Code propagation and app-level communication

Department of Computer and Communication Engineering
Volos, Greece
Group 2
Project Engineers: Petros Kalos - Asimina Vouronikoy
Submitted for: wireless sensor networks programming

Abstract

In the fifth assignment of the wireless sensors network programming class it is implemented a basic virtual machine supports for the execution of portable application code. The VM environment support the dynamic loading and unloading of an application without requiring a system restart. The VM environment also supports the execution of multiple applications at the same time, without any undesirable interference between them (each application is given the illusion of being the only one running in the system). This VM environment is extended in order to produce a "complete" platform for applications that implement long-running queries with built-in support for returning results back to the source of the query (root).

Table of Contents

The instruction set

Code (4 msbits)	Mnemonic	1st argument (4 lsbits)	2nd argument (1 byte)	Description
0x0_	ret	(none) 0	(none)	ends handler execution
0x1_	set	<rx> 1-6	<val> -127<=val<=127	rx = val
0x2_	cpy	<rx> 1-6	<ry> 1-6	rx = ry
0x3_	add	<rx> 1-6	<ry> 1-6	rx = rx + ry
0x4_	sub	<rx> 1-6	<ry> 1-6	rx = rx – ry
0x5_	inc	<rx> 1-6	(none)	rx = rx + 1
0x6_	dec	<rx> 1-6	(none)	rx = rx – 1
0x7_	max	<rx> 1-6	<ry> 1-6	rx = max(rx,ry)
0x8_	min	<rx> 1-6	<ry> 1-6	rx = min(rx,ry)
0x9_	bgz	<rx> 1-6	<off> -127<=off<=127	if (rx > 0) pc = pc + off
0xA_	bez	<rx> 1-6	<off> -127<=off<=127	if (rx == 0) pc = pc + off
0xB_	bra	(none) 0	<off> -127<=off<=127	pc = pc + off
0xC_	led	<val> 0-1	(none)	if (val != 0) turn led on else turn led off
0xD_	rdb	<rx> 1-6	(none)	rx = current brightness value
0xE_	tmr	<mode> 0-1	<val>	set timer to expire after val seconds (=0 cancels the timer); mode==0 for normal, mode==1 for aggregation
0xF_	snd	0 (send only r7), 1 (send r7 and r8)	none	send contents of r7-r8 towards the application sink; at the root, this instruction should send the message over serial to the PC

Each application has a set of 6 1-byte long general-purpose registers *r1-r6*.
Registers *r7-r8* and *r9-r10*, are used for storing the 2-byte payload of outgoing and incoming application-level result messages.

Implementation

The message payload structure goes as follows:

```
nx_struct radio_msg {
        nx_uint8_t      group_id;
        nx_uint8_t      parent_id;
        nx_uint8_t      msg_id;
        nx_uint8_t      sender_id;
        nx_uint8_t      hop;
        nx_uint8_t      flag;
        nx_uint8_t      sampler_id;
        nx_uint16_t     data_cnt;
        nx_uint16_t     data[2];
        vm_msg          app;

};
```

Four new fields were added in relation to the previous implementations:

1. The field **hop**, represents the hops from the source node (over how many links did the message "travel")
2. The field **app** is a struct as follows :

```
        nx_struct vm_msg {
           nx_uint8_t app_id;
           nx_uint8_t comm [MAXAPPSIZE];
        };
```

Where the fields represent the application's id and set of instructions.

3. The field **data [2]** is used to hold the data of the instruction snd*

4. The field **data_cnt**: If the first argument of the *instruction snd* is equal to 0 only r7 is sent and data_cnt is set to 1. If is equal to 1 both r7 and r8 are send and data_cnt is set to 2.

We represent the application with the following structure:

```
struct state {

        nx_uint8_t app_id;                       //application id

        unsigned char comm[MAXAPPSIZE];          //commands set

        char r [MAXREG];                         //registers

        unsigned char pc;                        //program counter

        unsigned char init_len;                  // init handler length

        unsigned char timer_len;                 //timer handler length

        unsigned char msg_len;                   //message handler length

        unsigned char state;                      //idle=2,active=1,halt=0

        unsigned char sensor;                     //if app has rdb instruction

        bool aggr_active;                        //if app’s timer mode is set for aggregation

        int aggr_delay;                          //aggregation delay

        char  msg_counter;                       //messages received from other applications

};
```

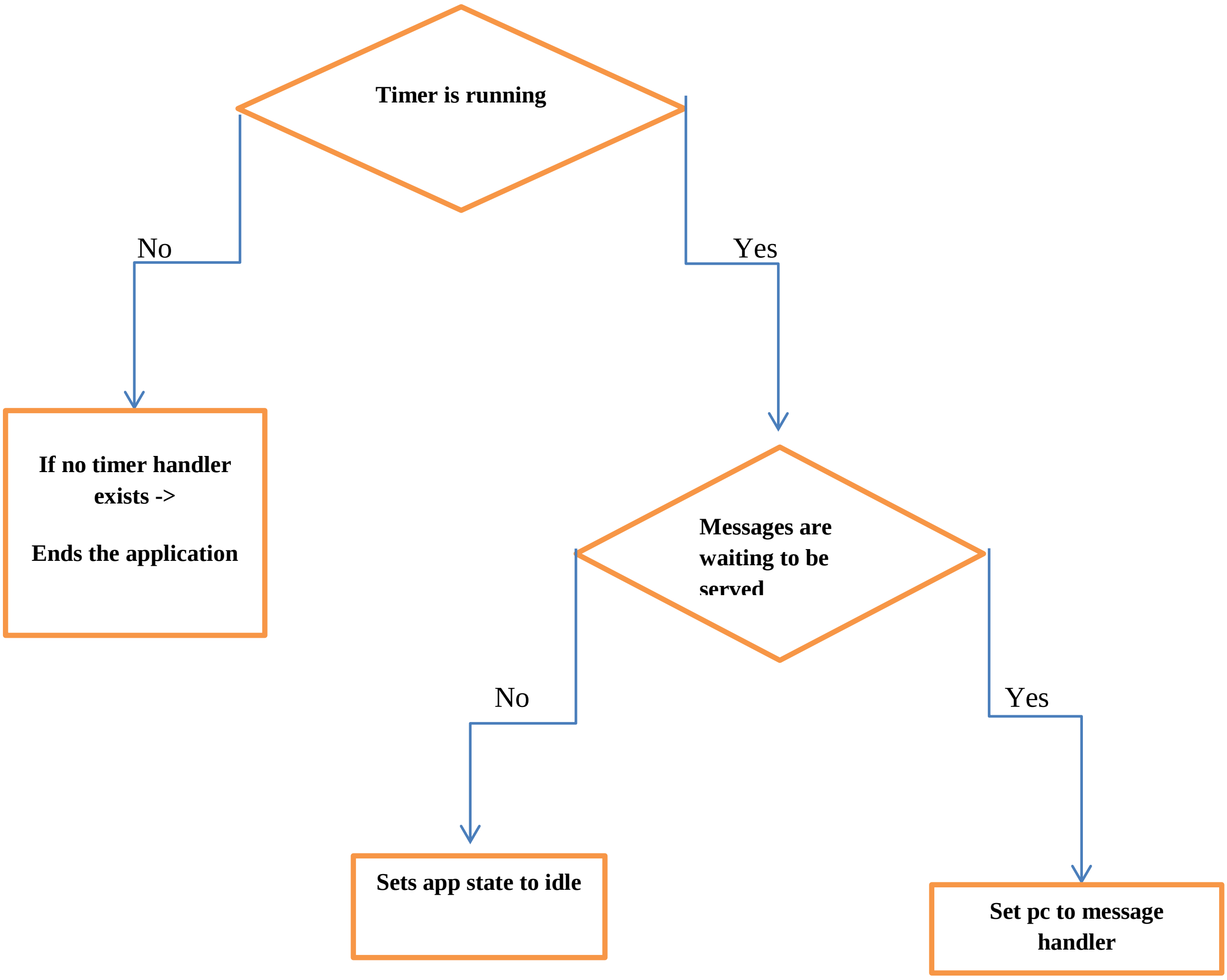
The implementation goes as follows:

When an application is sent via serial, the node connected to serial starts the application’s execution and propagates the app message to the network. When more than one application is running each node processes them round robin.

When a message is received, if it is an application propagation message, the node starts the application’s execution and propagates the message along network. Else, if it is a response message, if the node does not have message handler it simply forwards the message. Otherwise, the message data are copied to the registers of the application (to which was the message for) for handling. Exceptionally, priority is given to the application that received the message.If the node is the application source it sends the message data to serial.

Each instruction disassembles. Is worth to mention the cases of three instructions:

ret : In this instruction which ends handler execution we do as follows :


Timer is running
No
Yes
If no timer handler exists ->
Ends the application
Messages are waiting to be served
No
Yes
Sets app state to idle
Set pc to message handler

tmr: In this instruction if application’s aggr_active field is set to normal mode ,timer is set to expire after val seconds. Else if application’s aggr_active field is set to aggregation mode timer is set to expire after (val +number of nodes – hops) seconds.

rdb: In this instruction if during the reading of the sensor another application requests to read the sensor, in both is returned the same result. Read results downshifted from 16 bits to 7 bits(0-127) because registers are signed characters.

Measurements

The topology used is the same in assignment 3.Results are not presented because are the same with assignment 3.

Real Tests

Leds are used only for applications’ instruction use and not for demonstration purposes.