Introduction to IoT: Autumn 2019

Exercise set: 2

Due on 18th September 2019 by 16:00.

Instructions: All course participants are requested to submit their exercise solutions (in English) electronically to the instructors Agustin Zuniga (agustin.zuniga at helsinki.fi) and Prof. Petteri Nurmi (petteri.nurmi at cs.helsinki.fi) by the due date. Use the following subject in your email: *IoT_week[#]_[last name_first name]_[student number], (i.e. IoT_week2_Zuniga_Agustin_12345)*

Your submission have to contain no more than **three (3)** single-spaced and numbered pages. Use font type Arial or its equivalent with size no smaller than 10 points. Include the exercise set number, your full name and student Id in the upper right corner of the first page.

In all the exercises, do not just give the answer, but also the derivation how you obtained it. Participants are encouraged to review course material to answer the problems and in some cases write computer programs to derive solutions.

Learning objective: In this set of exercises we will take a look of operating systems in *IoT* environments. The exercises will help you to understand better the characteristics and requirements of operating systems in IoT applications.

Selecting Programming Base, Scheduling and OSs in IoT Application

Task 1 (4 pts.)

One car manufacturer wants to implement IoT features on its vehicles. The provider plans to use cars' elements as smart objects to improve driving quality and safety while driving. For instance, speed limit information will be automatically displayed according to GPS coordinates using on-line street maps and police information. Additionally, the manufacturer is periodically collecting smart objects' data to evaluate cars' condition on-line. This information is used to inform drivers about possible issues and suggest how to solve them in real-time. Additionally, information is also used to automatically re-calibrate cars' sensors. Your task is to describe the following about the implementation of this IoT application (give justified arguments for maximum points):

1. Would you use event or thread-based programmingg?
2. Would you use pre-emptive or non-preemptive scheduling?
3. Would you use Contiki or Linux based system such as Android?
4. Would you have hard, firm or soft constraints on real-time operation?

Task 2 (4 pts.)

Select two application domains, one focused on consumer level IoT and one focused on enterprise IoT.

1. Describe one smart object for each domain and what is its functionality (for maximum points, the system should have control, sensing, and actuation)
2. Describe what requirements would the object and the application impose on the Operating System
3. Which operating system would be best suited and why?
4. Which operating system would be worst suited and why?

You can find information about the limitations and use of OSs for different IoT applications in the second week lectures and the following sources:

- Section IV from Musaddiq A, Zikria YB, Hahm O, Yu H, Bashir AK, Kim SW. *A survey on resource management in IoT operating systems.* IEEE Access. 2018 Feb 21; 6:8459-82.
- Section XI (tables: VIII and XIV) from Javed F, Afzal MK, Sharif M, Kim BS. *Internet of Things (IoT) operating systems support, networking technologies, applications, and challenges: A comparative review.* IEEE Communications Surveys Tutorials. 2018 Mar 21; 20(3):2062-100.

Testing IoT OS simulator: Cooja

In this exercise we use *Cooja*, a simulator of IoT Contiki nodes. Complete steps 1, 2 and 3 of *Get starting with Contiki* guide (http://www.contiki-os.org/start.html, you can also find a copy of the steps as part of the weekly exercise files.)

Now that you are familiar with Cooja's interface, you will use it to interact with the nodes and analyse their performance.

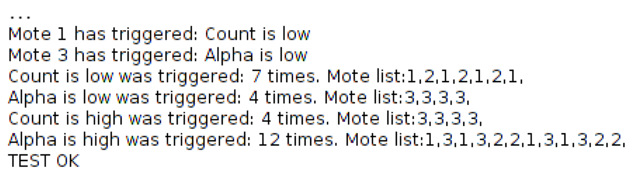
Task 3 (2 pts.)

- From the Cooja's menu select *File>Open simulation>Browse...*, and open the file /home/user/contiki/tools/cooja/contiki_tests/multithreading.csc [1]
- Once the file has been loaded, deactivate the simulation in the *Script editor* window and activate mote ID and type visualisation in the *Network* panel.
- Start the simulation using the *Simulation control* panel. In the *Mote output* window, you will observe the messages sent by mote 1.
- Add two new motes into the *Network* window (one of type Z1 and the other of type ESB). Create the motes using the following firmware: /home/user/contiki-2.6/examples/multi-threading/multi-threading.c
- Activate *Script editor*'s simulation, uncomment lines 16, 19, 22 and 25 and start the simulation.
  1. What is the script doing?
  2. Modify the script to satisfy the following requirements (put your source code in a .txt file and include it as part of your submission to get full points of this task):

  – Low-Count counter depends only on motes 1 and 2, high-Count and low-Alpha on mote 3, and high-Alpha on the three motes.

[1] http://contiki.sourceforge.net/docs/2.6/a00007.html

- The test is OK only when all the counters are greater than 3 or high-Alpha counter is greater than 12.
- The output prints how counters are triggered during the simulation and include the corresponding mote ID. Once test is OK, output shows the final values of counters and the list of motes that triggered each counter (see Figure 1 for an example).



```
...
Mote 1 has triggered: Count is low
Mote 3 has triggered: Alpha is low
Count is low was triggered: 7 times. Mote list:1,2,1,2,1,2,1,
Alpha is low was triggered: 4 times. Mote list:3,3,3,3,
Count is high was triggered: 4 times. Mote list:3,3,3,3,
Alpha is high was triggered: 12 times. Mote list:1,3,1,3,2,2,1,3,1,3,2,2,
TEST OK
```

Figure 1: Output example