Laboratory Project
Real-Time Systems
IST - 2023/2024

Monitoring and Alarm System (Part 2 – FreeRTOS)

1 Introduction

Many real-time embedded systems are developed using relatively simple platforms, with low resources, where the utilization of operating systems to support the application may not be viable. However, in several other cases, the existence of support at operating system level (even if with reduced functionality) may significantly facilitate the development of applications.

This part of the laboratory project has as main goal the familiarization of students with the use of multitasking kernels for the development of concurrent applications in real-time embedded systems. In particular, they should acquire some expertise in the utilization of communication and synchronization mechanisms between tasks, in the context of concurrent applications. The multitasking kernel (operating system) to use will be FreeRTOS [1, 2, 3], working on a "Mbed platform" (ARM-CortexM3 based) [4, 5, 6].

2 Problem description

The second part of the project corresponds to a new version of a "monitoring and alarm system" using a new environment with more resources, making it possible to have a concurrent application with several tasks supported by a multitasking kernel. Those tasks will interact among them, and will provide a more flexible user interface.

This part of the project is implemented using the development board **"mbed LPC1768"** [5], which includes a microcontroller **NXP LPC1768** (ARM-CortexM3 based), and can be connected to a "mbed Application Board" [6], which includes several devices. The application is programmed using the **C/C++ programming language**, and the mbed online development environment (Mbed Online Compiler / Keil Studio Cloud) [4]. Related documentation is available on the online site.

The main functions to be provided by this part of the project, corresponding to several different tasks, are:

- Sensor monitoring service (working in a periodic way and on demand, and with the capability of storing collected data and with the possibility of handling alarm situations).
- Clock service (to provide user meaningful time (wall clock) and to allow clock alarm management).
- Processing of information.
- Device actuation (to be handled by a resource manager task or in a distributed way).

- User interface (console).

The clock service is built based on the FreeRTOS system time, and must provide the current time in the form of hours, minutes and seconds. It should also be able to handle alarms associated with the clock.

The sensor task will monitor, both in a periodic fashion (period `pmon`) or on demand, the value of temperature (sensor LM75B, available on the application board), and "luminosity" (emulated using one of the potentiometers (pot1) also available on the application board, and converting the obtained value into 4 different levels {0,...,3}). If `pmon` is zero there will be no periodic information collected, but should still provide that service on demand.

The collected values are saved in a ring-buffer (global data structure accessible by all tasks). They are saved as a record that, in addition to the values obtained from the sensors, also contains a timestamp with the current time.

When alarms are enabled, they are generated when the specified clock alarm time is reached or when the new value of temperature or luminosity is **above** the defined threshold. The notification of an alarm situation is done through signalization on the LCD (letters CTL, respectively) **and** generation of a sound (using a buzzer available on the application board). The sound will have a duration of `tala`.

3 User interface

The user interface, in this part of the project, is performed using both an LCD, and through the use of a console. (When processing information it is also possible to use mbed LEDs and an RGB LED.)

3.1 LCD

The information presented on the LCD should be something similar to the following:

hh:*mm*:*ss*	`CTL A`
tt `C`	`L` *l*

The current values of clock ("`hh:mm:ss`"), temperature ("`tt`") and luminosity level ("`l`") are presented. Letter "`A`" (or "`a`") will indicate if alarms are enabled (or disabled). Letters "`CTL`" will appear only in the case where the respective alarm (clock, temperature or luminosity) has occurred and they are kept until cleared by the user.

Range values:

hh - hours [00 .. 23]

mm - minutes [00 .. 59]

ss - seconds [00 .. 59]

tt - temperature [00 .. 50]

l - luminosity level [0 .. 3]

a - alarm enabled/disabled [A,a]

3.2 Console

In this part of the project there is a task responsible for a user interface using a console, that makes it possible to execute a set of commands to interact with the system. The commands that should be made available are the following:

Available Commands		
cmd	args	description
rc		- read clock
sc	*h m s*	- set clock
rtl		- read temperature and luminosity
rp		- read parameters (pmon, tala, pproc)
mmp	*p*	- modify monitoring period (seconds - 0 deactivate)
mta	*t*	- modify time alarm (seconds)
mpp	*p*	- modify processing period (seconds - 0 deactivate)
rai		- read alarm info (clock, temperature, luminosity, active/inactive-A/a)
dac	*h m s*	- define alarm clock
dtl	*T L*	- define alarm temperature and luminosity
aa	*a*	- activate/deactivate alarms (A/a)
cai		- clear alarm info (letters CTL in LCD)
ir		- information about records (NR, nr, wi, ri)
lr	*n i*	- list *n* records from index *i* (0 - oldest)
dr		- delete records
pr	[*h*1 *m*1 *s*1 [*h*2 *m*2 *s*2]]	- process records (max, min, mean) between instants t1 and t2 (h,m,s)

In the first group of commands, some may be executed directly by the user task, and others through the interaction with specific tasks. In the last command (`pr`) the interaction is done with the processing task. The commands in the central part, related to record information (`ir`, `lr`, and `dr`), corresponding, respectively, to information, listing and deleting of records that are in the shared memory, are executed accessing directly that memory region (shared by the several tasks). This memory region is organized in the form of a ring-buffer, with capacity to `NR=20` records. Record update in this memory region is done by the sensor task, when sensor values are collected.

All the commands specified above that imply communication between the user task with other tasks have a reply message (synchronous interface).

In the command to list records from the shared memory, `n` is the number of records to list, obtained starting at the position defined by index `i`. Index zero corresponds to the oldest record in the ring-buffer.

In the interaction with the processing task, the user can make requests to process a set of records belonging to a time interval defined by `t1` and `t2` (specified in the form `h m s`). If these time instants are missing, it corresponds to consider the totality of records (or from `t1` to the end, in the case where only `t2` is missing).

Meaning, and assumed range, of data fields:

h - hours [0 .. 23]

m - minutes [0 .. 59]

s - seconds [0 .. 59]

T - temperature [0 .. 50]

L - luminosity [0 .. 3]

p - monitoring period or processing period (in seconds) [0 .. 99] (0 - inactive)

t - time alarm (in seconds) [0 .. 60]

a - alarms active or inactive [A,a]

NR - maximum size of buffer

nr - number of valid records in buffer [0 .. NR]

wi - buffer write index [0 .. NR-1]

ri - buffer read index [0 .. NR-1]

n - number of records to list [0 .. NR]

i - buffer index [0 .. NR-1] (0 - oldest)

Initial values for `pmon`, `tala` and `pproc` are 3, 5 and 0 seconds, respectively.

4 Processing of collected information

The processing task can work in a periodic fashion (period `pproc`) to read **new** records from the ring-buffer (one at a time, using the read index **ri**) and represent the values of temperature and luminosity using the RGB LED (blue: low temperature; red: high temperature) and the 4 mbed LEDs (number of LEDs on corresponding to the 4 levels of luminosity), respectively. (If `pproc` is zero this functionality is not active.)

The processing task also accepts requests to process a set of records that belong to the time interval defined by **t1** and **t2** (h1:m1:s1 - h2:m2:s2). It will determine maximum, minimum and mean values. This processing is done using the records that are in the shared memory region. The result of this processing is sent back to user interface task that is responsible to present it on the console.

As stated above, the collected information (records) should be kept in a shared memory region accessible by the sensor task and by the processing and user interface tasks. The consistent access to that memory region by the several tasks is of extreme importance for the correct operation of the application.

5 Project development

This second part of the project is also programmed using the C programming language (and possibly some C++). The development environment is the “Mbed Online Compiler”. However, the operating system that will support the application is FreeRTOS, and the final application, containing FreeRTOS, and possibly some mbed libraries, is transferred to the target board (mbed) that will be reinitialized with this application/operating system.

As was suggested for the first part of the project, in the development of the second part, the utilization of a modular structure with phased tests is also advised.

In what concerns the user interface (console related), we do **not** want to have anything too much complex (that is not the main goal). In order to simplify the implementation, students **should** use a simple command interpreter that is made available by faculty.

The characteristics of the IO devices that are used should be consulted on the manual of the board, and/or on the “Data Sheets” of the devices (this information can be obtained from the online site).

6 Project delivery

This part of the project should be delivered by $\boxed{\textbf{07/01/2024}}$. The delivery consists of a digital copy (ZIP file) containing all developed programs and a small report (3 or 4 pages) explaining the main requirements and solutions related to real-time aspects and communication/synchronization aspects. In addition to source files, the ZIP file should include the final executable file. All these elements should be correctly identified (group number and students).

Project demonstrations, based on the delivered material, and **oral exams** will be done on the week starting on **08/01/2024**.

References

[1] *FreeRTOS: Real-time operating system for microcontrollers.* 2022. URL: http://www.freertos.org.

[2] Amazon Web Services. *The FreeRTOS Reference Manual.* 2017.

[3] Richard Barry. *Mastering the FreeRTOS Real Time Kernel – A Hands-On Tutorial Guide.* Real Time Engineers Ltd. 2016.

[4] Mbed. *Mbed Online Compiler / Keil Studio Cloud.* 2022. URL: https://mbed.org.

[5] Mbed. *mbed LPC1768.* 2022. URL: https://os.mbed.com/platforms/mbed-LPC1768.

[6] Mbed. *mbed Application Board. Application board for mbed NXP LPC1768.* 2022. URL: https://os.mbed.com/components/mbed-Application-Board.