Introduction:

Arduino is very useful micro-controller, which has many usage in our life. Sometimes we need to monitor the CPU info of RAM, processor, temperature, GPU clock speed for doing big projects like Video Rendering, Android apps debugging. At that time of doing those heavy tasks, we need to check the info of CPU frequently. In that case, here is a simple idea could be arrived. With the help of Arduino we can set an external display which will show the info every seconds, and we can monitor those data so easily to do those heavy tasks.

About this project:

Character LCDs are one of the most common things one gets in an Arduino kit. They are very cheap and fun to work with. They are categorized in two categories: one that is directly hooked up to the Arduino board, and another that uses an I2C module between LCD and the main board. In this project, it can be found also one with pre-soldered I2C module to minimize the number of wires.

Objectives:

- To show CPU info on external display
- To Monitor info for big project concurrency, parallelism
- To check any processor or threads' clock speed usage
- To check disk usage
- To maintain external fan support
- To check RAM usage
- To check System Temperature
- To monitor usage in actual numbers (not in percentage)

Instructions to run the project (Methodology):

- Wire up the components using the schematic below.
- Upload the code mentioned below to your Arduino Nano/Uno.
- Make sure your Arduino is connected to a Windows Computer.
- Use the download button below to download the program.
- Run the program and enter the Serial Port number (ex., if it's COM4, type 4).

Equipment List:

- Arduino Nano R3
- 16x2 White on Blue Character LCD
- I2C/IIC module
- Jumper wires
- Breadboard
- LEDs

Built-in circuit picture:

This project is only support on displays 16x2 Character LCD.

A pre-built software using python language is used for the project to run and send info to the Arduino.

This command line software 1st selects a USB port for sending data. Then, after initialing, it transfers the info to the Arduino. The Arduino will convert the code to electric signal and sent it to the display.

The codes of Arduino uses to run the project:


```
#include <Wire.h>                //This library allows you to communicate with I2C / TWI devices.
#include <LiquidCrystal_I2C.h>   //include header file for the display to work


LiquidCrystal_I2C lcd(0x27,16,2); //Change address if this is not applicable.
                                  //Here we used 0x27 address for our display.
                                  //Because our display supports it. To check the display address,
                                  //arduino library code can be do that.


void setup(){
  lcd.init();                    // initialize the lcd
  lcd.backlight();               // to turn on the display backlit
  Serial.begin(9600);            // opens serial port, sets data rate to 9600 bps
}

void loop(){                     //run loop

  if (Serial.available()) {      //check if the Seral port is available
    delay(100);                  // set delay 100 millisecond
    lcd.clear();                 //Clears the LCD screen and positions the cursor in the upper-left corner

    while (Serial.available() > 0) { // read all the available characters

      lcd.write(Serial.read());     //To Display Message On LCD
    }
  }
}
```

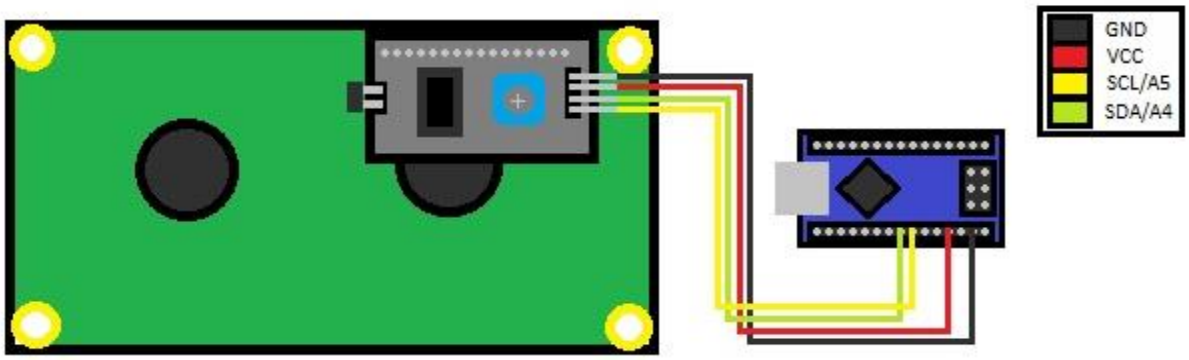
Arduino cannot send the data directly into the display. The display needs a driver. Display I2C converter is used for this purpose. It coverts data bit and send it to display. The display version is 0x27. The Liquid Crystal Header library code is given below (*This Library Code is taken from arduino.org*):

```
1   //LiquidCrystal_I2C Library for the 16x2 disply
2   //LiquidCrystal_I2C.h
3
4   #ifndef FDB_LIQUID_CRYSTAL_I2C_H
5   #define FDB_LIQUID_CRYSTAL_I2C_H
6
7   #include <inttypes.h>
8   #include <Print.h>
9
10  // commands
11  #define LCD_CLEARDISPLAY 0x01
12  #define LCD_RETURNHOME 0x02
13  #define LCD_ENTRYMODESET 0x04
14  #define LCD_DISPLAYCONTROL 0x08
15  #define LCD_CURSORSHIFT 0x10
16  #define LCD_FUNCTIONSET 0x20
17  #define LCD_SETCGRAMADDR 0x40
18  #define LCD_SETDDRAMADDR 0x80
19
20  // flags for display entry mode
21  #define LCD_ENTRYRIGHT 0x00
22  #define LCD_ENTRYLEFT 0x02
23  #define LCD_ENTRYSHIFTINCREMENT 0x01
24  #define LCD_ENTRYSHIFTDECREMENT 0x00
25
26  // flags for display on/off control
27  #define LCD_DISPLAYON 0x04
28  #define LCD_DISPLAYOFF 0x00
29  #define LCD_CURSORON 0x02
30  #define LCD_CURSOROFF 0x00
31  #define LCD_BLINKON 0x01
32  #define LCD_BLINKOFF 0x00
33
34  // flags for display/cursor shift
35  #define LCD_DISPLAYMOVE 0x08
36  #define LCD_CURSORMOVE 0x00
37  #define LCD_MOVERIGHT 0x04
38  #define LCD_MOVELEFT 0x00
39
40  // flags for function set
41  #define LCD_8BITMODE 0x10
42  #define LCD_4BITMODE 0x00
43  #define LCD_2LINE 0x08
44  #define LCD_1LINE 0x00
45  #define LCD_5x10DOTS 0x04
46  #define LCD_5x8DOTS 0x00
47
48  // flags for backlight control
49  #define LCD_BACKLIGHT 0x08
50  #define LCD_NOBACKLIGHT 0x00
51
52  #define En B00000100  // Enable bit
53  #define Rw B00000010  // Read/Write bit
54  #define Rs B00000001  // Register select bit
55
56  /**
57   * This is the driver for the Liquid Crystal LCD displays that use the I2C bus.
58   *
59   * After creating an instance of this class, first call begin() before anything else.
60   * The backlight is on by default, since that is the most likely operating mode in
61   * most cases.
62   */
63  class LiquidCrystal_I2C : public Print {
64  public:
65    /**
66     * Constructor
67     *
68     * @param lcd_addr  I2C slave address of the LCD display. Most likely printed on the
69     *          LCD circuit board, or look in the supplied LCD documentation.
```

```
70      * @param lcd_cols  Number of columns your LCD display has.
71      * @param lcd_rows  Number of rows your LCD display has.
72      * @param charsize  The size in dots that the display has, use LCD_5x10DOTS or
        LCD_5x8DOTS.
73      */
74     LiquidCrystal_I2C(uint8_t lcd_addr, uint8_t lcd_cols, uint8_t lcd_rows, uint8_t
        charsize = LCD_5x8DOTS);
75
76     /**
77      * Set the LCD display in the correct begin state, must be called before anything
        else is done.
78      */
79     void begin();
80
81      /**
82       * Remove all the characters currently shown. Next print/write operation will start
83       * from the first position on LCD display.
84       */
85     void clear();
86
87     /**
88      * Next print/write operation will will start from the first position on the LCD
        display.
89      */
90     void home();
91
92      /**
93       * Do not show any characters on the LCD display. Backlight state will remain
          unchanged.
94       * Also all characters written on the display will return, when the display in
          enabled again.
95       */
96     void noDisplay();
97
98     /**
99      * Show the characters on the LCD display, this is the normal behaviour. This method
        should
100     * only be used after noDisplay() has been used.
101     */
102    void display();
103
104    /**
105     * Do not blink the cursor indicator.
106     */
107    void noBlink();
108
109    /**
110     * Start blinking the cursor indicator.
111     */
112    void blink();
113
114    /**
115     * Do not show a cursor indicator.
116     */
117    void noCursor();
118
119    /**
120     * Show a cursor indicator, cursor can blink on not blink. Use the
121     * methods blink() and noBlink() for changing cursor blink.
122     */
123    void cursor();
124
125    void scrollDisplayLeft();
126    void scrollDisplayRight();
127    void printLeft();
128    void printRight();
129    void leftToRight();
130    void rightToLeft();
131    void shiftIncrement();
```

```
132     void shiftDecrement();
133     void noBacklight();
134     void backlight();
135     bool getBacklight();
136     void autoscroll();
137     void noAutoscroll();
138     void createChar(uint8_t, uint8_t[]);
139     void setCursor(uint8_t, uint8_t);
140     virtual size_t write(uint8_t);
141     void command(uint8_t);
142
143     inline void blink_on() { blink(); }
144     inline void blink_off() { noBlink(); }
145     inline void cursor_on() { cursor(); }
146     inline void cursor_off() { noCursor(); }
147
148   // Compatibility API function aliases
149     void setBacklight(uint8_t new_val);       // alias for backlight() and nobacklight()
150     void load_custom_character(uint8_t char_num, uint8_t *rows);  // alias for createChar()
151     void printstr(const char[]);
152
153   private:
154     void send(uint8_t, uint8_t);
155     void write4bits(uint8_t);
156     void expanderWrite(uint8_t);
157     void pulseEnable(uint8_t);
158     uint8_t _addr;
159     uint8_t _displayfunction;
160     uint8_t _displaycontrol;
161     uint8_t _displaymode;
162     uint8_t _cols;
163     uint8_t _rows;
164     uint8_t _charsize;
165     uint8_t _backlightval;
166   };
167
168   #endif // FDB_LIQUID_CRYSTAL_I2C_H
169
```

The Circuit Diagram made with software:



This circuit diagram is made with Adobe Illustrator.

The Python codes used for retrieving data from OS:

This code is a built-in code that used everywhere while sending CPU info into Arduino through USE port.

```
#!/usr/bin/env python
# -- coding: utf-8 --

import json
import os
import time
from urllib.error import URLError, HTTPError
from urllib.request import Request, urlopen

import serial
import serial.tools.list_ports

show_gpu_mem = None

def space_pad(number, length):
"""
Return a number as a string, padded with spaces to make it the given length

:param number: the number to pad with spaces
:param length: the specified length
:returns: the number padded with spaces as a string
"""

number_length = len(str(number))
spaces_to_add = length - number_length
return (' ' * spaces_to_add) + str(number)

def get_local_json_contents(json_filename):
"""
Returns the contents of a (local) JSON file

:param json_filename: the filename (as a string) of the local JSON file
:returns: the data of the JSON file
"""

try:
    with open(json_filename) as json_file:
        try:
            data = json.load(json_file)
        except ValueError:
            print('Contents of "' + json_filename + '" are not valid JSON')
            raise
except IOError:
    print('An error occurred while reading "' + json_filename + '"')
```

```
    raise

return data
```

def get_json_contents(json_url):
"""
Return the contents of a (remote) JSON file

```
:param json_url: the url (as a string) of the remote JSON file
:returns: the data of the JSON file
"""

data = None

req = Request(json_url)
try:
    response = urlopen(req).read()
except HTTPError as e:
    print('HTTPError ' + str(e.code))
except URLError as e:
    print('URLError ' + str(e.reason))
else:
    try:
        data = json.loads(response.decode('utf-8'))
    except ValueError:
        print('Invalid JSON contents')

return data
```

def find_in_data(ohw_data, name):
"""
Search in the OpenHardwareMonitor data for a specific node, recursively

```
:param ohw_data:    OpenHardwareMonitor data object
:param name:        Name of node to search for
:returns:           The found node, or -1 if no node was found
"""
if ohw_data['Text'] == name:
    # The node we are looking for is this one
    return ohw_data
elif len(ohw_data['Children']) > 0:
    # Look at the node's children
    for child in ohw_data['Children']:
        if child['Text'] == name:
            # This child is the one we're looking for
            return child
        else:
            # Look at this children's children
            result = find_in_data(child, name)
            if result != -1:
                # Node with specified name was found
                return result
# When this point is reached, nothing was found in any children
return -1
```

def get_hardware_info(ohw_ip, ohw_port, cpu_name, gpu_name, gpu_mem_size):
"""
Get hardware info from OpenHardwareMonitor's web server and format it
"""
global show_gpu_mem

```
# Init arrays
my_info = {}
gpu_info = {}
cpu_core_temps = []

ohw_json_url = 'http://' + ohw_ip + ':' + ohw_port + '/data.json'

# Get data from OHW's data json file
data_json = get_json_contents(ohw_json_url)

# Get info for CPU
cpu_data = find_in_data(data_json, cpu_name)

cpu_temps = find_in_data(cpu_data, 'Temperatures')
cpu_load = find_in_data(cpu_data, 'CPU Total')

# Look for CPU temperatures. For all children of the CPU temp. section...
for core_temp in cpu_temps['Children']:
    # Check that "Core" is in the name, to prevent using Intel's
    # "CPU Package" temperature, and should work with AMD too.
    if 'Core' in core_temp['Text']:
        # Remove '.0 °C' from end of value
        temp_value = core_temp['Value'][:-5]

        cpu_core_temps.append(temp_value)

my_info['cpu_temps'] = cpu_core_temps

# Get CPU total load, and remove ".0 %" from the end
cpu_load_value = cpu_load['Value'][:-4]

my_info['cpu_load'] = cpu_load_value

# Get info for GPU
gpu_data = find_in_data(data_json, gpu_name)

gpu_clocks = find_in_data(gpu_data, 'Clocks')
gpu_load = find_in_data(gpu_data, 'Load')

gpu_core_clock = find_in_data(gpu_clocks, 'GPU Core')
gpu_mem_clock = find_in_data(gpu_clocks, 'GPU Memory')
gpu_temp = find_in_data(find_in_data(gpu_data, 'Temperatures'), 'GPU Core')
gpu_core_load = find_in_data(gpu_load, 'GPU Core')
fan_percent = find_in_data(find_in_data(gpu_data, 'Controls'), 'GPU Fan')

# Get GPU Fan RPM info (check both Fans > GPU and Fans > GPU Fan)
fan_rpm = find_in_data(find_in_data(gpu_data, 'Fans'), 'GPU')
if fan_rpm == -1:
    fan_rpm = find_in_data(find_in_data(gpu_data, 'Fans'), 'GPU Fan')
```

```
# Check if the GPU has used memory information, and remember it
if show_gpu_mem is None:
    gpu_mem_percent = find_in_data(gpu_load, 'GPU Memory')
    show_gpu_mem = (gpu_mem_percent != -1)
    # show_gpu_mem = False

# Get GPU Memory percentage if it exists, otherwise GPU voltage
if show_gpu_mem:
    # Get GPU memory percentage
    gpu_mem_percent = find_in_data(gpu_load, 'GPU Memory')

    # Calculate used MBs of GPU memory based on the percentage
    used_percentage = float(gpu_mem_percent['Value'][:-2])
    used_mb = int((gpu_mem_size * used_percentage) / 100)

    # Add to GPU info object
    gpu_info['used_mem'] = used_mb
else:
    # Get GPU voltage
    voltages = find_in_data(gpu_data, 'Voltages')
    core_voltage = find_in_data(voltages, 'GPU Core')
    gpu_info['voltage'] = core_voltage['Value'][:-2]

# Add rest of GPU info to GPU object
gpu_info['temp'] = gpu_temp['Value'][:-5]
gpu_info['load'] = gpu_core_load['Value'][:-4]
gpu_info['core_clock'] = gpu_core_clock['Value'][:-4]
# Memory clock divided by 2 so it is the same as GPU-Z reports
gpu_info['mem_clock'] = int(int(gpu_mem_clock['Value'][:-4]) / 2)
gpu_info['fan_percent'] = fan_percent['Value'][:-4]
gpu_info['fan_rpm'] = fan_rpm['Value'][:-4]

# Add GPU info to my_info
my_info['gpu'] = gpu_info

return my_info
```

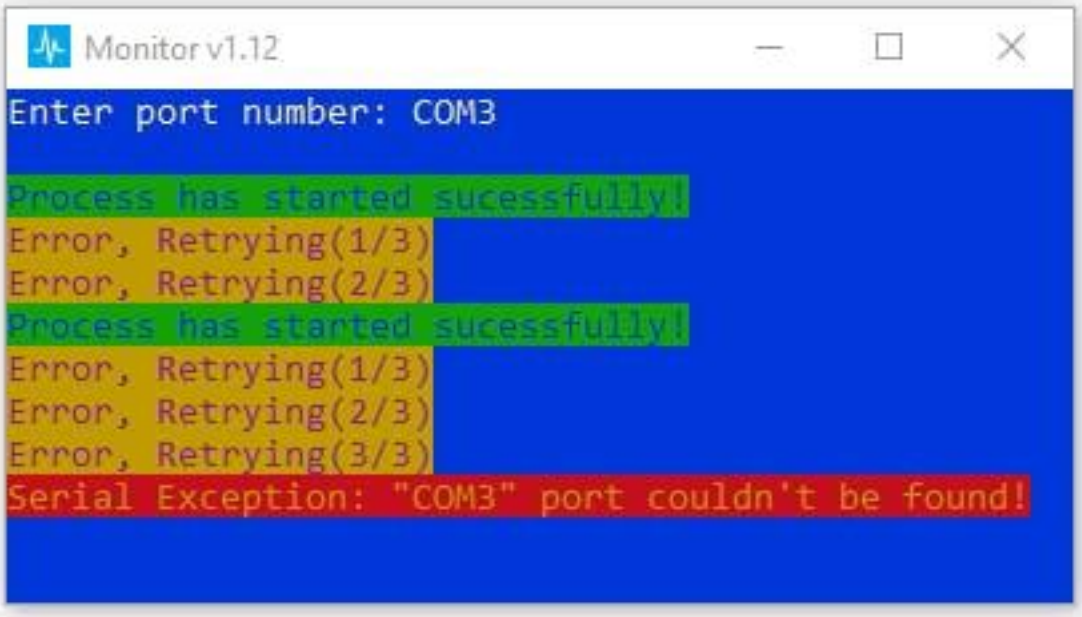
def main():
# Get serial ports
ports = list(serial.tools.list_ports.comports())

```
# Load config JSON
cd = os.path.join(os.getcwd(), os.path.dirname(__file__))
__location__ = os.path.realpath(cd)
config = get_local_json_contents(os.path.join(__location__, 'config.json'))

# If there is only 1 serial port (so it is the Arduino) connect to that one
if len(ports) == 1:
    # Connect to the port
    port = ports[0][0]
    print('Only 1 port found: ' + port + '. Connecting to it...')
    ser = serial.Serial(port)

    while True:
```

```
        # Get current info
        my_info = get_hardware_info(
            config['ohw_ip'],
            config['ohw_port'],
            config['cpu_name'],
            config['gpu_name'],
            config['gpu_mem_size']
        )

        # Prepare CPU string
        cpu_temps = my_info['cpu_temps']  # [:1]
        cpu = space_pad(int(my_info['cpu_load']), 3) + '% '
        for index, temp in enumerate(cpu_temps):
            if index >= 4:
                # Can't fit more than 4 temperatures in Arduino screen
                break
            cpu += space_pad(int(temp), 2) + 'C '

        # Prepare GPU strings
        gpu_info = my_info['gpu']
        gpu1 = \
            space_pad(int(gpu_info['load']), 3) + '% ' + \
            space_pad(int(gpu_info['temp']), 2) + 'C '
        if 'used_mem' in gpu_info:
            gpu1 += space_pad(int(gpu_info['used_mem']), 4) + 'MB'
        else:
            gpu1 += str(gpu_info['voltage']) + 'V'

        gpu2 = \
            space_pad(int(gpu_info['fan_percent']), 3) + '% F ' + \
            space_pad(int(gpu_info['fan_rpm']), 4) + ' RPM'

        gpu3 = \
            space_pad(int(gpu_info['core_clock']), 4) + '/' + \
            space_pad(int(gpu_info['mem_clock']), 4)

        # Send the strings via serial to the Arduino
        arduino_str = \
            'C' + cpu + '|G' + gpu1 + '|F' + gpu2 + '|g' + gpu3 + '|'
        # print(arduino_str)
        ser.write(arduino_str.encode())

        # Wait until refreshing Arduino again
        time.sleep(2.5)

    ser.close()
else:
    print('Number of ports is not 1, can\'t connect!')
```

if **name** == '**main**':
main()

The Picture of that command line tool:



Things that could possibly be causing trouble:

- You have not changed the address of the LCD from the above code.
- Dimensions of your LCD is different from what is required (16x2).
- You have entered wrong serial address.
- You are using LCD_I2C library on your non I2C display or vice versa.
- LCD isn't properly connected.
- Arduino isn't properly connected.
- Your system doesn't have driver installed (CH340 [For Chinese Arduino]).

Problem Faced while building this Project:

- We faced the 1st problem with the display address value. The address 0x3F was selected by default. We have changed the value with 0x27.
- We have to adjust display brightness with the star pin with a star pinned Screw-Driver. It was set low by default.
- We have to select Arduino new Library.

Project Cost:

- Arduino Nano R3 – 350 Taka
- 16x2 White on Blue Character LCD – 174 Taka
- I2C/IIC module – 99 Taka
- Jumper wires – 50 Taka
- Breadboard – 100 Taka
- LEDs – 20 Taka

Conclusion:

In a word, it can be a very useful project, made to monitor system resources without opening Task Manager. We can used it to build a high tech PC with proper monitoring system. In a data center, it can be used to monitor the whole system without look into the task manager. Those usages make this little project unique.

References:

- arduino.org
- github.com
- YouTube video tutorials
- Google images
- https://pythonhosted.org/pyserial/shortintro.html
- https://create.arduino.cc/projecthub/code_files/191075/download
- create.arduino.cc

Thank you