

Lab Session 2: Hello World

Lecturers: Dr Mohammad Shojafar, Dr Chuan H Foh

Disclaimer: *These notes have not been subjected to the usual scrutiny reserved for formal publications. They may be distributed outside this class only with the permission of the Instructor.*

2.1 Introduction

During this lab session we are going to compile and run our first program on sensor hardware. As tradition demands, we are going to run a simple Hello World! program that outputs "Hello World!".

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

2.2 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 (if you have a mote);
- How to load a Cooja simulation environment, start and stop the simulation;
- Be familiar with Cooja environment and Contiki programming structure.

In this lab session, you will learn:

- The basic structure of Contiki C-programming code;
- The limitation of embedded C and Contiki library;
- Find information from the ContikiOS documentation;

2.3 Hello World! Code Example

```
1 #include "contiki.h"
2
3 #include <stdio.h> /* For printf() */
4 /*-----*/
5 PROCESS(hello_world_process, "Hello world process");
6 AUTOSTART_PROCESSES(&hello_world_process);
7 /*-----*/
8 PROCESS_THREAD(hello_world_process, ev, data)
9 {
10     PROCESS_BEGIN();
11
12     printf("Hello, world\n");
13
14     PROCESS_END();
15 }
16 /*-----*/
```

In the above listing we see our first program. To understand the code, we go through it line by line.

In line 1 we include the Contiki header files that include the Contiki OS into the compiled program and allow to access the scheduling and abstraction APIs of Contiki.

Contiki follows Protothreads (see also <https://en.wikipedia.org/wiki/Protothreads>), a concurrent programming model with a low-overhead.

Line 3 includes the standard input/output library needed to write to the standard output.

Line 5 defines the processes to be executed and included during runtime. In this case we include only one process: the Hello World process. It is possible and usually the case to define several concurrent processes, for example one for processing and collecting data, one for transmitting or receiving data.

Line 6 tells the operating system which process to start on startup of the sensor node.

Lines 8-10 are the head of a function and line 14 defines the end of the process.

Line 12 is where the magic happens and the String gets to the console via the `printf()` command. Most of the Contiki commands rely on standard-C.

2.4 The Make Files

In order to run the code on the sensor node we need make files telling the compiler how to compile the programs for which specific platform. In this case we need two make files, one for telling the compiler where the Contiki headers can be found, and one defining the platform the code.

The following is the make file: "Makefile"

```
1 CONTIKI_PROJECT = hello-world
2 all: $(CONTIKI_PROJECT)
3
4 #UIP_CONF_IPV6=1
5
6 CONTIKI = ../..
7 include $(CONTIKI)/Makefile.include
```

The following is the make target: "Makefile.target" (optional)

```
1 TARGET=xm1000
```

Note that if you do not have "Makefile.target", you may provide the TARGET in the command line during compilation.

2.5 Testing on XM1000 (skip this section if you cannot collect a mote)

You can find "Hello World" program under `"/home/user/contiki-2.6/surrey/S2"`. You need the following files to compile:

- hello-world.c
- Makefile
- Makefile.target (optional)

Execute the following commands to compile, deploy and run the program.

```
$ sudo make TARGET=xm1000 hello-world.upload
```

```
$ sudo make TARGET=xm1000 login
```

The first command compiles and uploads the program to the connected sensor node. The second command logs in to the console of the node. After the successful login, restart the sensor node by pressing the red reset button on it and see the Hello World program running.

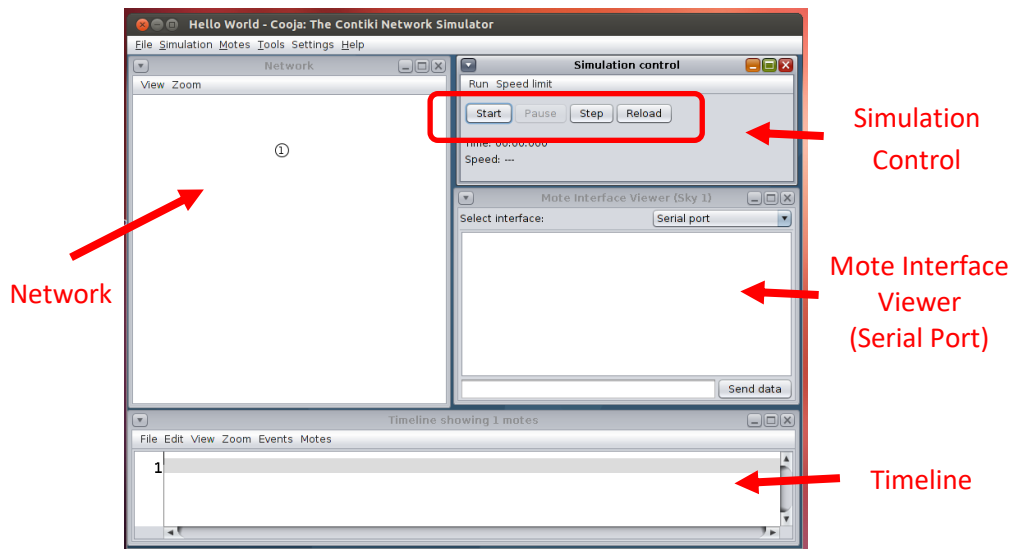
2.6 Testing in Cooja Simulator

You can find "Hello World" program under `"/home/user/contiki-2.6/surrey/S2"`. The source file is "hello-world.c".

You need to follow the steps below to load the simulation environment:

- Run Cooja by double-clicking Cooja icon on the desktop
- From the “File” menu, choose “Open simulation”, then “Browse...”, a dialog will pop-up
- On the dialog, click “Home” icon, you should see a list of folders showing up on the dialog
- From the list of folders, you should find “contiki-2.6”, double-click it to go into the folder
- After entering “contiki-2.6”, you should find “surrey”, double-click it to go into the folder
- After entering “surrey”, you should find “S2”, double-click it to go into the folder
- After entering “S2”, you should find “cooja_helloworld.csc”, double-click it to load the simulation environment. It may take a while to load the environment.

After loading the environment, you should see the following environment:



Do the following to run the simulation:

- Click the [Start] button on the “Simulation Control” window to run the mote. The “Mote Interface Viewer (Sky 1)” window will show the initialization message of the mote, followed by your message “Hello, world”.

Do the following to stop the simulation:

- Click the [Pause] button on the “Simulation Control” window to stop the simulation.

You can modify your source code to change the message or add features. After saving the changes, you need to click the [Reload] button on the “Simulation Control” window to reload and compile the code. If your code cannot be compiled successfully, the simulation will show you the error messages for you to correct the code. Otherwise, the simulation will return back to the normal environment.

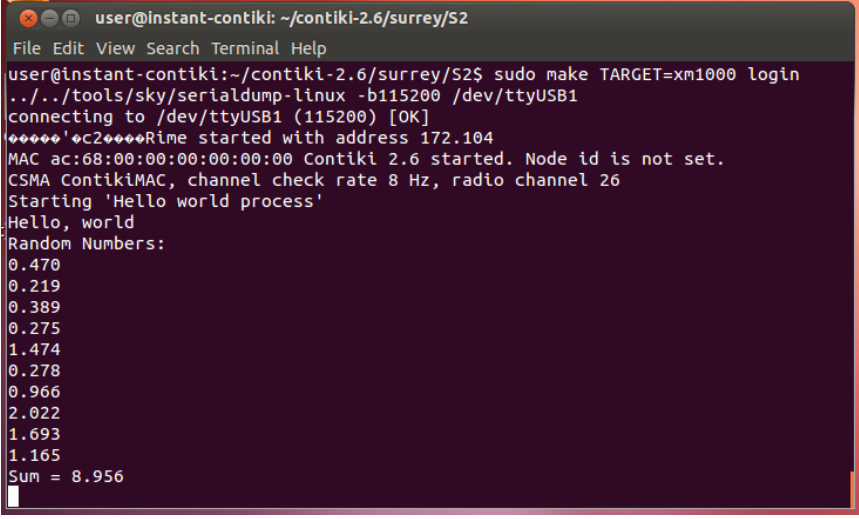
You may now run your modified code by repeating the above.

2.7 Exercise

Extend your code to print some random numbers between 0 and 2.5, and then display the sum of the random numbers. You may find two challenges:

- **Challenge1:** Standard C random number generator will not work in Contiki embedded library. Contiki has its own random generator. Your task is to find the function by going through its documentation at: <http://contiki.sourceforge.net/docs/2.6/index.html>.
- **Challenge2:** You will encounter problem printing floating point numbers. This is because Contiki library does not support printing of floating points. Your task is to implement a method that can print a floating point.

In the following, we show the screenshot of the intended outcome. The screenshot is taken from the terminal and the output is produced by XM1000 mote. If you use Cooja, the output will appear on “Mote Interface Viewer” panel.

A terminal window titled 'user@instant-contiki: ~/contiki-2.6/surrey/S2' with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows the execution of 'sudo make TARGET=xm1000 login' and subsequent output from the XM1000 mote. The output includes connection status, MAC address, node ID, CSMA parameters, and a list of random numbers with their sum.

```
user@instant-contiki: ~/contiki-2.6/surrey/S2$ sudo make TARGET=xm1000 login
../../tools/sky/seriaIdump-linux -b115200 /dev/ttyUSB1
connecting to /dev/ttyUSB1 (115200) [OK]
*****c2*****Rime started with address 172.104
MAC ac:68:00:00:00:00:00:00 Contiki 2.6 started. Node id is not set.
CSMA ContikiMAC, channel check rate 8 Hz, radio channel 26
Starting 'Hello world process'
Hello, world
Random Numbers:
0.470
0.219
0.389
0.275
1.474
0.278
0.966
2.022
1.693
1.165
Sum = 8.956
```