**Embedded Systems Fall 2020**

**Project 1 : Cooperative Scheduler on Renode**

Team members:

Ahmed Saleh ---- 900114441

Mariam Mohammed -----900172118

**Project Components:**

1. Keil uVision MDK
2. Renode Simulator
3. Visual Studio Complier(for Unit Test)

**Project Compilation Steps:**

1. Open the project with Keil uVision.
2. Build it.
3. Open Renode folder and open the single node scripts folder.

**<Renode\_Installation\_path>\scripts\single-node**

1. Open stm32f4\_discoveryf4.resc using a text editor.
2. Change the value after **$bin?=@** tobe the path of the axf file built.

**<keil\_Project\_Path>/Project1/Demo1/Objects/Demo1.axf**

For demo 1

**<keil\_Project\_Path>/Project1/Demo2/Objects/Demo2.axf**

For demo 2

1. Under the path make sure the showAnalyzer uses sysbus.uart2.

**showAnalyze sysbus.uart2**

1. Save the text file.
2. Open Renode.
3. In the monitor shell type

**s @scripts/single-node/stm32f4\_discovery.resc**

1. A UART window will open showing the tasks as they progress.
2. In the monitor shell if u wish to Press a button type

**sysbus.gpioPortA.UserButton Press**

1. To release the button also use the monitor shell and type

**sysbus.gpioPortA.UserButton Release**

**Project Overview:**

The objective of this short project is to develop a cooperative scheduler for embedded systems. This scheduler is similar to the function queue scheduler introduced in the lecture. The project is developed on Keil uVison IDE and run on the STM32F4\_discovery kit simulated using Renode. This project can be divided into four main parts:

**The first part** of the project contains different queues and global variables to handle the running and delayed tasks. Each queue element contains task information such as:

1. Task ID
2. Task Priority which has an integer value between 1 to 8, where 1 is the highest priority and 8 is the lowest.
3. Task pointer that points to the function to be executed when it is ready and based on its priority
4. Task delay ticks, where each tick is 100ms and the task will be waiting in the delayed queue as long as delay tick ≠ 0.

Three queues have been used in this project:

1. Tasks queue that contains information about each task. This information are retrieved when needed.
2. Delay queue that contains the delayed tasks based on the delay tick property of each task. Tasks are sorted in descending order in the delay queue based on their delay ticks and if two task has the same delay ticks they will be sorted based on their priorities. Tasks are being held in the delay queue as long as delay ticks ≠ 0. Otherwise it will be sent to ready queue.
3. Ready queue contains all ready task to be executed. Tasks are sorted in ascending order in the ready queue based on their priority.

This part contains global variables such as:

1. Running task ID that keep track to the ID of the running task
2. ReadyQueueTail\_id which holds the ID of the last task in the ready queue
3. DelayQueueHead\_id that contains the ID of the task that has the lowest dely ticks

**The second part** of the project contains the configuration functions for UART and GPIOs of the STM32F407\_Discovery board. Such configuration functions are constructed in Keil uVison IDE to run on the STM32F4\_discovery kit simulated using Renode emulator. This configuration was developed based on the given tutorial for Renode emulator.

**The third part** of this project contains the defined initialization function, scheduling functions and task to be executed in the system such as:

1. Init function that creates and initializes all needed queues and variables in the system.
2. QueTask function that insert ready tasks in the Tasks queue and ready queue. If the queues are empty, the coming task is saved in the first location in the queues. Otherwise, task are stored in the queue based on its priority. Tasks are added to task queues only once as it is used to hold task information only.
3. Dispatch function remove the highest priority task from the ready queue to execute it. This function set the RunningTask\_id to the ready queue head and shift all remaining task in the ready queue after removing the highest priority task. Then called RunTask function.
4. RunTask function is a Helper function that receive pointer to the highest ready task to execute it.
5. QueDelay function is used to is used to insert a task to the delay queue by passing a pointer to the function and delay value. Delayed tasks are stored in descending order based on the delay ticks. If the delay ticks is equal to 0, the task is sent to QueTask. Other wise will be saved in the delay queue. If two task has the same delay ticks they will be stored in the delay queue based on their priority.
6. DecrementDelay function is used to decrement the delay ticks of all tasks stroed in the delay queue every 100 ms. This function remove the task that has delay ticks equal to 0 and send them to QueTask function to save it in the ready queue.
7. ReRunMe function is used to re-run the current task after delayed ticks received as an argument. This function enqueue the running task to the ready queue if the received ticks equal to 0. Otherwise enqueue the running task in the delay queue.
8. GetTaskParameters function is used to retrieve task arguments if needed from task queue. It can retrieve task priority and ID if received the task pointer or retrieve task pointer and priority if receive the task ID. For example, this function is used in the ReRunMe function as we do not know any information about running task except the running task ID.

**In Demo 1**

1. Task1 function prints message on UART and toggle the value of a global variable called Switchon, then it enqueue Task2 to delay queue. Otherwise it rerun it self after certain delay ticks.
2. Task2 function prints message on UART and toggle the value of a global variable called Switchon, then it enqueue Task1 to delay queue. Otherwise it rerun it self after certain delay ticks..
3. Task3 function prints message only and then rerun it self after certain delay ticks.

**In Demo 2**

1. Task 1,2,3,4 have different behaviors depending on if the button is pressed or released. If it is pressed they queue other tasks, if it is released they rerun themselves after certain delay ticks.

**The last part is**, the main function that call initialization functions and call the QueTask function to enqueue tasks in the ready queue to be executed. Main function contains an infinite loop that calls the Dispatch function to run the ready tasks exist in the ready queue.

**Project Components:**

1. Keil uVision MDK
2. Renode Simulator

Note: The file Unit\_Test.cpp is used to test that each function is working properly by displaying message present what is happening right now. It keep tracks to the running function and delay queue and function that moves from delay queue to running queue.

I ran it using Visual studio compiler.