Søren Hansen <shan@iha.dk>
V1.0

OS Api

Introduction

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 application

Prerequisites

You must have:

- Knowledge about mutex, conditional and semaphore implementations.

Goal

Upon successful completion of this exercise, you will:

- Have gained insight into how such an API can be constructed and compiled.
- Have acquired knowledge about how certain elements that are OS "near" are to be implemented.
- Have gained insight in how to use the abstract OS API to swap between target platforms.
- Have an understanding for the build and library inclusion mechanisms of a C++ program.
- Have been provided with an easy-to-use OS API for use in other courses.

In the exercise folder, where you found this document you will find the *OSApi library* in a zip file and the file `ThreadTest.cpp` that will aid you in your endeavor. Remember to refresh you knowledge of the OS Api in general. Read "Specification of an OS Api.pdf"

Exercise 1 Completing the Linux skeleton of the OO OS Api

In this exercise you will be tasked with completing the *OSApi* library. The following files are missing or have not been completed.

- `int/osapi/linux/Mutex.hpp` - *Missing*
- `linux/Mutex.cpp` - *Missing*
- `linux/Conditional.cpp` - *Already started*
- `int/osapi/linux/Semaphore.hpp` - *Missing*
- `linux/Semaphore.cpp` - *Missing*
- `linux/Utility.cpp` - *Missing*
- `linux/Thread.cpp` - *Already started*

Use the Linux documentation either via `man` pages, the net and remember to inspect the Win32-edition of the OS API. Also remember to read the "Specification of an OO OS API" (available on CampusNet) to write a Linux-edition of the OO OS API.

It is of course very important that the interface to the OS resources is exactly as specified in the *OS API specification.* Otherwise, it will be impossible to interchange the OS APIs as seen from the user when he/she tries to compile it for another platform.

Søren Hansen <shan@iha.dk>
V1.0

OS Api

To compile your revised version of the OSApi, use the following command:
`make DEBUG=1 TARGET=host all`

The `makefile` has multiple different combinatorial uses, which means that you should take the time to study it and determine what it can and what it can't. It is a ***simple*** implementation with some interesting approaches that you might benefit by, however "feel free to improve".

Having completed part or all of the *OSApi* library appropriate tests are needed. This means that you have to create simple tests that verify that the library does indeed work. *Be sure to test make individual tests before using the combined OSApi as a whole.* Otherwise it will be much more difficult to determine what the programming error actually is.

To help verify your program thread-wise use the source file `ThreadTest.cpp`.

Questions to answer:

- Why does the `ThreadTest.cpp` program behave differently depending on whether you are running it on a *single* processor or *multi* processor system and how is this experienced?
- The file `linux/Conditional.cpp`
  - In completing this file you will encounter the term CLOCK_MONOTONIC. What does this mean and how does it distinguish itself from the other *clock setup options* that are possible?
  - Why could this possibly be important? Hint: NTP and setting time.
- Inspecting the *OSApi* library, describe the chosen setup and how it works.
  Hint: Directory structure, define usage, handling platforms etc.

Exercise 2 On target

Verify that your OSApi library can compile for both Linux host and target. The same applies to your test applications.

Questions to answer:

- Did you do need to change code for this to work, if so what?

Exercise 3 PCLS now the OS Api

At this point we have created the PCLS system and it works great with Message and Message queues :-).

The next natural step is obviously to port your PCLS application such that it now is based on your newly created *OS Api.*

Remember that the loop in the thread function `run()` ***must*** look like this:

Listing 3.1: Loop in thread function

```
void MyThread::run()
{
  while(running_)
  {
    unsigned long id;
    osapi::Message* msg = mq_.receive(id);
    handleMsg(msg, id);
    delete msg;
  }
}
```

What is important here is that the `receive()` function must be in this loop and NOT anywhere else!

Questions to answer:

- Which changes did you make and why?
- Does this modification whereby you utilize the *OSApi* library improve your program or not. Substantiate you answer with reasoning.

Finally the complete solution for this exercise must be available for reading!