Project 4B

Week 9

CS 111

What is this project about ?

- Project 4 is an IoT project. The final result will be a networked temperature sensor, communicating on a (potentially) encrypted channel

- This week, the goal is to run an application using external sensors and log results on the Beaglebone

- The main difficulty should be reading data from the sensor correctly
    - You will use the temperature sensor for reading
    - You will use the button for shutdown

Assemble Beaglebone

- Plug the Base Shield in the BeagleBone
  - Notice that there are 2 kinds of pins : analog and digital
  - The temperature sensor is analog, the button is digital
- Plug the temperature sensor to A0 / A1
  - Will be I/O pin #1
- Plug the push-button to GPIO 50
  - Will be I/O pin #60
- Turn the voltage on the base cape to 5V

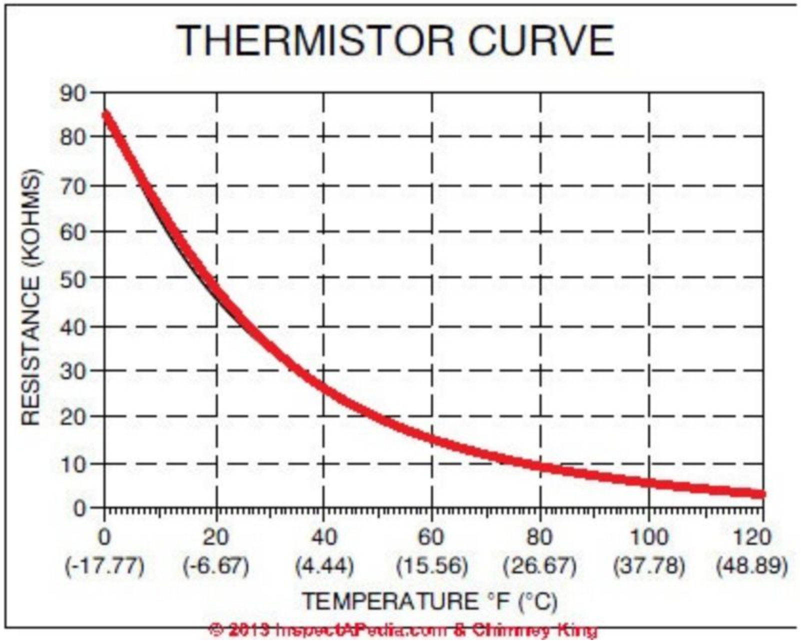
Breakdown of the tasks

- Arguments to your program:
    - Period : interval (s) between 2 temperature measurements
    - Scale : choose the reading scale between Celsius and Fahrenheit
    - Log : choose the file where measurements are saved

- You should also accept parameters from stdin:
    - Scale, to switch units during execution
    - Period, to change the period during execution
    - Stop : stop generating reports (you are not exiting, you are still processing input parameters). If already stopped, do nothing.
    - Start : resume reports (if stopped)
    - Log <text> : add <text> to logfile
    - OFF : output and log a timestamped shutdown message, and exit

The Temperature sensor

- Is a thermistor :



- You should set your base cape to 5V for more accurate readings
    - The readings **will** be inaccurate (~15F from real value), this isn't a problem

Temperature sensor

- The equation for determining the temperature is :

$$\frac{1}{T} = \frac{1}{T_o} + (\frac{1}{\beta}) \cdot \ln\left(\frac{R}{R_o}\right)$$

Reference Temperature: 298.15 K

Beta-value : 4275

Reference resistance at To: 100000 Ohms

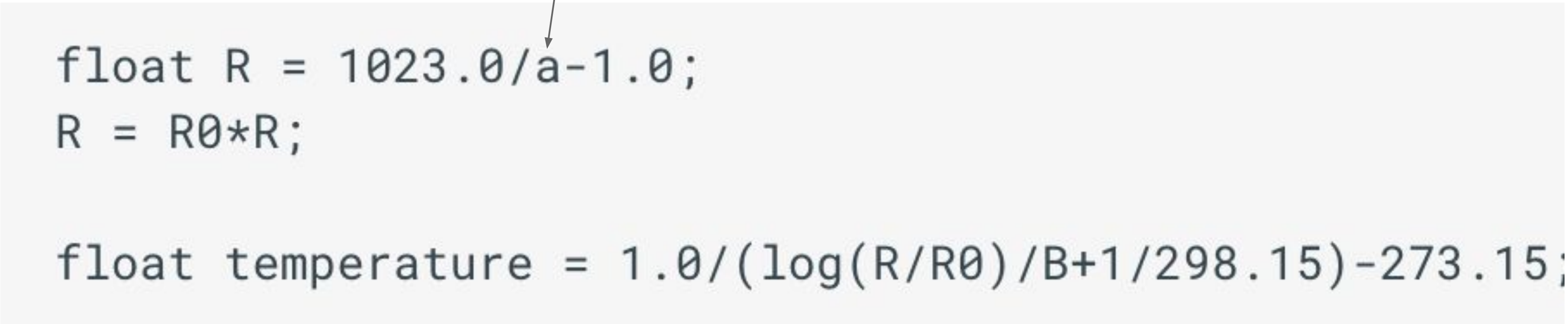
- The above is some background, the implementation is found at :

  http://wiki.seeedstudio.com/Grove-Temperature_Sensor_V1.2/

Temperature Sensor

The lines you’re interested in on the previous page are :

Your reading



```
float R = 1023.0/a-1.0;
R = R0*R;

float temperature = 1.0/(log(R/R0)/B+1/298.15)-273.15;
```

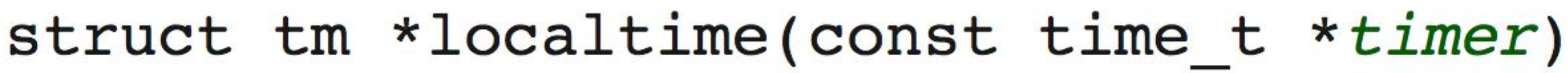
- As a reminder :
  - Kelvin to Celsius : K - 273.15
  - Celsius to Farenheit: C * 9/5 + 32

MRAA : I/O library

- Include headers
- Allocate sensors as mraa_gpio_context and mraa_aio_context (argument is the pin number from the board)
- Initialize them. The button is an input
  - mraa_gpio/aio_init(context)
  - mraa_gpio_dir(context, direction) (don't need direction for aio)
- Read from them
  - mraa_gpio/aio_read(context)
  - The button will return 0 or 1
  - The temperature sensor will return a voltage
  - Both will return -1 on error
- Close them
  - mraa_gpio/aio_close(context)

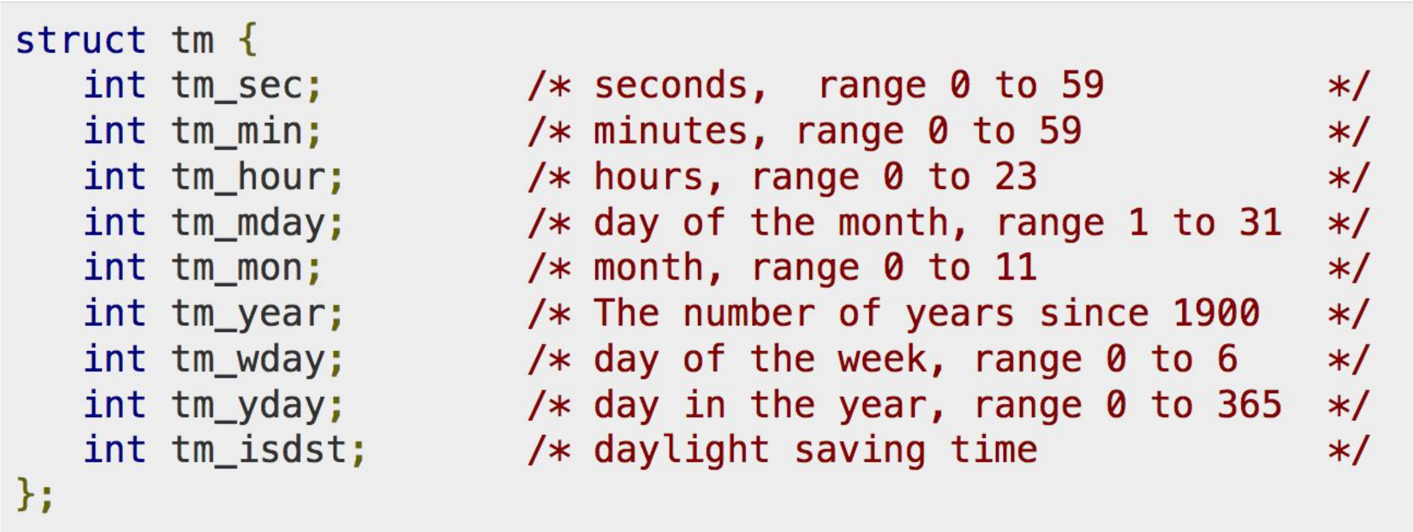
localtime()

Goal: Return local time



```
struct tm *localtime(const time_t *timer)
```

It will fill up the following structure:



```
struct tm {
   int tm_sec;         /* seconds,  range 0 to 59          */
   int tm_min;         /* minutes, range 0 to 59           */
   int tm_hour;        /* hours, range 0 to 23             */
   int tm_mday;        /* day of the month, range 1 to 31  */
   int tm_mon;         /* month, range 0 to 11             */
   int tm_year;        /* The number of years since 1900   */
   int tm_wday;        /* day of the week, range 0 to 6    */
   int tm_yday;        /* day in the year, range 0 to 365  */
   int tm_isdst;       /* daylight saving time             */
};
```

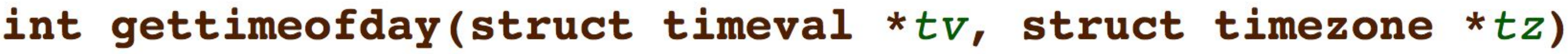
localtime()

To use it properly, you need your timezone to be set on your Beaglebone:

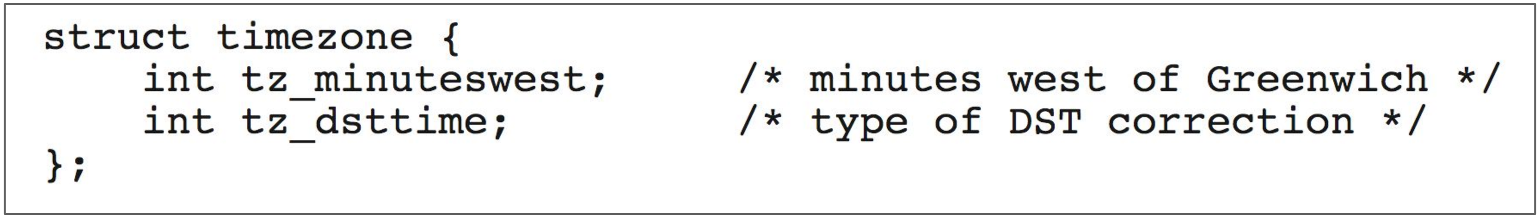
- You can check your current setting using ‘date’
- Several ways to change this setting, an easy one would be :
    - apt- get install tzdata
    - dpkg-reconfigure tzdata
    - Follow the steps
- This is optional… we will test your code on another device

Do I need a new measurement?

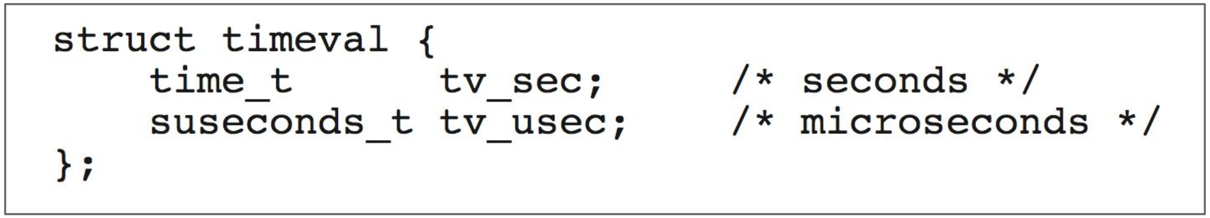
- Several ways to go about this
- You could use :



```
int gettimeofday(struct timeval *tv, struct timezone *tz)
```



```
struct timezone {
    int tz_minuteswest;     /* minutes west of Greenwich */
    int tz_dsttime;         /* type of DST correction */
};
```



```
struct timeval {
    time_t      tv_sec;     /* seconds */
    suseconds_t tv_usec;    /* microseconds */
};
```

- If enough time has passed and you read, set the time when the next reading is due

Generating reports

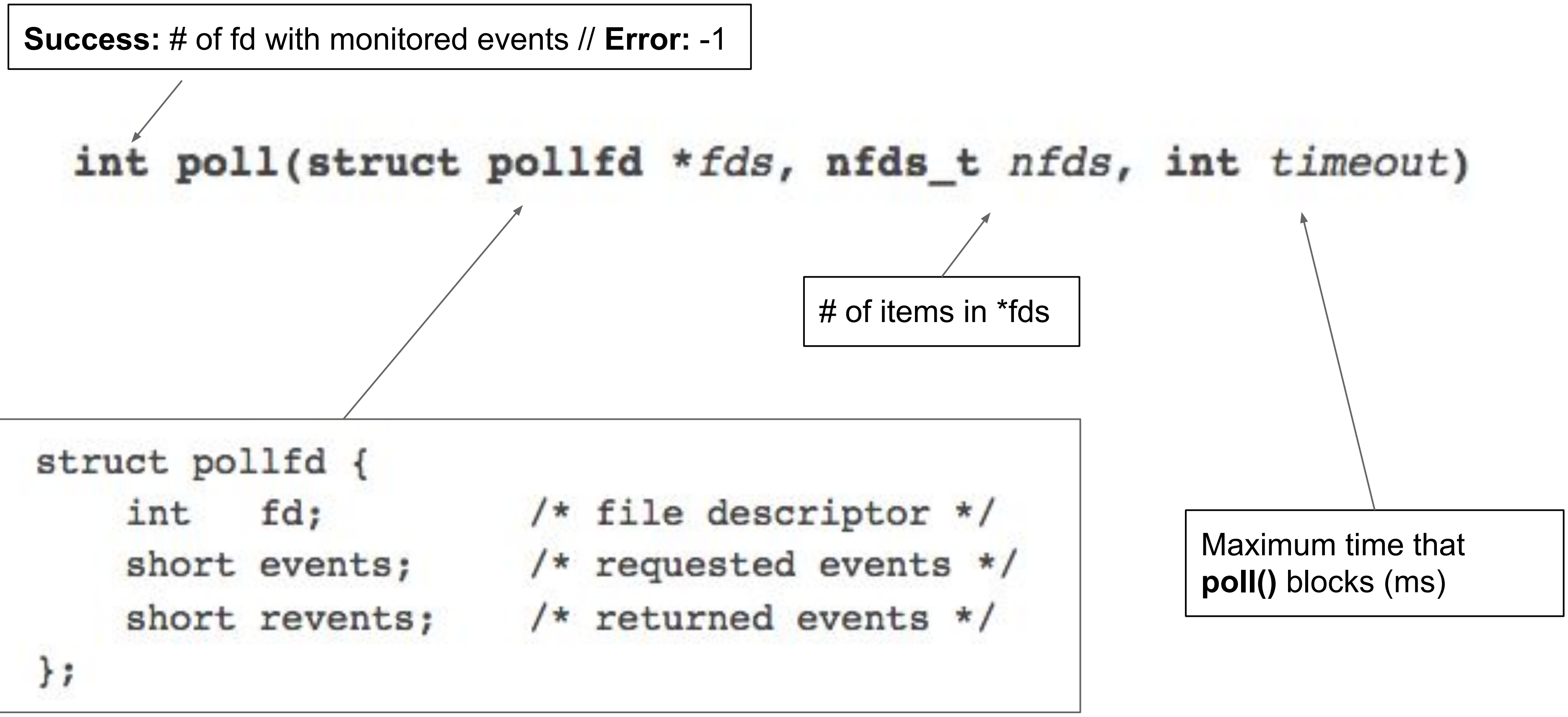
- Create an outgoing buffer
- Print the formatted time and temperature to that buffer
  - Watch out, the temperature returned by default is not in the correct format!
- Push that buffer to stdout
- If the logfile is enabled, also push that buffer to the file

Receiving commands

- Commands will come from a pipe, not a keyboard
  - A single read may return partial or multiple lines
- Therefore, use a buffer
  - Check at every iteration if commands can be found
- To wait on commands, poll() is appropriate
  - You can't poll() on the button
  - You can use several threads (1 for commands, one for sensors)
  - You can simply check the status of the button every second (that frequency is high enough for this project)

poll(2)

Goal: Wait for some event on a file descriptor (for I/O)



Success: # of fd with monitored events // **Error:** -1

```
int poll(struct pollfd *fds, nfds_t nfds, int timeout)
```

# of items in *fds

```
struct pollfd {
    int   fd;         /* file descriptor */
    short events;     /* requested events */
    short revents;    /* returned events */
};
```

Maximum time that **poll()** blocks (ms)

poll(2)

- Some of the bits that may be set/returned :
  - POLLIN : Data may be read without blocking
  - POLLOUT : Data may be written without blocking
  - POLLERR* : (revents only) Error has occurred on device / stream
  - POLLHUP* : (revents only) Device disconnected / pipe closed <- Mutually exclusive with POLLOUT
  - POLLNVAL* : (revents only) Invalid fd
- When you fill up the pollfd structure:
  - Indicate which events you want to monitor in the events field
    - Eg: pollfd.events = POLLIN (same as POLLIN & POLLER)
  - When poll() returns, it will fill out the revent field
    - Eg: pollfd.revents = POLLERR (there was an error)
    - Note that poll() automatically reports on * fields
      - You don't have to include them in the events field!
    - However if you don't specify POLLIN, poll() will not check for input

DUMMY

- Choose that option if you want to be able to test the base functionality of your code without the board

- In that case, you cannot import the headers

- Instead, define the functions yourself
    - Simply have them return the correct type of data, and take in the correct type of input

- That way you'll be able to test functionality of your code on your laptop, before debugging sensor reads

FAQ

- My program segfaults!
  - This is likely due to the initialization of your I/O. If your sensors aren’t initialized properly, the init function will return NULL, and you will segfault when trying to read().
    - Flash your board
    - Run your code from root

- What edge case order of commands should we handle?
  - There will be **no** tricky edge cases, such as:
    - A period of 0
    - Stop and start within a single period
    - Stop and Stop, start and start generate no behavior
    - Changing the period can take effect after the next report
  - On startup, generate first reading before processing input

FAQ

- My program hangs indefinitely?
  - If your shutdown didn't go through, it's possible that your program is still running in the background.
  - top -U <username> will help you verify this
  - You can kill your program using its pid if this happens

- UCLA_WEB blocks ntp messages, you won't be able to download the sanity script functions on that network (Use eduroam, your mobile hotspot, or home router)