Лабораторная работа №7. Многопоточное программирование

Освоение принципов многопоточного программирования и синхронизации работы нескольких потоков для организации управления периферийными подсистемами микроконтроллера.

Учебное время 8 часов

Цель работы: Привитие практических навыков по работе с библиотекой cmsis\_os.

* 1. Содержание работы

1. Ознакомиться с документацией по режимам работы: контроллера ILI9325 TFT LCD; DMA, USART, I2C микроконтроллера STM32F303x. Изучить функции библиотеку cmsis\_os
2. Настроить параметры интерфейсов
3. Выполнить работу согласно варианту:

1.2 Пример программной реализации

**Краткие сведения о RTOS Keil RTX.**

RTOS (Real-Time Operating System) Keil RTX - это многозадачная операционная система реального времени, встроенная в среду разработки Keil, которая обеспечивает вытесняющую многозадачность. Она упрощает разработку сложных систем, улучшая управление проектами и способствуя повторному использованию кода. Основным компромиссом использования ОСРВ является увеличение объёма используемой памяти и возможное удлинение времени реакции на прерывания. Однако современные микроконтроллеры с объёмом ОЗУ от 32 КБ и выше предоставляют достаточные ресурсы для работы ОСРВ, типичный размер которой составляет всего 5 КБ.

Keil RTX реализует стандарт CMSIS-RTOS, что обеспечивает совместимость с другими ОСРВ, соответствующими этому стандарту. Это позволяет разработчикам легко переносить код между различными платформами и упрощает интеграцию ОСРВ в существующие проекты.

На сегодняшний день доступны две версии RTX: RTOS v1 (Keil RTX 4) и RTOS v2 (Keil RTX 5). Последняя версия, RTX5, основана на API стандарта CMSIS-RTOS2 и предоставляет расширенные возможности, включая:

Улучшенную поддержку многозадачности:

* Поддержка динамического создания и удаления потоков, что делает систему более гибкой.
* Настраиваемые приоритеты потоков, позволяющие оптимизировать использование процессора.

Синхронизация потоков и управление ресурсами:

* Мьютексы (mutex) для предотвращения взаимного исключения при работе с общими ресурсами.
* Семафоры (semaphore) и флаги событий (event flags) для координации потоков.
* Таймеры и очереди сообщений для передачи данных между потоками.

Также присутствую возможности, упрощающие взаимодействие:

* RTX5 интегрирована с механизмами энергосбережения, что важно для IoT-устройств и встраиваемых систем с ограниченным энергопотреблением.
* Поддержка режима "спящего" состояния микроконтроллера, который активируется, если все задачи находятся в состоянии ожидания.
* RTX5 позволяет отслеживать состояние потоков в режиме реального времени через средства отладки Keil uVision.
* Интеграция с RTOS Event Viewer для визуализации работы ОСРВ, включая временные диаграммы переключения потоков.

**Архитектура RTX.**

RTX состоит из следующих основных компонентов:

* **Потоки (tasks):** это базовые элементы выполнения. Каждый поток выполняется в своём контексте с отдельным стеком.
* **Планировщик задач (scheduler):** управляет распределением процессорного времени между потоками, используя механизм вытесняющей многозадачности.
* **Ядро ОСРВ:** обрабатывает события, управляет таймерами и взаимодействием между потоками.
* **Интерфейс взаимодействия с оборудованием:** использует системный таймер (SysTick) для управления временем выполнения задач.

Для наших задач мы будем использовать RTX5, у него, в отличии от предыдущих версий, присутствуют следующие особенности:

* RTX5 оптимизирована для современных микроконтроллеров с ядрами ARM Cortex-M. Она поддерживает два режима планирования задач:
  + **Round Robin (циклический опрос):** каждому потоку выделяется фиксированное время на выполнение.
  + **Priority-based scheduling (планирование на основе приоритетов):** потоки с более высоким приоритетом вытесняют задачи с низким приоритетом.
* RTX5 поддерживает функции защиты памяти (MPU), доступные в микроконтроллерах Cortex-M. Это позволяет разработчикам изолировать критически важные потоки, что повышает надёжность и безопасность системы.

**Многопоточности и сихронизация потоков.**

В отличие от функций, выполнение которых завершается после выполнения операций, потоки в ОСРВ (операционных системах реального времени) работают непрерывно. Их выполнение продолжается до завершения всей системы, благодаря наличию бесконечного цикла while(1) в их коде. Управление этими потоками осуществляется специальным компонентом ОСРВ — планировщиком задач. Планировщик использует системный таймер (обычно SysTick) для создания прерываний, которые распределяют процессорное время между потоками. Например, поток 1 может выполняться в течение 100 мс, затем управление передаётся потоку 2, потом потоку 3, и цикл повторяется. Такое чередование создаёт иллюзию одновременного выполнения нескольких задач, хотя фактически они выполняются последовательно.

**Управление потоками**

Каждому потоку выделяется определённый интервал времени, который можно настроить. Планировщик использует вытесняющую многозадачность, что позволяет более высоким приоритетам прерывать выполнение задач с низким приоритетом. Потоки могут находиться в одном из трёх состояний:

* **Running** - поток выполняется.
* **Ready** - поток готов к выполнению, ожидает выделения процессорного времени.
* **Wait** - поток заблокирован, ожидает наступления события (например, сигнала от другого потока, завершения операции ввода/вывода или истечения таймера).

**Приоритеты потоков**

Планировщик выбирает, какой поток выполнить следующим, исходя из их приоритетов. Если несколько потоков имеют одинаковый приоритет и находятся в состоянии Ready, они обрабатываются в порядке очереди или в режиме кругового распределения времени (**Round Robin**).

Список приоритетов, от низкого к высокому:

1. **osPriorityIdle** - минимальный приоритет (обычно для потоков, выполняющихся в "фоновом режиме").
2. **osPriorityLow**
3. **osPriorityBelowNormal**
4. **osPriorityNormal** - стандартный приоритет, присваиваемый по умолчанию.
5. **osPriorityAboveNormal**
6. **osPriorityHigh**
7. **osPriorityRealTime** - наивысший приоритет (используется для критически важных задач).
8. **osPriorityError** - используется для обозначения ошибки.

**Идентификация потоков**

При создании потока ему присваивается уникальный идентификатор и, по умолчанию, приоритет **osPriorityNormal**. Этот идентификатор используется для управления потоком, установки параметров, а также передачи данных. Для взаимодействия с потоками и их управления в RTX используются функции, начинающиеся с префикса os, такие как:

* **osKernelStart()** - запуск ядра ОСРВ (планировщика задач).
* **osThreadNew()** - создание нового потока.
* **osThreadTerminate()** - завершение выполнения потока.
* **osDelay()** - временная задержка выполнения потока.

**Флаги событий**

Флаги событий позволяют потокам ждать наступления определённого события или набора событий. Они представляют собой 32-битное значение, где каждый бит может быть установлен (1) или сброшен (0). Потоки могут проверять отдельные биты или их комбинации.

Основные функции:

* **osEventFlagsSet()** - устанавливает заданный флаг (или флаги) для события.
* **osEventFlagsClear()** - сбрасывает указанные флаги.
* **osEventFlagsWait()** - поток переходит в состояние ожидания, пока не будет установлен нужный флаг.
* **osEventFlagsGet()** - считывает текущие значения флагов.

Пример работы с флагами событий:

osEventFlagsId\_t eventFlags = osEventFlagsNew(NULL);

// Поток 1: Устанавливает флаг после завершения работы

osEventFlagsSet(eventFlags, 0x01);

// Поток 2: Ждёт установки флага

osEventFlagsWait(eventFlags, 0x01, osFlagsWaitAny, osWaitForever);

**Мьютексы**

Мьютекс используется для предотвращения одновременного доступа нескольких потоков к одному ресурсу. В отличие от семафора, мьютекс всегда имеет значение 1 (занят/свободен) и поддерживает механизм приоритетного наследования для предотвращения инверсии приоритетов.

Основные функции:

* **osMutexAcquire()** - захватывает мьютекс. Если ресурс занят, поток переходит в состояние ожидания.
* **osMutexRelease()** - освобождает мьютекс, разрешая доступ другим потокам.
* **osMutexNew()** - создаёт новый мьютекс.

Пример:

osMutexId\_t myMutex = osMutexNew(NULL);

// Поток 1: Захватывает ресурс

osMutexAcquire(myMutex, osWaitForever);

// Работа с ресурсом...

osMutexRelease(myMutex);

// Поток 2: Ждёт, пока мьютекс будет освобождён

osMutexAcquire(myMutex, osWaitForever);

**Семафоры**

Семафоры представляют собой счётчик, который ограничивает количество потоков, имеющих доступ к ресурсу. Семафоры бывают двух типов:

* Бинарные - аналог мьютекса, значение - либо 0 (занят), либо 1 (свободен).
* Счётные - используются для управления доступом к пулу ограниченных ресурсов (например, нескольким слотам в очереди).

Основные функции:

* **osSemaphoreAcquire()** - уменьшает счётчик семафора. Поток блокируется, если значение счётчика равно нулю.
* **osSemaphoreRelease()** - увеличивает счётчик семафора, разрешая доступ заблокированным потокам.
* **osSemaphoreNew()** - создаёт новый семафор с заданным начальным значением.

Пример:

osSemaphoreId\_t mySemaphore = osSemaphoreNew(3, 3, NULL); // Семафор на 3 ресурса

// Поток 1: Захватывает один ресурс

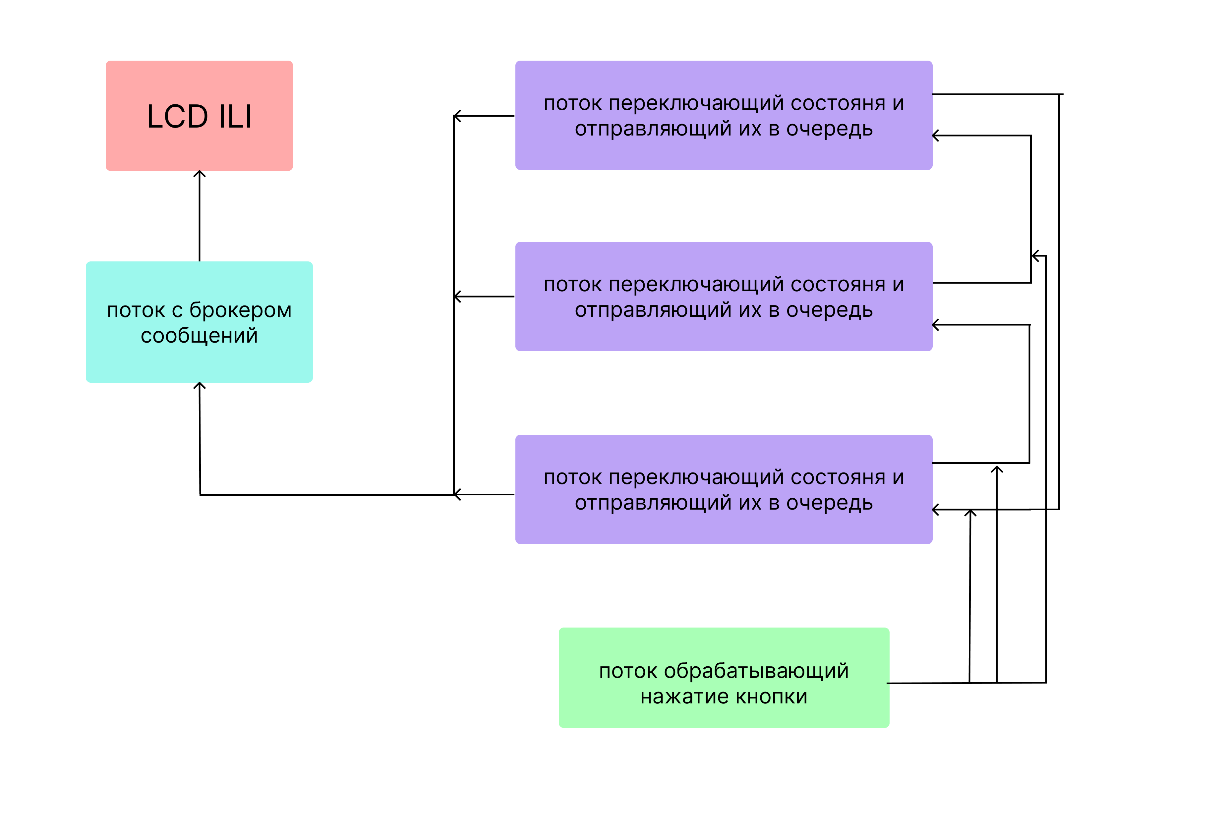
osSemaphoreAcquire(mySemaphore, osWaitForever);

// Работа с ресурсом...

osSemaphoreRelease(mySemaphore);

// Поток 2: Захватывает другой ресурс

osSemaphoreAcquire(mySemaphore, osWaitForever);

Предложенная схема к реализации:  


Создадим для каждого светофора свой поток, который будет отвечать за обновление его состояния, назовём эти потоки потоками светофора. Поток светофора оповещает поток с брокером сообщений о своём состоянии. Поток с брокером сообщений, получает все состояния светофоров и в зависимости от этих данных обновляет дисплей. Создадим поток, обрабатывающий нажатие на кнопки. Нажатие на кнопки может:

* Привести все светофоры в определённое состояние
* Увеличивать состояние потока на 1 раз в n мс, пока кнопка нажата, тем самым ускорим работу светофоров.

Важно рассмотреть, как должны обновляться состояния в потоке светофора. Светофор должен содержать внутри себя своё текущее состояние. Сделаем это состояние доступным для других потоков, чтобы они могли изменять его.

… !!!!!!! Паша вставь код небольшой урезанный код switch

Светофор каждую итерацию проходится по switch, и сопоставив своё состояние начинает ожидать в нём некоторое, заданное вариантом время. После завершения ожидания поток меняет своё внутреннее состояние на следующее по счёту. Важно, чтобы функция ожидания учитывала прерывания с помощью кнопок.

… !!!!!!! Паша вставь код небольшой урезанный код custom delay

Светофор может изменить своё состояние и изменить состояние другого светофора. Для гарантии, что только один поток будет иметь доступ к переменной состояния используем семафоры.

… !!!!!! Паша вставь код небольшой урезанный код логики в switch изменения состояния другого потока.

При нажатии на кнопку тоже будем обращаться к внутреннему состоянию потока светофора, чтобы изменить и при этом гарантировать доступ к переменной состояния потока используем семафор.

… !!!!!! Паша вставь код небольшой урезанный код логики изменения состояния потока в кнопке.

**Варианты**

Возможные состояния для одного сигнала светофора:

R01[(КЯ), 3 сек]

Y01[(ЖТ), 2 сек]

A01[(АМ), 4 сек]

G01[(ЗЯ), 1 сек]

O01[(ОМ), 3 сек]

R02[(КТ), 2 сек]

Y02[(ЖМ), 4 сек]

A02[(АЯ), 1 сек]

G02[(ЗМ), 3 сек]

O02[(ОТ), 2 сек]

R03[(КМ), 4 сек]

Y03[(ЖЯ), 3 сек]

A03[(АТ), 2 сек]

G03[(ЗТ), 1 сек]

O03[(ОЯ), 4 сек]

Возможные состояния для двух сигналов светофора:

R02[(КЯ, ОТ), 3 сек]

Y02[(ЖТ, АЯ), 2 сек]

A02[(АМ, ЗТ), 4 сек]

G02[(КТ, ЖМ), 1 сек]

O02[(ОМ, ЗЯ), 3 сек]

RG2[(КЯ, ЖТ), 2 сек]

YA2[(ЖМ, АЯ), 4 сек]

AG2[(АМ, ЗМ), 1 сек]

GO2[(ЗЯ, ОМ), 3 сек]

OR2[(ОТ, КТ), 2 сек]

RY2[(КМ, ЖЯ), 4 сек]

YB2[(ЖТ, АМ), 3 сек]

BG2[(АМ, ЗЯ), 2 сек]

GR2[(ЗТ, КЯ), 1 сек]

OG2[(ОМ, ЗМ), 3 сек]

Возможные состояния для трёх сигналов светофора:

R03[(КЯ, ОТ, ЖМ), 3 сек]

G03[(КТ, ОЯ, ЖТ), 2 сек]

Y03[(КЯ, ОТ, АМ), 4 сек]

A03[(КТ, ЖТ, ЗЯ), 1 сек]

O03[(КЯ, ЖМ, АТ), 3 сек]

RG3[(КМ, ОТ, ЖТ), 2 сек]

YA3[(ЖЯ, АМ, ЗТ), 4 сек]

AG3[(КЯ, ЖМ, ЗМ), 1 сек]

GO3[(КМ, ЗЯ, ОТ), 3 сек]

OR3[(КТ, ОМ, ЖТ), 2 сек]

RY3[(КЯ, ЖЯ, ОТ), 4 сек]

YB3[(ЖМ, АЯ, ЗМ), 3 сек]

BG3[(КТ, ОЯ, ЖТ), 2 сек]

GR3[(ЗТ, КМ, ОМ), 1 сек]

OG3[(ОМ, ЗМ, КЯ), 3 сек]

Светофоры должны переключаться автоматически, как описано ранее, но сигналы должны следовать строго в указанной последовательности: следующий сигнал (n) включается только после полного завершения текущего сигнала (n-1).

Вариант 1:

1R01 1Y01 2A01 2G01 3O01 3R02 4Y02 4A02 5G02 5O02 6R03 6Y03 7A03 7G03 8O03 8R01 1Y01 2A01 3G01 4O01

Вариант 2:

1R02 1Y02 2A02 2G02 3O02 3R03 4Y03 4A03 5G03 5O03 6R01 6Y01 7A01 7G01 8O01 8R02 1Y02 2A02 3G02 4O02

Вариант 3:

1R03 1Y03 2A03 2G03 3O03 3R01 4Y01 4A01 5G01 5O01 6R02 6Y02 7A02 7G02 8O02 8R03 1Y03 2A03 3G03 4O03

Вариант 4:

1R01 1Y02 2A01 2G02 3O01 3R03 4Y01 4A03 5G01 5O03 6R02 6Y03 7A02 7G03 8O02 8R01 1Y02 2A01 3G02 4O01

Вариант 5:

1R02 1Y03 2A02 2G03 3O02 3R01 4Y02 4A01 5G02 5O01 6R03 6Y01 7A03 7G01 8O03 8R02 1Y03 2A02 3G03 4O02

Вариант 6:

1R03 1Y01 2A03 2G01 3O03 3R02 4Y03 4A02 5G03 5O02 6R01 6Y02 7A01 7G02 8O01 8R03 1Y01 2A03 3G01 4O03

Вариант 7:

1R01 1Y03 2A01 2G03 3O01 3R02 4Y01 4A02 5G01 5O02 6R03 6Y02 7A03 7G02 8O03 8R01 1Y03 2A01 3G03 4O01

Вариант 8:

1R02 1Y01 2A02 2G01 3O02 3R03 4Y02 4A03 5G02 5O03 6R01 6Y03 7A01 7G03 8O01 8R02 1Y01 2A02 3G01 4O02

Вариант 9:

1R03 1Y02 2A03 2G02 3O03 3R01 4Y03 4A01 5G03 5O01 6R02 6Y01 7A02 7G01 8O02 8R03 1Y02 2A03 3G02 4O03

Вариант 10:

1R02 1Y02 2A02 2RG2 3YA2 3GO2 4OR2 4RY2 1Y02 2A02 3GO2 4RY2 1R02 2YA2 3OR2 4GO2 1Y02 2RG2 3YA2 4RY2

Вариант 11:

1RG2 1YA2 2GO2 2OR2 3RY2 3RG2 4YA2 4GO2 1OR2 2YA2 3RY2 4RG2 1GO2 2RG2 3OR2 4YA2 1RY2 2YA2 3RG2 4GO2

Вариант 12:

1RY2 1RG2 2YA2 2GO2 3OR2 3RY2 4RG2 4YA2 1GO2 2RY2 3OR2 4YA2 1RG2 2OR2 3YA2 4GO2 1RY2 2GO2 3RG2 4OR2

Вариант 13:

1YA2 1GO2 2RG2 2RY2 3YA2 3OR2 4GO2 4RY2 1RG2 2YA2 3RY2 4OR2 1GO2 2RG2 3OR2 4YA2 1RY2 2YA2 3RG2 4GO2

Вариант 14:

1OR2 1RG2 2GO2 2YA2 3RY2 3OR2 4RG2 4GO2 1YA2 2RG2 3OR2 4YA2 1RY2 2GO2 3YA2 4RG2 1GO2 2RY2 3RG2 4YA2

Вариант 15:

1RG2 1OR2 2YA2 2RY2 3GO2 3YA2 4RY2 4RG2 1YA2 2GO2 3OR2 4YA2 1RY2 2RG2 3YA2 4GO2 1RG2 2GO2 3OR2 4YA2

Вариант 16:

1RY2 1YA2 2OR2 2GO2 3RG2 3RY2 4YA2 4OR2 1GO2 2RG2 3YA2 4RY2 1YA2 2GO2 3RG2 4OR2 1RY2 2YA2 3GO2 4RG2

Вариант 17:

1GO2 1RG2 2RY2 2OR2 3YA2 3RG2 4GO2 4RY2 1YA2 2OR2 3RG2 4GO2 1RY2 2YA2 3OR2 4RG2 1GO2 2RY2 3YA2 4OR2

Вариант 18:

1OR2 1RG2 2YA2 2GO2 3RG2 3RY2 4GO2 4YA2 1RY2 2RG2 3OR2 4GO2 1YA2 2GO2 3YA2 4RG2 1RG2 2YA2 3RY2 4OR2

Вариант 19:

1R03 1YA3 2GO3 2RG3 3OR3 3YA3 1R03 2YA3 3GO3 1RG3 2OR3 3YA3 1R03 2GO3 3RG3 1YA3 2RG3 3OR3 1GO3 2YA3

Вариант 20:

1GO3 1RG3 2OR3 2YA3 3GO3 3RG3 1R03 2GO3 3YA3 1RG3 2OR3 3YA3 1GO3 2YA3 3RG3 1YA3 2RG3 3OR3 1R03 2GO3

Вариант 21:

1RG3 1YA3 2OR3 2GO3 3RG3 3YA3 1R03 2RG3 3OR3 1YA3 2GO3 3YA3 1RG3 2OR3 3YA3 1GO3 2RG3 3OR3 1YA3 2GO3

Вариант 22:

1YA3 1GO3 2RG3 2YA3 3OR3 3RG3 1R03 2OR3 3YA3 1RG3 2GO3 3YA3 1YA3 2RG3 3OR3 1GO3 2YA3 3YA3 1RG3 2OR3

Вариант 23:

1RG3 1YA3 2GO3 2OR3 3RG3 3YA3 1R03 2YA3 3RG3 1GO3 2OR3 3YA3 1YA3 2RG3 3OR3 1RG3 2GO3 3YA3 1R03 2YA3

Вариант 24:

1OR3 1RG3 2YA3 2GO3 3OR3 3YA3 1R03 2RG3 3GO3 1YA3 2OR3 3YA3 1RG3 2YA3 3RG3 1GO3 2RG3 3OR3 1YA3 2GO3

Вариант 25:

1YA3 1GO3 2RG3 2OR3 3YA3 3RG3 1R03 2GO3 3OR3 1RG3 2YA3 3YA3 1GO3 2RG3 3YA3 1YA3 2OR3 3RG3 1R03 2GO3

Вариант 26:

1GO3 1RG3 2OR3 2YA3 3RG3 3GO3 1R03 2YA3 3OR3 1RG3 2GO3 3YA3 1OR3 2RG3 3YA3 1YA3 2GO3 3RG3 1R03 2YA3

Вариант 27:

1RG3 1YA3 2GO3 2RG3 3YA3 3OR3 1R03 2OR3 3RG3 1YA3 2YA3 3GO3 1RG3 2RG3 3YA3 1GO3 2OR3 3YA3 1R03 2YA3

Вариант 37:

1YA3 1RG3 2OR3 2YA3 3GO3 3RG3 1R03 2RG3 3OR3 1YA3 2GO3 3YA3 1RG3 2OR3 3YA3 1GO3 2YA3 3RG3 1R03 2GO3