Real-Time Systems

Prof. Dr. rer. nat. Karsten Weronek

Prof. Dr.-Ing. Peter Tawdross

FreeRTOS

Fachbereich 2 Informatik und Ingenieurwissenschaften

Operating Systems

- Why do we need an operating system?
  - Abstraction of hardware
  - Scheduling
  - Resource management
  - Coordination of concurrent actions and events
  - Security

Real-Time Operating System (RTOS)

- Operating system especially for Real-Time Systems
  - **Meeting the deadlines**
  - Preemptive scheduling
  - Interrupt handling

- Example of RTOS
  - FreeRTOS, OSEK, QNX, VxWorks, LynxOS, RTLinux, Symbian, Windows CE

Real-Time Operating System
Difference between Real Time and non-Real-Time OS

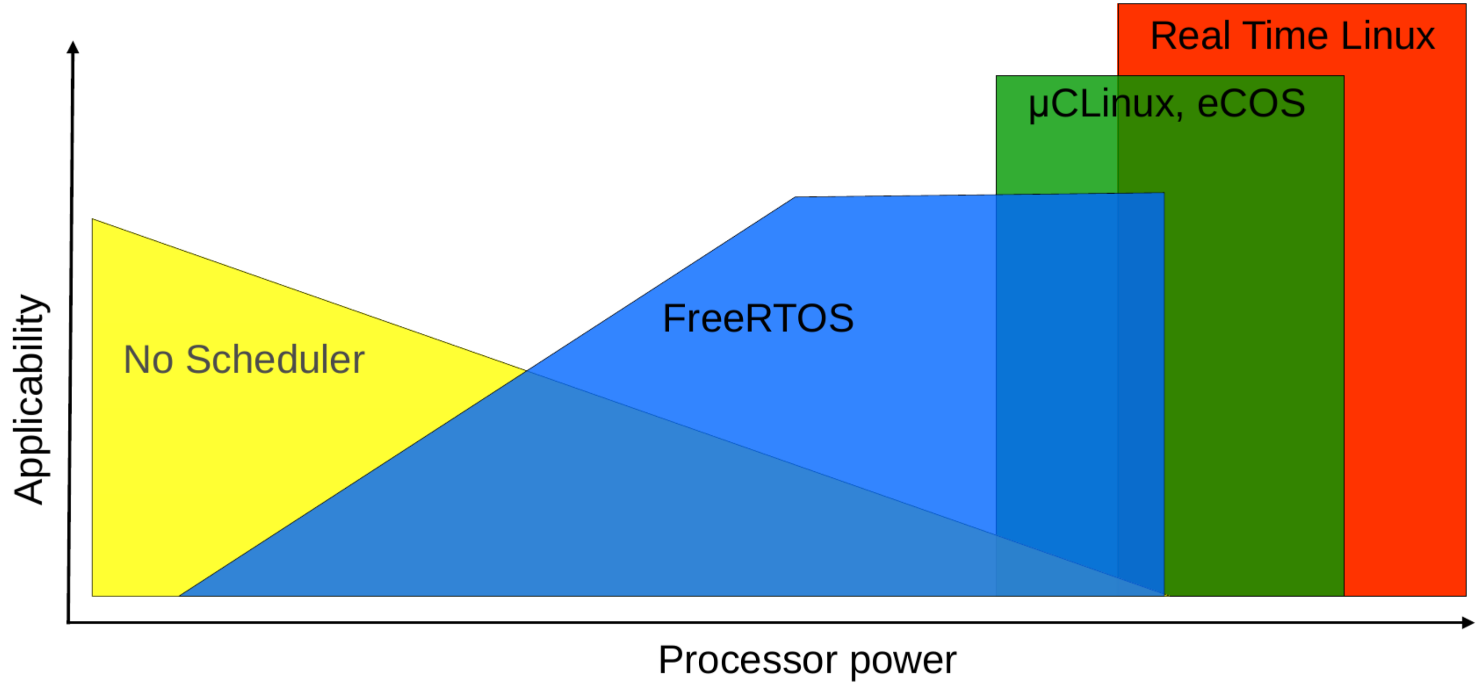
The main differences are:

	Non-Real Time OS	Real-Time
Application	Normal computers, etc.	Embedded systems, etc.
Target	Fairness, flexibility, scalability	Meet **deadlines**, consume **less resources**
	Optimized for **average cases**	Meet deadlines in **worst case**
Communication	**Maximize** average transfer rate	**Guaranty** for transfer rate
Latency	**Minimize** average latency	**Guaranty** for latency

Real-Time Operating System (RTOS)
freeRTOS

- FreeRTOS came top in class in every EETimes Embedded Market Survey since 2011
- Supports more then 35 architectures
- FreeRTOS is downloaded every 170 seconds (on average, during 2019)
- Opensource MIT license

Real-Time Operating System (RTOS)
Domain of FreeRTOS



Real-Time Operating System (RTOS)
freeRTOS Features

- Pre-emptive or co-operative operation
- Very flexible task priority assignment
- Flexible, fast and light weight task notification mechanism
- Queues
- Binary semaphores
- Counting semaphores
- Mutexes
- Recursive Mutexes
- Software timers
- … …

Real-Time Operating System (RTOS) Task Function

In RTOS, each task is executed by executing a task function

```
void vATaskFunction( void *pvParameters )
{
    for( ;; )
    {
        -- Task application code here. --
    }

    /* Tasks must not attempt to return from their implementing
    function or otherwise exit.  In newer FreeRTOS port
    attempting to do so will result in an configASSERT() being
    called if it is defined.  If it is necessary for a task to
    exit then have the task call vTaskDelete( NULL ) to ensure
    its exit is clean. */
    vTaskDelete( NULL );
}
```

[https://www.freertos.org/implementing-a-FreeRTOS-task.html]

Real-Time Operating System (RTOS)
Create a Task

- To create a task, the function xTaskCreate can be used

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

BaseType_t xTaskCreate( TaskFunction_t pvTaskCode,
                        const char * const pcName,
                        unsigned short usStackDepth,
                        void *pvParameters,
                        UBaseType_t uxPriority,
                        TaskHandle_t *pxCreatedTask );
```

- **Complete manual** for RTOS is found at the following link: https://www.freertos.org/fr-content-src/uploads/2018/07/FreeRTOS_Reference_Manual_V10.0.0.pdf

Real-Time Operating System (RTOS)
Example 1

```
void vTask1 ( void ∗ pvParameters ) ;
void vTask2 ( void ∗ pvParameters ) ;

void setup()
{
        ...
        xTaskCreate ( vTask1 , ”Task 1 ”, 100 , NULL, 1 , NULL ) ;
        xTaskCreate ( vTask2 , ”Task 2 ”, 100 , NULL, 1 , NULL ) ;
        // vTaskStartScheduler( ) ; //not needed in Arduino
}
```

Real-Time Operating System (RTOS) Example 1

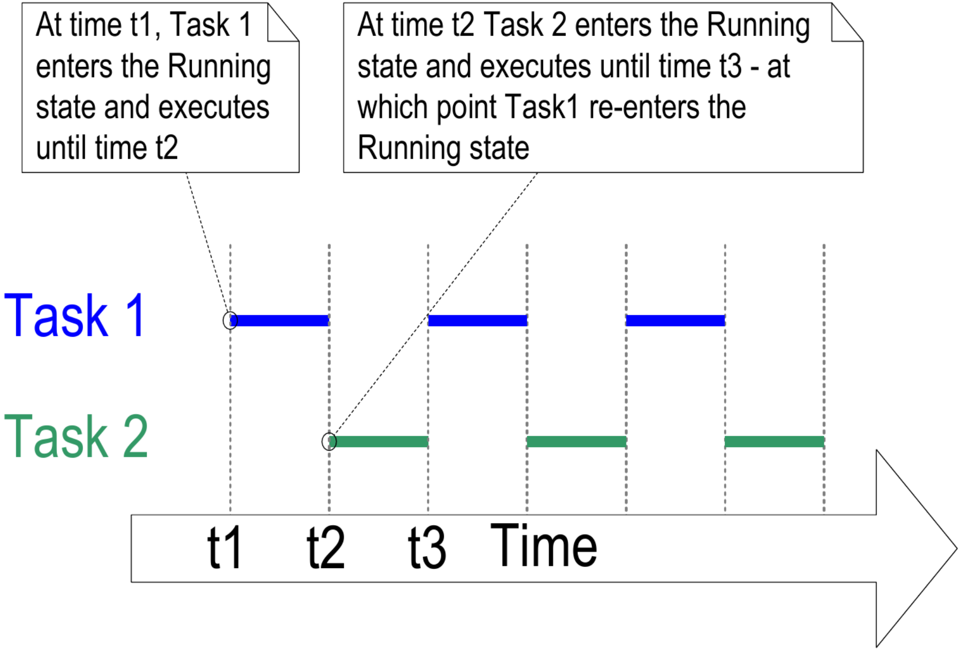
- **vTaskDelay** is used tell the kernel that this task will wait for a given number of ticks
  - In vTask1, **vTaskDelay** is waiting 100 ticks
  - In vTask2, the function **pdMS_TO_TICKS** is used to calculate how many ticks in 96 mSec before calling **vTaskDelay**.

```
void vTask1(void* pvParameters){
  (void) pvParameters;
  const char Task1String[]="Task1 is running \n\r";
   for (;;) // A Task shall never return or exit.
   {
       Serial.println(Task1String);

       vTaskDelay(100);
    }
}
void vTask2(void* pvParameters){
  (void) pvParameters;
  const char Task2String[]="Task2 is running \n\r";
   for (;;) // A Task shall never return or exit.
   {
       Serial.println(Task2String);

       vTaskDelay(pdMS_TO_TICKS(96));
    }
}
```

Real-Time Operating System (RTOS) freeRTOS Scheduling

- The **dispatchable task** with the **highest priority** will interrupt the other tasks

- **Time slicing** is used to **switch between multiple tasks** with the **same priority** that all want to own the processor ➔ each task gets a time slice



Real-Time Operating System (RTOS)
Task Stack

- Each Task needs **a stack to store its local data**
- This stack can be allocated either **statically or dynamically**
  - Dynamic stack allocation
    - The **operating system** automatically **allocates** it when the **task is created** and **deletes** it when the **task is deleted**
  - Static stack allocation
    - The stack is **created statically** and not deleted again
- In the **same design**, some tasks can be created with **dynamically** allocated stacks and others with **statically** allocated stacks
- **Most embedded system** applications use **statically allocated stacks** because there is no expectation that a task will be terminated

Real-Time Operating System (RTOS)
Task Stack

- In freeRTOS **xTaskCreateStatic** is used to create a new task is **static stack**
- The stack itself should be **allocated by the developer as a global variable**
    - Global variables are **mapped by the compiler**

```
TaskHandle_t xTaskCreateStatic( TaskFunction_t pxTaskCode,
                                const char * const pcName,
                                const uint32_t ulStackDepth,
                                void * const pvParameters,
                                UBaseType_t uxPriority,
                                StackType_t * const puxStackBuffer,
                                StaticTask_t * const pxTaskBuffer );
```

Real-Time Operating System (RTOS) Example 2

- To create two tasks with static stack
  - Before setup() …

```
#define Task1StackSize      100
#define Task2StackSize      100

StaticTask_t xTask1Buffer;
StaticTask_t xTask2Buffer;

StackType_t puxTask1StackBuffer[Task1StackSize];
StackType_t puxTask2StackBuffer[Task2StackSize];
```

- Creating the task

```
xTaskCreateStatic( vTask1 , "Task 1", Task1StackSize , NULL, 2 , puxTask1StackBuffer, &xTask1Buffer );
xTaskCreateStatic( vTask2 , "Task 2", Task2StackSize , NULL, 2 , puxTask2StackBuffer, &xTask2Buffer );
```

FRANKFURT UNIVERSITY OF APPLIED SCIENCES

Real-Time Operating System (RTOS) freeRTOS Create Task Using Parameters

- The task functions can also take parameters
  - e.g. the two task functionally in the previous example can **be combined to one function**

```
void vTask_Param(void* pvParameters){
  char *TaskName=(char*) pvParameters;
   for (;;) // A Task shall never return or exit.
   {
        Serial.println(TaskName);

        vTaskDelay(pdMS_TO_TICKS(96));
    }
}
```

- Advantage:
  - **Reduce code size**
  - **Code changes** are done in **one place → less effort**

Real-Time Operating System (RTOS)
freeRTOS Task Function Using Parameters

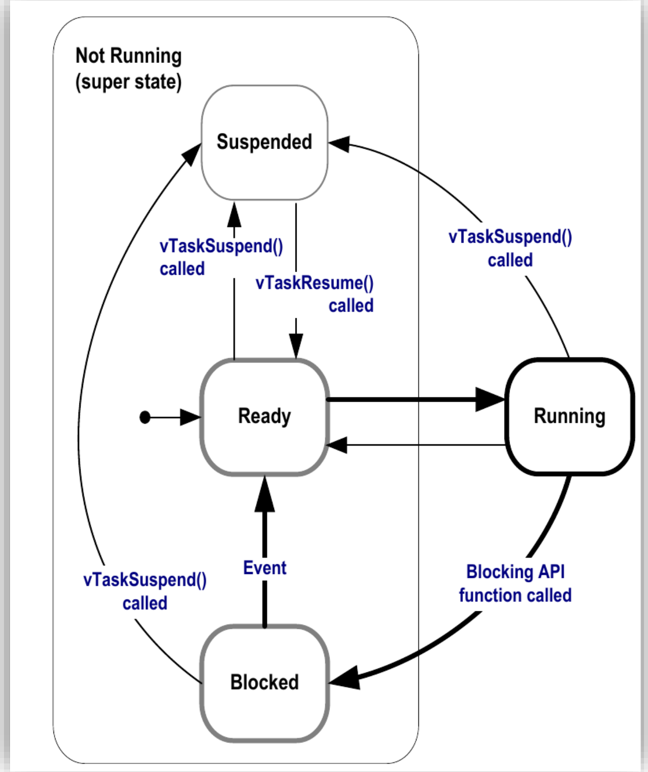
- In this case, both tasks can use the same function

```
static const char* pcTextForTask1 = "Task 1 is running";
static const char* pcTextForTask2 = "Task 2 is running";
```

```
xTaskCreateStatic( vTask_Param , "Task 1", Task1StackSize , pcTextForTask1,
        2 , puxTask1StackBuffer, &xTask1Buffer );
xTaskCreateStatic( vTask_Param , "Task 2", Task2StackSize , pcTextForTask2,
        2 , puxTask2StackBuffer, &xTask2Buffer );
```

Real-Time Operating System (RTOS) Task States

- Running:
  - The task is currently active
- Suspended:
  - Task is suspended by calling vTaskSuspend()
- Ready:
  - Task is ready to run
- Blocked:
  - Task is waiting for a resource or waiting for some time (ticks)

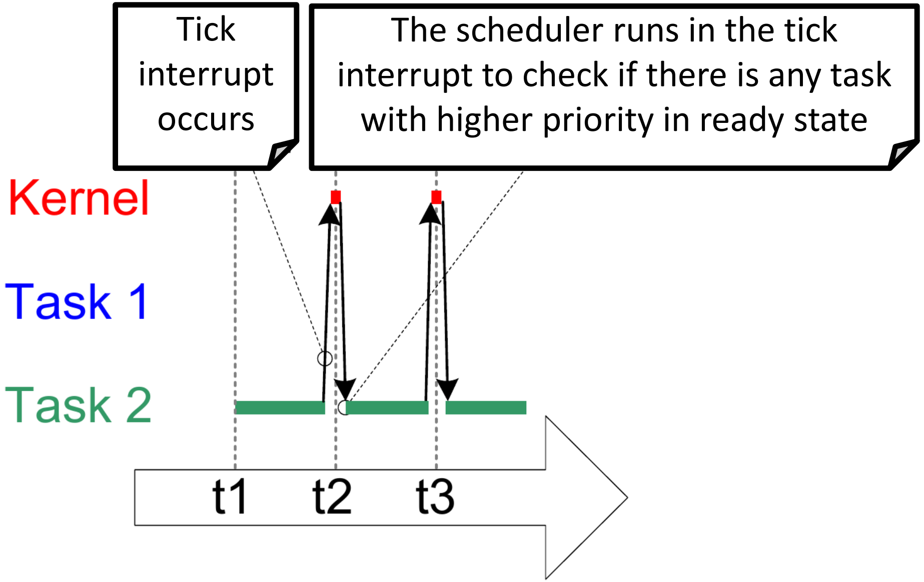


Real-Time Operating System (RTOS)
Task Priority

- The **number of task priorities** in freeRTOS is **configurable**
- Operating system **tick interrupt**
  - The **operating system** receives an **interrupt** every predefined **time interval** (e.g. every 1 ms) to check whether the **dispatchable task** with the **highest priority** can be activated
  - The **minimum tick** interrupt time interval in **freeRTOS** is **1 mSec**

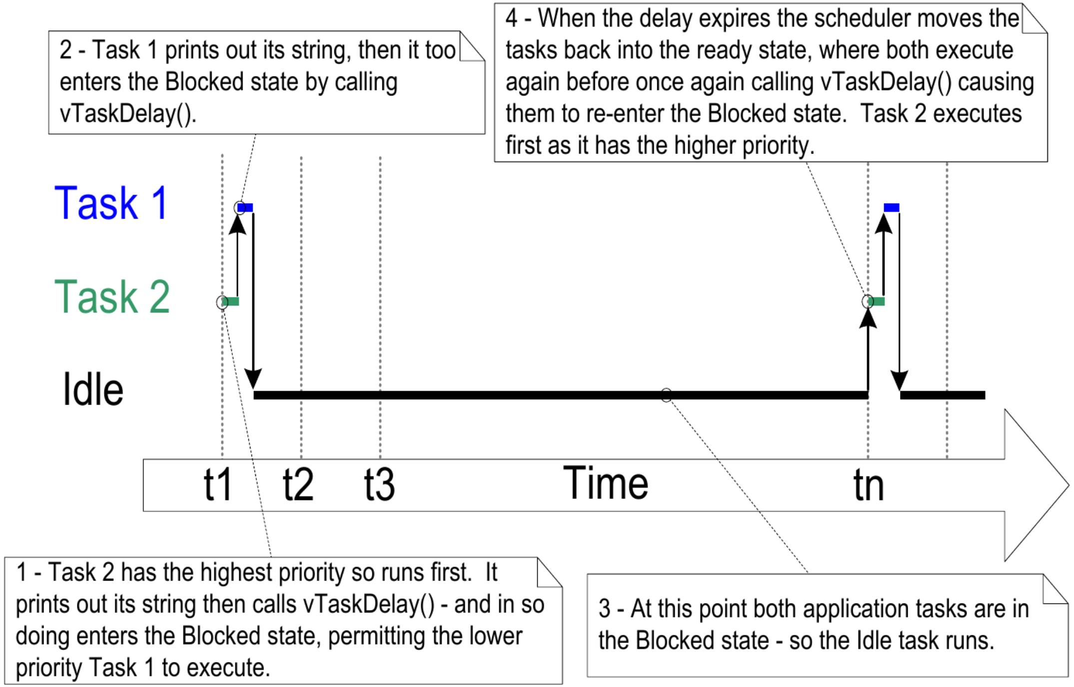
- In addition to the tick interrupt, the **scheduler also check** if a dispatchable task with higher priority can be activated **each time a task state is changed**

Real-Time Operating System (RTOS)
Task Priority - Example



Real-Time Operating System (RTOS)
freeRTOS vTaskDelay - scheduling

- **vTaskDelay** tells the scheduler that this **task needs to wait** for a certain **number of ticks**



FRANKFURT UNIVERSITY OF APPLIED SCIENCES

Real-Time Operating System (RTOS)
freeRTOS - Details

- FreeRTOSConfig.h
- Interrupt
- Memory Management
- Task functions
- Delay
- Statistic information

Real-Time Operating System (RTOS)
freeRTOS - FreeRTOSConfig.h - API Functions

```
/* Set the following definitions to 1 to include the
 API function, or zero to exclude the API function. */
#define INCLUDE_vTaskPrioritySet            1
#define INCLUDE_uxTaskPriorityGet           1
#define INCLUDE_vTaskDelete                 1
#define INCLUDE_vTaskCleanUpResources       0
#define INCLUDE_vTaskSuspend                1
#define INCLUDE_vTaskDelayUntil             0
#define INCLUDE_vTaskDelay                  1
#define INCLUDE_xTaskGetSchedulerState      1
```

Real-Time Operating System (RTOS)
freeRTOS - FreeRTOSConfig.h - Microcontroller Configs

```
#define configUSE_PREEMPTION                    1
#define configSUPPORT_STATIC_ALLOCATION         1
#define configSUPPORT_DYNAMIC_ALLOCATION        1
#define configUSE_IDLE_HOOK                     0
#define configUSE_TICK_HOOK                     0
#define configCPU_CLOCK_HZ                      ( SystemCoreClock )
#define configTICK_RATE_HZ                      ((TickType_t)1000)
#define configMAX_PRIORITIES                    ( 7 )
#define configMINIMAL_STACK_SIZE                ((uint16_t)128)
#define configTOTAL_HEAP_SIZE                   ((size_t)3072)
#define configMAX_TASK_NAME_LEN                 ( 16 )
#define configUSE_16_BIT_TICKS                  0
```

Real-Time Operating System (RTOS)
freeRTOS - FreeRTOSConfig.h – freeRTOS Configs

```c
#define configUSE_MUTEXES                       1
#define configQUEUE_REGISTRY_SIZE               8
#define configUSE_PORT_OPTIMISED_TASK_SELECTION 1

/* Co-routine definitions. */
#define configUSE_CO_ROUTINES                   0
#define configMAX_CO_ROUTINE_PRIORITIES         ( 2 )
```

Real-Time Operating System (RTOS)
freeRTOS - FreeRTOSConfig.h – freeRTOS Configs

- The number of **priorities** in freeRTOS is **limited to 32** if the configuration **configUSE_PORT_OPTIMISED_TASK_SELECTION** is **enabled**

```
#define configPRIO_BITS         4

/* The lowest interrupt priority that can be used in
 a call to a "set priority" function. */
#define configLIBRARY_LOWEST_INTERRUPT_PRIORITY   15
```

Real-Time Operating System (RTOS)
freeRTOS - FreeRTOSConfig.h - freeRTOS Interrupt

- **Interrupts** have **different priorities**
- Interrupts could be **disabled** (and **enabled**)
  - void taskDISABLE_INTERRUPTS(void)
  - void taskENABLE_INTERRUPTS(void)
- Lowest priority interrupt **interrupt any task** even the highest priority task as long as the interrupts are not disabled

Real-Time Operating System (RTOS)
freeRTOS - FreeRTOSConfig.h - freeRTOS Interrupt

- FreeRTOS **two groups** of interrupts:
  - Interrupts that can be **masked** by freeRTOS **critical sections**
  - Interrupts that are **not masked** by **critical sections**
- The boundary between these two groups is set by the macro **configMAX_SYSCALL_INTERRUPT_PRIORITY**
  - As example, `#define configMAX_SYSCALL_INTERRUPT_PRIORITY  5` means the **priorities 0 to 4 are not masked by critical sections**
- Only freeRTOS functions ends with …**_FROM_ISR**(…) are allowed to be called from interrupts, other freeRTOS functions can call issues if they are called from interrupt

Real-Time Operating System (RTOS)
freeRTOS - FreeRTOSConfig.h - freeRTOS Interrupt

- API for **critical sections** (no task switching during execution, only interrupts with priority 0 to configMAX_SYSCALL_INTERRUPT_PRIORITY are permitted)
  - void taskENTER_CRITICAL(void);
  - void taskEXIT_CRITICAL(void);
  - UBaseType_t taskENTER_CRITICAL_FROM_ISR(void);
  - void taskEXIT_CRITICAL_FROM_ISR(UBaseType_t uxSavedInterruptStatus);

Real-Time Operating System (RTOS)
freeRTOS - FreeRTOSConfig.h - freeRTOS Interrupt

- Example
  - ```
    #define configLIBRARY_LOWEST_INTERRUPT_PRIORITY   15
    #define configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY 5
    ```
  - This means we have minimum interrupt priority 15 (as defined in STM32f103VET6 datasheet) and interrupts 0 to 4 are not masked by OS critical sections
    ```

Real-Time Operating System (RTOS)
freeRTOS - Task Handle

- Methods for receiving a task handle:
  - TaskHandle_t xTaskGetCurrentTaskHandle(void)
  - TaskHandle_t TaskGetIdleTaskHandle(void)
  - TaskHandle_t xTaskGetHandle(const char *pcNameToQuery)
  - UBaseType_t uxTaskGetNumberOfTasks(void)
  - UBaseType_t uxTaskGetStackHighWaterMark(TaskHandle_t xTask)
  - eTaskState eTaskGetState(TaskHandle_t pxTask)
  - void vTaskGetTaskInfo(TaskHandle_t xTask, TaskStatus_t *pxTaskStatus, BaseType_t xGetFreeStackSpace, eTaskState eState)

Real-Time Operating System (RTOS)
freeRTOS - Task Handle

- Methods for receiving a task handle:

  …

  - void vTaskList(char* pcWriteBuffer)
    - example

```
Name            State   Priority  Stack   Num
*********************************************
Print           R       4         331     29
Math7           R       0         417     7
Math8           R       0         407     8
QConsB2         R       0         53      14
QProdB5         R       0         52      17
QConsB4         R       0         53      16
SEM1            R       0         50      27
SEM1            R       0         50      28
IDLE            R       0         64      0
Math1           R       0         436     1
Math2           R       0         436     2
```

[freeRTOS reference manual]

Real-Time Operating System (RTOS) freeRTOS –Statistical Information

- freeRTOS - Statistical information
  - void vTaskGetRunTimeStats(char* pcWriteBuffer)
    - Example:

```
Task              Abs Time         % Time
****************************************

uIP               12050            <1%
IDLE              587724           24%
QProdB2           2172             <1%
QProdB3           10002            <1%
QProdB5           11504            <1%
QConsB6           11671            <1%
PolSEM1           60033            2%
PolSEM2           59957            2%
IntMath           349246           14%
MuLow             36619            1%
GenQ              579715           24%
```

[freeRTOS reference manual]

  - BaseType_t xTaskGetSchedulerState(void)
  - TickType_t xTaskGetTickCount(void)

Interprocess Communication

- Interprocess communication is a set of **mechanisms** that are provided by the **operating system** to allow different processes to manage shared data and resources
- Examples
  - Semaphore
  - Message Queue
  - Stream Buffers

Task Synchronization
Semaphore

- Semaphore is a like a token
- A task can acquire semaphore
- Only the task with a semaphore can access a resource
- Otherwise, the task remains in the block state and can not execute
- As soon as the token is released it could be acquired by the other task

Semaphore
Free-RTOS Semaphore-API

- xSemaphoreCreateBinary()
- xSemaphoreCreateMutex()
  - Like binary semaphore but includes priority inheritance mechanism
- xSemaphoreTake()
- xSemaphoreGive()
- xSemaphoreCreateCounting()
- vSemaphoreDelete()
- uxSemaphoreGetCount()

Semaphore
Example Part1

```
…
SemaphoreHandle_t xSemaphore = NULL;
…
void main() {
    …
    xSemaphore = xSemaphoreCreateBinary();
    xTaskCreate( vTask1, "Task1", 100, NULL, 1, NULL );
    xTaskCreate( vTask2, "Task2", 100, NULL, 1, NULL );
    xSemaphoreGive( xSemaphore );
    vTaskStartScheduler();
}
```

Semaphore
Example Task1

```
void vTask1( void * pvParameters ) {
  while(1) {
    if( xSemaphore != NULL ) {
      if( xSemaphoreTake( xSemaphore, ( TickType_t ) 0 ) == pdPASS ) {
          for (uint8_t i = 0; i < 5; i++) {
            snprintf(cbuffer,30,“Task1:i=%d\n\r",i);
            Serial.printfln(cbuffer);
            vTaskDelay(100);
          }
          xSemaphoreGive( xSemaphore );
          vTaskDelay(1);
       }
    }
  }
}
```

Semaphore
Example Task2

```
void vTask2( void * pvParameters ) {
  while(1) {
    if( xSemaphore != NULL ) {
      if( xSemaphoreTake( xSemaphore, ( TickType_t ) 0 ) == pdPASS ) {
          for (uint8_t j = 9; j >=0; j--) {
            snprintf(cbuffer,30,“Task2:j=%d\n\r",j);
            Serial.printfln(cbuffer);
            vTaskDelay(50);
          }
          xSemaphoreGive( xSemaphore );
          vTaskDelay(1);
        }
    }
  }
}
```

Semaphore
Example Output

Task 1 : i = 0

Task 1 : i = 1

Task 1 : i = 2

Task 1 : i = 3

Task 1 : i = 4

Task 2 : j = 9

Task 2 : j = 8

Task 2 : j = 7

Task 2 : j = 6

Task 2 : j = 5

Task 2 : j = 4

Task 2 : j = 3

Task 2 : j = 2

Task 2 : j = 1

Task 2 : j = 0

Task 1 : i = 0

Task 1 : i = 1

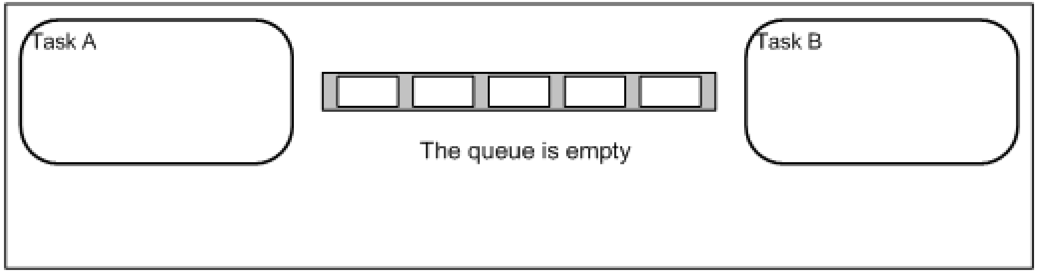
Interprocess Communication
Message Queue

- Message queue can be used to send data from one task to another
  - Messages are stored until they are read by a task
  - Message queue preserve the message order
    - New message can be inserted to the front (xQueueSendToFront) or to the back of the queue (freeRTOS: xQueueSendToBack)
- Each Queue in FreeRTOS:
  - Consists of a predefined number of message entries
  - All messages have same predefined size
    - Can be a variable, array, structure, …

FRANKFURT UNIVERSITY OF APPLIED SCIENCES

Interprocess Communication
Message Queue

- Advantage over other interprocess communication mechanisms:
  - Messages are queued in order
  - Messages not lost



https://www.freertos.org/Embedded-RTOS-Queues.html

Interprocess Communication
Message Queue

- Creating a message
  - QueueHandle_t xQueueCreate(UBaseType_t uxQueueLength,
    UBaseType_t uxItemSize);

  - QueueHandle_t xQueueCreateStatic(UBaseType_t uxQueueLength,
    UBaseType_t uxItemSize,
    uint8_t *pucQueueStorageBuffer,
    StaticQueue_t *pxQueueBuffer);

Message Queue
Sending Item to a Queue

- Sending Item to the back of a queue
  - BaseType_t **xQueueSendToBack**(QueueHandle_t xQueue,
    const void * pvItemToQueue,
    TickType_t xTicksToWait);
  - BaseType_t **xQueueSendToBackFromISR**(…);

- Sending Item to the front of a queue
  - BaseType_t **xQueueSendToFront**(…);
  - BaseType_t **xQueueSendToFrontFromISR**(…);

Message Queue
Sending Item to the Queue

- **Overwriting** the last item in a Queue
  - BaseType_t **xQueueOverwrite**(QueueHandle_t xQueue,
    const void * pvItemToQueue,
    BaseType_t *pxHigherPriorityTaskWoken);

  - BaseType_t **xQueueOverwriteFromISR** (QueueHandle_t xQueue,
    const void * pvItemToQueue,
    BaseType_t *pxHigherPriorityTaskWoken);

Message Queue
Receiving Item from Queue

- Receiving Item from a Queue
  - BaseType_t **xQueueReceive(QueueHandle_t xQueue,**
    void *pvBuffer,
    TickType_t xTicksToWait);
  - BaseType_t **xQueueReceiveFromISR**(…)

- Receiving Item from the Queue **without consuming** it
  - BaseType_t **xQxQueuePeek**(…)
  - BaseType_t **xQueuePeekFromISR**(…)

Message Queue
Other Functions

- BaseType_t **xQueueIsQueueFullFromISR**(const QueueHandle_t xQueue);
- BaseType_t **xQueueIsQueueEmptyFromISR**(const QueueHandle_t xQueue);
- BaseType_t **xQueueReset**(QueueHandle_t xQueue);
- void **vQueueDelete**(QueueHandle_t xQueue);