UMass Lowell 16.480/552 Microprocessor Design II and Embedded Systems

Lab 4: Multithreaded Programming

Due Date: See the course schedule web page.

Revision: 11/16/2017

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 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 multithreaded C application that handles user inputs, gathers sensors data, triggers a camera and sends data to a remote server.

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** *(Undergrads ONLY)* or **gesture** *(Grads ONLY)* 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 including (1) RESET the PIC sensor; (2) check PIC status by sending PING command (if you don't get an ACK, report it as "Error" status; if you get an ACK, report it as "Online" status); (3) GET the PIC ADC returned values and (4) TURNxxx (TURN30, TURN90 and TURN120) the servo-motor arm 30, 90 and 120 degrees *(as in 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 *(for Undergrad groups)* or **color measurement/proximity/touch-less gesture** value from APDS-9960 (*for Grad groups)*

C. The user will be able to set an new **temperature** or **Proximity/Color/Gesture threshold.** Retain a print of the current sensor threshold close to the user menu**.**

D. User will also be able to enable *(start)* or disable *(stop)* ***Camera Scanning mode*** and change the camera scanning view range (***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** ***(speed close to 1 degree angle change per 500ms)*** 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 *(being in an infinite loop moving back and forth)* for ***Scanning Option 90*** (from 180 to 0 for ***Scanning Option 180*** correspondingly).

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

This thread is responsible for sending control commands to the sensors (LDR, TMP102, APDS-9960 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.

In addition, if the user have enabled Scanning mode via the menu option *(thread #1)* **AND** the predefined sensor threshold is exceeded, then the camera should capture an image and store it locally *(such as Lab3)*. Each captured image should retain an incremental filename: *image_name + image number + .jpg (for example: "**`image03.jpg`**").*

Bonus part: if the user defined sensor threshold is exceeded, the Camera Scanning mode starts. ONLY when a face is detected in the camera plane, camera should capture an image and store it locally *(for implementation details see **Bonus** section below)*.

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. The client thread should send data to the server every 2 seconds. If there is no image captured *(camera is not triggered yet)* , client application should send an appropriate message. All messages and data are specified in the Protocol section 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&timestamp=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 such as "**342**")
	timestamp	string	Local time and date (example: "**2017-11-09_09:12:34**")
	filename	string	Name of the image file (example: "**picture02.jpg**"). If there is no image captured yet should send "**No face detected**"

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***, make all the appropriate changes to your PIC code in order to accommodate the execution of the novel ***Scanning Option 90*** and ***Scanning Option 180*** as described above in *requirement 4 , Thread #1, D. (for example: add three more new messages to your PIC parallel communication protocol such as `MSG_SCAN_90=0x19,MSG_SCAN_180=0x18 ,MSG_STOP_SCAN=0x20)`* You are free to decide about it.

Bonus (3 points) : Face detection

Implement a function that gets as an argument an image and detects if a face is present in the image or not returning an integer for each case. Follow the implementation steps which are provided in the prototype function in "`face_detection`" directory. You will need to use the front face cascade .xml file along with your c/c++ code.

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++ and organized in subfolders
3. Lab Reports in PDF format (All the team members' Lab Reports)
4. Include Readme.txt file with compilation instructions in case you created more than one c/c++ files , include **makefiles** if it is required for compilation

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