Лабораторная работа № 19.
Тема: Операционные системы реального времени.

Цели работы:

- Изучить основные принципы работы операционных систем реального времени (ОСРВ).
- Научиться использовать ОСРВ для разработки приложений, работающих в реальном времени.
- Реализовать на практике проект, демонстрирующий применение ОСРВ для управления внешними устройствами.

Оборудование:

- Компьютер с Arduino IDE и Proteus

Основные понятия RTOS (Операционной системы реального времени)

RTOS (Real-Time Operating System) - это операционная система, предназначенная для выполнения задач в **режиме реального времени**. Это означает, что RTOS **гарантирует** выполнение задач в **строго определенные моменты времени**, независимо от других факторов.

Основные понятия RTOS:

1. Задачи:

- **Базовая единица** выполнения в RTOS.
- **Содержит** код, который необходимо выполнить.
- **Может** иметь **приоритет**, определяющий ее **важность** относительно других задач.
- **Запускаются** и **останавливаются** диспетчером задач.

2. Диспетчер задач:

- **Ядро** RTOS, которое отвечает за **выбор** и **запуск** задач.
- Использует **алгоритм планирования**, чтобы задачи выполнялись **справедливо** и **эффективно**.
- **Обеспечивает контекстное переключение** между задачами.

3. Очереди:

- **Структуры данных**, используемые для **хранения** элементов в **определенном порядке**.
- **Элементы** очереди могут быть **задачами**, **данными** или **сообщениями**.
- **Очереди** используются для **синхронизации** задач и **передачи** данных между ними.

4. Семафоры:

- **Механизмы** синхронизации, которые **регулируют** доступ к **общим ресурсам**.
- **Обеспечивают исключительный** доступ к ресурсу одной задаче в **определенный момент времени**.
- **Предотвращают** конфликты при **доступе** к **общим ресурсам**.

5. Приоритеты задач:

- **Уровни** важности задач.
- **Задают** порядок выполнения задач диспетчером задач.
- **Задачи** с **более высоким** приоритетом **выполняются первыми**.

Пример:

- **Задача** с **высоким** приоритетом **считывает** данные с датчика.
- **Задача** с **низким** приоритетом **отображает** данные на дисплее.

Диспетчер задач будет **сначала выполнять** задачу с **высоким** приоритетом, чтобы **гарантировать своевременное считывание** данных с датчика.

Структура программы с использованием FreeRTOS обычно включает в себя следующие основные элементы:

Включение заголовочных файлов: Для использования FreeRTOS необходимо включить соответствующие заголовочные файлы в ваш проект.

```
#include <Arduino_FreeRTOS.h>
#include <task.h>
#include <semphr.h>
#include <queue.h>
```

Определение задач (Tasks): Задачи являются основными исполняемыми элементами программы в FreeRTOS. Каждая задача обычно представлена как бесконечный цикл.

```
void TaskFunction(void *pvParameters) {
  while(1) {
    // Код задачи здесь
  }
}
```

Создание задач: Задачи создаются с использованием функции xTaskCreate() в функции setup(). Параметры задачи включают в себя имя, размер стека, приоритет и т. д.

```
xTaskCreate(TaskFunction, "TaskName", 128, NULL, 1, NULL);
```

Семафоры (Semaphores): Семафоры используются для синхронизации доступа к общим ресурсам между задачами.

```
SemaphoreHandle_t xSemaphore;
xSemaphore = xSemaphoreCreateBinary();
```

Очереди (Queues): Очереди используются для передачи данных между задачами.

```
QueueHandle_t xQueue;
xQueue = xQueueCreate(10, sizeof(int));
```

Задержка выполнения задачи: Временная задержка между выполнением задач может быть обеспечена с помощью функции vTaskDelay().

```
vTaskDelay(1000 / portTICK_PERIOD_MS); // Задержка на 1 секунду
```

Операции над семафорами и очередями: Для захвата, освобождения семафоров и операций чтения/записи в очереди используются соответствующие

функции, такие как xSemaphoreTake(), xSemaphoreGive(), xQueueSend() и xQueueReceive().
Остановка планировщика: В режиме остановки планировщика все задачи приостанавливаются, и управление возвращается в функцию loop().

```
vTaskSuspendAll(); // Остановить все задачи
vTaskResumeAll();  // Возобновить все задачи
```

Пример:

```
#include <Arduino_FreeRTOS.h>
#include <semphr.h> //Подключаем заголовочный файл для работы с семафорами.

SemaphoreHandle_t xSemaphore; // Переменная для хранения семафора

// Определяем функцию задачи, которая не имеет параметров.
void TaskFunction(void *pvParameters) {
  while(1) {
    // Пытается "занять" семафор xSemaphore
    // portMAX_DELAY: Указывает, что задача должна блокироваться до тех пор,
    // пока семафор не станет доступным
    // Функция возвращает pdTRUE, если семафор был успешно "взят", и
pdFALSE
    // в противном случае.
    if (xSemaphoreTake(xSemaphore, portMAX_DELAY) == pdTRUE) {
    // Критическая секция
    // выполняться тогда, когда задача "владеет" семафором.

      xSemaphoreGive(xSemaphore); // "Освобождает" семафор, делая его
                                  // доступным для других
задач.
    }
    // Задерживает выполнение задачи на заданное время.
    vTaskDelay(1000 / portTICK_PERIOD_MS);
  }
}
// функция setup, выполняется один раз при запуске программы.
void setup() {
  xSemaphore = xSemaphoreCreateBinary(); // Создаем бинарный семафор
    // Создает задачу TaskFunction с именем "Task".
  xTaskCreate(TaskFunction, "Task", 128, NULL, 1, NULL);
  // 128: Указывает размер стека задачи в байтах.
    // NULL: Указывает, что задача не имеет параметров.
    // 1: Приоритет задачи.
    // NULL: Указывает, что у задачи нет контекстного хранилища.
}

void loop() {
  // Основной код программы
}
```

Пояснения к коду:

- Семафор xSemaphore используется для синхронизации доступа к **общему ресурсу**.
- Задача TaskFunction **периодически** пытается "занять" семафор.
- Если семафор доступен, задача получает **исключительный доступ** к ресурсу (**критическая секция**).
- После выполнения действий в критической секции задача "освобождает" семафор, делая его доступным для других задач.
- Функция main() создает задачу TaskFunction и инициализирует семафор.
- Функция loop() содержит основной код программы, который выполняется **независимо** от задачи.

Структура программы с FreeRTOS без семафора:

Пример:

```
#include <Arduino_FreeRTOS.h>

void Task1(void *pvParameters) {
  while (1) {
    // Выполняем код задачи 1
    Serial.println("Task 1: Hello!");
    vTaskDelay(1000); // Задержка 1 секунды
  }
}

void Task2(void *pvParameters) {
  while (1) {
    // Выполняем код задачи 2
    Serial.println("Task 2: World!");
    vTaskDelay(500); // Задержка 500 мс
  }
}

void setup() {
  // Инициализация FreeRTOS
  vTaskStartScheduler();
}

void loop() {
  // Не достигается, так как задачи выполняются бесконечно
}
```

Пояснение:

Заголовочный файл: #include <Arduino_FreeRTOS.h> - подключает библиотеку FreeRTOS для Arduino.

Функции задач:

Task1(): Выводит сообщение "Task 1: Hello!" в Serial каждые 1000 мс.

Task2(): Выводит сообщение "Task 2: World!" в Serial каждые 500 мс.

Функция setup: Запускает планировщик задач FreeRTOS.

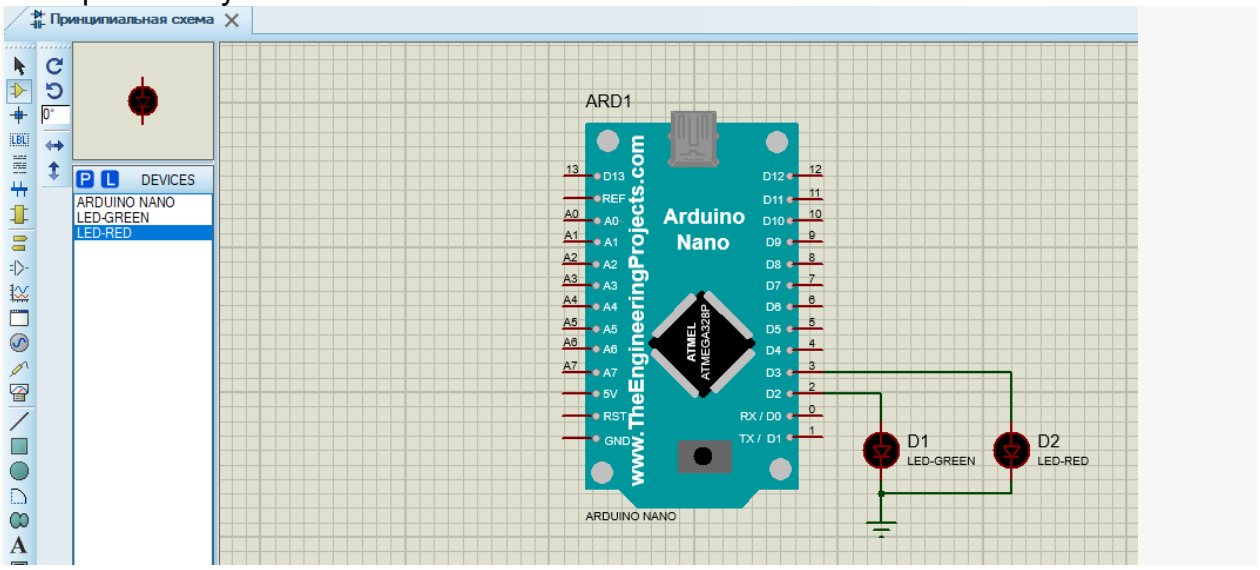
Функция loop: Не достигается, так как задачи выполняются бесконечно в своих циклах.

В этой программе:
Диспетчер задач FreeRTOS распределяет время выполнения между задачами. Задачи не используют семафоры, поэтому не синхронизируют доступ к ресурсам. Выводимые сообщения могут перекрываться, так как задачи выполняются одновременно.

Важно:
Использование семафоров необходимо, если задачи должны синхронизировать доступ к общим ресурсам. Синхронизация предотвращает конфликты и повреждение данных.

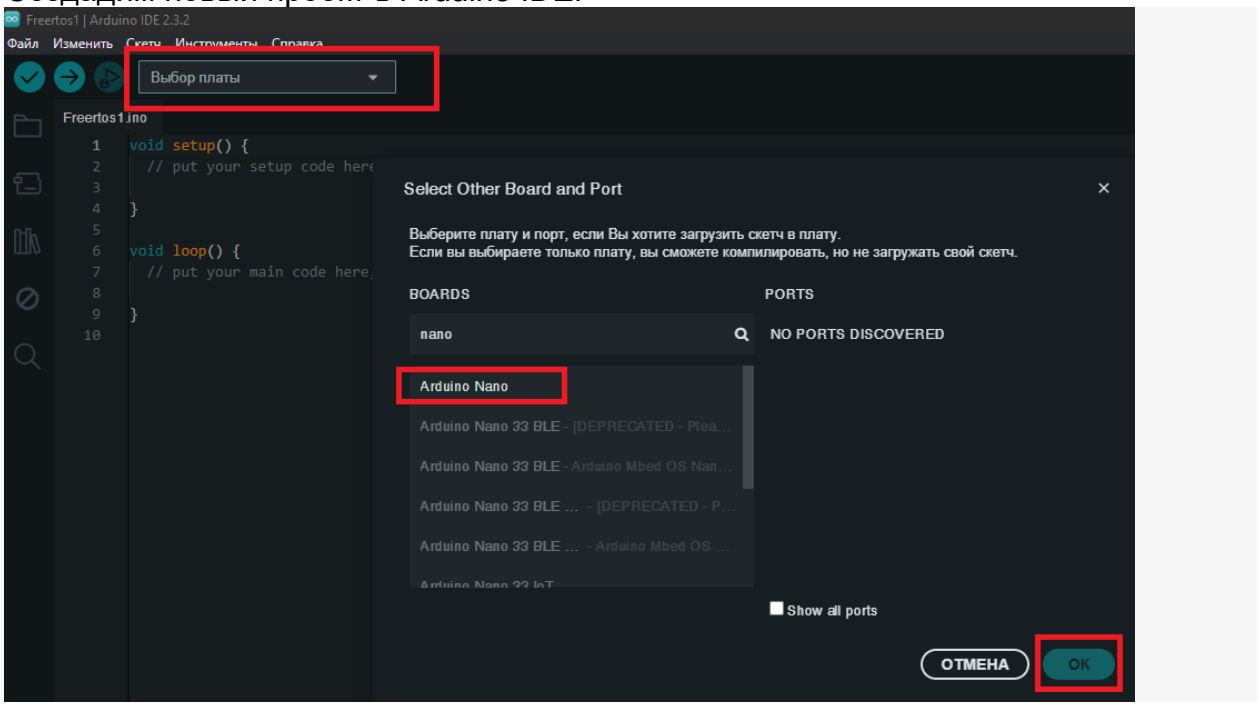
Пример использования FreeRTOS.

Соберем схему в Proteus:

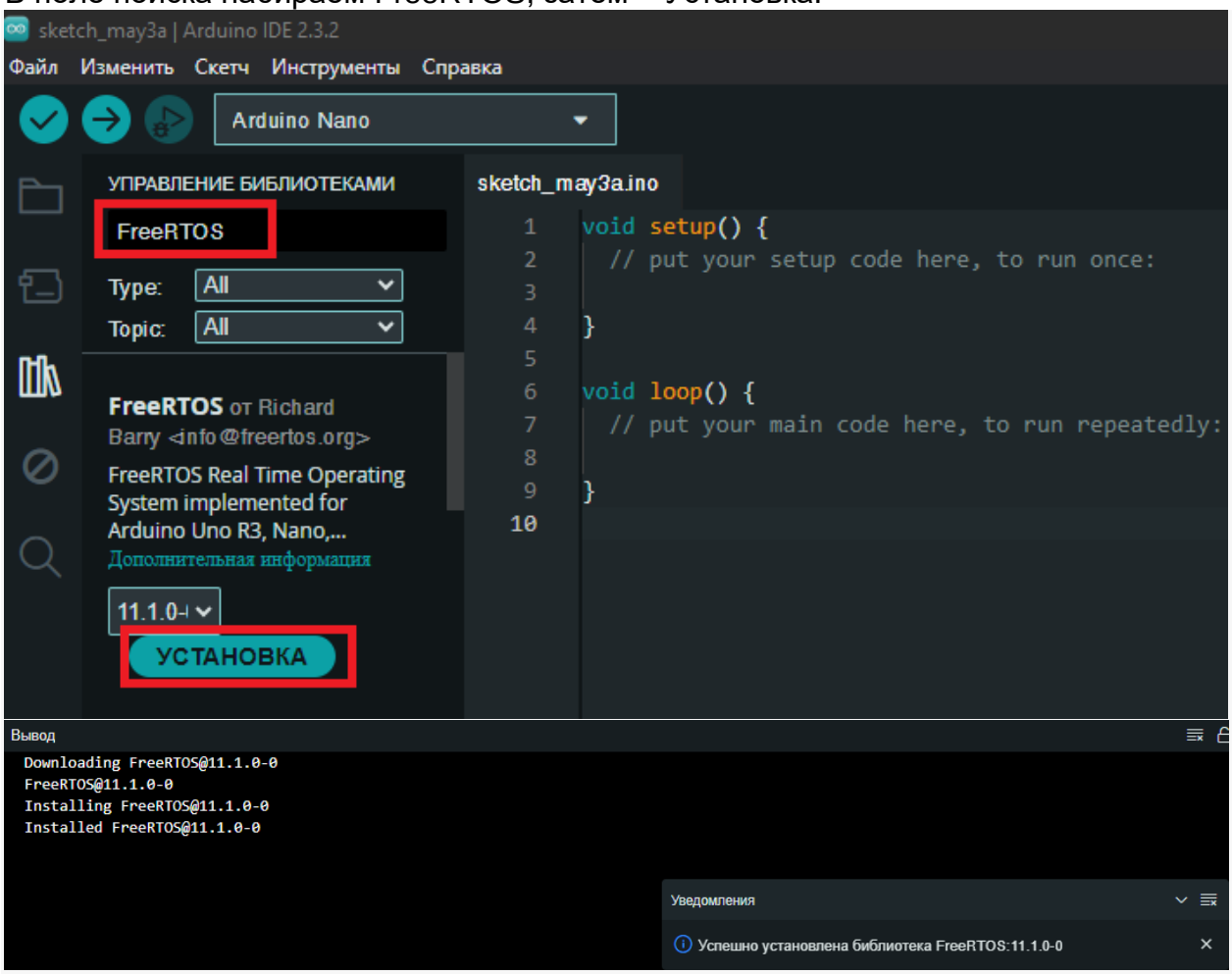


К Arduino Nano подключены два светодиода, задача – заставить их мигать одновременно, но с разной частотой.

Создадим новый проект в Arduino IDE:



Далее нужно добавить в проект библиотеку FreeRTOS.
Скетч => Подключить библиотеку => Управление библиотеками.
В поле поиска набираем FreeRTOS, затем – Установка:

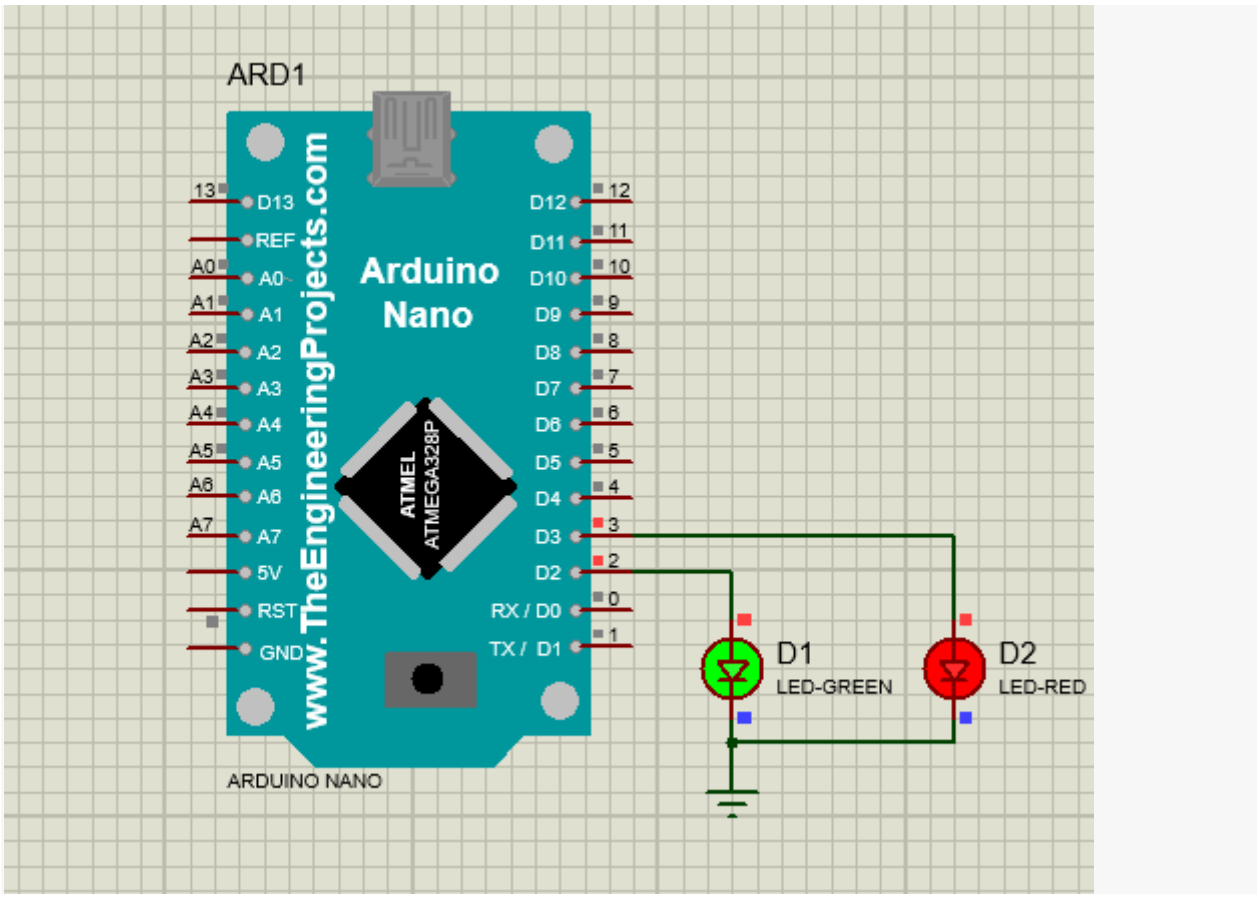


Добавим код и скомпилируем:

```
#include <Arduino_FreeRTOS.h>

// Определите пины, к которым подключены светодиоды
#define LED1_PIN 2
#define LED2_PIN 3

// Прототипы функций для задач FreeRTOS
void TaskBlinkLED1(void *pvParameters);
void TaskBlinkLED2(void *pvParameters);

void setup() {
  // Инициализируем пины светодиодов как выходы
  pinMode(LED1_PIN, OUTPUT);
  pinMode(LED2_PIN, OUTPUT);

  // Создаем задачи
  xTaskCreate(
    TaskBlinkLED1,          // Функция, которая реализует задачу
    "BlinkLED1",            // Имя задачи
    128,                    // Размер стека (в словах)
    NULL,                   // Параметр, передаваемый в задачу
```

```
    1,                  // Приоритет задачи
    NULL                // Указатель на задачу
  );

  xTaskCreate(
    TaskBlinkLED2,
    "BlinkLED2",
    128,
    NULL,
    1,
    NULL
  );
}

void loop() {
  // Эта функция пуста, так как всю работу выполняют задачи FreeRTOS
}

// Задача мигания первым светодиодом (1 Гц)
void TaskBlinkLED1(void *pvParameters) {
  (void) pvParameters;

  for (;;) // Бесконечный цикл
  {
    digitalWrite(LED1_PIN, HIGH);   // Включить светодиод
    vTaskDelay(500 / portTICK_PERIOD_MS); // Пауза на 500 мс
    digitalWrite(LED1_PIN, LOW);    // Выключить светодиод
    vTaskDelay(500 / portTICK_PERIOD_MS); // Пауза на 500 мс
  }
}

// Задача мигания вторым светодиодом (5 Гц)
void TaskBlinkLED2(void *pvParameters) {
  (void) pvParameters;

  for (;;) // Бесконечный цикл
  {
    digitalWrite(LED2_PIN, HIGH);
    vTaskDelay(100 / portTICK_PERIOD_MS); // Пауза на 100 мс
    digitalWrite(LED2_PIN, LOW);
    vTaskDelay(100 / portTICK_PERIOD_MS); // Пауза на 100 мс
  }
}
```

Затем: Скетч => Экспортировать скомпилированный бинарный файл. Полученный бинарный файл (ищем в папке C:\Temp\arduino\sketches) загружаем в Arduino в Proteus и запускаем симуляцию:



Работает как задумано.

Задание:

1. Собрать схему как в примере. Добавить еще один светодиод. Изменить код так, что бы 3-й светодиод мигал с частотой равной номеру вашего варианта.

2. Создать две задачи: Task1() и Task2().
Task1() должна периодически (например, каждые 2 секунды) измерять температуру с помощью датчика. Сохраняет значение в глобальной переменной.
Task2() должна отображать измеренную температуру на LCD-дисплее каждые 0.5 секунды. Периодически считывает значение глобальной переменной. Преобразует значение в строку. Отображает строку на LCD-дисплее.

3. Решить задание № 2 с использованием семафора.
Семафор будет регулировать доступ к датчику температуры и файлу.
Task1(). "Занимает" семафор перед измерением температуры.
Измеряет значение с датчика. Записывает значение в файл.
"Освобождает" семафор. Задерживается на 2 секунды.
Task2(). "Занимает" семафор перед считыванием значения из глобальной переменной. Считывает значение глобальной переменной.

Преобразует значение в строку для отображения. Отображает строку на LCD-дисплее. "Освобождает" семафор. Задерживается на 0.5 секунды.

4. Простой термостат.
Создайте две задачи:
Task1(): Считывает температуру с датчика. Сохраняет значение температуры в глобальной переменной.
Task2(): Управляет реле для включения или выключения нагревателя в зависимости от температуры. Периодически проверяет глобальную переменную. Сравнивает температуру с заданным уровнем.
Если температура ниже уровня: Включает нагреватель.
Если температура выше уровня: Выключает нагреватель.
Используйте:
Семафор для синхронизации доступа к глобальной переменной.
Библиотеку для управления реле. (например, Tone).

5. Метеостанция.
Задача: Создайте несколько задач:
Task1(): Считывает температуру с датчика.
Task2(): Считывает влажность с датчика.
Task3(): Считывает атмосферное давление с датчика.
Task4(): Отображает измеренные данные на LCD-дисплее. Периодически проверяет глобальные переменные. Отображает измеренные данные (температуру, влажность, атмосферное давление) на LCD-дисплее.
Используйте:
Семафоры для синхронизации доступа к глобальным переменным.
Библиотеки для работы с датчиками и LCD-дисплеем (например, LiquidCrystal, DHT, BMP180).