Teesside University
School of Computing
2011 - 2012

Final Year Project
BSc Computer Science

Social network library development

Thibaut Havel
L1247434

Supervisor: João F. Ferreira
Second reader: Erika Downs

Abstract

The main aim of this project is to develop a low-level library that is able to grab and store data from a social network. This library was designed for embedded devices and allows data to be stored and to be used to provide a simple service. To demonstrate the effectiveness of the final library, I created a demo service that interacts with a known social network (for example, Twitter).

Acknowledgements

I would like to thank my supervisor João Ferreira for his support all over the development of my project.

Contents

1 Introduction

Our society tends to use more and more social networks (for instance, Twitter or Facebook). At the same time, we are increasingly dependent on the use of embedded devices on a day-to-day basis (for instance, home automation). The goal of this project is to develop a software platform that allows an effective and efficient communication between embedded devices and social networks.

The main aim is to develop a low-level library that is able to grab and store data from a social network. This library was designed for embedded devices and allows data to be stored and to be used to provide a simple service. To demonstrate the effectiveness of the final library, I created a demo service that interacts with a known social network: Twitter.

One of my personal objectives was to improve my knowledge in low-level development and become familiar with the C language. Also I was looking forward to improving my software development skill while working with a very specific hardware.

Firstly, this report will present the way I started my initial research and how I designed the library according to my functional choices. Secondly, it will present how I've implemented these functionalities, and what I did to test my library. Finally, this report will end by an evaluation of the final product regarding the design choices, the way I've implemented them, and his reliability.

2 Methodology

2.1 The initial project scheduling

In my project specification, I set my schedule as following:

- January: Research (2 weeks), design (2 weeks).
- Febuary: Design (1 week), Implementation (3 weeks).
- March: Implementation (3 weeks), Testing (1 week).
- April: Evaluation and report compilation (2 weeks).

My supervisor João and I met every week or every two weeks to discuss about the planning and progresses of my project.

2.2 Research plan

My initial approach was to become familiar with the embedded device technology. I had to find an adapted, a small and a simple operating system to work with, thereby I chose **FreeRTOS**[1] supported from my supervisor.

My supervisor had already used this system and he had developed applications before. He provided me one of his own device running with FreeRTOS. So, thanks to João and his knowledge about this operating system, I had a platform in addition to a massive support from him and the online community to develop my library. As I didn't have any knowledge about FreeRTOS, I read several articles which deals with how it works, and how tasks are scheduled in an real-time way.

As a consequence of this chose and because one the goals of this project is to gain knowledge about low-level development, the library has been built using the C language.

Then, I defined every functionality of the final applications. Basically, the library's features are simple: it should allow a user to receive and send text from and to a social network. For instance

[1] FreeRTOS is a light-weight Real-Time Operating System.

a *message on the wall* in **Facebook** or a *tweet* in **Twitter**. Facebook and Twitter are both well known social networks and after some research about them, I choose to build my library suitable for Twitter because of the solid support for **OAuth** which is a secured protocol to access data. Once again, this choice was supported by João.

At this point, I had to defined how to receive and to send tweets, so I've started by looking for any existing solutions. In the next chapter, I will discuss why I chose to build my own library from scratch only using OAuth.

After this key decision, I've learned how OAuth works and how to register an application on Twitter it in order to access to the data.

As I was now aware about what the tools I will use and the way to use them, I designed the library according to the features I wanted to implement.

The next step of the development was to implement the abstract structure of the library. At this stage, I faced lot of issues concerning the use of OAuth, the C language and its requirements. During my previous years of studies I learnt the basics of this language but to build the library I read a lot of articles, books and tutorials along the implementation.

Once I finished the draft version of the library, I tested every functionality by receiving and sending tweets over my own account and I improved the reliability according to the results of these tests.

2.3 The support tools

To help myself into the research and the development of the library, I used some additional appropriate tools:

- A diary: to keep every relevant informations but also as a memory trail of the development chronology.
- Github[2]: to back up the source code and share my progress with my supervisor.
- FreeRTOS POSIX[3] simulator: to develop and to test my library without any embedded device.

[2]Github is an online project hosting.
[3]Standards to maintaining compatibility between UNIX operating systems.

3 Research

3.1 Operating System: FreeRTOS

3.1.1 Overview

FreeRTOS is a light-weight and real-time[1] operating system suitable for embedded devices. It is distributed under the GPL[2] even then with certain exceptions: a developer is required to maintain the kernel as open source, whereas he could remain closed source his applications.

Figure 3.1: FreeRTOS logo

The FreeRTOS source code is tiny and simple, thus it is really easy to port. It is mostly written in C and assembly. The kernel has been ported to around thirty micro-controllers.

3.1.2 Real-Time System

One of the main feature provided by FreeRTOS is the way the system performs threads. All tasks are scheduled depending on their priority and sorted according to a round-robin algorithm.

Every files which manage tasks should include the FreeRTOS header *task.h* in order to use the appropriate functions.

A task can be defined as follows and should be of endless loop style:

[1] A system that should fit to time constraints.
[2] General public licence is used for free and open source software.

```c
void vMyTask (void * pvParameters)
{
    for(;;)
    {
        /* Task code content */
    }
}
```

The FreeRTOS thread API[3] allows to manage tasks. For instance, to create or delete them:

```c
xTaskHandle xHandle;

xTaskCreate( vMyTask, "NAME", STACK_SIZE, &parameters, tskIDLE_PRIORITY, &xHandle );
vTaskDelete( xHandle );
```

The *xTaskHandle* element allows to keep a reference to the task that can be then used by other functions

Depending on its presence or its position in the queue, a task can take different states:

- Ready: the task is ready to run but it is not currently executing because another task is running. From this state, it might become *Suspended* or *Running*.
- Running: this task is currently executing. From this state, it might become *Blocked*, *Ready* or *Suspended*.
- Blocked: this task is not available for scheduling until a defined delay period. From this state, it might become *Suspended* or *Ready*.
- Suspend: this task is not available for scheduling without any timeout. From this state, it might become *Ready*.

These states can be changed using the appropriate functions from the API, for instance:

```c
/* To suspend a specific task */
vTaskSuspend( xHandle );

/* To resume a suspended task */
vTaskResume( xHandle );

/* To raise a task priority */
vTaskPrioritySet( xHandle, tskIDLE_PRIORITY + 1 );
```

[3]An Application Programming Interface is a set of routines or functions given by a library.

3.1.3 POSIX simulator

To develop and test demo tasks in order to get knowledge about FreeRTOS, I worked with a simulator. This is a ported version of the operating system that allows a embedded application to be simulated on a computer running another system.

As I'm a Linux user, so I chose the POSIX simulator. This simulator consist of a set of files representing the kernel and some demo applications which can all be compiled using GCC[4].

3.2 Twitter authentication: OAuth protocol

3.2.1 Overview

(Common authentication mechanism: token, secret key system, include graphic representations)

3.2.2 Existing library in C

(Downloaded and tested library: samples hard to understand, idea: create a simple-to-use library layer)

3.2.3 Register an application on Twitter

(Way and proprieties of the registered application)

3.2.4 Required libraries

(libcurl: overview and it's seem hard to adapt to FreeRTOS, idea: create a very simple HTTP request library)
(OpenSSL: overview and it's seem hard to adapt to FreeRTOS, idea)

[4]GNU Compiler Collection is a compiler system which can compile several programming languages including C and C++.

4 Design of the library

4.1 Functional design

4.1.1 Interactions with Twitter

Basically, the final library should allow a developer:

- To authenticate its application to Twitter.
- To send tweets to Twitter.
- To receive tweets from Twitter.

And these features have to work whatever the operating system and the hardware architecture.

I chose not implement any storing properties because it is more flexible to let the developer chose the way he wants to store tweets.

4.1.2 Easy layer to user

This library has to be a user-friendly layer. The developer does not have to know how OAuth works to build its own application to access to Twitter. He just have to give the basic informations about the registered application (the public and secret key provided by Twitter) and informations about its Twitter account (the login and password).

The first main functionality is the authentication process which gathers all OAuth operations and returns an authentication entity (for instance, typed as a C structure) which could be use by the developer in a further step to send and to receive tweets. This entity contains every required informations needed to allow OAuth to connect to Twitter.

The send functionality is one of the two behaviours which could use the authentication entity in order to send a tweet on a Twitter profile.

Finally, the receive functionality use the authentication entity as well in order to receive tweets from a Twitter account's **timeline**[1]. This functionality include parsing functions which allows to return a set of tweets entities.

Each functionality is represented by a single function. Nevertheless, the content of a functionality could be split into several operation each represented by another function.

4.2 Implementation design

4.2.1 Functions implementations

Every steps of the authentication process are gather into the main authentication function. Basically each use of the OAuth library for a specific stage of the synchronisation is surrounded by a set of operations, for this very reason each stage is defined into a distinctive function. As explained above in the chapter Research, to authenticate an application to Twitter and then be able to access to the timeline or to send a tweet is simple but it requires few steps:

- 1- Request token: it requests the first token to the Twitter service provider.
- 2- Direct token: it requests the token needed to obtain user authorization.
- 3- Verifier: it uses the direct token in order to request the PIN code (or verifier).
- 4- Access token: it uses the verifier to request the final token which will be use to send or receive tweets.

This main function gives to the user an authentication entity, that is the one he provides to the behaviour functions (send and receive). This entity is typed as a C structure.

As every behaviour functions, the send function need the access token to be able to send a text message over Twitter. The main function retrieves the needed informations from the authentication entity which are given as parameters in sub-functions[2]. Whatever the sub-function, no field of the authentication entity is directly used, only the main function holds this responsibility.

To get the tweets from the user's timeline, a request is firstly send to the Twitter service provider. The result is a XML content which is parsed by some sub-functions. These parsing functions determine how many tweets are there in the timeline, and for each of them a new structure is created. Thus, the main function gives to the developer a collection of tweets each represented by a C structure which contains the most significant informations about it (e.g. the date, the text content).

[1]The timeline is the part of a Twitter profile which contains all tweets sent by a user.
[2]A sub-function is used by the main function in a distributed way to perform the goal.

4.2.2 Functional diagram of the library

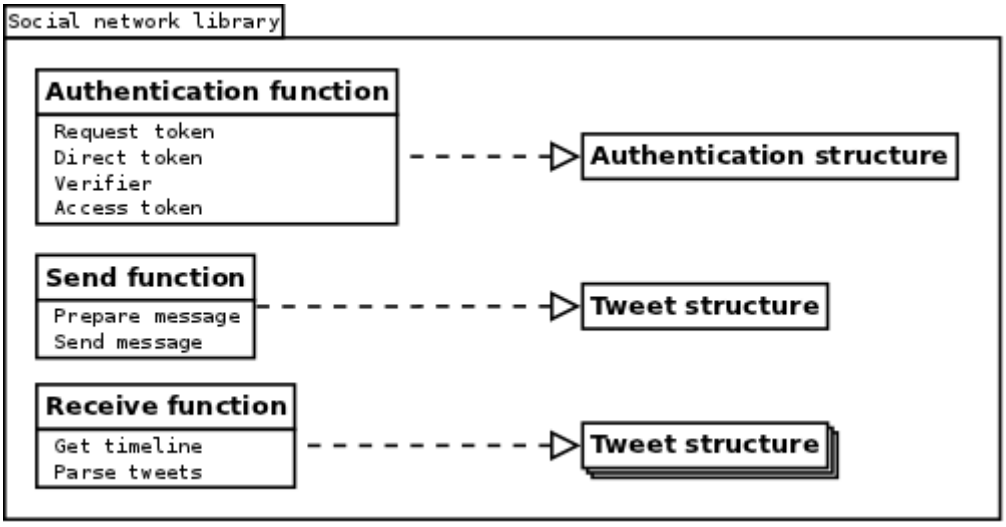

Figure 4.1: Functional diagram representing the implemented design

5 Implementation

5.1 The library

5.1.1 Required libraries

- OAuth which use OpenSSL and Libcurl are required. - OAuth and OpenSSL could be both included in the final library package. - The use of Libcurl could be replace by sockets.

5.1.2 Authentication process

- Request token - Direct token - Access token - Authentication structure returned

5.1.3 Send a tweet

- How use the authentication structure - Apply these informations to a behaviour

5.1.4 Receive a tweet

- Receiving process - XML parsing: get each tweet - Store temporary informations as a structure

5.2 A demo application

(Graphic representation of the use of my library layer)

6 Testing of the library

7 Evaluation of the ptoject

7.1 Goal

(Is my goal achieved, why/why not?)
(Is my work could be use by someone else, why/why not?)

7.2 Schedule

(Did I follow my schedule, why/why not?)

7.3 Improvements

(What is it possible to do to improve my library?)

8 Conclusion

Bibliography

[1] F. Surname, "Title," 2000.

[2] D. E. Knuth, *The TEXbook*. Addison-Wesley, 1990.