Politecnico di Torino

Department of Control and Computer Engineering

INTRODUCTION TO FREERTOS

STEFANO DI CARLO

OUTLINE

- What is a Real-Time Application (RTA)?
- What is a Real-Time Operating System (RTOS)?
- RTOS vs General Purpose Operating System (GPOS)
- What is FreeRTOS?
- FreeRTOS kernel
- FreeRTOS configuration
- FreeRTOS demos
- First operations with FreeRTOS

WHAT IS A REAL-TIME APPLICATION (RTA)

- Correctness of the computation doesn't depend on the result only, but also on the **completion time**
- **Predictability** is more important than speed: need a constant response time

HARD REAL-TIME

- Strict time deadline
- Missing the deadline implies damages to people or to the system itself

Examples: Airbag Deployment Systems, Pacemakers, Anti-lock Braking Systems (ABS), Flight Control Systems, Industrial Control Systems

SOFT REAL-TIME

- Time deadline
- A delay is tolerable

Examples: Video Streaming, Online Gaming, VoIP (Voice over IP), Video Conferencing, E-commerce Systems

WHAT IS A REAL-TIME OPERATING SYSTEM (RTOS)

- Operating system designed to reach **time requirements**
- Applications executed with very **precise and deterministic timing**

µC/OS by Micrium

Embedded With RTEMS™ www.rtems.org

QNX SOFTWARE SYSTEMS

ERIKA 3

VxWorks

freeRTOS

WHAT IS A REAL-TIME OPERATING SYSTEM (RTOS)

- Operating system designed to reach **time requirements**
- Applications executed with very **precise and deterministic timing**

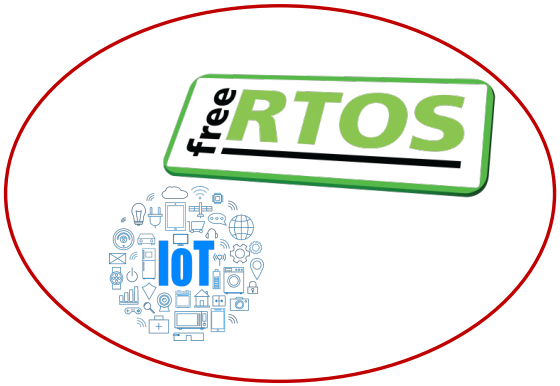
µC/OS by Micriµm

Embedded With RTEMS™ www.rtems.org

QNX SOFTWARE SYSTEMS

ERIKA 3

VxWorks



REAL-TIME OPERATING SYSTEM (RTOS)

- Priority-based preemptive scheduling
- Bounded interrupt and scheduling latency
- Accepts lower throughput in favor of determinism
- Minimal kernel

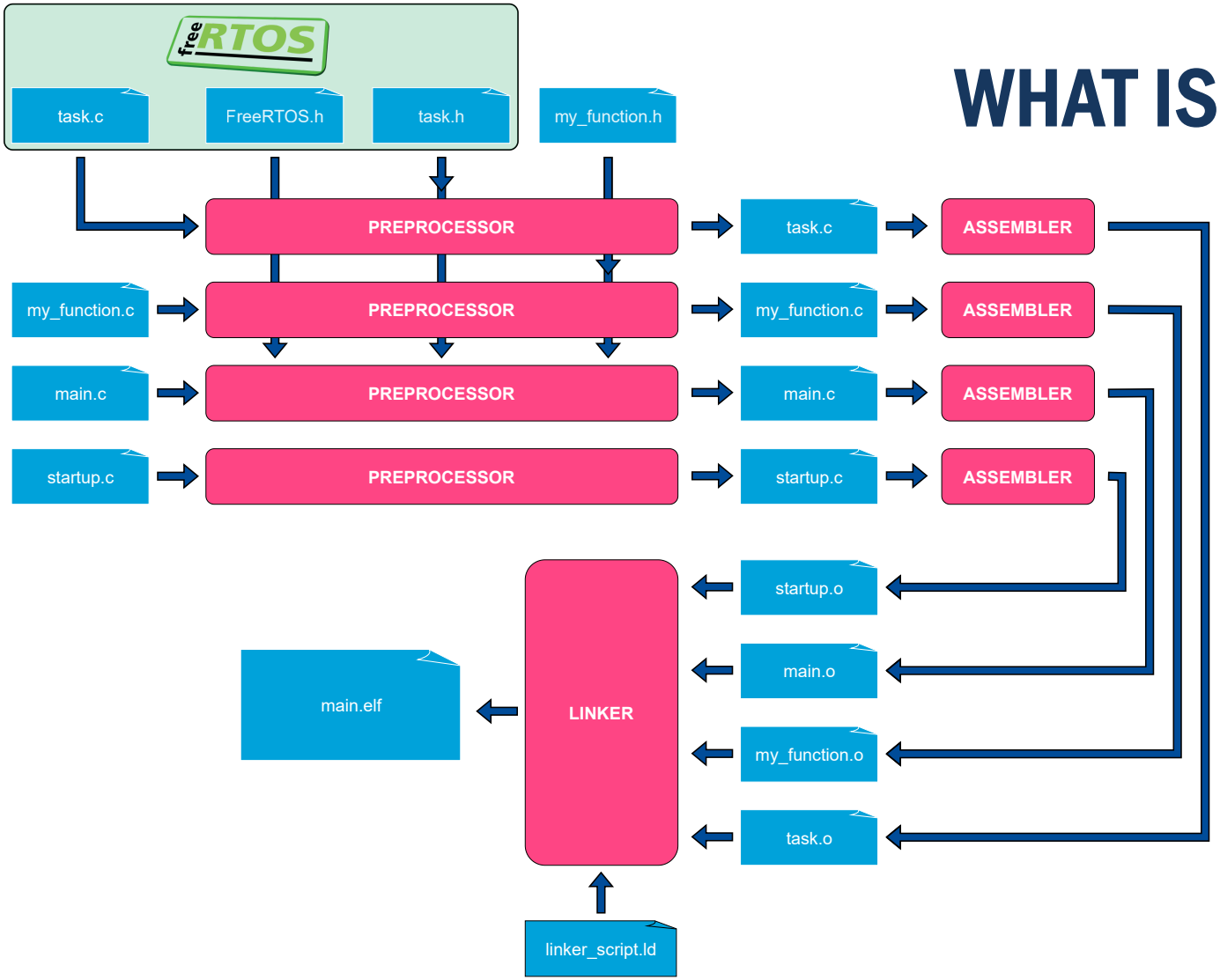
GENERAL PURPOSE OPERATING SYSTEM (GPOS)

- Scheduling for performance
- Unbounded interrupt and scheduling latency
- Maximizes throughput
- Larger kernel with more features

WHAT IS FREERTOS

- Real-time operating system
- Designed for embedded systems with **limited resources** (e.g. microcontrollers)
- **Open-source**, distributed under MIT license
- **Portable** on a large number of embedded boards
- Tiny, power-saving kernel
- Small memory footprint, low overhead, and fast execution
- Extensive documentation

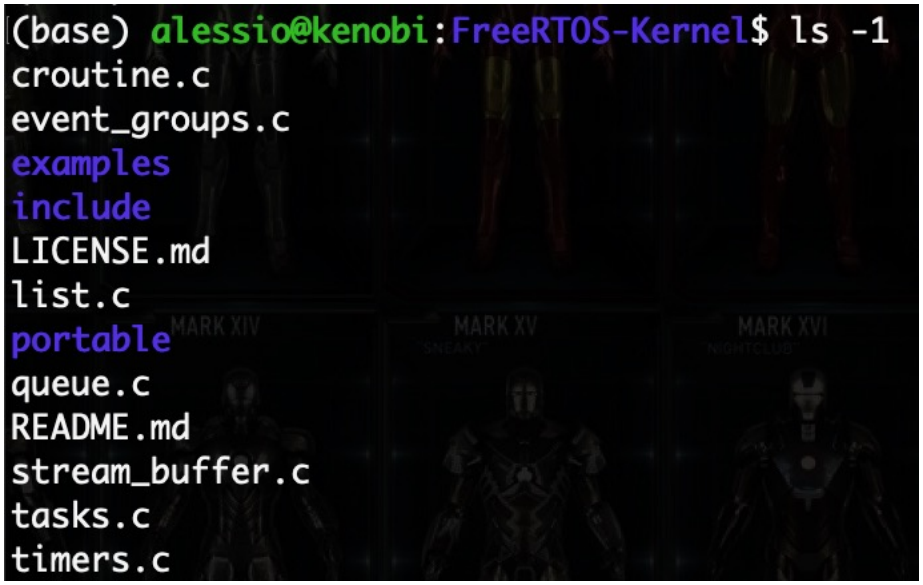
WHAT IS FREERTOS



- Flat operating system
- Set of functions that are linked to your executable to provide the OS functionalities

FREERTOS KERNEL

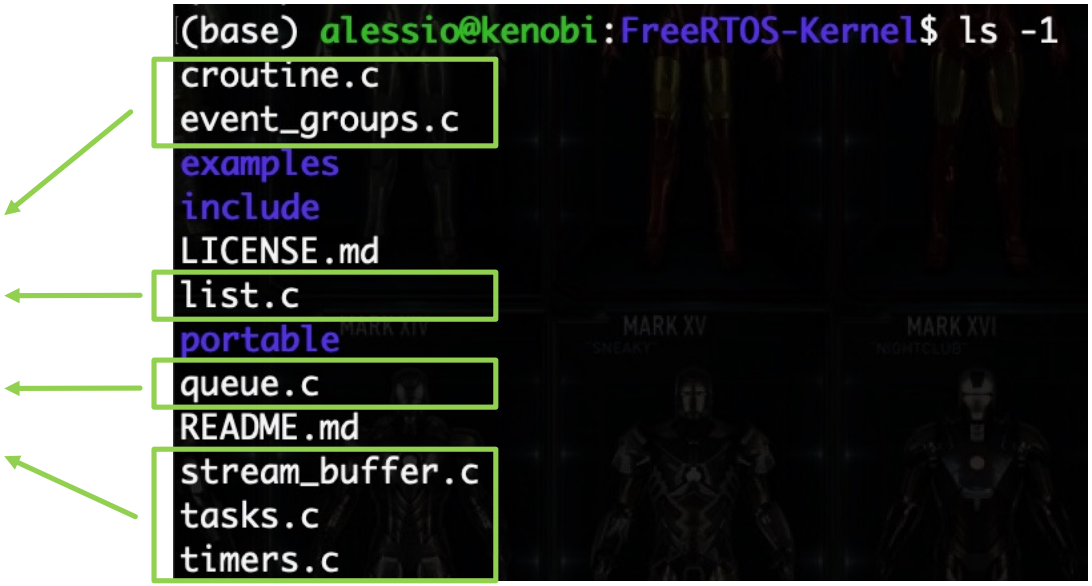
https://github.com/FreeRTOS/FreeRTOS-Kernel.git



FREERTOS KERNEL

https://github.com/FreeRTOS/FreeRTOS-Kernel.git

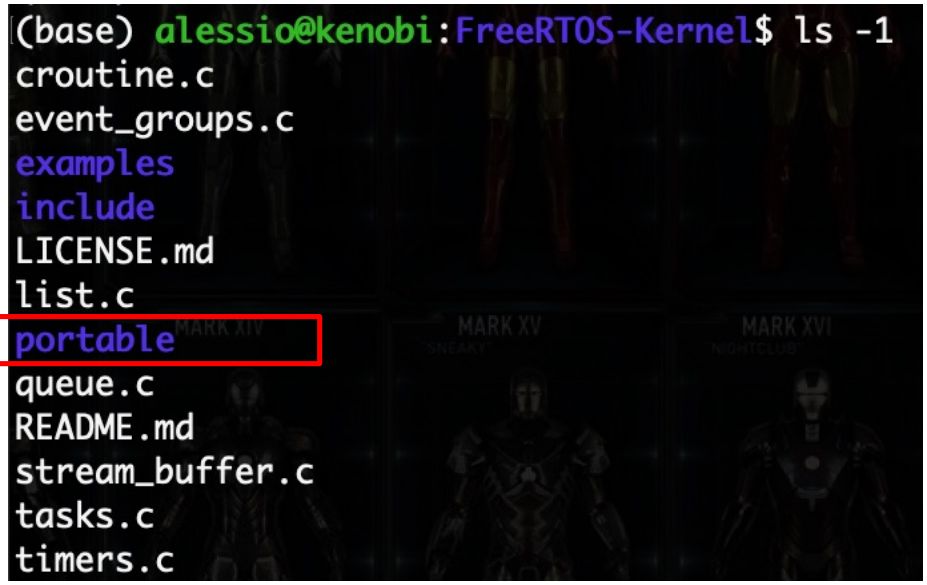
Architecture independent kernel functionalities



FREERTOS KERNEL

https://github.com/FreeRTOS/FreeRTOS-Kernel.git

Architecture **specific** kernel functionalities



FREERTOS KERNEL

https://github.com/FreeRTOS/FreeRTOS-Kernel.git

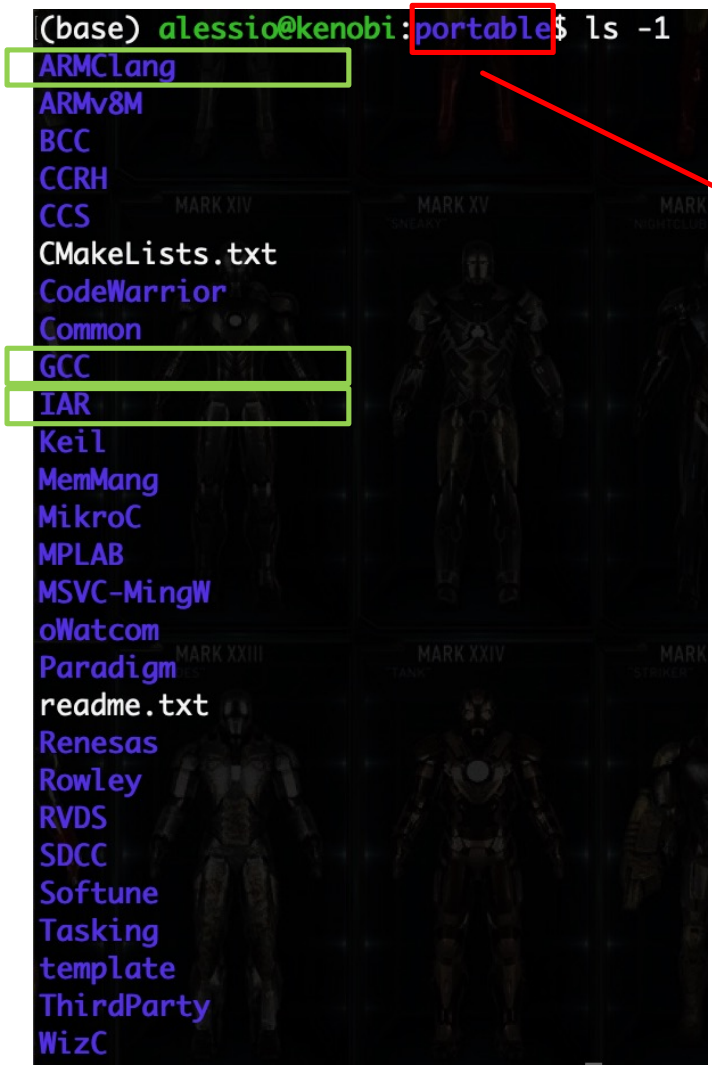
- Toolchain specific files

Architecture **specific** kernel functionalities

FREERTOS KERNEL

https://github.com/FreeRTOS/FreeRTOS-Kernel.git

- Toolchain specific files
- E.g. GCC, IAR, ARMClang toolchains



Architecture **specific** kernel functionalities

FREERTOS KERNEL

https://github.com/FreeRTOS/FreeRTOS-Kernel.git

- Architecture specific kernel functionalities
- E.g. ARM cores

```
(base) alessio@kenobi:ARM_CM4F$ ls -1
port.c
portmacro.h
```

- **Source code**: need to compile and link port.c to the application
- **Header**: automatically included by FreeRTOS. Need to add the right include path during compilation

Toolchain specific kernel functionalities

FREERTOS KERNEL

https://github.com/FreeRTOS/FreeRTOS-Kernel.git

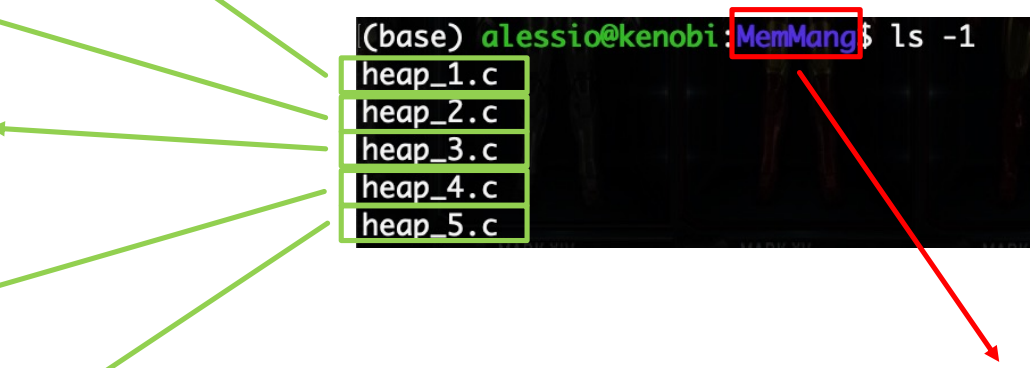
- Toolchain specific files
- E.g. GCC, IAR, ARMClang toolchains
- Memory management functionalities

Architecture specific kernel functionalities

FREERTOS KERNEL

https://github.com/FreeRTOS/FreeRTOS-Kernel.git

- Simplest. Allocation only. No free
- Allocation and free
- Uses standard C malloc() and free() wrapping them in FreeRTOS memory management interface
- Improves heap_2: adiacent blocks merging. Reduces fragmentation
- Most advanced: allow allocation in multiple, non-contiguous regios



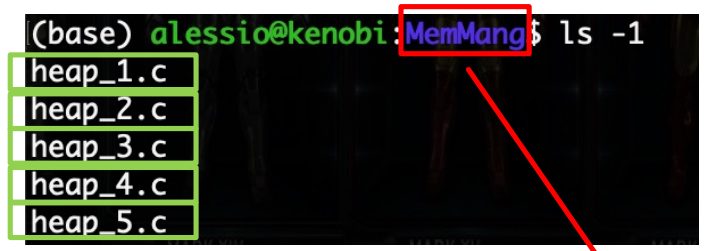
Memory management functionalities

FREERTOS KERNEL

https://github.com/FreeRTOS/FreeRTOS-Kernel.git

- Simplest. Allocation only. No free
- Allocation and free
- Uses standard C malloc() and free() wrapping them in FreeRTOS memory management interface
- Improves heap_2: adiacent blocks merging. Reduces fragmentation
- Most advanced: allow allocation in multiple, non-contiguous regios

- Compile and link only the desired memory management code (e.g. heap_4.c)

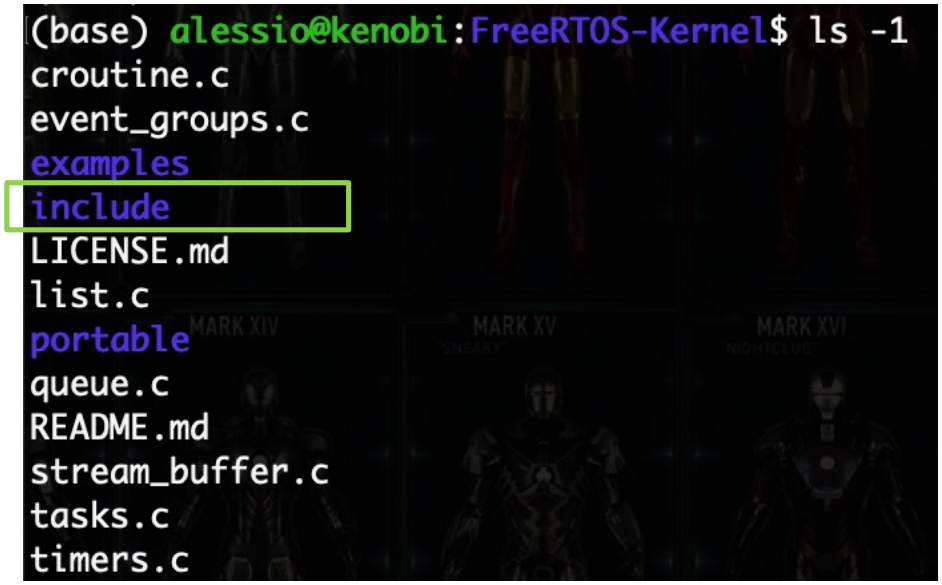


Memory management functionalities

FREERTOS KERNEL

https://github.com/FreeRTOS/FreeRTOS-Kernel.git

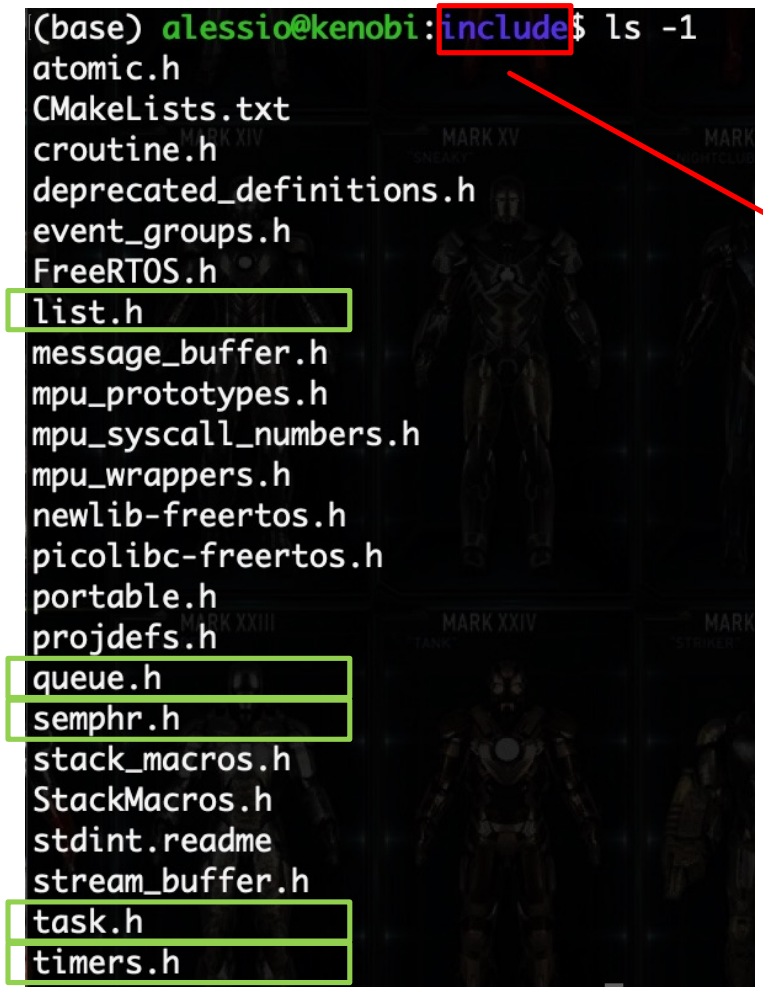
Custom types, functions interface etc.



FREERTOS KERNEL

https://github.com/FreeRTOS/FreeRTOS-Kernel.git

- Declaration and function prototypes to use FreeRTOS functionalities

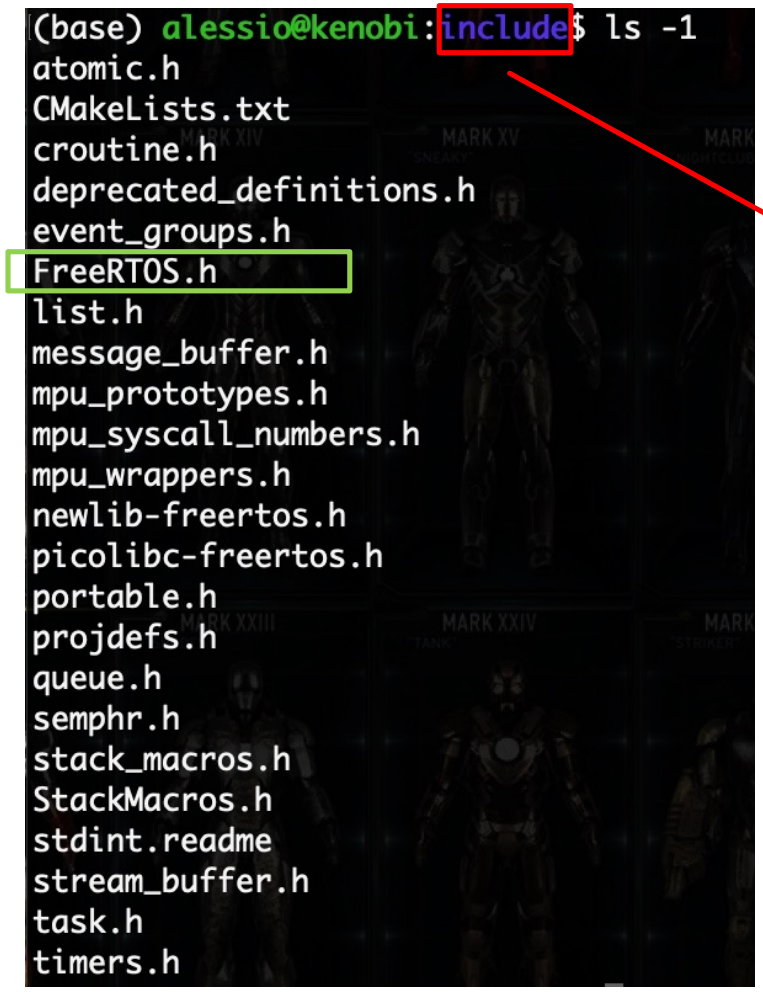


Custom types, functions interface etc.

FREERTOS KERNEL

https://github.com/FreeRTOS/FreeRTOS-Kernel.git

- Declaration and function prototypes to use FreeRTOS functionalities
- FreeRTOS custom types definitions and configurations



Custom types, functions interface etc.

FREERTOS KERNEL

https://github.com/FreeRTOS/FreeRTOS-Kernel.git

- Declaration and function prototypes to use FreeRTOS functionalities
- FreeRTOS custom types definitions and configurations

```
#ifndef LIST_H
#define LIST_H

#ifndef INC_FREERTOS_H
    #error "FreeRTOS.h must be included before list.h"
#endif
```

```
(base) alessio@kenobi:include$ ls -1
atomic.h
CMakeLists.txt
croutine.h
deprecated_definitions.h
event_groups.h
FreeRTOS.h
list.h
message_buffer.h
mpu_prototypes.h
mpu_syscall_numbers.h
mpu_wrappers.h
newlib-freertos.h
picolibc-freertos.h
portable.h
projdefs.h
queue.h
semphr.h
stack_macros.h
StackMacros.h
stdint.readme
stream_buffer.h
task.h
timers.h
```

Custom types, functions interface etc.

FREERTOS CONFIGURATION

https://github.com/FreeRTOS/FreeRTOS-Kernel.git

FreeRTOS.h

```
/* Application specific configuration options. */
#include "FreeRTOSConfig.h"

#if !defined( configUSE_16_BIT_TICKS ) && !defined( configTICK_TYPE_WIDTH_IN_BITS )
    #error Missing definition:  One of configUSE_16_BIT_TICKS and configTICK_TYPE_WIDTH_
IN_BITS must be defined in FreeRTOSConfig.h     See the Configuration section of the Fre
eRTOS API documentation for details.
#endif

#if defined( configUSE_16_BIT_TICKS ) && defined( configTICK_TYPE_WIDTH_IN_BITS )
    #error Only one of configUSE_16_BIT_TICKS and configTICK_TYPE_WIDTH_IN_BITS must be
defined in FreeRTOSConfig.h.  See the Configuration section of the FreeRTOS API document
ation for details.
#endif
```

- FreeRTOS must be configured for the specific application
- Your project must include a FreeRTOSConfig.h file to configure application-specific options

FREERTOS CONFIGURATION

https://github.com/FreeRTOS/FreeRTOS-Kernel.git

- Scheduler configuration

FreeRTOSConfig.h

```
#define configUSE_PREEMPTION                    1
#define configCPU_CLOCK_HZ                      ( ( unsigned long ) 25000000 )
#define configTICK_RATE_HZ                      ( ( TickType_t ) 1000 )
#define configMINIMAL_STACK_SIZE                ( ( unsigned short ) 80 )
#define configTOTAL_HEAP_SIZE                   ( ( size_t ) ( 60 * 1024 ) )
#define configMAX_TASK_NAME_LEN                 ( 12 )
#define configUSE_TRACE_FACILITY                0
#define configUSE_16_BIT_TICKS                  0
#define configUSE_MUTEXES                       1
#define configUSE_RECURSIVE_MUTEXES             1
#define configCHECK_FOR_STACK_OVERFLOW          0
#define configUSE_QUEUE_SETS                    1
#define configUSE_COUNTING_SEMAPHORES           1

#define configMAX_PRIORITIES                    ( 9UL )
#define configSUPPORT_STATIC_ALLOCATION         0

/* Timer related defines. */
#define configUSE_TIMERS                        0
#define configTIMER_TASK_PRIORITY               ( configMAX_PRIORITIES - 4 )
#define configTIMER_QUEUE_LENGTH                20
#define configTIMER_TASK_STACK_DEPTH            ( configMINIMAL_STACK_SIZE * 2 )
```

FREERTOS CONFIGURATION

https://github.com/FreeRTOS/FreeRTOS-Kernel.git

- Scheduler configuration
- Stack and heap configuration

FreeRTOSConfig.h

```
#define configUSE_PREEMPTION                    1
#define configCPU_CLOCK_HZ                      ( ( unsigned long ) 25000000 )
#define configTICK_RATE_HZ                      ( ( TickType_t ) 1000 )
#define configMINIMAL_STACK_SIZE                ( ( unsigned short ) 80 )
#define configTOTAL_HEAP_SIZE                   ( ( size_t ) ( 60 * 1024 ) )
#define configMAX_TASK_NAME_LEN                 ( 12 )
#define configUSE_TRACE_FACILITY                0
#define configUSE_16_BIT_TICKS                  0
#define configUSE_MUTEXES                       1
#define configUSE_RECURSIVE_MUTEXES             1
#define configCHECK_FOR_STACK_OVERFLOW          0
#define configUSE_QUEUE_SETS                    1
#define configUSE_COUNTING_SEMAPHORES           1

#define configMAX_PRIORITIES                    ( 9UL )
#define configSUPPORT_STATIC_ALLOCATION         0

/* Timer related defines. */
#define configUSE_TIMERS                        0
#define configTIMER_TASK_PRIORITY               ( configMAX_PRIORITIES - 4 )
#define configTIMER_QUEUE_LENGTH                20
#define configTIMER_TASK_STACK_DEPTH            ( configMINIMAL_STACK_SIZE * 2 )
```

FREERTOS CONFIGURATION

https://github.com/FreeRTOS/FreeRTOS-Kernel.git

- Scheduler configuration
- Stack and heap configuration
- Mutexes and semaphores for synchronization

FreeRTOSConfig.h

```
#define configUSE_PREEMPTION                    1
#define configCPU_CLOCK_HZ                      ( ( unsigned long ) 25000000 )
#define configTICK_RATE_HZ                      ( ( TickType_t ) 1000 )
#define configMINIMAL_STACK_SIZE                ( ( unsigned short ) 80 )
#define configTOTAL_HEAP_SIZE                   ( ( size_t ) ( 60 * 1024 ) )
#define configMAX_TASK_NAME_LEN                 ( 12 )
#define configUSE_TRACE_FACILITY                0
#define configUSE_16_BIT_TICKS                  0
#define configUSE_MUTEXES                       1
#define configUSE_RECURSIVE_MUTEXES             1
#define configCHECK_FOR_STACK_OVERFLOW          0
#define configUSE_QUEUE_SETS                    1
#define configUSE_COUNTING_SEMAPHORES           1

#define configMAX_PRIORITIES                    ( 9UL )
#define configSUPPORT_STATIC_ALLOCATION         0

/* Timer related defines. */
#define configUSE_TIMERS                        0
#define configTIMER_TASK_PRIORITY               ( configMAX_PRIORITIES - 4 )
#define configTIMER_QUEUE_LENGTH                20
#define configTIMER_TASK_STACK_DEPTH            ( configMINIMAL_STACK_SIZE * 2 )
```

FREERTOS CONFIGURATION

https://github.com/FreeRTOS/FreeRTOS-Kernel.git

- Scheduler configuration
- Stack and heap configuration
- Mutexes and semaphores for synchronization
- Timers configuration

FreeRTOSConfig.h

```
#define configUSE_PREEMPTION                    1
#define configCPU_CLOCK_HZ                      ( ( unsigned long ) 25000000 )
#define configTICK_RATE_HZ                      ( ( TickType_t ) 1000 )
#define configMINIMAL_STACK_SIZE                ( ( unsigned short ) 80 )
#define configTOTAL_HEAP_SIZE                   ( ( size_t ) ( 60 * 1024 ) )
#define configMAX_TASK_NAME_LEN                 ( 12 )
#define configUSE_TRACE_FACILITY                0
#define configUSE_16_BIT_TICKS                  0
#define configUSE_MUTEXES                       1
#define configUSE_RECURSIVE_MUTEXES             1
#define configCHECK_FOR_STACK_OVERFLOW          0
#define configUSE_QUEUE_SETS                    1
#define configUSE_COUNTING_SEMAPHORES           1

#define configMAX_PRIORITIES                    ( 9UL )
#define configSUPPORT_STATIC_ALLOCATION         0

/* Timer related defines. */
#define configUSE_TIMERS                        0
#define configTIMER_TASK_PRIORITY               ( configMAX_PRIORITIES - 4 )
#define configTIMER_QUEUE_LENGTH                20
#define configTIMER_TASK_STACK_DEPTH            ( configMINIMAL_STACK_SIZE * 2 )
```

FREERTOS CONFIGURATION

https://github.com/FreeRTOS/FreeRTOS-Kernel.git

- Scheduler configuration
- Stack and heap configuration
- Mutexes and semaphores for synchronization
- Timers configuration
- Others

FreeRTOSConfig.h

```
#define configUSE_PREEMPTION                    1
#define configCPU_CLOCK_HZ                      ( ( unsigned long ) 25000000 )
#define configTICK_RATE_HZ                      ( ( TickType_t ) 1000 )
#define configMINIMAL_STACK_SIZE                ( ( unsigned short ) 80 )
#define configTOTAL_HEAP_SIZE                   ( ( size_t ) ( 60 * 1024 ) )
#define configMAX_TASK_NAME_LEN                 ( 12 )
#define configUSE_TRACE_FACILITY                0
#define configUSE_16_BIT_TICKS                  0
#define configUSE_MUTEXES                       1
#define configUSE_RECURSIVE_MUTEXES             1
#define configCHECK_FOR_STACK_OVERFLOW          0
#define configUSE_QUEUE_SETS                    1
#define configUSE_COUNTING_SEMAPHORES           1

#define configMAX_PRIORITIES                    ( 9UL )
#define configSUPPORT_STATIC_ALLOCATION         0

/* Timer related defines. */
#define configUSE_TIMERS                        0
#define configTIMER_TASK_PRIORITY               ( configMAX_PRIORITIES - 4 )
#define configTIMER_QUEUE_LENGTH                20
#define configTIMER_TASK_STACK_DEPTH            ( configMINIMAL_STACK_SIZE * 2 )
```

FREERTOS DEMOS

https://github.com/FreeRTOS/FreeRTOS/tree/main/FreeRTOS/Demo

- Architecture-specific or board-specific demos and examples
- Each demo project has its own FreeRTOSConfig.h
- Examples for different toolchains and compilation tools
- When developing code for a supported board:
  - Start from the demo
  - Copy the FreeRTOSConfig.h in your project
  - Customize it as you need

FIRST OPERATIONS WITH FREERTOS

- In a RTOS everything is based on **tasks**
- Depending on the context tasks can be calles **processes** as well
- Example of FreeRTOS function to interact with tasks:

  xTaskCreate --> create task

  vTaskDelete --> delete task

  vTaskDelay --> sleep for specified amount of time

  xSemaphore --> create semaphore for synchronization

CREATE TASK

```
BaseType_t xTaskCreate(
                TaskFunction_t                  pvTaskCode,
                const char * const              pcName,
                const configSTACK_DEPTH_TYPE    uxStackDepth,
                void                            *pvParameters,
                UBaseType_t                     uxPriority,
                TaskHandle_t                    *pxCreatedTask
);
```

CREATE TASK

```
BaseType_t xTaskCreate(


);
```

xTaskCreate

Type specifier. Examples

- x : FreeRTOS custom type (e.g. BaseType_t, TickType_t)
- px : pointer to FreeRTOS custom
- v : void
- pv : pointer to void
- us : unsigned short
- Others

CREATE TASK

```
BaseType_t xTaskCreate(
                TaskFunction_t                  pvTaskCode,
                const char * const              pcName,
                const configSTACK_DEPTH_TYPE    uxStackDepth,
                void                            *pvParameters,
                UBaseType_t                     uxPriority,
                TaskHandle_t                    *pxCreatedTask
);
```

- Pointer to the task handler: function which implements the task

CREATE TASK

```
BaseType_t xTaskCreate(
                TaskFunction_t                  pvTaskCode,
                const char * const              pcName,
                const configSTACK_DEPTH_TYPE    uxStackDepth,
                void                            *pvParameters,
                UBaseType_t                     uxPriority,
                TaskHandle_t                    *pxCreatedTask
);
```

- String which identifies the task name.
- Human readable identifiers, useful for debug

CREATE TASK

```
BaseType_t xTaskCreate(
                TaskFunction_t              pvTaskCode,
                const char * const          pcName,
                const configSTACK_DEPTH_TYPE uxStackDepth,
                void                        *pvParameters,
                UBaseType_t                 uxPriority,
                TaskHandle_t                *pxCreatedTask
);
```

- Size of the stack dedicated to the task

CREATE TASK

```
BaseType_t xTaskCreate(
                TaskFunction_t                  pvTaskCode,
                const char * const              pcName,
                const configSTACK_DEPTH_TYPE    uxStackDepth,
                void                            *pvParameters,
                UBaseType_t                     uxPriority,
                TaskHandle_t                    *pxCreatedTask
);
```

- Pointer to the parameter that will be passed to the task handler

CREATE TASK

```
BaseType_t xTaskCreate(
                TaskFunction_t                  pvTaskCode,
                const char * const              pcName,
                const configSTACK_DEPTH_TYPE    uxStackDepth,
                void                            *pvParameters,
                UBaseType_t                     uxPriority,
                TaskHandle_t                    *pxCreatedTask
);
```

- Priority of the task

CREATE TASK

```
BaseType_t xTaskCreate(
                TaskFunction_t                  pvTaskCode,
                const char * const              pcName,
                const configSTACK_DEPTH_TYPE    uxStackDepth,
                void                            *pvParameters,
                UBaseType_t                     uxPriority,
                TaskHandle_t                    *pxCreatedTask
);
```

- Pointer to the created task

FREERTOS HELLO WORLD

```c
#include "FreeRTOS.h"
#include "task.h"

#include "uart.h"

#define mainTASK_PRIORITY    ( tskIDLE_PRIORITY + 2 )

void vTaskFunction(void *pvParameters);

int main(int argc, char **argv){

    UART_init();

    xTaskCreate(
        vTaskFunction,
        "Task1",
        configMINIMAL_STACK_SIZE,
        NULL,
        mainTASK_PRIORITY,
        NULL
    );

    vTaskStartScheduler();

    for( ; ; );
}

void vTaskFunction(void *pvParameters) {

    for (;;) {

        UART_printf("Hello, World!\n");

        // Delay for 1 second
        vTaskDelay(pdMS_TO_TICKS(1000));
    }
}
```

FREERTOS HELLO WORLD

- Initialize hardware peripherals

```c
#include "FreeRTOS.h"
#include "task.h"

#include "uart.h"

#define mainTASK_PRIORITY    ( tskIDLE_PRIORITY + 2 )

void vTaskFunction(void *pvParameters);

int main(int argc, char **argv){

    UART_init();

    xTaskCreate(
        vTaskFunction,
        "Task1",
        configMINIMAL_STACK_SIZE,
        NULL,
        mainTASK_PRIORITY,
        NULL
    );

    vTaskStartScheduler();

    for( ; ; );
}

void vTaskFunction(void *pvParameters) {

    for (;;) {

        UART_printf("Hello, World!\n");

        // Delay for 1 second
        vTaskDelay(pdMS_TO_TICKS(1000));
    }
}
```

FREERTOS HELLO WORLD

- Initialize hardware peripherals
- Create task

```c
#include "FreeRTOS.h"
#include "task.h"

#include "uart.h"

#define mainTASK_PRIORITY    ( tskIDLE_PRIORITY + 2 )

void vTaskFunction(void *pvParameters);

int main(int argc, char **argv){

    UART_init();

    xTaskCreate(
        vTaskFunction,
        "Task1",
        configMINIMAL_STACK_SIZE,
        NULL,
        mainTASK_PRIORITY,
        NULL
    );

    vTaskStartScheduler();

    for( ; ; );
}

void vTaskFunction(void *pvParameters) {

    for (;;) {

        UART_printf("Hello, World!\n");

        // Delay for 1 second
        vTaskDelay(pdMS_TO_TICKS(1000));
    }
}
```

FREERTOS HELLO WORLD

- Initialize hardware peripherals
- Create task
- Task handler

```c
#include "FreeRTOS.h"
#include "task.h"

#include "uart.h"

#define mainTASK_PRIORITY    ( tskIDLE_PRIORITY + 2 )

void vTaskFunction(void *pvParameters);

int main(int argc, char **argv){

    UART_init();

    xTaskCreate(
        vTaskFunction,
        "Task1",
        configMINIMAL_STACK_SIZE,
        NULL,
        mainTASK_PRIORITY,
        NULL
    );

    vTaskStartScheduler();

    for( ; ; );
}

void vTaskFunction(void *pvParameters) {

    for (;;) {

        UART_printf("Hello, World!\n");

        // Delay for 1 second
        vTaskDelay(pdMS_TO_TICKS(1000));
    }
}
```

FREERTOS HELLO WORLD

- Initialize hardware peripherals
- Create task
- Task handler
- Give control to the scheduler

```c
#include "FreeRTOS.h"
#include "task.h"

#include "uart.h"

#define mainTASK_PRIORITY    ( tskIDLE_PRIORITY + 2 )

void vTaskFunction(void *pvParameters);

int main(int argc, char **argv){

    UART_init();

    xTaskCreate(
        vTaskFunction,
        "Task1",
        configMINIMAL_STACK_SIZE,
        NULL,
        mainTASK_PRIORITY,
        NULL
    );

    vTaskStartScheduler();

    for( ; ; );
}

void vTaskFunction(void *pvParameters) {

    for (;;) {

        UART_printf("Hello, World!\n");

        // Delay for 1 second
        vTaskDelay(pdMS_TO_TICKS(1000));
    }
}
```

FREERTOS HELLO WORLD

- Initialize hardware peripherals
- Create task
- Task handler
- Give control to the scheduler
- If everything is ok should never reach here

```c
#include "FreeRTOS.h"
#include "task.h"

#include "uart.h"

#define mainTASK_PRIORITY    ( tskIDLE_PRIORITY + 2 )

void vTaskFunction(void *pvParameters);

int main(int argc, char **argv){

    UART_init();

    xTaskCreate(
        vTaskFunction,
        "Task1",
        configMINIMAL_STACK_SIZE,
        NULL,
        mainTASK_PRIORITY,
        NULL
    );

    vTaskStartScheduler();

    for( ; ; );
}

void vTaskFunction(void *pvParameters) {

    for (;;) {

        UART_printf("Hello, World!\n");

        // Delay for 1 second
        vTaskDelay(pdMS_TO_TICKS(1000));
    }
}
```

USEFUL LINKS

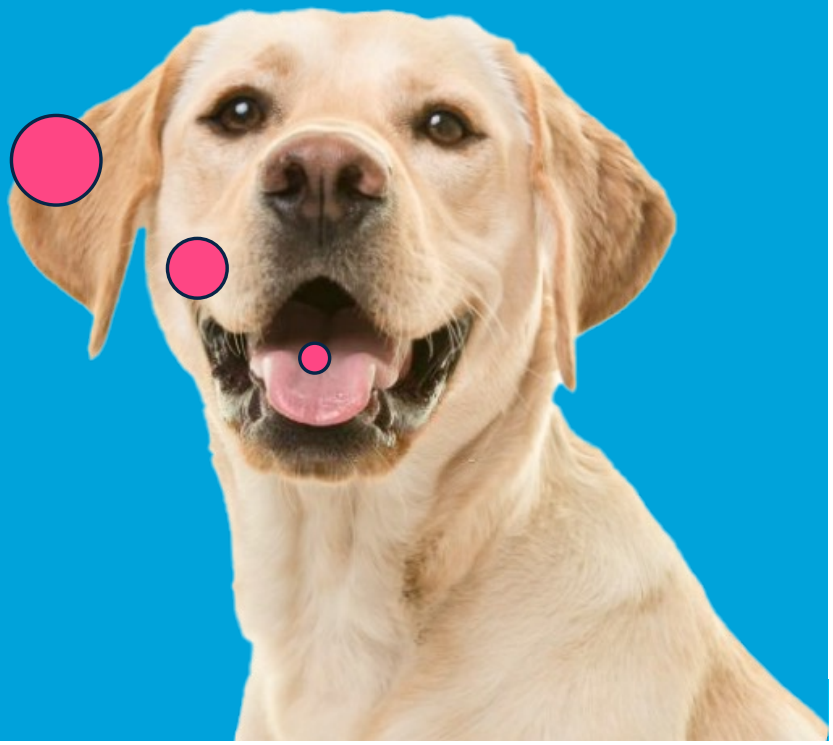
- FreeRTOS GitHub repo: https://github.com/FreeRTOS/FreeRTOS
- Reference manual: https://www.freertos.org/media/2018/FreeRTOS_Reference_Manual_V10.0.0.pdf
- Hands-on tutorial guide: https://github.com/FreeRTOS/FreeRTOS-Kernel-Book/releases/download/V1.1.0/Mastering-the-FreeRTOS-Real-Time-Kernel.v1.1.0.pdf

SMILIES
reSilient coMputer archItectures
and LIfE Sciences

Politecnico di Torino
Department of Control and Computer Engineering

DAUIN

QUESTIONS?



THANK YOU!