Course: COMP 413 - Internet of Things **Fall 2023**

Lab Session 3: Numerical Computations

Lecturer: Dr. Abdulkadir Köse

Acknowledgement: The lab session follows IoT module taught by University of Surrey.

3.1 Introduction

Data processing is an important feature in IoT. In a typical design, we prefer to offload the data processing tasks to the cloud servers in order to reduce the power consumption of a mote. However, it is inevitable that in some situations, the mote must perform some simple data processing to either avoid transmitting excessive data to the servers or respond promptly to a sensed event.

Tasks in data processing often use techniques developed in numerical computations. Numerical computations require iterative calculation or loops to perform the corresponding calculation. It is critical to manage a loop in your program to ensure your program is not holding up the CPU for a long time which may cause resource starvation to other critical background tasks.

In ContikiOS, if your program holds the CPU for over a second in a single process, the built-in watchdog will trigger a soft reset of the mote. You will see the mote keep booting (showing the initial message again and again). This usually happens when you run a loop that takes more than a second to complete. The use of watchdog has many benefits, for example, if your program runs into a rare bug and got stuck in a forever loop, or your program suffers from a memory leak problem after a long period of running, an automatic soft reset will help to recover the situation.

You will also learn how to setup a timer in the program for periodic task.

Equipment and materials:

- ❑ Lab PC or your own computer
- ❑ An XM1000 mote (optional, collect it during the lab)
- ❑ InstantContiki-2.6 image file, see "Lab Session 1" for the download link

3.2 Learning Objectives

We expect you to know the following from the previous lab sessions:

- How to launch Contiki-2.6 in the lab (if you are attending the lab) or from your own PC (if you are working remotely)
- How to compile and upload your code onto XM1000 mote, and debug your code by logging debugging messages onto the console (if you have a mote);
- How to load a Cooja simulation environment, start and stop the simulation, and debug your code by logging debugging messages onto the console;
- Be familiar with Cooja environment and Contiki programming structure.

In this lab session, you will learn:

- Understand the limitation of CPU resource in an embedded system and careful use of loops in a program;
- How to implement own numerical computation;

- How to use timer to run periodic task;

3.3 Periodic Tasks and Timer

In Contiki programming, we can use timer to schedule a task to run periodically. We use a loop with a timer control to rerun it repeatedly. First, we need to construct a timer data structure:

```
static struct etimer timer;
```

After `PROCESS_BEGIN()`, the timer has to be configured, and the duration has to be set. This is how:

```
etimer_set(&timer, CLOCK_CONF_SECOND);
```

Then we create an infinite while loop that runs our code periodically:

```
while(1) {

  PROCESS_WAIT_EVENT_UNTIL(ev=PROCESS_EVENT_TIMER); // wait for
                                                    // the timer

  // do the magic here ... that gets executed periodically

  etimer_reset(&timer); // reset the timer
}
```

Note: even we have created an infinite loop, the process actually periodically returns back the control to the operating system while waiting for the timer. As a result, the process is not holding the CPU resource. However, since the process returns the control, contents in the local variables will be lost. To maintain the variable contents, use static variables instead.

3.4 Exercise

The task you need to calculate the square root of a value. You can find the skeleton code in "/home/user/contiki-2.6/labs/L3". In the skeleton code, a random value is periodically generated and shown on the console. The square root of the generated random value should be calculated and shown on the console. Currently, there is no implementation of the square root function in the skeleton code. You need to implement your own square root numerical computation. While there are many numerical methods, we recommend the Babylonian method:

```
To compute $\sqrt{S}$:
Step 1: Pick a guess $x_0$ which is near the value of $\sqrt{S}$
Step 2: Run a loop to compute $x_{n+1} = \frac{1}{2}\left(x_n + \frac{S}{x_n}\right)$ for $n = 0,1,2,\ldots$
Step 3: For each iteration, check the result accuracy by performing $(x_{n+1})^2$
        If the difference between $(x_{n+1})^2$ and $S$ is deemed small, break the loop.
Step 4: Output $x_{n+1}$ as the outcome of $\sqrt{S}$.
```

In the folder, you will also find a prepared simulation environment named "cooja_calculation.csc" for this experiment if you are working with Cooja Simulator.