UMass Lowell 16.480/552 Microprocessor Design II and Embedded Systems

Lab 4: Multithreaded Programming

Due Date: See the course schedule web page.

Revision: 3/27/2018

Objectives:

- Learn how to design multithreaded programs for embedded multicore systems
- Understand the principles of synchronization and mutual exclusion
- Learn POSIX thread libraries.
- Learn HTTP protocol and libcurl library
- Understand image processing concepts
- Implement image processing functions

Description:

You should by now have an embedded system with a variety of sensor devices such as light intensity, temperature, proximity/gesture and camera. In this lab, you are asked to send these data to a web server and keep save the captured image in your Galileo. You will need to design a multithreaded client-sensor software program that communicates with connected sensors and a web server. Your client-sensor application is a realtime multithreaded C application that handles user inputs, gathers sensors data, triggers a camera and sends data to a remote server in realtime.

Your program should meet the following requirements:

1. The client application starts all the required threads from the beginning.

2. Gathered data are sent to the server only when **temperature** sensor triggers the camera.

3. The camera is attached to servo-motor arm *(see "mounted_camera.pdf" for details)* and it is moving slowly back and forth (*scanning mode*). When the sensor reaches the predefined *(by the user)* threshold, the servo freezes and then the camera captures an image.

4. Use at least three threads to perform the following tasks respectively:

Thread #1: User Command Interface - (*User Menu & Results Display*)

A. This thread allows user to input commands such as RESET the PIC sensor, check PIC status (send PING command: if you don't get an ACK report it as "Error" status, if you get an ACK report it as "Online" status), GET the PIC ADC returned values and TURNxxx (TURN30, TURN90 and TURN120) the servo-motor arm 30, 90 and 120 degrees *(such as Lab2)*. There is no LDR threshold anymore in PIC code. You can also specify the LED operation as you will for Lab 4 (i.e: turn the LED on when the camera is in ***scanning mode -*** *see D below*).

B. It will also print/display the current **ambient temperature** value from TMP102 sensor

C. The user will be able to set an new **temperature** that triggers the camera trigger *(such as Lab3)*.

D. User will also be able to enable or disable ***Camera Scanning mode*** and change the camera scanning angle (***Scanning Option 90: 90 -> 0 -> 90*** to ***Scanning Option 180: 180 -> 0 ->180 degrees***). This is called "**Camera Scanning mode**" that consists a very slow angle change of the fixed usb camera on the servo *(see "mounted_camera.pdf")* motor from 0 degree position to 90 degrees and then from 90 degrees to 0 for ***Scanning Option 90*** (or for ***Scanning Option 180*** correspondingly).

Thread #2: ***Sensor Control Center***

This thread is responsible for sending control commands to the sensors (LDR, TMP102, and Camera) and obtaining sensors responses. This thread also needs to check the sensors status (i.e: new image was taken, ambient light intensity-LDR changed dramatically, temperature/gesture sensor passed the threshold, refreshing all sensors global variables, etc.) and report any error occurs (i.e: failed to read I2C bus or getting PIC response, etc.) by printing error messages. It will use OpenCV to capture an image and store it locally.

Thread #3: *Client-Server Communication*

This thread communicates with the provided remote web server using HTTP protocol and *libcurl* library in order to send all the gathered static and dynamic data to the server. The web based communication protocol is defined below.

PROTOCOL:

The client-sensor application (more specifically the communication thread) reports PIC sensor status and data using **HTTP POST method.** It will use the following URL to supply status and data:

```
http://server_hostname:portnumber/update?
id=var_xxxx&password=var_xxxx&name=var_xxxx&data=var_xxxx&status=var_xxxx&tim
estamp=var_xxxx&filename=var_xxxx
```

The client sensor application must send the following data as part of the URL request:

	Data to be sent to the remote server		
Static	**id**	int	A unique numerical identifier. This is your group number (example: if you are group 3 use “3” as ID #)
	password	string	A unique password to authenticate the sensor application (leave as default value “password”)
	name	string	Student name (example: “Tom”)
Dynamic	**status**	string	The current status of the PIC sensor (example: “Online”, “Error”)
	data	int	a 10bit integer value returned from PIC ADC (Built-in ADC of PIC16F18856)
	timestamp	string	Local time and date (example: ”2017-11-09_09:12:34”)
	filename	string	Name of the image file (example: “picture02.jpg”)

Notes:

- for **timestamp**, you need to create a function that acquires the local time and date and make a string such as the example above. You need to include time.h library.

- for **filename**, you need to create a function that counts the captured pictures and assembles the image filename such as the example above.

- for remote ***server hostname*** and **service port number** see lab4_instructions.txt

- for ***Camera Scanning mode***, perform corresponding changes to your PIC code in order to execute ***Scanning Option 90*** and ***Scanning Option 180*** as described above in *requirement 4 , Thread #1, D*.

Deliverables

A zipped file containing:

1. Schematic of the design (in png/jpg/pdf format)
2. Source code (for Galileo Board) written in C/C++
3. Lab Reports in PDF format (All the team members' Lab Reports)

Zip filename should be in the following format: "GroupXX_LAB4.zip"

(XX is the group number, for more details see at the Github posted Micro2_Lab_Introduction_Nov1st.pdf presentation and Piazza related announcements)

References

1. Posix Thread programming. Available at: https://computing.llnl.gov/tutorials/pthreads/

2. libcurl APIs. Available at: http://curl.haxx.se/libcurl/

3. Useful links:

Find gcc compilation flags for CURL lib here: https://ubuntuforums.org/showthread.php?t=1175115