



Computer Architecture and Operating System

HackOSSim Project

Master's Degree in Cybersecurity

Giuseppe Famà, Youssef Rachid Grib, Vincenzo Longo

September 20, 2024



**Politecnico
di Torino**



Table of Contents

1 Introduction

- ▶ Introduction
- ▶ Environment Setup
- ▶ Task Management
- ▶ Queue and Task Synchronization
- ▶ Memory Management



The Goal of this Project

1 Introduction

The goal of this project consists in analysing and using the real time OS **FreeRTOS** exploiting the **QEMU**¹ simulator. In the following you find a detailed tutorial for the **installation** and usage procedures and some **practical examples** and **new implementations** to demonstrate the functionality of the operating system.

¹It allows to virtualize several types of hardware architecture.



Table of Contents

2 Environment Setup

► Introduction

► **Environment Setup**

► Task Management

► Queue and Task Synchronization

► Memory Management



Installation

2 Environment Setup

To setup the environment the following steps has been followed:

- Downloading the FreRTOS repository;
- Downloading of QEMU emulator;
- Setup of other tools:
 - ARM GNU Toolchain;
 - CMake;
 - Make;
- Finally we proceed with the environment configuration;



Intro to the Demo Applications

2 Environment Setup

In this project, each single demo application can be selected by properly setting the `mainCREATE_SIMPLE_DEMO` value in the `main.c` file.

```
/* The mainCREATE_SIMPLE_DEMO setting is described at the top
 * of this file. It selects the proper demo application */
#if ( mainCREATE_SIMPLE_DEMO == 1 )
{
    main_three_tasks_CRUDE();
}
#elif ( mainCREATE_SIMPLE_DEMO == 2 )
{
    main_three_tasks();
}
#elif ( mainCREATE_SIMPLE_DEMO == 3 )
{
    main_priority();
}
#elif ( mainCREATE_SIMPLE_DEMO == 4 )
{
    main_queue();
}
#elif ( mainCREATE_SIMPLE_DEMO == 5 )
{
    main_semaphore();
}
#elif ( mainCREATE_SIMPLE_DEMO == 6 )
{
    main_semaphore2();
}
#elif ( mainCREATE_SIMPLE_DEMO == 7 )
{
    main_memManagement();
}
#endif
```



Table of Contents

3 Task Management

► Introduction

► Environment Setup

► **Task Management**

► Queue and Task Synchronization

► Memory Management



Demo Applications

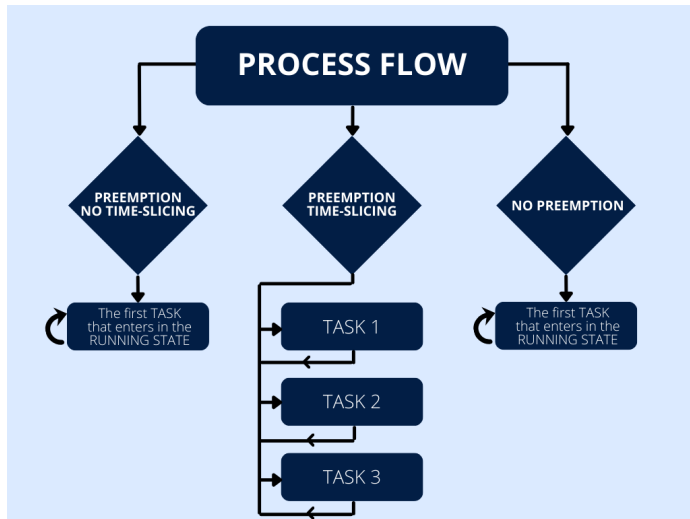
3 Task Management

- **main_three_task_CRUDE.c**
 - Tasks with the **SAME priority**.
 - Each task implements the same function that **prints a message**.
 - After printing the message, the task enters in a **loop** whose the unique functionality is the implementation of a **CRUDE delay**, i.e., which does not move the task in the waiting list.
- **main_three_task.c**
- **main_priority.c**



Process Flow - main_three_task_CRUDE.c

3 Task Management





Demo Applications

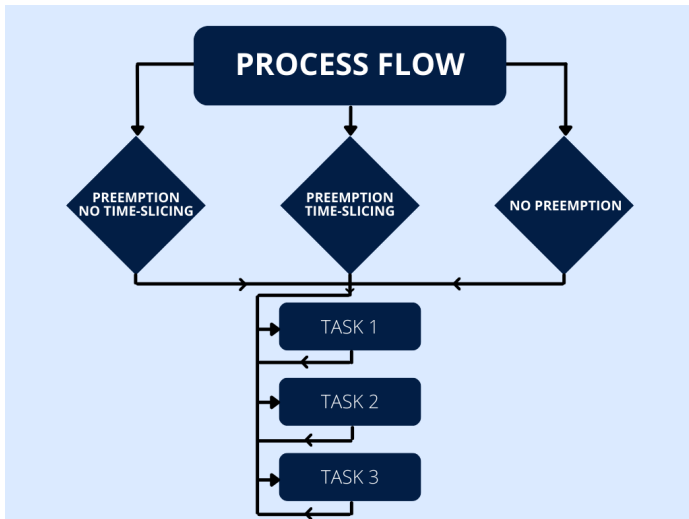
3 Task Management

- `main_three_task_CRUDE.c`
- **`main_three_task.c`**
 - **NO CRUDE DELAY!**
 - API function `VTaskDelayUntil()`.
 - This function just moves the task in the blocked state, making room for tasks in the ready state.
- `main_priority.c`



Process Flow - main_three_task.c

3 Task Management





Demo Applications

3 Task Management

- `main_three_task_CRUDE.c`
- `main_three_task.c`
- **`main_priority.c`**
 - More advanced example to show how **FreeRTOS scheduler** works.
 - One of the two tasks has dynamic priority.



Process Flow - main_priority.c

3 Task Management

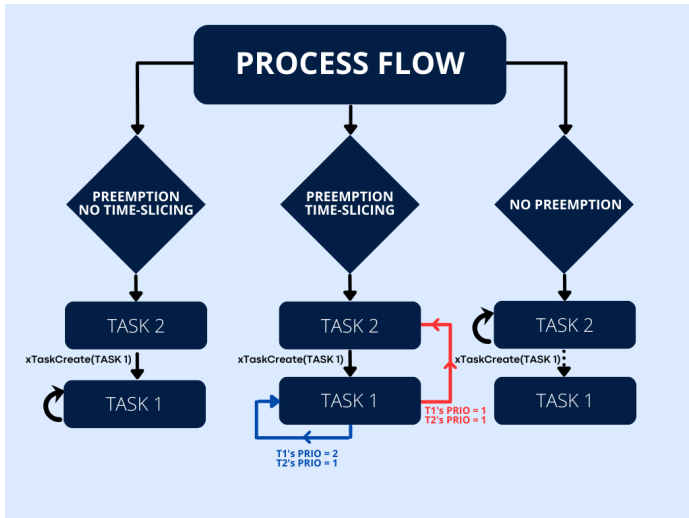




Table of Contents

4 Queue and Task Synchronization

- ▶ Introduction
- ▶ Environment Setup
- ▶ Task Management
- ▶ Queue and Task Synchronization
- ▶ Memory Management



Demo Applications

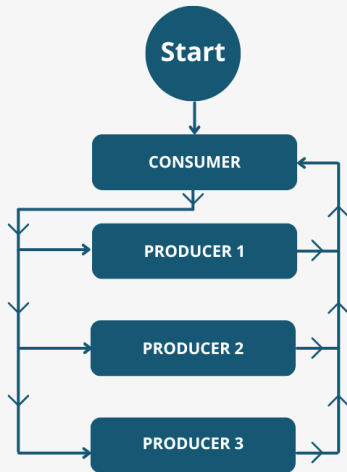
4 Queue and Task Synchronization

- **main_queue.c**
 - Three **PRODUCERS** and one **CONSUMER** (**highest priority**).
 - The **CONSUMER** is blocked when the queue is empty, unblocking the **PRODUCERS**.
- **main_semaphore.c**
- **main_semaphore2.c**



Precedence Diagrams - main_queue.c

4 Queue and Task Synchronization



```
QEMU RTOSdemo started
Queue created
-----
Queue is empty - waiting for data...
-----
Producer 1 about to send 100 to the queue
-----
Consumer Received = 100
-----
Queue is empty - waiting for data...
-----
Producer 2 about to send 200 to the queue
-----
Consumer Received = 200
-----
Queue is empty - waiting for data...
-----
Producer 3 about to send 300 to the queue
-----
```




Demo Applications

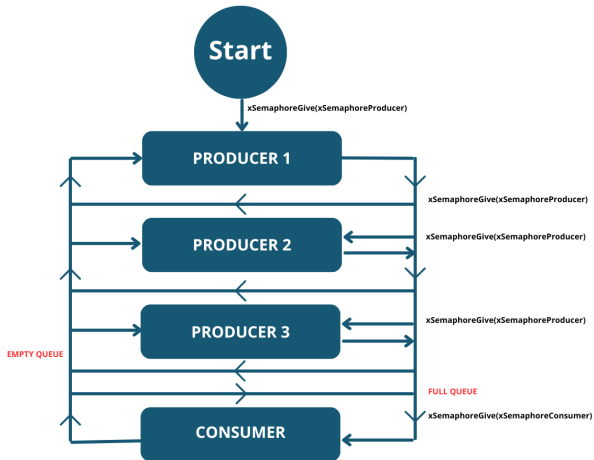
4 Queue and Task Synchronization

- main_queue.c
- main_semaphore.c
 - Three **PRODUCERS** and one **CONSUMER** handled with **two binary semaphores**.
 - At the beginning the **PRODUCERS** enter in the critical section **until the queue is fulfilled**.
 - Then the **CONSUMER** reads out all the data.
 - **Any priority can be used!** Semaphores synchronize the tasks.
- main_semaphore2.c



Precedence Diagrams - main_semaphore.c

4 Queue and Task Synchronization



```
-----
Producer 3 sent: 30
-----
Producer 1 sent: 10
The queue is full, consumer can start. 20
Consumer received: 10
-----
Consumer received: 20
-----
Consumer received: 30
-----
Consumer received: 10
-----
Consumer received: 20
-----
Consumer received: 30
-----
```



Demo Applications

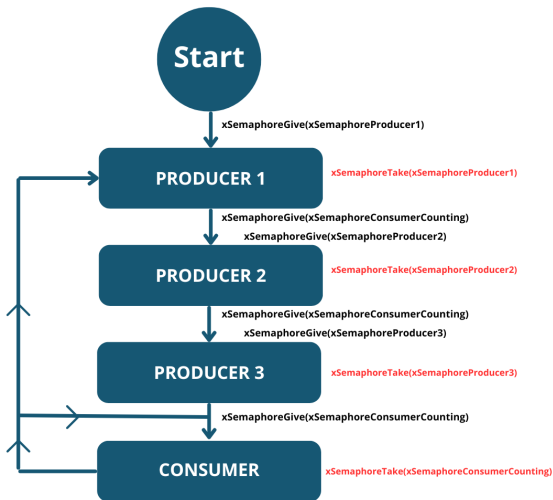
4 Queue and Task Synchronization

- `main_queue.c`
- `main_semaphore.c`
- **`main_semaphore2.c`**
 - Three **PRODUCERS** and one **CONSUMER** handled with **four semaphores** (3 binary and 1 counting).
 - Each **PRODUCER** inserts its value and unblocks the next task.
 - Once all the **PRODUCERS** inserted their values, the **CONSUMER** reads out all the items, then unblocks the first **PRODUCER**.
 - **Any priority can be used!** Semaphores synchronize the tasks.



Precedence Diagrams - main_semaphore2.c

4 Queue and Task Synchronization



Producer 1 sent: 10

Producer 2 sent: 20

Producer 3 sent: 30

Consumer received: 10

Consumer received: 20

Consumer received: 30

Producer 1 sent: 10

Producer 2 sent: 20



Table of Contents

5 Memory Management

- ▶ Introduction
- ▶ Environment Setup
- ▶ Task Management
- ▶ Queue and Task Synchronization
- ▶ **Memory Management**



Implementation

5 Memory Management

To better analyze the Memory Management the provided `heap_4.c` file was revised to implement:

- **Best-Fit:** The process is allocated in smallest available memory block that is large enough for the process.
- **Worst-Fit:** The process is allocated in the largest available memory block.
- **First-Fit:** The process is allocated in the first available memory block.



Performance Evaluation

5 Memory Management

The main perform the following steps:

1. Allocates 1000 bytes
2. Allocates 1000 bytes
3. Allocates 1500 bytes
4. Allocates 100 bytes
5. Allocates 100 bytes
6. Allocates 100 bytes
7. Allocates 100 bytes
8. De-allocates 1000 bytes (second block)
9. De-allocates 100 bytes (fourth block)
10. De-allocates 100 bytes (fifth block)
11. De-allocates 100 bytes (sixth block)
12. Allocates 300 bytes
13. Allocates 1000 bytes
14. Creates a task (TASK 1)



First-Fit and Worst-Fit output 1/2

5 Memory Management

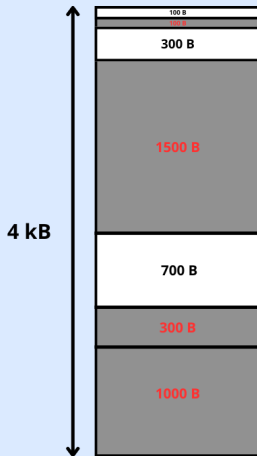
```
QEMU RTOSdemo started
Message | Free Heap (bytes) | Minimum Ever Free Heap (bytes)
-----|-----|-----
Before allocating memory blocks | 0 | 0
After allocated 1000 bytes | 3072 | 3072
After allocated 1000 bytes | 2064 | 2064
After allocated 1500 bytes | 552 | 552
After allocated 100 bytes | 440 | 440
After allocated 100 bytes | 328 | 328
After allocated 100 bytes | 216 | 216
After allocated 100 bytes | 104 | 104
After deallocated the second block (1000 bytes) | 1112 | 104
After deallocated the fourth block (100 bytes) | 1224 | 104
After deallocated the fifth block (100 bytes) | 1336 | 104
After deallocated the sixth block (100 bytes) | 1448 | 104
After allocated 300 bytes | 1136 | 104

Oops...Malloc failed
```




First-Fit and Worst-Fit output 2/2

5 Memory Management





Best-Fit output 1/2

5 Memory Management

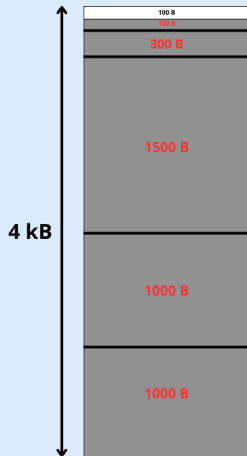
```
QEMU RTOSdemo started
Message                                | Free Heap (bytes) | Minimum Ever Free Heap (bytes)
-----
Before allocating memory blocks | 0                  | 0
After allocated 1000 bytes      | 3072               | 3072
After allocated 1000 bytes      | 2064               | 2064
After allocated 1500 bytes      | 552                | 552
After allocated 100 bytes       | 440                | 440
After allocated 100 bytes       | 328                | 328
After allocated 100 bytes       | 216                | 216
After allocated 100 bytes       | 104                | 104
After deallocated the second block (1000 bytes) | 1112               | 104
After deallocated the fourth block (100 bytes) | 1224               | 104
After deallocated the fifth block (100 bytes) | 1336               | 104
After deallocated the sixth block (100 bytes) | 1448               | 104
After allocated 300 bytes       | 1136               | 104
After allocated 1000 bytes      | 128                | 104

Oops...Malloc failed
```



Best-Fit output 2/2

5 Memory Management





Thanks for listening