Politecnico di Torino

Operating Systems for Embedded Systems
02NPSOV

Master's Degree in Computer Engineering

Smart bicameral pacemaker
for diagnostic and clinical monitoring

Candidates:
Fabio Quazzolo (s290806)

Referents:
Stefano Di Carlo

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

The objective of this report is to describe the development of a Real Time Operating System (from now on referred as RTOS) for a smart bicameral pacemaker.
In the following chapters the reader will understand better the hardware chosen, the RT-Thread operating system and the main tasks that the device must perform within strict deadlines.
Since safety and reliability of the device plays a key role in the development of this, great emphasis has been given to the execution of tasks, their periods, their priorities, their WCETs (worst case execution times) and most importantly to the scheduling algorithm performed.

The device under analysis is composed of:

- 2 sensors for the electrical potential measurements of the atrium cells
- 1 heartbeats sensor measuring the HB/min of the patient
- 1 external sensor (a button) for the user to send data request to the device
- 1 actuator for triggering the electric impulses in case of anomaly
- 1 external interface (a display) to provide the user the information needed

The report is organized as follows:

1. RT-Thread OS
2. The Development Board
3. The Scheduling Algorithm
4. Threads
5. Emulation

In *RT-Thread OS* and *The Development Board* a brief introduction of the rtos and hardware chosen is given to the reader, focusing on the main features, resources and components.
In *The Scheduling Algorithm* the scheduling policy will be explained, through timing diagrams the reader will discover how the scheduler schedules the right task at the right moment to send in execution.
In *Threads* each thread implementing a specific task is explained in timing an functional terms.
In *Emulation* an example of emulation of the device is provided and explained through screenshot taken directly from the console when the os is running.

CHAPTER 2

RT-Thread OS

RT-Thread is an open source, easy to use, and community-based RTOS designed especially for the development of embedded systems or IoT devices.
Among the most important features to mention:

- Lightweight: is designed to be light in terms of both memory footprint and processing power (crucial for embedded systems with limited resources)
- Extensibility: it's very easy to add additional module and components through an easy-to-use graphical configuration interface
- Portability: can be easily portable to different hardware platforms and it supports the POSIX standards so also applications are easy to develop.

Besides it can be developed using a specific IDE called RT-Thread Studio that supports emulation of boards and MCUs by means of QEMU, so it what the right choice since not having a physical board like in this project.

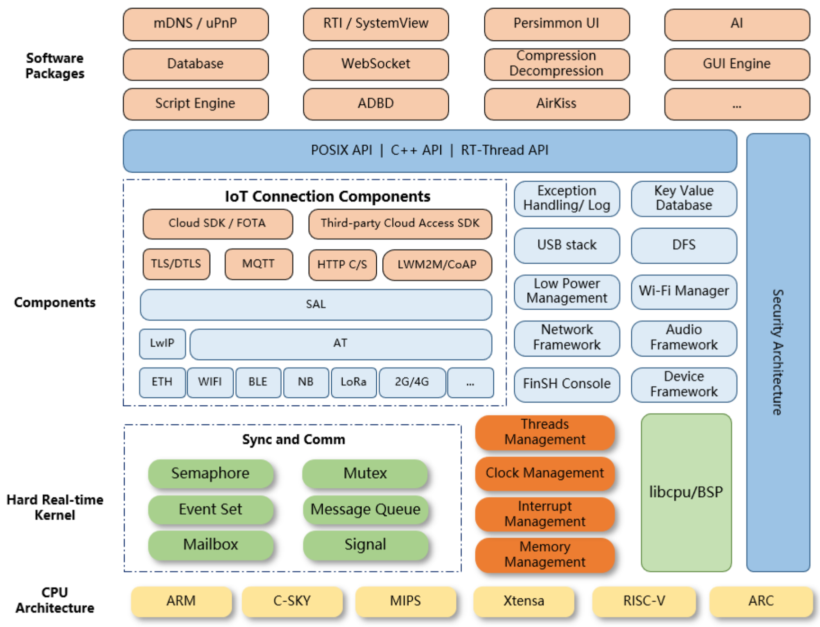


Figure 2.1: RT-Architecture

CHAPTER 3

The Development Board

The chosen board is the *Explorer STM32F407* a development board based on the ARM Cortex-M4 core launched by ATK.

- MCU: STM32F407ZGT6, 168MHz, 1024KB FLASH , 192KB RAM
- External RAM: IS62WV51216(1MB)
- External FLASH: W25Q128(SPI, 16MB)
- 2 LEDs: DS0(Red, PB1), DS1(Green, PB0)
- Buttons: KEY-UP(Wake-up, PIN0), K0(PIN68), K1(PIN67), K2(PIN66)
- Interface: USB-Serial, SD Card, Ethernet, LCD
- Debug: JTAG/SWD

Figure 3.1: The development board

The decision to use this particular board was mainly dictated by two factors:

1. in terms of resources (memory) and computational power it was more than sufficient for the development of our application and management of our tasks
2. easily supported by RT-thread studio and emulated through QEMU

CHAPTER 4

The Scheduling Algorithm

For what concerns the scheduling policy implemented by the scheduler, the algorithm chosen is the *priority-based full preemptive*.
So the scheduler has to manage different threads with different priorities and will always look to schedule the highest-priority *and* ready thread available at the given time. (NB: these 2 conditions need to be in AND logic).
"Preemptive" implies that any process can be preempted by one with a higher priority, in our case for process we intend a thread in execution.
In case of ready threads with *same priorities*, these are scheduled with a *time-slice rotation* scheduling. Every thread when is created is initialized with an integer number representing the time-slice, this number will be decreased during the execution. When the 0 value is reached the scheduler will then schedule the other same priority task, in a rotation fashion.
Every time a thread is scheduled or preempted its state changes accordingly as well as its "context": this process is called *Context Switch* and can be summarized in the image below.
When no other ready thread exists in the system, the scheduler will schedule the idle thread, which is usually an infinite loop and can never be suspended.
Timing diagrams for our specific application will be explained in the *Emulation* chapter.

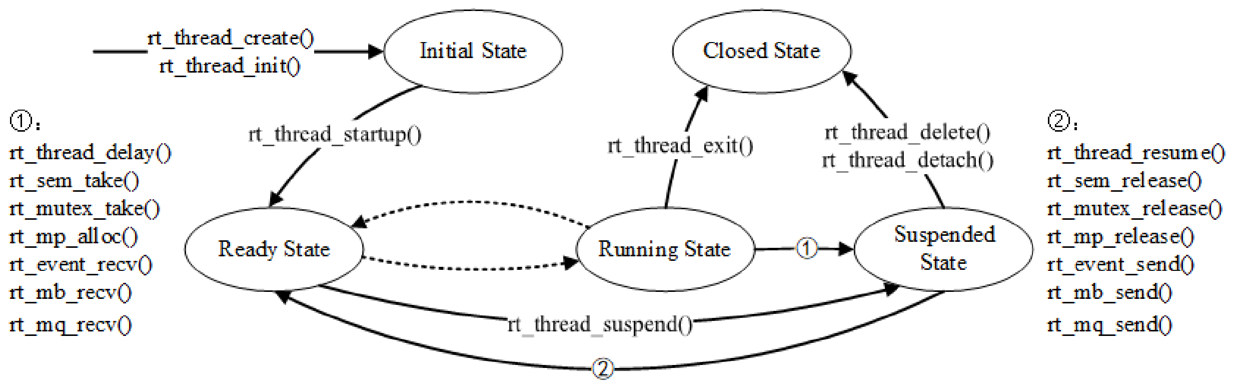


Figure 4.1: Threads States

CHAPTER 5

Threads

Threads are processes in execution which have the function of performing and achieving a specific task that the device implements.
Specifically, our pacemaker must perform 5 different functions that we have already introduced previously and therefore requires as many threads. All these threads were created *dynamically* through the function

```
rt_thread_t rt_thread_create(const char* name, void (*entry)(void* parameter),
                               void* parameter, rt_uint32_t stack_size,
                               rt_uint8_t priority, rt_uint32_t tick);
```

which allocates dynamically the needed space for the thread from the heap memory.
All threads are created the same way but may differ from each other both in terms of functionality and scheduling times.

Hard Tasks	**Soft Tasks**
TaskA TaskB TaskD	TaskC TaskE

Table 5.1: Hard vs Soft Threads

In our application the above mentioned *hard threads* have strict deadlines that *must* be met otherwise the patient is in possible danger.*Soft tasks* on the other hand have deadlines too, but missing one of these does not necessarily lead to system failure.
The other difference regards the *Periodic vs. Aperiodic* feature. In our system we have some threads that are periodic and so need to be executed at regular intervals defined by their periods. Aperiodic tasks are present too and can be triggered by some external event at any given time, so asynchronously (like a User requesting information to the device).

Periodic Tasks	**Aperiodic Tasks**
TaskA TaskB TaskC	TaskD TaskE

Table 5.2: Periodic vs Aperiodic Threads

All hard tasks have been given a high priority and smaller periods so that they are performed consistently at short intervals of time and they can preempt other tasks if needed.

In the following subchapters all threads will be deeply explained.
As a starting point to develop the application these periods have been taken as specifications: To

Thread	**T[ms]**	**WCET[ms]**	**Priority**
TaskA	100	20	7
TaskB	200	50	8
TaskC	1000	50	9
TaskD	100	20	1
TaskE	1000	50	8

Table 5.3: Original Specifications

evaluate and verify that the WCETs are not exceeded, within each thread these functions were called and the following calculations performed:

```
/*At the start of the ThreadX execution*/
        tick_start_task = rt_tick_get_millisecond();
/*Computing the execution time of the task */
        tick_end_task = rt_tick_get_millisecond();
        exec_time = tick_end_task - tick_start_task;
        rt_kprintf("Task X execution time is: %d \n ", exec_time);

/*Checking WCET constraint */
        if(exec_time > WCET_b){
            rt_kprintf("Attention WCET of task X exceeded of  %d ms \n ", exec_time-WCET_X);
        }
```

To make threads periodic the event mechanism was used: for each periodic task an event and a specific timer were created as shown below.

```
 /*Creation of event */
        threadA_event = rt_event_create("TaskAEvent", RT_IPC_FLAG_PRIO);

/* Creation of timer  */
        timer_threadA = rt_timer_create("TaskATimer", timer_threadA_callback,
        RT_NULL, Ta, RT_TIMER_FLAG_PERIODIC);

/* Startup of Thread */
        rt_thread_startup(thread_A);
        rt_timer_start(timer_threadA);
```

When a thread is started, its timer is also started and when this one expires the corresponding callback function is executed and an event is sent to the thread that needs to be executed again

```
/* Callback Funtion */
        static void timer_threadA_callback(void *parameter){
            rt_event_send(threadA_event, TASK_A_TRIGGER_EVENT1);
        }
-----------------------------------------------------------------
```

```
/* Thread A periodically scheduled */
    static void thread_A_entry(void *param){
         while(1){
        /*Every Ta[ms] (period) the task must be scheduled */
            rt_event_recv(threadA_event, TASK_A_TRIGGER_EVENT1, RT_EVENT_FLAG_AND |
            RT_EVENT_FLAG_CLEAR, RT_WAITING_FOREVER, RT_NULL);
                ...
        }
```

5.1 Thread A

This thread has the function of handling the sensor related to the sensing of the electrical potential of the right and left atrium cells of the heart. Every Ta[ms] the task is scheduled, and the electrical potential values are read through two designated functions

```
int read_sensor_sx(void);
int read_sensor_dx(void);
```

At this point the task should check whether or not these values from the sensors are 'normal' or they are showing some problem with the patient. This has been implemented by comparing the average of the 2 values just read with the last 10 average values stored in a circular buffer. If the delta between the 2 compared values is too large, it means that the patient is in danger and a stimulus impulse must be sent to the heart. To perform this action Thread A dynamically creates Thread D assigning to it maximum priority and does the startup.

5.1.1 Thread A sensing

Since no physical sensors are attached to the boards, the electrical potential values have been generated by means of a random function that produces integer numbers within two specific intervals:

- Interval 1 [0-100]: bad values that leads to danger situations can be produced
- Interval 2 [45-65]: normal values produced

The decision of which interval to use for the generation of electrical potential depends on a global flag named *flag sens*. If this is set to 1 means that Thread D has been recently triggered and so is likely that following generated values are "normal" ones and are not potentially dangerous. After some time this flag is set back to 0 automatically.

5.2 Thread B

This task has the function of sensing the heartbeats for min. It has been considered as an hard task has a high priority.
The sensing part was implemented similarly to Thread A.

5.3 Thread C

This thread is considered as soft task because all it has to do is to check information coming from an external source like the press of the button. The patient can ask any time the device to provide him the recent history and data of his heart most important parameters. After Tc[ms] the task is scheduled but if the Signal *SIGUSR1* has not been launched the task is not executed.

5.3.1 Thread C: emulating the user request

To emulate an asynchronous request from the user *signals* have been used, here an example:

```
/* Send signal SIGUSR1 after 10 seconds to thread C */
        rt_thread_mdelay(10000);
        rt_thread_kill(thread_C, SIGUSR1);
-------------------------------------------------------
/* Thread C receiving the signal*/
        // link SIGUSR1 with thread E execution
        rt_signal_install(SIGUSR1, threadE_signal_handler);
        rt_signal_unmask(SIGUSR1);
-------------------------------------------------------
/* Calling the handler of the signal*/
     void threadE_signal_handler(int sig){
        rt_kprintf("Task C running: receive signal %d from the User
        Launching thread E\n", sig);
        /* Create thread E */
        thread_E = rt_thread_create("threadE",thread_E_entry, RT_NULL, 1024, 8, 5);
        if (thread_E != RT_NULL){
               rt_thread_startup(thread_E);
        }
        rt_thread_delete(thread_E);
}
```

After the signal is sent and received by Thread C, the handler will be called and Thread E created and started as shown in the code above.

5.4 Thread D

This thread is the most important thread of the application and thus has the highest priority possible. When needed it may preempt all other running threads. Is created and started when Thread A detects an anomaly in the values just sensed. Every time is launched it stores in an array of struct, named *History*, all the important parameters detected in that precise moment so that the patient may consult them whenever he wants.

```
/*Definition of a new type: d_occurence*/
    typedef struct {
        int occurrence;
        int  at_time;
        int hr;
        int last_values[BUFFER_SIZE];
    } d_occurence;

/*Array of d_occurence elements*/
    d_occurence history[HISTORY_SIZE];
```

5.4.1 Thread D: emulating the actuation

To emulate the triggering of the leads the easiest way was to act directly on the generation of the electrical values of the atrium cells. When Thread D runs *"flag sens"* is set and the following generated values will be ”normal ones” meaning that the heart potential has been correctly restored to normal behavior. This flag after some execution cycles will be reset back to 0 automatically.

5.5 Thread E

The purpose of this task is simply to send the data to the patient that request them.This thread is created and launched when thread C receives the User signal. All the recent history of the anomalies will be sent and shown to the patient.

CHAPTER 6

Emulation

6.1 How to start an emulation

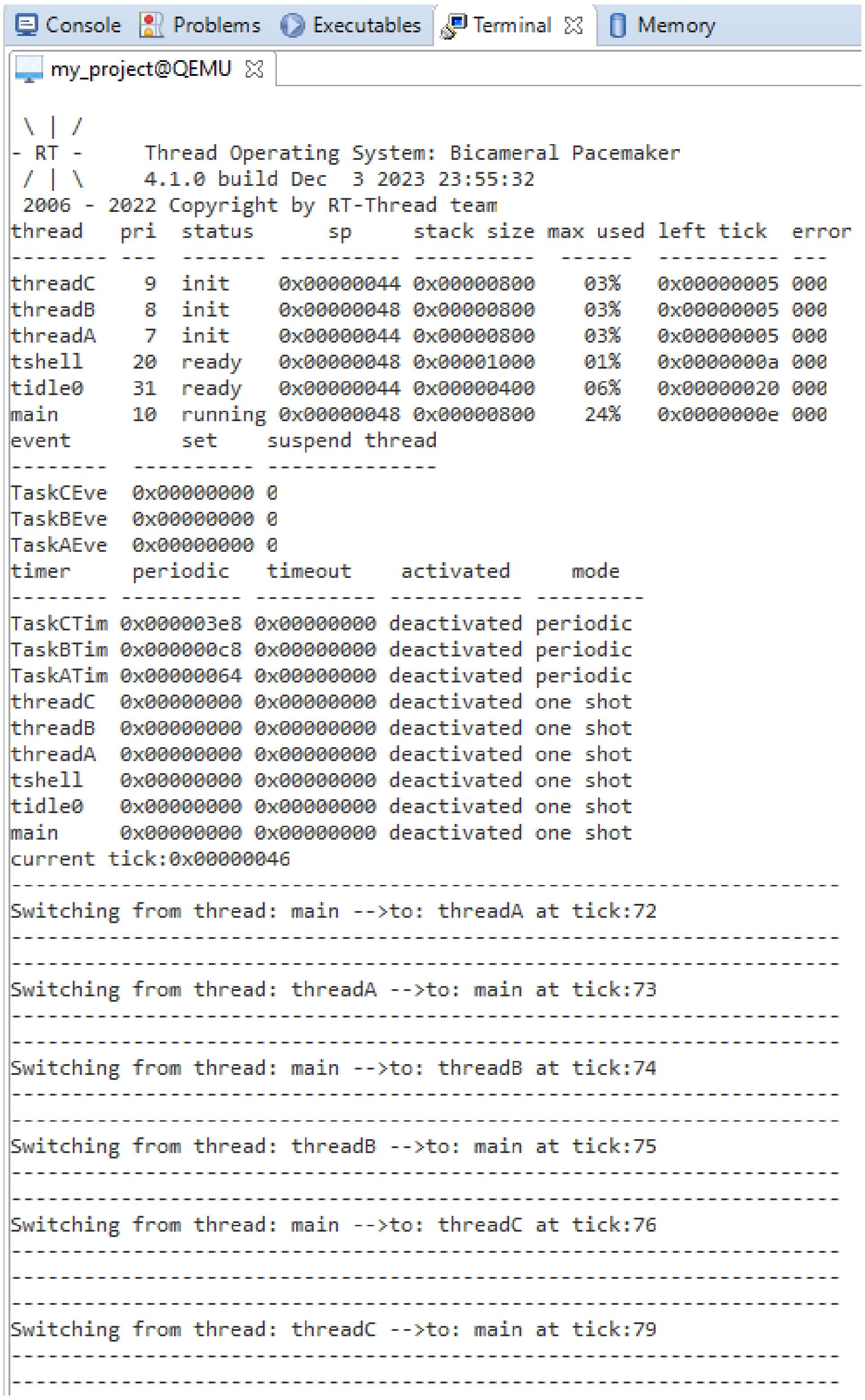
After building the project and selecting the board that we want to emulate, we can flash the *rtthread.elf* file and the RTOS starts running using QEMU.
The operations performed can be seen through the console. For a better understanding of the context switch between tasks and the checking of the correct behavior of the scheduling algorithm a scheduler hook has been instantiated

```
/*Definition of a new type: d_occurence*/
    rt_scheduler_sethook(hook_of_scheduler)

/*To understand when we switch from a thread to another */
static void hook_of_scheduler(struct rt_thread* from, struct rt_thread* to){
    rt_kprintf("Switching from thread: %s -->to: %s at tick:%d\n",
    from->name , to->name, rt_tick_get_millisecond());
}
```

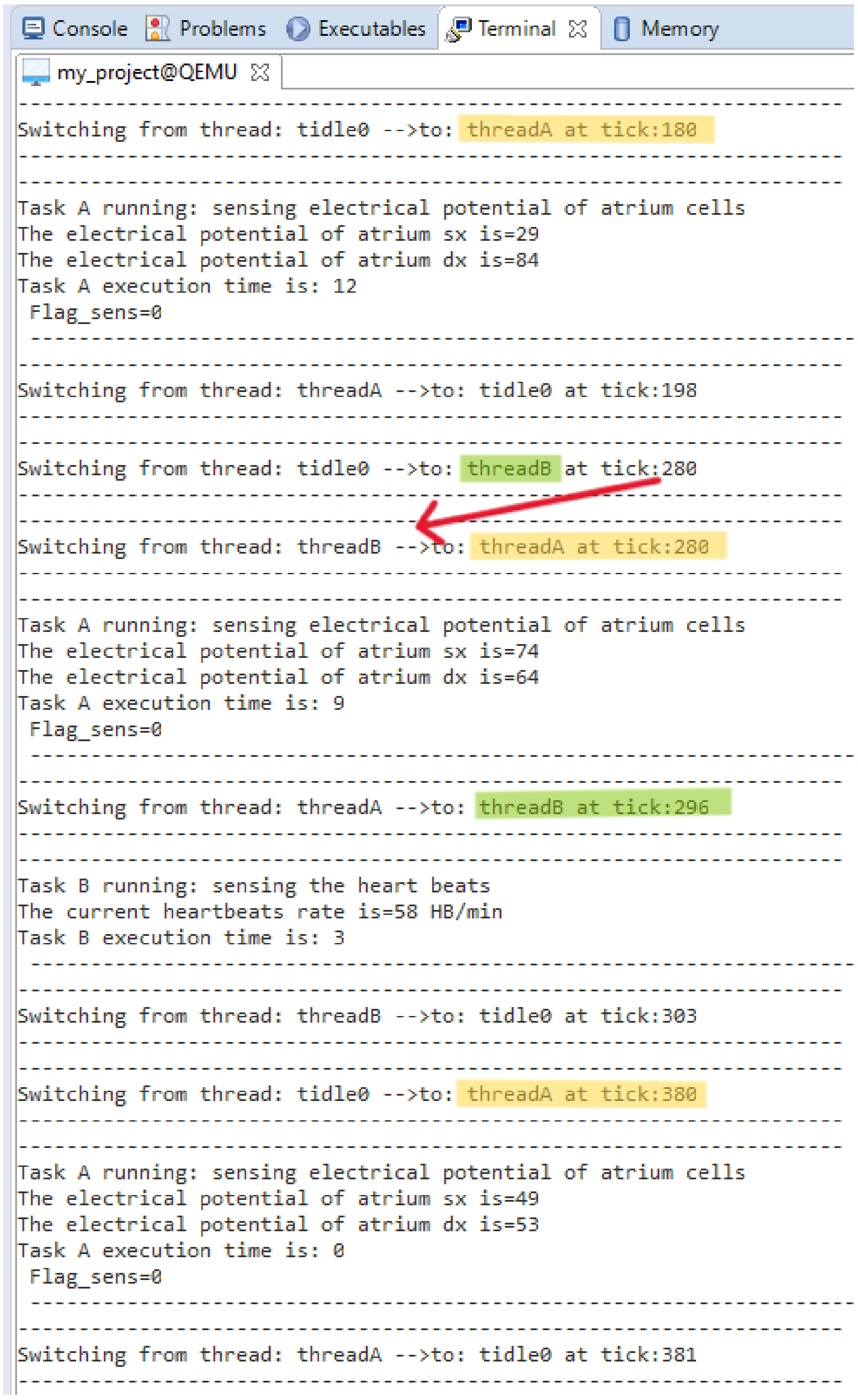
So every time a new thread is scheduled and a context switch takes place the systems will print it on the console.

6.2 Results



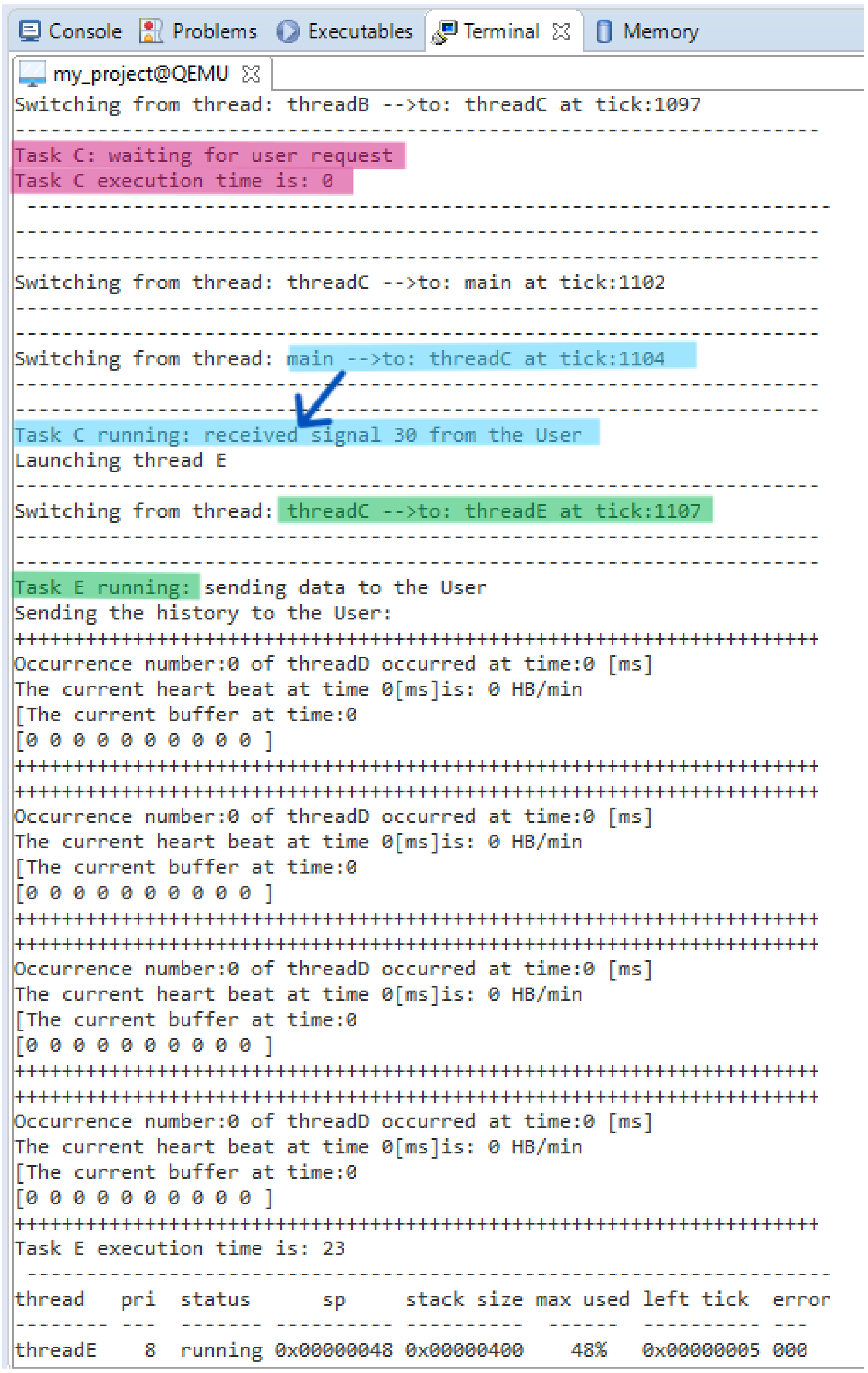
```
 \ | /
- RT -     Thread Operating System: Bicameral Pacemaker
 / | \     4.1.0 build Dec  3 2023 23:55:32
 2006 - 2022 Copyright by RT-Thread team
thread   pri  status      sp     stack size max used left tick  error
-------- ---  ------- ---------- ----------  ------  ---------- ---
threadC    9  init    0x00000044 0x00000800    03%   0x00000005 000
threadB    8  init    0x00000048 0x00000800    03%   0x00000005 000
threadA    7  init    0x00000044 0x00000800    03%   0x00000005 000
tshell    20  ready   0x00000048 0x00001000    01%   0x0000000a 000
tidle0    31  ready   0x00000044 0x00000400    06%   0x00000020 000
main      10  running 0x00000048 0x00000800    24%   0x0000000e 000
event         set    suspend thread
--------  ---------- -------------
TaskCEve  0x00000000 0
TaskBEve  0x00000000 0
TaskAEve  0x00000000 0
timer     periodic   timeout    activated     mode
-------- ---------- ---------- ----------- ---------
TaskCTim 0x000003e8 0x00000000 deactivated periodic
TaskBTim 0x000000c8 0x00000000 deactivated periodic
TaskATim 0x00000064 0x00000000 deactivated periodic
threadC  0x00000000 0x00000000 deactivated one shot
threadB  0x00000000 0x00000000 deactivated one shot
threadA  0x00000000 0x00000000 deactivated one shot
tshell   0x00000000 0x00000000 deactivated one shot
tidle0   0x00000000 0x00000000 deactivated one shot
main     0x00000000 0x00000000 deactivated one shot
current tick:0x00000046
--------------------------------------------------------------------
Switching from thread: main -->to: threadA at tick:72
--------------------------------------------------------------------
--------------------------------------------------------------------
Switching from thread: threadA -->to: main at tick:73
--------------------------------------------------------------------
--------------------------------------------------------------------
Switching from thread: main -->to: threadB at tick:74
--------------------------------------------------------------------
--------------------------------------------------------------------
Switching from thread: threadB -->to: main at tick:75
--------------------------------------------------------------------
--------------------------------------------------------------------
Switching from thread: main -->to: threadC at tick:76
--------------------------------------------------------------------
--------------------------------------------------------------------
--------------------------------------------------------------------
Switching from thread: threadC -->to: main at tick:79
--------------------------------------------------------------------
--------------------------------------------------------------------
```

Figure 6.1: Emulation just started



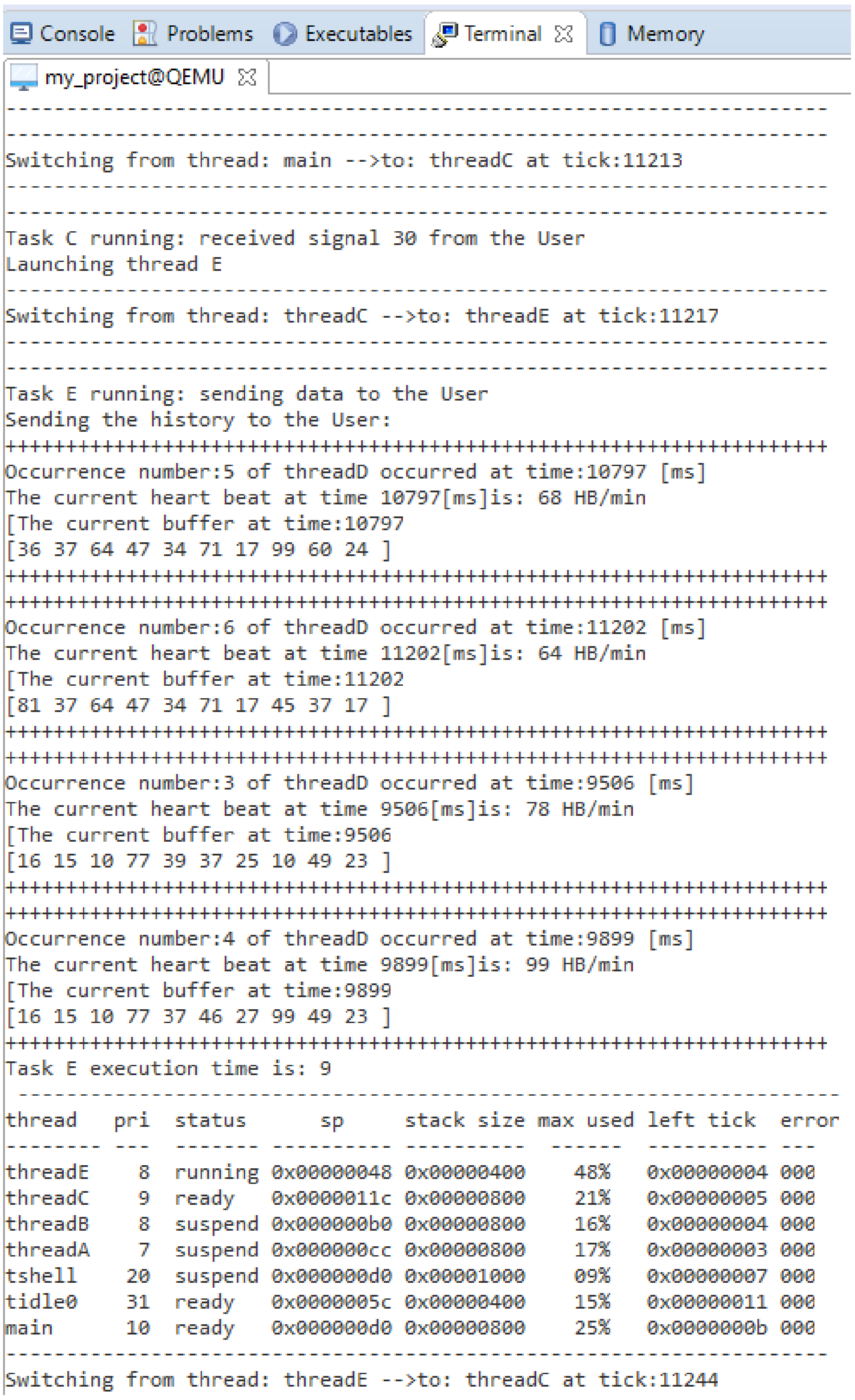
```
--------------------------------------------------------------------
Switching from thread: tidle0 -->to: threadA at tick:180
--------------------------------------------------------------------
--------------------------------------------------------------------
Task A running: sensing electrical potential of atrium cells
The electrical potential of atrium sx is=29
The electrical potential of atrium dx is=84
Task A execution time is: 12
 Flag_sens=0
 --------------------------------------------------------------------
--------------------------------------------------------------------
Switching from thread: threadA -->to: tidle0 at tick:198
--------------------------------------------------------------------
--------------------------------------------------------------------
Switching from thread: tidle0 -->to: threadB at tick:280
--------------------------------------------------------------------
--------------------------------------------------------------------
Switching from thread: threadB -->to: threadA at tick:280
--------------------------------------------------------------------
--------------------------------------------------------------------
Task A running: sensing electrical potential of atrium cells
The electrical potential of atrium sx is=74
The electrical potential of atrium dx is=64
Task A execution time is: 9
 Flag_sens=0
 --------------------------------------------------------------------
--------------------------------------------------------------------
Switching from thread: threadA -->to: threadB at tick:296
--------------------------------------------------------------------
--------------------------------------------------------------------
Task B running: sensing the heart beats
The current heartbeats rate is=58 HB/min
Task B execution time is: 3
 --------------------------------------------------------------------
--------------------------------------------------------------------
Switching from thread: threadB -->to: tidle0 at tick:303
--------------------------------------------------------------------
--------------------------------------------------------------------
Switching from thread: tidle0 -->to: threadA at tick:380
--------------------------------------------------------------------
--------------------------------------------------------------------
Task A running: sensing electrical potential of atrium cells
The electrical potential of atrium sx is=49
The electrical potential of atrium dx is=53
Task A execution time is: 0
 Flag_sens=0
 --------------------------------------------------------------------
--------------------------------------------------------------------
Switching from thread: threadA -->to: tidle0 at tick:381
--------------------------------------------------------------------
```

Figure 6.2: Thread B preempted by Thread A



```
Switching from thread: threadB -->to: threadC at tick:1097
---------------------------------------------------------------------
Task C: waiting for user request
Task C execution time is: 0
 ---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
Switching from thread: threadC -->to: main at tick:1102
---------------------------------------------------------------------
---------------------------------------------------------------------
Switching from thread: main -->to: threadC at tick:1104
---------------------------------------------------------------------
---------------------------------------------------------------------
Task C running: received signal 30 from the User
Launching thread E
---------------------------------------------------------------------
Switching from thread: threadC -->to: threadE at tick:1107
---------------------------------------------------------------------
---------------------------------------------------------------------
Task E running: sending data to the User
Sending the history to the User:
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Occurrence number:0 of threadD occurred at time:0 [ms]
The current heart beat at time 0[ms]is: 0 HB/min
[The current buffer at time:0
[0 0 0 0 0 0 0 0 0 0 ]
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Occurrence number:0 of threadD occurred at time:0 [ms]
The current heart beat at time 0[ms]is: 0 HB/min
[The current buffer at time:0
[0 0 0 0 0 0 0 0 0 0 ]
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Occurrence number:0 of threadD occurred at time:0 [ms]
The current heart beat at time 0[ms]is: 0 HB/min
[The current buffer at time:0
[0 0 0 0 0 0 0 0 0 0 ]
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Occurrence number:0 of threadD occurred at time:0 [ms]
The current heart beat at time 0[ms]is: 0 HB/min
[The current buffer at time:0
[0 0 0 0 0 0 0 0 0 0 ]
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Task E execution time is: 23
 ---------------------------------------------------------------------
thread   pri  status      sp     stack size max used left tick  error
-------- ---  ------- ---------- ----------  ------  ---------- ---
threadE    8  running 0x00000048 0x00000400    48%   0x00000005 000
```

Figure 6.3: Thread C when receiving and not receiving the SIGUSR1



```
--------------------------------------------------------------------
--------------------------------------------------------------------
Switching from thread: main -->to: threadC at tick:11213
--------------------------------------------------------------------
--------------------------------------------------------------------
Task C running: received signal 30 from the User
Launching thread E
--------------------------------------------------------------------
Switching from thread: threadC -->to: threadE at tick:11217
--------------------------------------------------------------------
--------------------------------------------------------------------
Task E running: sending data to the User
Sending the history to the User:
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Occurrence number:5 of threadD occurred at time:10797 [ms]
The current heart beat at time 10797[ms]is: 68 HB/min
[The current buffer at time:10797
[36 37 64 47 34 71 17 99 60 24 ]
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Occurrence number:6 of threadD occurred at time:11202 [ms]
The current heart beat at time 11202[ms]is: 64 HB/min
[The current buffer at time:11202
[81 37 64 47 34 71 17 45 37 17 ]
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Occurrence number:3 of threadD occurred at time:9506 [ms]
The current heart beat at time 9506[ms]is: 78 HB/min
[The current buffer at time:9506
[16 15 10 77 39 37 25 10 49 23 ]
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Occurrence number:4 of threadD occurred at time:9899 [ms]
The current heart beat at time 9899[ms]is: 99 HB/min
[The current buffer at time:9899
[16 15 10 77 37 46 27 99 49 23 ]
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Task E execution time is: 9
 --------------------------------------------------------------------
thread    pri  status      sp     stack size max used left tick  error
-------- ---  ------- ---------- ----------  ------  ---------- ---
threadE    8  running 0x00000048 0x00000400    48%   0x00000004 000
threadC    9  ready   0x0000011c 0x00000800    21%   0x00000005 000
threadB    8  suspend 0x000000b0 0x00000800    16%   0x00000004 000
threadA    7  suspend 0x000000cc 0x00000800    17%   0x00000003 000
tshell    20  suspend 0x000000d0 0x00001000    09%   0x00000007 000
tidle0    31  ready   0x0000005c 0x00000400    15%   0x00000011 000
main      10  ready   0x000000d0 0x00000800    25%   0x0000000b 000
--------------------------------------------------------------------
Switching from thread: threadE -->to: threadC at tick:11244
```

Figure 6.4: Thread E showing the history

CHAPTER 7

Open Issues

These are the issue that I was not able to solve:

- mounting an elm-FatFS and make it works
- using real emulated sensors