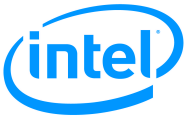
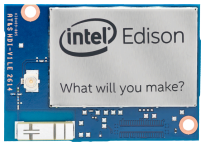




Intel® Edison Tutorial: IoT Distributed Environmental System Reference Design

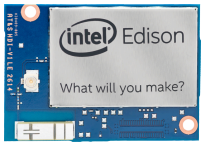
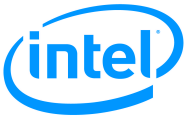
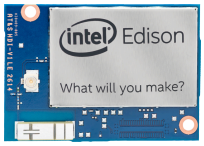


Table of Contents

Introduction **3**
Things Needed **3**
Hardware Setup **4**
Software Setup **7**
Running the code **8**
Exercises **12**

Revision history		
Version	**Date**	**Comment**
1.0	07/15/2016	Final Draft v1.0
1.1	08/11/2016	Final Draft v1.1





Introduction

In this tutorial you will

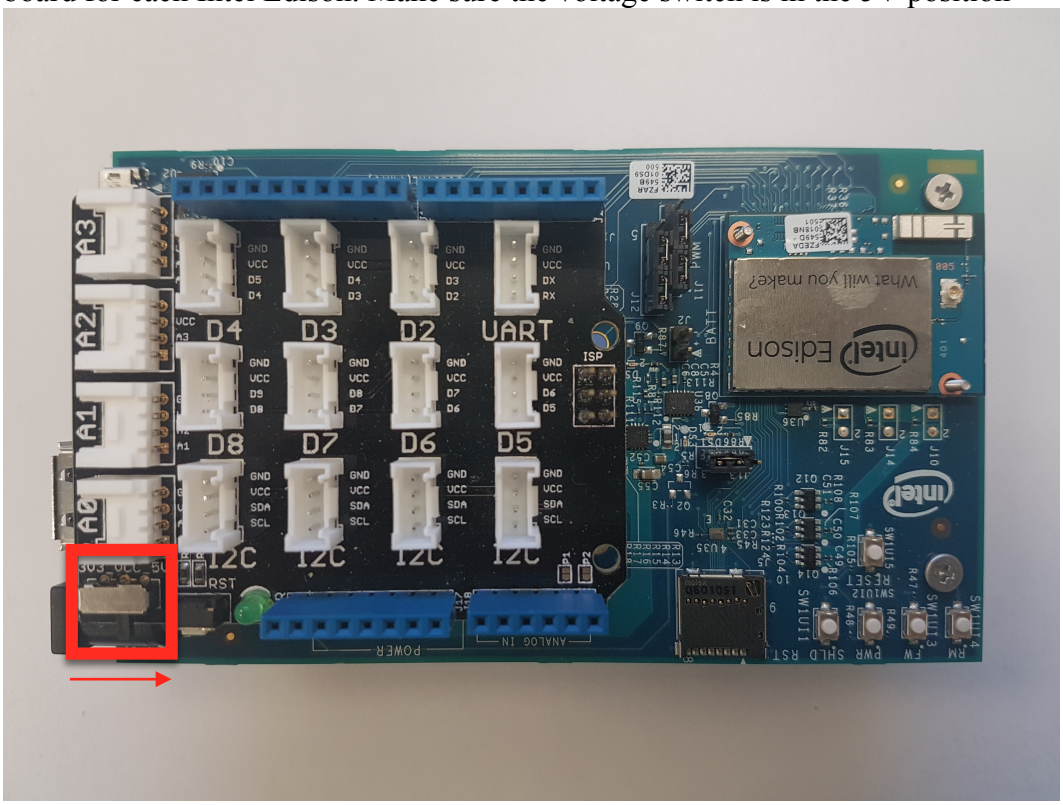
1. Build a networked embedded system using:
    a. Sensors available in the Seeed Grove Starter Kit
    b. Multiple Intel Edison boards

Things Needed

1. 2x Intel® Edison (additional boards can be incorporated for more complex designs)
2. A PC or a Mac
3. 1x Seeed Grove Starter Kit per Intel Edison
4. 2x Micro USB cables per Intel Edison
5. An internet connection

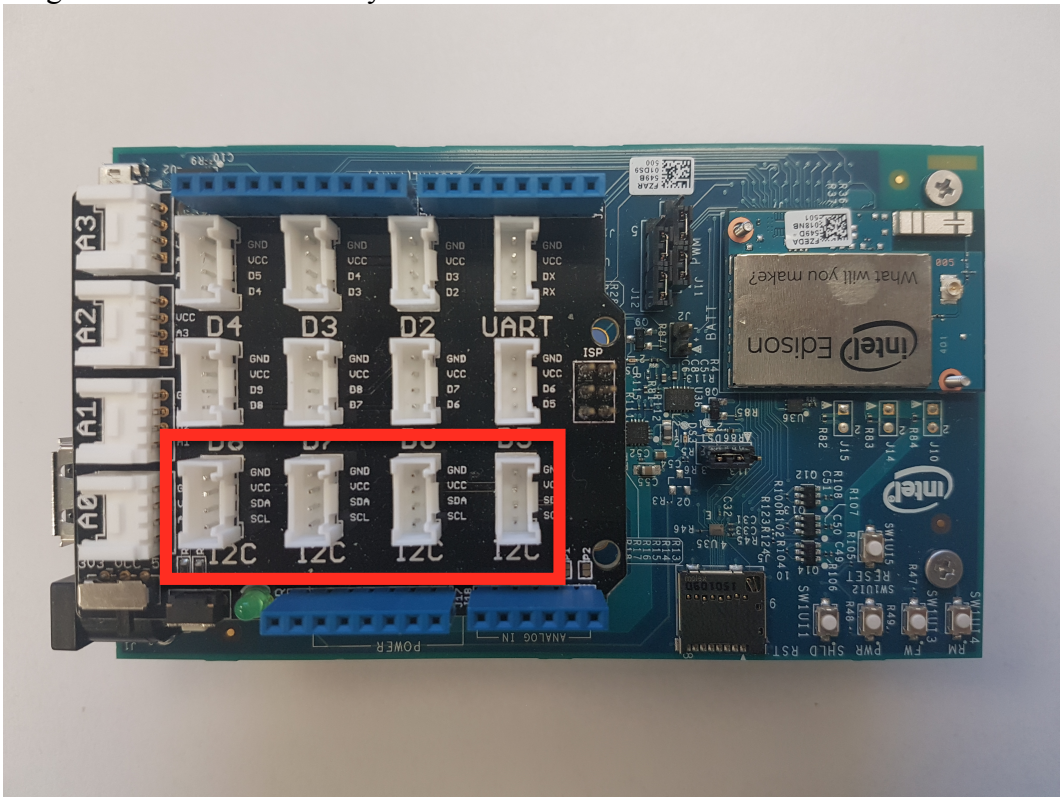
Hardware Setup

1. Place the shield from the Seeed Grove Starter Kit onto the Arduino compatible breakout board for each Intel Edison. Make sure the voltage switch is in the 5V position

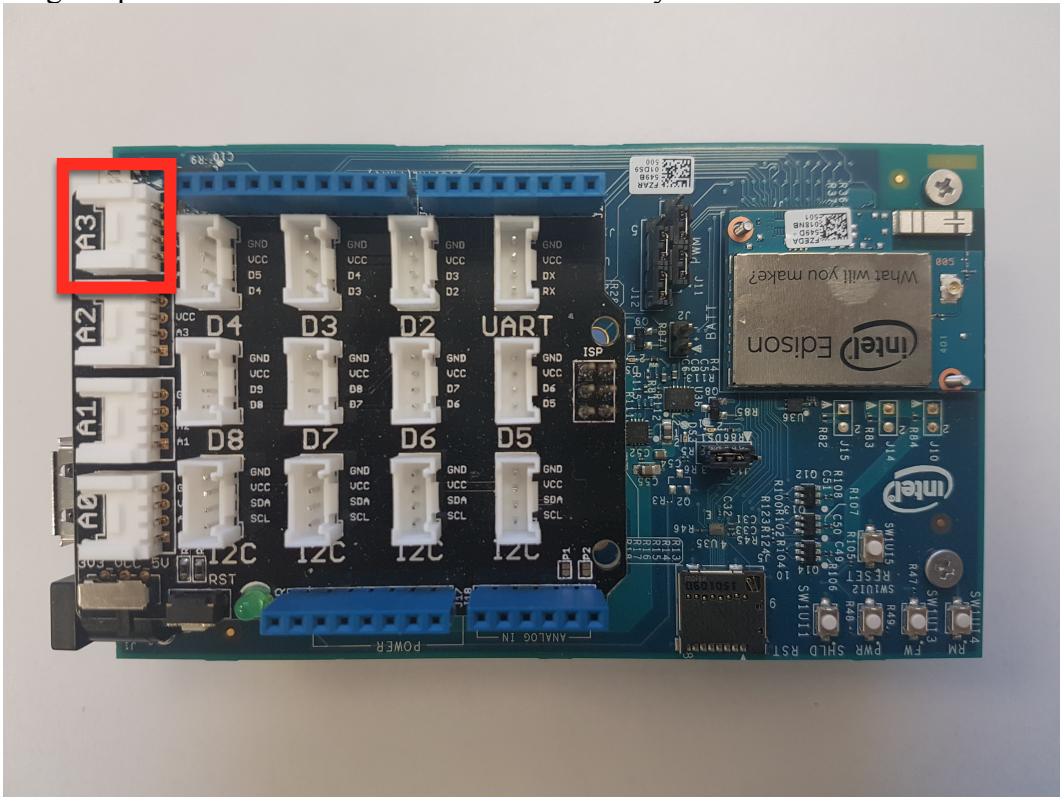


2. Select an Intel Edison board. This Intel Edison will be the server

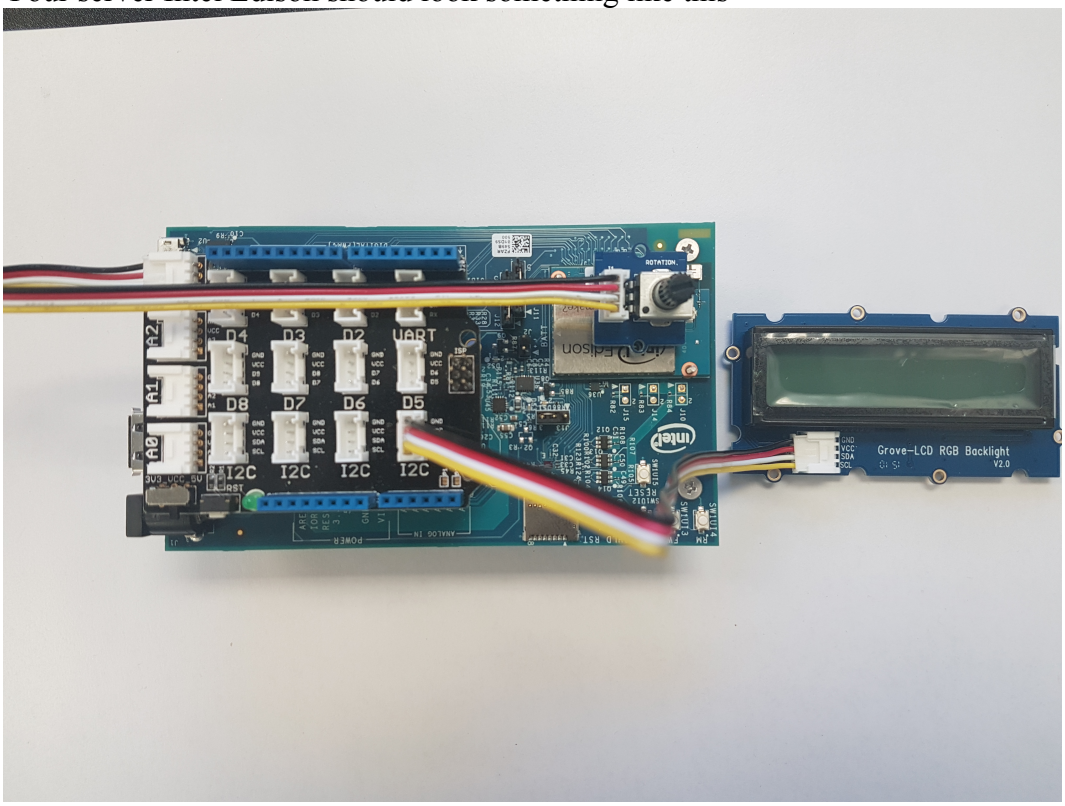
3. Plug the LCD screen into any of the I2C slots of the server Intel Edison



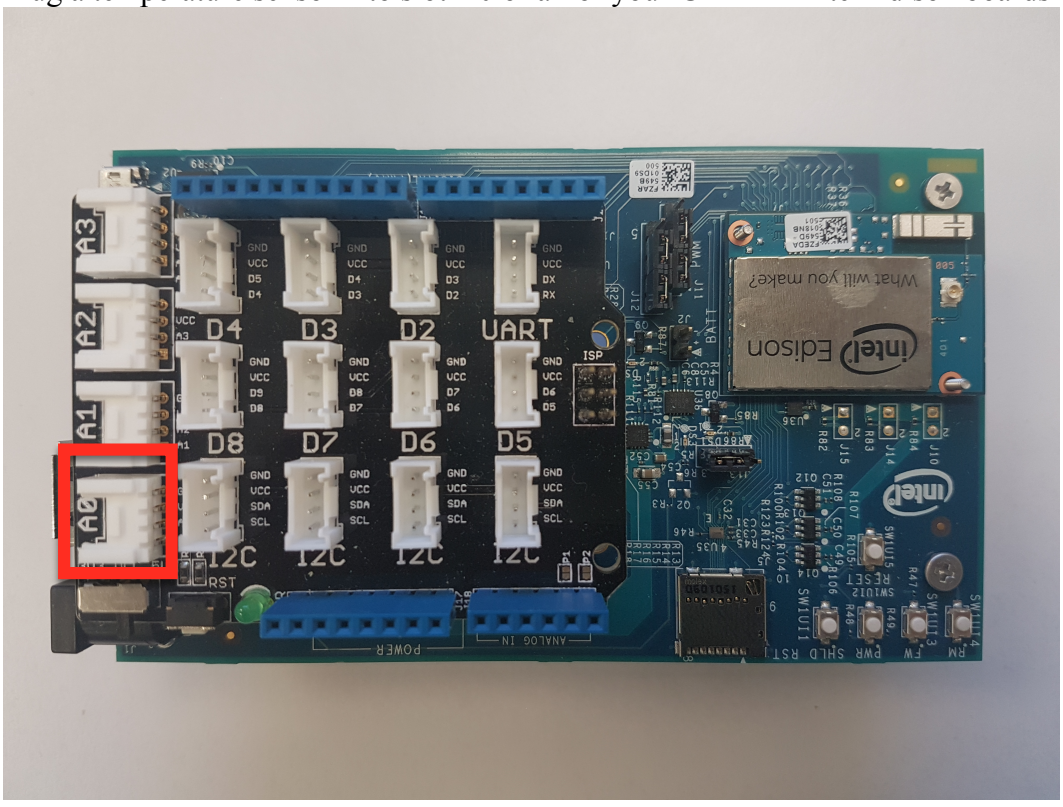
4. Plug the potentiometer into the slot labelled A3 of your server Intel Edison



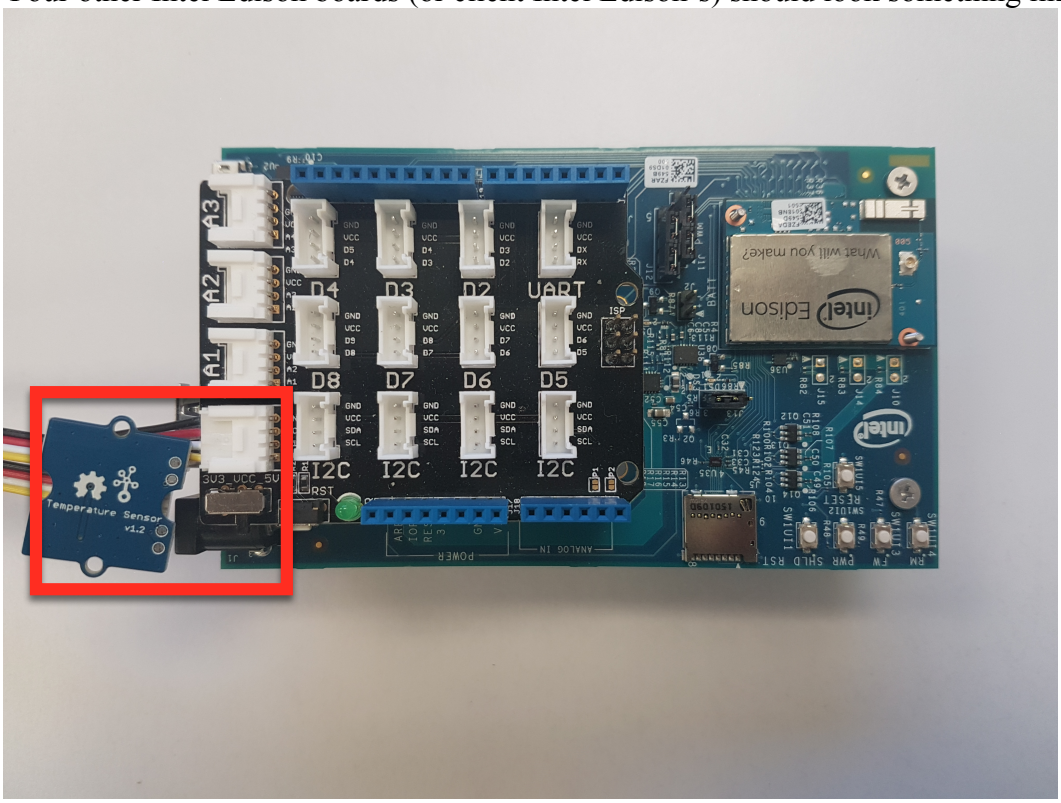
5. Your server Intel Edison should look something like this



6. Plug a temperature sensor into slot A0 of all of your **OTHER** Intel Edison boards



7. Your other Intel Edison boards (or client Intel Edison's) should look something like this



N.B.: Notice that the sensor has "Temperature Sensor" written on the bottom.

Software Setup

1. Download the code available in the same folder where you downloaded this PDF

2. Connect each Intel Edison to the internet
3. SSH into the server Intel Edison (the board used in steps 2-5 in Hardware Setup)
4. Upload the code available under the folder labelled "server" to the server Intel Edison
5. Use the makefile labelled "Makefile" to compile the project

```
root@server_edison:~/SUSP/pth# ls
Makefile   knob.h     rgb_lcd.c  server.c
knob.c     main.c     rgb_lcd.h  server.h
root@server_edison:~/SUSP/pth# make
gcc -c -Wall main.c -o main.o
gcc -c -Wall rgb_lcd.c -o rgb_lcd.o
gcc -c -Wall knob.c -o knob.o
gcc -c -Wall server.c -o server.o
gcc -lmraa -pthread main.o rgb_lcd.o knob.o server.o -o main
root@server_edison:~/SUSP/pth#
```

What the output should look like after compilation of the server files

6. SSH into your other Intel Edison board
7. Upload the code available under the folder labelled “Client” to the Edison from step 6
8. Use the makefile labelled “Makefile” to compile the project

```
root@client_edison_1:~/SUSP/Client# ls
Makefile  client.c  client.h  main.c  thermistor.c  thermistor.h
root@client_edison_1:~/SUSP/Client# make
gcc -c -Wall main.c -o main.o
gcc -c -Wall client.c -o client.o
gcc -c -Wall thermistor.c -o thermistor.o
gcc -lmraa -pthread -lm main.o client.o thermistor.o -o main
root@client_edison_1:~/SUSP/Client#
```

What the output should look like after compilation of the client files

9. Repeat step 6-8 for each Intel Edison board you have that isn’t the server Intel Edison

Running the code

1. Find the IP address of your server Intel Edison board.

```
son:~# configure_edison --showWiFiIP
131.179.54.225
son:~#
```

2. Execute the main file on the server Intel Edison

$ ./main

```
root@server_edison:~/SUSP/pth# ./main
server init: 0.0.0.0:5000
Temp Setting: 72
Temp Setting: 71
ERROR on accept: Resource temporarily unavailable
Latest child process is waiting for an incoming client connection.
ERROR on accept: Resource temporarily unavailable
Latest child process is waiting for an incoming client connection.
ERROR on accept: Resource temporarily unavailable
Latest child process is waiting for an incoming client connection.
ERROR on accept: Resource temporarily unavailable
Latest child process is waiting for an incoming client connection.
ERROR on accept: Resource temporarily unavailable
Latest child process is waiting for an incoming client connection.
```

Showing a successful run of main. Remember this port number for use in step 3

If the main function exits and shows a message similar to below, follow the below steps

```
root@server_edison:~/SUSP/pth# ./main
ERROR on binding: Address already in use

...cleanup operations complete. Exiting main.
root@server_edison:~/SUSP/pth#
```

Binding error, the port is already in use. Let’s try a different port.

$ vi main.c

```
 1 #include <stdio.h>
 2 #include <string.h>
 3
 4 #include <sys/types.h>
 5 #include <signal.h>
 6 #include <pthread.h>
 7
 8 #include "server.h"
 9 #include "rgb_lcd.h"
10 #include "knob.h"
11
12 #define PORTNO 5000
13
14 static volatile int run_flag = 1;
15
16 void do_when_interrupted()
17 {
18         run_flag = 0;
19         printf("\nThreads exiting. Please wait for cleanup operations to complete...\n");
20 }
```

Change line 12 such that it has a port in the range of 2000-8000. Let's try port 8000

```
 1 #include <stdio.h>
 2 #include <string.h>
 3
 4 #include <sys/types.h>
 5 #include <signal.h>
 6 #include <pthread.h>
 7
 8 #include "server.h"
 9 #include "rgb_lcd.h"
10 #include "knob.h"
11
12 #define PORTNO 8000
13
14 static volatile int run_flag = 1;
15
16 void do_when_interrupted()
17 {
18         run_flag = 0;
19         printf("\nThreads exiting. Please wait for cleanup operations to complete...\n");
20 }
```

Save and exit the main.c file, then recompile the project

```
root@server_edison:~/SUSP/pth# make
gcc -c -Wall main.c -o main.o
gcc -lmraa -pthread main.o rgb_lcd.o knob.o server.o -o main
root@server_edison:~/SUSP/pth#
```

Try to run the program again

```
root@server_edison:~/SUSP/pth# ./main
server init: 0.0.0.0 8000
Temp Setting: 71
ERROR on accept: Resource temporarily unavailable
Latest child process is waiting for an incoming client connection.
```

New port number. Required for step 3 of this tutorial.

If this still doesn't fix the code, try to reboot the board and see if that solves the problem. If the problem still persists, please see your instructor.

3. Run main on each of the client Intel Edison's

$ ./main <IP_ADDR_SERVER> <PORT>

```
root@client_edison_1:~/SUSP/client# ls
Makefile      client.h      main          main.o        thermistor.h
client.c      client.o      main.c        thermistor.c  thermistor.o
root@client_edison_1:~/SUSP/client# ./main 131.179.54.225 5000
msg from server: 131.179.18.148 sent the server: 1468606431.0704898834,75.42
msg from server: 131.179.18.148 sent the server: 1468606431.5793359280,75.42
msg from server: 131.179.18.148 sent the server: 1468606432.2777819633,75.42
msg from server: 131.179.18.148 sent the server: 1468606432.7964000702,75.42
msg from server: 131.179.18.148 sent the server: 1468606433.3111279011,75.42
msg from server: 131.179.18.148 sent the server: 1468606433.8248019218,75.42
msg from server: 131.179.18.148 sent the server: 1468606434.3400969505,75.42
msg from server: 131.179.18.148 sent the server: 1468606434.8472480774,75.42
msg from server: 131.179.18.148 sent the server: 1468606435.3536989689,75.42
msg from server: 131.179.18.148 sent the server: 1468606435.8659710884,75.42
msg from server: 131.179.18.148 sent the server: 1468606436.3776130676,75.42
msg from server: 131.179.18.148 sent the server: 1468606436.8887650967,75.42
```

<IP_ADDR_SERVER>** will vary based on step 1. **<PORT>** will vary based on step 2. For this example, the IP address is **131.179.54.225,** and the port is **5000

4. To kill either a client or the server, press “CTRL+C”

```
msg from server: 131.179.18.148 sent the server: 1468606774.8177680969,75.42
msg from server: 131.179.18.148 sent the server: 1468606775.3310348988,75.42
msg from server: 131.179.18.148 sent the server: 1468606775.8372011185,75.42
msg from server: 131.179.18.148 sent the server: 1468606776.3601720333,75.42
msg from server: 131.179.18.148 sent the server: 1468606776.8726670742,75.42
msg from server: 131.179.18.148 sent the server: 1468606777.3788759708,75.42
msg from server: 131.179.18.148 sent the server: 1468606777.8909010887,75.42
msg from server: 131.179.18.148 sent the server: 1468606778.4022259712,75.42
msg from server: 131.179.18.148 sent the server: 1468606779.0011510849,75.42
msg from server: 131.179.18.148 sent the server: 1468606779.6216130257,75.42
msg from server: 131.179.18.148 sent the server: 1468606780.1411230564,75.42
msg from server: 131.179.18.148 sent the server: 1468606780.6493310928,75.42
msg from server: 131.179.18.148 sent the server: 1468606781.1642179489,75.42
msg from server: 131.179.18.148 sent the server: 1468606781.6715939045,75.42
msg from server: 131.179.18.148 sent the server: 1468606782.2822749615,75.42
^CERROR reading from socket: Interrupted system call
root@client_edison_1:~/SUSP/client#
```

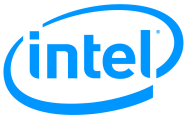
Killing the client with ctrl+c

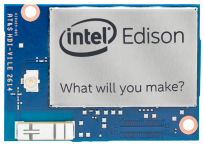
```
131.179.18.148 says: 1468606929.3963649273,75.42
131.179.18.148 says: 1468606929.9521520138,75.42
^C
Threads exiting. Please wait for cleanup operations to complete...
131.179.18.148 says: 1468606930.4628500938,75.42
ERROR on accept: Resource temporarily unavailable
Latest child process is waiting for an incoming client connection.

...cleanup operations complete. Exiting main.
root@server_edison:~/SUSP/pth#
```

Killing the server with ctrl+c

5. Once you have killed the server, examine the .csv files created on the server Intel Edison

```
root@server_edison:~/SUSP/pth# ls -l *.csv
-rw-r--r--    1 root     root           923 Jul 15 18:26 output_file_IP_131.179.13.243_TIMESTAMP_1468607145.csv
root@server_edison:~/SUSP/pth# head output_file_IP_131.179.13.243_TIMESTAMP_1468607145.csv
time (epoch),temperature/C
1468607145.6926860809,17.38
1468607146.2030038834,24.08
1468607146.7926421165,24.69
1468607147.3114590645,24.43
1468607147.8286709785,24.61
1468607148.3538000584,24.78
1468607148.8631141186,24.52
1468607149.4177949429,24.43
1468607149.9618151188,24.35
root@server_edison:~/SUSP/pth#
```





Exercises

- Swap the sensor from a temperature sensor to a light intensity sensor. Write code to make the system function with the new light sensor
- Add the temperature sensor back in, send the server data for both temperature and light levels
- Based on the temperature information received from the server, display whether the temperature recorded by the decentralized nodes is less than, equal to, or greater than the temperature displayed on the LCD screen