extended acces to the class ThreadFunctor.

Explain how the pimpl/cheshire cat idiom is used for the windows implementation and what is achieved in this particular situation.

> In the OSApi winx86 folde there's a mutex and a folder details containing MutexDetails. The Mutex's implementations each contains a pointer to the details. This means you'll be able to change the implementation of the MutexDetils, and thereby Mutex, without having to recompile.

Why is the class Conditional a friend to class Mutex? What does it mean to be a friend?

> A friend class is able to access both private and protected members of the classes in which it is declared as a friend. This means the class Conditional is allowed to access Mutex' private members, which is necessary to use mut_

Exercise 2 - On target

To compile to the target instead of the host, all we needed to change were the target in the terminal:

Command	Function
make ARCH=THREAD TARGET=target	Building TestThread
make ARCH=LOG TARGET=taget	Building TestLog
make ARCH=TIME TARGET=target	Building TestTime
make ARCH=TIMER TARGET=target	Building TestTimer

Making to targetMaking to target

makeTarget.png

> Making to the target

The output on the RPi terminal was the same as in Linux.

Exercise 3 - PLCS now the OS Api

Before implementation the PLCS to use the APi, we made a class diagram. We're have used the same sequence diagram as we made for lecture 5.

Class Diagram - Parking LotClass Diagram - Parking Lot cd_parkinglot.png

Class Diagram for PLCS

Sequence Diagram - Parking lotSequence Diagram - Parking lot

sq_plcs.png

This is the same Sequence Diagram as in Lecture 5 Exercises

After implementating the PLCS to use the OSApi, the program still executed as before.

Terminal outputTerminal output

test_5_cars.png

Testing the PLCS with 5 cars}}

Questions

Which changes did you make and why?

Since we didn't manage to get exercise 5 to work properly, we didn't technically change anything, as much as we just made it all from scratch. kek.

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

Since we were able to implement the PLCS with the OSApi library, we agreeded in the group that this way was an improvement.

Filer

Testing_files.png	113 KB	2018-11-13	Mie Nielsen
makeTestFiles.png	105 KB	2018-11-13	Mie Nielsen
makeTarget.png	192 KB	2018-11-13	Mie Nielsen
cd_parkinglot.png	70,9 KB	2018-11-13	Mie Nielsen
sq_plcs.png	42,6 KB	2018-11-13	Mie Nielsen
test_5_cars.png	81,6 KB	2018-11-13	Mie Nielsen