Computer Architectures and Operating Systems

HackOSsim

Francesco Laterza, 323518 Leonardo Rizzo, 328764 Massimo Aresca, 328743
Yann Freddy Dongue Dongmo, 322787

February 19, 2024

Abstract

The HackOSsim project consist in the exploration and improvement of FreeRTOS embedded OS within the context of QEMU, an ISA simulator.

Among the primary objectives of the project is the mastery of QEMU, a comprehensive tutorial is provided that outlines the installation and the usage procedures of the ISA simulator to emulate an ARM system. Moreover practical examples are developed to illustrate the functionalities of FreeRTOS.

The last part of the project lies in the development of a new functionality in the memory management of FreeRTOS and testing the correct functioning of the developed code.

Contents

1 QEMU

QEMU, short for Quick EMUlator, is an open-source emulator that allows users to run virtual machines and emulate various architectures. Originally designed for x86, it has evolved over the years to support a wide range of architectures, including ARM, MIPS, PowerPC, and more. QEMU enables developers to create virtual environments for testing, debugging, and development without the need for physical hardware.

1.1 The QEMU Packages

QEMU packages follow a consistent naming convention to reflect their purpose and target architecture. Typically, package names begin with `qemu` followed by the specific architecture or target type. For user-mode emulation, where QEMU runs specific applications on a different architecture, package names start with `qemu-<arch>` (e.g., `qemu-arm` for ARM architecture). On the other hand, for full system emulation, where QEMU simulates an entire computer system, the package names begin with `qemu-system-<arch>` (e.g., **`qemu-system-arm`** for ARM architecture).

Install QEMU QEMU can be installed on a Debian-based system using the following command:

```
sudo apt install qemu-system
```

For other distributions, the package name and the package manager may vary.

1.2 Run QEMU

In order to emulate a 32-bit ARM system the package `qemu-system-arm` is used. Mandatory arguments are machine, CPU, and kernel image. The machine argument specifies the machine type to emulate, while the CPU argument specifies the CPU model, the kernel image is the kernel to be loaded.

The following command is used to run QEMU:

```
qemu-system-arm -machine mps2-an385 \
                -cpu cortex-m3      \
                -kernel kernel.elf
```

In this example the machine is `mps2-an385`, the CPU is `cortex-m3` and the kernel image is `kernel.elf`.

2 FreeRTOS

2.1 Real-Time Operating System

FreeRTOS is a real-time operating system (RTOS) for embedded devices, it is the market-leading RTOS. The OS provide a kernel around which software applications can be built, regardless of the specific hardware platform.

The need for a RTOS arises from the inherent complexity of managing multiple tasks in real-time environments. Embedded systems often have to respond to events in a timely and predictable manner. FreeRTOS provides the necessary features to achieve this, such as task scheduling, inter-task communication, and synchronization.

2.2 FreeRTOS Usage

In order to use FreeRTOS, the user must include the FreeRTOS source code in their project. It is designed to be simple and easy to use, only three source files common to all ports and one microcontroller specific file are required.

In the official FreeRTOS project, the file to be included are located in:

- `FreeRTOS/Source/tasks.c`
- `FreeRTOS/Source/queue.c`
- `FreeRTOS/Source/list.c`
- `FreeRTOS/Source/portable/[compiler]/[architecture]/port.c`
- `FreeRTOS/Source/portable/MemMang/heap_x.c`

The following header files paths must be included in the project:

- `FreeRTOS/Source/include`

- `FreeRTOS/Source/portable/[compiler]/[architecture]`
- `[path to FreeRTOSconfig.h]`

Every project must include a `FreeRTOSConfig.h` file, which is used to configure the RTOS to the specific needs of the application.

3 Practical Projects

Practical projects have been developed to demonstrate how to develope applications using QEMU and FreeRTOS on a Cortex-M3 microcontroller.

For all projects, the **NUCLEO-F103RB** development board is used. The choice of the board is based on the fact that the Nucleo boards are designed to be very affordable, and they are also very popular among hobbyists and students.

The projects are startupped with the help of the STM32CubeMX tool, which is a graphical software configuration tool, provided by STMicroelectronics, that allows generating C initialization code using a graphical interface.

3.1 LED Blink Project

This project focuses on the interaction with a microcontroller's hardware, specifically controlling a LED. The goal is to make a LED blink every 1000 milliseconds. To achieve this, we use the HAL (Hardware Abstraction Layer) library provided by STM.

3.1.1 Hardware Abstraction Layer

The HAL library is an Application Programming Interface (API) that abstracts the details of the microcontroller's hardware. This means that we can write code that interacts with the hardware in a high-level and portable way.

The HAL library provides a function, `HAL_GPIO_TogglePin`, that allows us to alter the state of a pin. This function takes two arguments: the GPIO port and the pin number. For the LED, the GPIO port is `GPIOA` (0x40000000) and the pin number is 5 (0x20 is the bitmask):

3.1.2 Pin Control

To control the LED on a NUCLEO-F103RB board, the HAL library needs to write some bits in memory. From the documentation of the board, the GPIO Register Map has the structure showed in the figure 1.

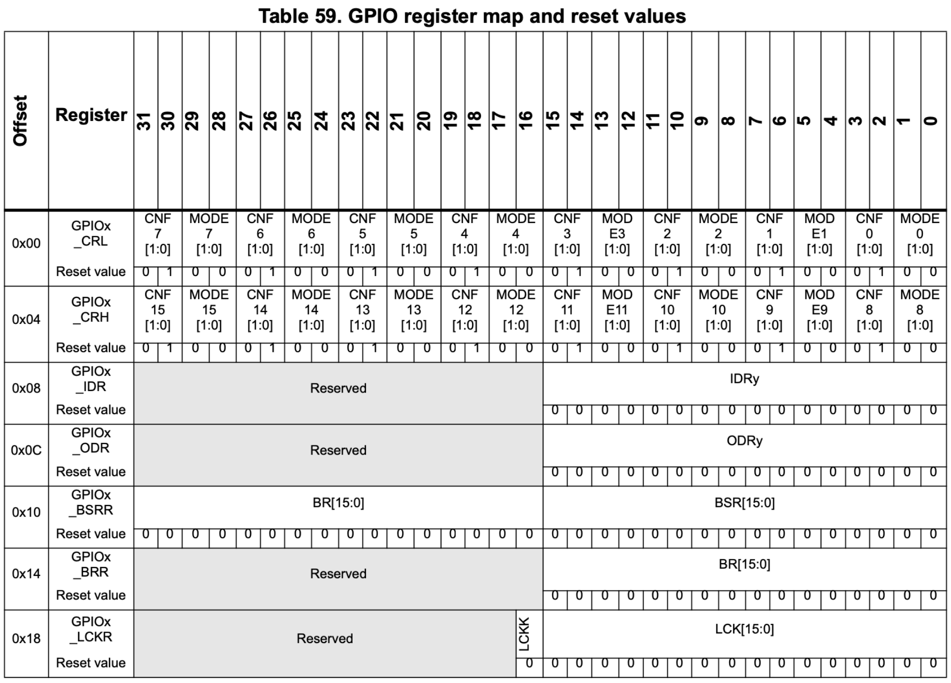

Table 59. GPIO register map and reset values

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	GPIOx _CRL	CNF 7 [1:0]		MODE 7 [1:0]		CNF 6 [1:0]		MODE 6 [1:0]		CNF 5 [1:0]		MODE 5 [1:0]		CNF 4 [1:0]		MODE 4 [1:0]		CNF 3 [1:0]		MODE3 [1:0]		CNF 2 [1:0]		MODE 2 [1:0]		CNF 1 [1:0]		MODE1 [1:0]		CNF 0 [1:0]		MODE 0 [1:0]	
	Reset value	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0
0x04	GPIOx _CRH	CNF 15 [1:0]		MODE 15 [1:0]		CNF 14 [1:0]		MODE 14 [1:0]		CNF 13 [1:0]		MODE 13 [1:0]		CNF 12 [1:0]		MODE 12 [1:0]		CNF 11 [1:0]		MODE11 [1:0]		CNF 10 [1:0]		MODE 10 [1:0]		CNF 9 [1:0]		MODE9 [1:0]		CNF 8 [1:0]		MODE 8 [1:0]	
	Reset value	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0
0x08	GPIOx _IDR	Reserved																IDRy															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x0C	GPIOx _ODR	Reserved																ODRy															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x10	GPIOx _BSRR	BR[15:0]																BSR[15:0]															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x14	GPIOx _BRR	Reserved																BR[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x18	GPIOx _LCKR	Reserved															LCKK	LCK[15:0]															
	Reset value																0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 1: GPIO Register Map

To set or reset a pin, the BSSR (Bit Set/Reset Register) is used. The first 16 bits of the BSSR register are used to set, while the last 16 bits are used to reset.

To toggle a pin, first the current value is taken from the ODR (Output Data Register) and then the BSSR low register is written with the inverse of the current value, while the BSSR high register is written with the current value.

3.2 Task Scheduling Project

This project is a practical exploration of task scheduling in FreeRTOS. It involves two tasks, each of which prints a message to the console. The priority of these tasks and the preemption setting of the scheduler can be adjusted to observe different behaviors.

3.2.1 Project Initialization and Execution

The project is initialized with two tasks of equal priority. Each task prints a message ("Hello from Task x") to the console every second, thanks to a **delay** added to each task. The scheduler is set to **preemptive** mode, which means that it can interrupt a currently running task to start executing a higher-priority task.

When the project runs, the console output shows that each task is executed in a **round-robin** way, as expected in a preemptive scheduler with tasks of equal priority in FreeRTOS.

3.2.2 Disabling Preemption

When preemption is disabled, the scheduler cannot interrupt a running task until it has finished. The expected behavior is that the first task runs indefinitely, and the second task is never executed. Though we observe the same behavior as in the preemptive mode.

The reason for this is that each task includes a delay using the `vTaskDelay` function, which put the task in the **blocked** state for a specified period. This allows the other task to be executed, despite the lack of preemption.

If the delay is removed from both tasks and preemption is disabled, the first task runs indefinitely, and the second task is never executed.

3.2.3 Changing Task Priorities

If the priority of Task 2 is set higher than that of Task 1, then when the project is run, Task 2 is executed indefinitely, and Task 1 is never executed. This is because the scheduler always chooses the highest-priority task to run.

Even if preemption is re-enabled, the output does not change, because Task 2 still has a higher priority than Task 1. The scheduler will always choose Task 2 over Task 1, regardless of whether it can preempt the currently running task.

3.3 FreeRTOS Features Project

This last project is designed to demonstrate how some of the features of FreeRTOS can be used. A timer, a semaphore and a queue are used to show how tasks can be synchronized and communicate with each other.

3.3.1 Project Design

The project is design as shown in the figure 2. It works as follows:

Timer The timer is a software timer that is configured to expire periodically every 1000 milliseconds. Upon expiration, it triggers a callback function. The primary role of this callback function is to give a semaphore.

Semaphore The semaphore is used to synchronize the operation of Task TX and the timer's callback function. When the timer's callback function is executed, it gives the semaphore. Task TX waits for the semaphore and performs its operation when it is given.

Task TX Task TX is designed to demonstrate how a task can wait for a semaphore and perform an operation once the semaphore is given, it sends a value to a queue. This value is a counter that is incremented in each loop, serving as a simple demonstration of a task performing work in response to a semaphore.

Queue The queue is used to send the value produced by Task TX to Task RX.

Task RX Task RX is designed to demonstrate how tasks can wait for data on a queue and process it. It prints the received value and the current tick count. This demonstrates a task that consumes data produced by another task and the use of the system tick count to measure time in a real-time system.

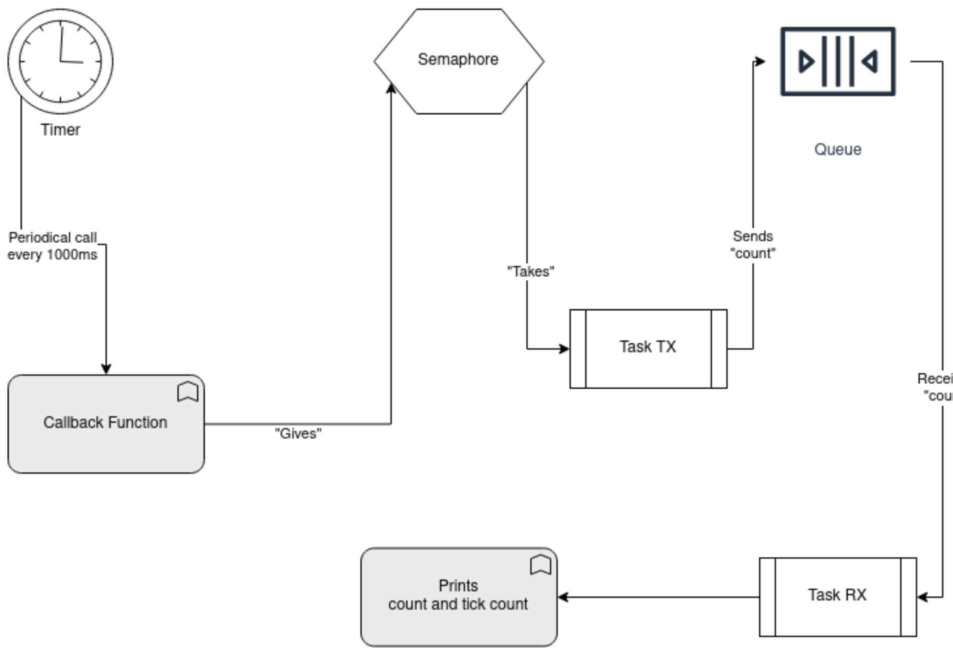

Figure 2: FreeRTOS Features Project Schema

4 Memory Management

A custom memory management library has been implemented. The library is a modified version of the `heap_4.c` provided by FreeRTOS.

In the custom implementation it has been developed the *realloc* function, among with *best-fit* and *worst-fit* allocation strategy.

4.1 Realloc

4.1.1 Implementation

The implementation of the realloc function is the following:

```
void *pvPortRealloc(void *pv, size_t xWantedSize);
```

The function takes two arguments, the pointer to the memory block to be reallocated and the new size of the memory block. The function returns a pointer to the reallocated memory block.

In case a `NULL` pointer is passed as the first argument, the function behaves as a malloc call. In case the second argument is 0, the function behaves as a free call.

In any other case the funcion will try to reallocate the memory block, after checking that the pointer corresponds to a valid memory block.

The reallocation is implemented in the following way:

```
BlockLink_t * pxLink;
void * pvReturn = NULL;
uint8_t * puc = ( uint8_t * ) pv;
size_t xBlockSize;
size_t xMoveSize;

/* The memory being reallocated will have an BlockLink_t structure
    immediately
    * before it. */
puc -= xHeapStructSize;

/* This casting is to keep the compiler from issuing warnings. */
pxLink = ( void * ) puc;
```

```
    /* The size of the block being reallocated */
    xBlockSize = ( pxLink->xBlockSize & ~heapBLOCK_ALLOCATED_BITMASK ) -
        xHeapStructSize;

    /* Allocate new memory block */
    pvReturn = pvPortMalloc( xWantedSize );

    if( pvReturn != NULL )
    {
        /* The size of the memory to copy */
        xMoveSize = xBlockSize < xWantedSize ? xBlockSize : xWantedSize;

        /* Copy the memory to the new location */
        memcpy( pvReturn, pv, xMoveSize );

        /* Free old memory block */
        vPortFree(pv);
    }
```

Notice that FreeRTOS use the first bit of the block size to store the allocation status of the block. For this reason if a block is allocated the size of the block is obtained by clearing the first bit of the block size.

4.1.2 Testing

The realloc function has been tested on the following cases:

Reallocating a block to a larger size To test the realloc function a block of memory of 10 bytes is allocated and then reallocated to a size of 30 bytes. It can be observed that the memory is reallocated to a new address and the old memory is freed.

Reallocating a block to a smaller size The test is similar to the previus one but with initial size of 30 bytes and reallocation to 10 bytes. It can be observed that the memory is reallocated to a new address and the old memory is freed. Only the first 10 bytes of the old memory are copied to the new memory.

Reallocating a NULL pointer The `NULL` pointer is reallocated, it can be observed that the memory is allocated to a new address, same as a malloc call.

Reallocating a block to size 0 A block is reallocated to size 0, it can be observed that the memory is freed, same as a free call.

4.2 Best-Fit and Worst-fit Strategy

4.2.1 Implementation

FreeRTOS by default use a *first-fit* allocation strategy and handles the free blocks in a linked list. In order to allocate a block the list is traversed until a block greater than the requested size is found.

The *best-fit* strategy is implemented by traversing the list and keeping track of the smallest block greater than the requested size. The first block greater than the requested size is returned.

Following the implementation

Best-fit Traverse the list and find the smallest difference between the block size and the requested size.

```
#if (configHEAP_ALLOCATION_TYPE == 1)
    configASSERT( heapSUBTRACT_WILL_UNDERFLOW( pxBlock->xBlockSize,
        xWantedSize ) == 0 );
    /* traverse the whole free block list */
    while( pxBlock->pxNextFreeBlock != heapPROTECT_BLOCK_POINTER( NULL ) )
    {
        /* Check if the current block is a valid option and if another valid
            block
            was found before and check wheter is a best fit */
        if  (   ( pxBlock->xBlockSize >= xWantedSize )
                &&
                (   pxBlockTmp == NULL
                    ||
                    ( ( pxBlock->xBlockSize - xWantedSize ) < ( pxBlockTmp->
                        xBlockSize - xWantedSize ) )
                )
            )
        {
            pxPreviousBlockTmp = pxPreviousBlock;
            pxBlockTmp = pxBlock;
        }
        pxPreviousBlock = pxBlock;
        pxBlock = heapPROTECT_BLOCK_POINTER( pxBlock->pxNextFreeBlock );
        heapVALIDATE_BLOCK_POINTER( pxBlock );
    }
    pxPreviousBlock = pxPreviousBlockTmp;
    pxBlock = pxBlockTmp;
```

Worst-fit Traverse the list and find the largest difference between the block size and the requested size.

```
#elif (configHEAP_ALLOCATION_TYPE == 2)
    configASSERT( heapSUBTRACT_WILL_UNDERFLOW( pxBlock->xBlockSize,
        xWantedSize ) == 0 );
    /* traverse the whole free block list */
    while( pxBlock->pxNextFreeBlock != heapPROTECT_BLOCK_POINTER( NULL ) )
    {
        /* Check if the current block is a valid option and if another valid
            block
            was found before and check wheter is a worst fit */
        if  (   ( pxBlock->xBlockSize >= xWantedSize )
                &&
                (   pxBlockTmp == NULL
                    ||
                    ( ( pxBlock->xBlockSize - xWantedSize ) > ( pxBlockTmp->
                        xBlockSize - xWantedSize ) )
                )
            )
        {
            pxPreviousBlockTmp = pxPreviousBlock;
            pxBlockTmp = pxBlock;
        }
        pxPreviousBlock = pxBlock;
        pxBlock = heapPROTECT_BLOCK_POINTER( pxBlock->pxNextFreeBlock );
        heapVALIDATE_BLOCK_POINTER( pxBlock );
    }
    pxPreviousBlock = pxPreviousBlockTmp;
    pxBlock = pxBlockTmp;
```

First-fit Traverse the list and return the first block greater than the requested size. Default case.

```
#else
    while( ( pxBlock->xBlockSize < xWantedSize ) && ( pxBlock->
        pxNextFreeBlock != heapPROTECT_BLOCK_POINTER( NULL ) ) )
    {
        pxPreviousBlock = pxBlock;
        pxBlock = heapPROTECT_BLOCK_POINTER( pxBlock->pxNextFreeBlock );
        heapVALIDATE_BLOCK_POINTER( pxBlock );
    }
#endif
```

4.2.2 Testing

Best-fit The best-fit strategy has been tested by creating a memory layout with 5 blocks of memory as follow:

	Block 1	**Block 2**	**Block 3**	**Block 4**	**Block 5**
Allocated	No	Yes	No	Yes	No
Size (Bytes)	64	32	32	32	2856

A new block of 32 bytes is requested, the best-fit strategy should return the block 3, the smallest block greater than the requested size.

Worst-fit The worst-fit strategy has been tested by creating a memory layout with 3 blocks of memory as follow:

	Block 1	**Block 2**	**Block 3**
Allocated	No	Yes	No
Size (Bytes)	64	32	2936

A new block of 32 bytes is requested, the worst-fit strategy should return the block 3, the largest block greater than the requested size.

5 Conclusion

Our project focused on mastering FreeRTOS on QEMU through a tutorial and three exercises. Beginning with a LED blink project, we interfaced with microcontroller hardware using the HAL library to achieve a 1000-millisecond blink interval. Next, we explored task scheduling, adjusting task priorities and preemption settings to observe different behaviors. The third exercise showcased FreeRTOS features like timers, semaphores, and queues, exemplified through a producer-consumer scenario, illustrating task synchronization and communication. In our final endeavor, we enhanced memory management by modifying the heap4.c implementation,introducing the realloc function and best-fit/worst-fit allocation strategies. Practical examples validated the efficacy of these enhancements. Overall, this project equipped us with essential skills in FreeRTOS development, empowering us to navigate real-time operating systems and embedded systems with confidence and proficiency. Through practical implementations and comprehensive exploration, we've gained invaluable insights and capabilities for future endeavors in embedded systems development.

6 Statement Of Work

The work done by each member of the group is summarized in the following table:

Member	**Work Done**
Francesco Laterza	Project Management Demo Developement Projects Developement MemMang Implementation MemMang Testing
Leonardo Rizzo	QEMU Tutorial Features Project Projects Readmes
Massimo Aresca	FreeRTOS Tasks Demo Theoretical Researches
Yann Freddy Dongue Dongmo	FreeRTOS Tasks Demo Features Project

The workload has been divided among the members based on their skills, preferences and previous experiences. The work has been done in a collaborative way, with the members helping each other to solve problems and to understand the different aspects of the project. The project has been carried out in a way that each member has been able to learn new things and to improve his skills.