EEC 172 - Spark Core - Lyrics Search

1 INTRODUCTION

Using the Spark Core along with Nokia's 5110 display, we are able to create an embedded application that wirelessly searches the lyrics of a song from user inputs. We use AT&T's IR remote to transmit data in a series of binary signals at 36 KHz and is received by Vishay IR module. We use our own timer interrupt that double-samples incoming messages at 2350 Hz to decode which button was pressed. Texting through the remotes numeric pad allows users to enter an ARTIST name and SONG title onto the screen display. The Spark Core then pushes the information up to our proxy server, where it then gathers data from ChartLyrics's API. The proxy server we created from OpenShift then sends specific data down to the Spark Core, where it displays the lyrics onto the display.

User Directions:
1. Start Spark Core application
2. Enter ARTIST name, press Vol-up when done
3. Enter SONG title, press Vol-up when done
4. Lyrics are being pulled from the web
5. Lyrics are displayed onto the screen display
6. Press Ch-up to restart

2 CLIENT-SIDE (Spark Core)

The app installed onto the Spark Core is similar to what was done in Lab 3. The timer interrupt, button decoding and setup are the same. What is different here is that instead of sending text messages to an X-Bee through a serial terminal, we create a TCP client connection and use GET to access web data. There are several "states" that the Spark Core will be in, controlled by the Vol-up button. The first state accepts data for the ARTIST variable. The next state accepts data for SONG variable. We first use **client.connect()** to connect to the proxy server. Then with **client.print()** and **client.println()**, Spark Core sends out a GET request to the with the query strings of the artist name and song. After the Spark Core sends a GET request, it is put in a waiting state that checks for any data from the TCP-client. Once data arrives from the client, the lyrics are printed out onto the display. When all the data is done being sent, the client is closed with **client.stop()**. The application is reset with Ch-up button on the remote.

3 SERVER-END

3.1 Method

python framework: Flask
Requests in python: Apache2 Licensed HTTP library
HTTP GET request

Flask is the python framework used for this lab handle all the data received by our proxy server. The ***request*** object in Flask was used to retrieve the the parameters from a HTTP GET request. The format of the query string in bold for an HTTP GET request should be:

http://example.com/page***?parameter=value&also=another***

In our case ***keys*** are ***artists,*** and ***song.*** And the ***values*** would be the name of the artist and the song that you want the lyrics for. We imported an HTTP library called ***requests*** to send a GET request to the Chart Lyrics API and to retrieve lyrics in ".xml" format. Instead of parsing the .xml text format on the client end (Spark core), we decided to do it on the server end (proxy). We imported a library called ***BeautifulSoup*** and used beautifulsoup to extract only the Lyrics from the .xml text format.

3.2 Detailed code for Server

3.2.1 Importing modules:

```
from flask import Flask, request #
from bs4 import BeautifulSoup
import requests #
```

3.2.2 Grabbing parameters from GET request sent from client

```
#We use request to look up the key to obtain the string values from the GET request:

SONG = request.args.get('song')
ARTIST = request.args.get('artist')
```

3.2.3 Using imported HTTP library *requests*

```
param1 = 'artist='
ampersand = '&'
param2 = 'song='
r = requests.get(url+param1+ARTIST+ampersand+param2+SONG).content

#example format for 'url+param1+ARTIST+ampersand+param2+SONG' would be:
#http://api.chartlyrics.com/apiv1.asmx/SearchLyricDirect?artist=kanye&song=mama
```

4 PROBLEMS ENCOUNTERED

- Data loss on client-end (Spark core) due to delay and limited buffer size.

    - Fix: Create a proxy server that can handle larger http responses
- Importing modules in Flask so that the python environment on the proxy server could install and use the libraries.
    - Fix: Including correct library names in setup.py
- Sending the parsed .xml text from the proxy server to the Spark core.
    - Type-cast the text into a string structure using **str()**
- Beautiful Soup module did not include the xml parser (lxml)
    - Fix: Included lxml library, in addition to beautifulsoup4, in setup.py