# 1. Теоретическую часть:

## ○ Принцип работы FreeRTOS;

FreeRTOS – многозадачная операционная система реального времени для встраиваемых систем. За переключение между задачами отвечает диспетчер, работающий по прерыванию от таймера. Задача выглядит как обычная функция Си, которая выполняет некоторое действие в бесконечном цикле.

## ○ Назначение и применение FreeRTOS;

The FreeRTOS kernel is a real-time operating system that supports numerous architectures. It is ideal for building embedded microcontroller applications. It provides:

* A multitasking scheduler.
* Multiple memory allocation options (including the ability to create completely statically-allocated systems).
* Intertask coordination primitives, including task notifications, message queues, multiple types of semaphore, and stream and message buffers.

The FreeRTOS kernel never performs non-deterministic operations, such as walking a linked list, inside a critical section or interrupt. The FreeRTOS kernel includes an efficient software timer implementation that does not use any CPU time unless a timer needs servicing. Blocked tasks do not require time-consuming periodic servicing. Direct-to-task notifications allow fast task signaling, with practically no RAM overhead. They can be used in most intertask and interrupt-to-task signaling scenarios.

## ○ Преимущества и недостатки FreeRTOS;

Преимущества:

* многозадачность;
* временная база для измерения интервалов времени;
* обмен данными между задачами через очередь и синхронизацию.

Недостатки:

* увеличение потребной памяти программ для реализации ядра;
* большее количество памяти для хранения стека каждой задачи, семафоров, очередей, мьютексов и других объектов ядра системы;
* задержки при переключении между задачами на сохранение контекста.

## ○ Примитивы синхронизации;

Synchronization primitives are simple software mechanisms for the purposes of supporting thread or process synchronization. Mutex, event, conditional variables and semaphores are all synchronization primitives.

## ○ Кооперативную и вытесняющую многозадачность;

Кооперативная многозадачность — тип многозадачности, при котором следующая задача выполняется только после того, как текущая задача явно объявит себя готовой отдать процессорное время другим задачам.

Вытесняющая многозадачность — это вид многозадачности, при которой операционная система может временно прервать текущий процесс без какой-либо помощи с его стороны.

## ○ Принцип действия планировщика задач;

To provide deterministic real-time behavior, the FreeRTOS tasks scheduler allows tasks to be assigned strict priorities. RTOS ensures the highest priority task that is able to execute is given processing time. This requires sharing processing time between tasks of equal priority if they are ready to run simultaneously. FreeRTOS also creates an idle task that executes only when no other tasks are ready to run.

## ○ Состояния задачи;

A task can exist in one of the following states:

* **Running**

task is executing. If the processor on which the RTOS is running only has a single core then there can only be one task in the Running state at any given time.

* **Ready**

tasks are able to execute (they are not in the Blocked or Suspended state) but are not currently executing because a different task of equal or higher priority is already in the Running state.

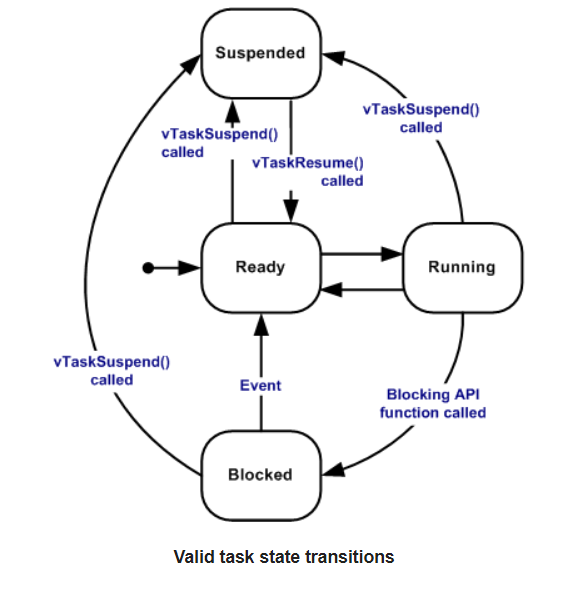
* **Blocked**

A task is waiting for either a temporal or external event. For example, if a task calls vTaskDelay() it will block (be placed into the Blocked state) until the delay period has expired – a temporal event. Tasks can also block to wait for queue, semaphore, event group, notification or semaphore event. Tasks in the Blocked state normally have a ‘timeout’ period, after which the task will be timeout, and be unblocked, even if the event the task was waiting for has not occurred.

Tasks in the Blocked state do not use any processing time and cannot be selected to enter the Running state.

* **Suspended**

Like tasks that are in the Blocked state, tasks in the Suspended state cannot be selected to enter the Running state, but tasks in the Suspended state do not have a time out. Instead, tasks only enter or exit the Suspended state when explicitly commanded to do so through the vTaskSuspend() and xTaskResume() API calls respectively.



## ○ Квант времени работы планировщика;

квант времени работы планировщика (tick) — это жестко фиксированный отрезок времени, в течение которого планировщик не вмеши­вается в выполнение задачи. По истечении кванта времени планировщик получает воз­можность приостановить текущую задачу и возобновить следующую, готовую к выполнению.

## ○ Приоритет задачи.

Each task is assigned a priority from 0 to ( configMAX\_PRIORITIES – 1 ), where configMAX\_PRIORITIES is defined within FreeRTOSConfig.h.

Low priority numbers denote low priority tasks. The [idle task](https://www.freertos.org/RTOS-idle-task.html) has priority zero (tskIDLE\_PRIORITY).

The FreeRTOS scheduler ensures that tasks in the Ready or Running [state](https://www.freertos.org/RTOS-task-states.html) will always be given processor (CPU) time in preference to tasks of a lower priority that are also in the ready state. In other words, the task placed into the Running state is always the highest priority task that is able to run.

Any number of tasks can share the same priority. If configUSE\_TIME\_SLICING is not defined, or if configUSE\_TIME\_SLICING is set to 1, then Ready state tasks of equal priority will share the available processing time using a time sliced round robin scheduling scheme.

# 2. Программную часть:

## ○ Установку и использование FreeRTOS в STM32CubeMX и в STM32CubeIDE с использованием фреймворка HAL;

## ○ Структуру проекта, использующего операционную систему FreeRTOS.

**if (xSemaphoreTake(myBinarySem01Handle, portMAX\_DELAY))** : pdTRUE if the semaphore was obtained. pdFALSE if xTicksToWait expired without the semaphore becoming available.

**xSemaphoreGive()** release a semaphore.

## 1. How FreeRTOS works

FreeRTOS is a real-time multitasking operating system for embedded systems. Switching between tasks is the responsibility of a dispatcher working on a timer interrupt. The task looks like a regular C function that performs some action in an infinite loop.

## 2. Purpose and application of FreeRTOS

The FreeRTOS kernel is a real-time operating system that supports multiple architectures. It is ideal for building applications for embedded microcontrollers. This provides:

• Scheduler for multitasking.

• Multiple options for memory allocation (including the ability to create fully statically distributed systems).

• Cross-task coordination primitives, including task notifications, message queues, several types of semaphores, and stream and message buffers.

The FreeRTOS kernel never performs non-deterministic operations such as traversing a linked list inside a critical section or interrupting. The FreeRTOS kernel includes an efficient software timer implementation that does not use CPU time unless the timer requires maintenance. Locked tasks do not require time-consuming periodic maintenance. Direct task notifications allow you to quickly signal tasks, with little or no memory overhead. They can be used in most inter-task signaling and interrupt scenarios.

## 3. Advantages and disadvantages of FreeRTOS

Benefits:

• multitasking;

• time base for measuring time intervals;

• data exchange between tasks via queue and synchronization.

Disadvantages:

• increasing the required program memory for the implementation of the kernel;

• more memory for storing the stack of each task, semaphores, queues, mutexes and other objects of the system kernel;

• delays when switching between tasks to save the context.

## 4. Synchronization primitives

Synchronization primitives are simple software mechanisms designed to support synchronization of threads or processes. Mutex, event, condition variables, and semaphores are all synchronization primitives.

## 5. Cooperative and preemptive multitasking

Cooperative multitasking is a type of multitasking in which the next task is executed only after the current task has explicitly declared itself ready to devote CPU time to other tasks.

Preemptive multitasking is a type of multitasking in which the operating system can temporarily interrupt the current process without any assistance from it.

## 6. How the task scheduler works

To provide deterministic real-time behavior, FreeRTOS Task Scheduler allows you to assign strict priorities to tasks. RTOS ensures that the highest priority task that can be completed receives processing time. This requires splitting processing time between tasks of equal priority if they are ready to work at the same time. FreeRTOS also creates an idle task that only runs when other tasks are not ready to run.

## 7. Task states

A task can exist in one of the following states:

• Running: task is executing. If the processor on which the RTOS is running only has a single core then there can only be one task in the Running state at any given time.

• Ready: tasks are able to execute (they are not in the Blocked or Suspended state) but are not currently executing because a different task of equal or higher priority is already in the Running state.

• Blocked: A task is waiting for either a temporal or external event. For example, if a task calls vTaskDelay () it will block (be placed into the Blocked state) until the delay period has expired - a temporal event. Tasks can also block to wait for queue, semaphore, event group, notification or semaphore event. Tasks in the Blocked state normally have a 'timeout' period, after which the task will be timeout, and be unblocked, even if the event the task was waiting for has not occurred.

Tasks in the Blocked state do not use any processing time and cannot be selected to enter the Running state.

• Suspended: Like tasks that are in the Blocked state, tasks in the Suspended state cannot be selected to enter the Running state, but tasks in the Suspended state do not have a time out. Instead, tasks only enter or exit the Suspended state when explicitly commanded to do so through the vTaskSuspend () and xTaskResume () API calls respectively.

## 8. Time quantum of the scheduler

A scheduler time quantum (tick) is a rigidly fixed amount of time during which the scheduler does not interfere with the execution of a task. When the time slice expires, the scheduler is able to suspend the current task and resume the next one ready to run.

## 9. Task priority

Each task is assigned a priority from 0 to (configMAX\_PRIORITIES - 1), where configMAX\_PRIORITIES is defined in FreeRTOSConfig.h.

Low priority numbers indicate low priority tasks. The inactive task has a priority of zero (tskIDLE\_PRIORITY).

The FreeRTOS scheduler ensures that tasks in a ready or running state will always be allocated processor time (CPU), rather than lower priority tasks that are also in a ready state. In other words, a task that is in the Running state is always the highest priority task that can run.

Any number of tasks can have the same priority. If configUSE\_TIME\_SLICING is not defined or configUSE\_TIME\_SLICING is set to 1, then ready state tasks with equal priority will share the available processing time using a time-sliced ​​scheduling scheme.