МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра ВТ

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Операционные системы реального времени»

Тема: СОВМЕСТНЫЙ ДОСТУП К РЕСУРСАМ.

Студенты гр. 6492 _____ Мурашко А. С.
Огурецкий Д. В.

Преподаватель _____ Гречухин М. Н

Санкт-Петербург
2020

Цель: изучить организацию безопасного совместного доступа к ресурсам.

Задание:

1. Открыть IDE Keil MDK-ARM, создать новый проект по алгоритму из работы 1. В файле конфигурации FreeRTOSConfig.h необходимо разрешить использование нужных в работе функций (с помощью директив вида #define INCLUDE_vTaskDelay 1).

2. Создать две функции-задачи, которые будут обращаться к одному и тому же ресурсу – расположенному на плате диоду. Первая задача будет мигать диодом равномерно с частотой 1 Гц, вторая – повторять сигнал SOS (три коротких вспышки, три длинных, снова три коротких). Создать мьютекс для организации совместного доступа к ресурсу.

3. Сделать так, чтобы обе задачи в течение 4 с мигали диодом по своей программе по очереди.

4. Скомпилировать проект. Загрузить его на плату, наблюдать работу с подключенной периферией.

5. Намеренно «забыть» освободить мьютекс в первой задаче. Пересобрать проект, наблюдать за работой.

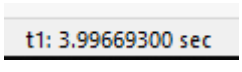
Ход работы.

В данной лабораторной работе предлагается создать две задачи, которые будут обращаться к одному и тому же ресурсу – расположенному на плате диоду. Первая задача будет мигать диодом равномерно с частотой 1 Гц, вторая – повторять сигнал SOS (три коротких вспышки, три длинных, снова три коротких).

Настраиваем наш проект как в лабораторной работе 1 и 2 и переходим к выполнению задачи.

В лабораторной работе используется режимов работы планировщика вытесняющей многозадачности с разделением времени (pre-emptive mode with time-slicing), использующий RMS-подобный алгоритм. Данный режим работы планировщика гарантирует, что задачи с равным приоритетом, находящиеся в состоянии «Готова», будут получать равное время выполнения. (согласно информации, найденной на форумах, 2 тика (около 2 мс)). Соответственно задачам задан равный приоритет.

При каждой итерации выполнения 1 задачи производятся попытки захватить мьютекс с помощью функции xSemaphoreTake(xSemaphore, 0) при этом время ожидания освобождения мьютекса равно нулю и задача не переходит в блокированное состояние для ожидания мьютекса.

В задаче SOS способ ограничения времени работы с ресурсам реализован с помощью метода цикла for и специально подобранными временами задержек времени мигания светодиода. Способ задержки был выбран именно такой, потому что vTaskDelay() позволяет задаче входить в заблокированное состояние, давая время другим задачам на выполнение. Таким образом сигнал SOS посылается 2 раза за 4 сек. Можно было реализовать ограничение времени с помощью функции xTaskGetTickCount(), возвращающую тики, но тогда время выполнения индикации сигнала SOS было бы больше 4 секунд и пришлось бы вместо vTaskDelay() делать цикл for. Время работы практически 4 секунды 

.

В задаче Blinking точное время работы с ресурсом определяется с помощью цикла for.

Строчка кода vTaskDelay(1) необходима для того, чтобы создать задержку в 1 тик (1 мс). Без неё после освобождения мьютекса, задача 2 не успеет выполниться из за того, что не произойдет вызова планировщика, и не

захватит мьютекс, тем самым 1 задача снова завладеет мьютексом и по такой логике будет бесконечно выполняться. Можно было вместо задержки сделать макрос taskYIELD() для вызова планировщика, но vTaskDelay и так приводит к вызову планировщика.

Код программы:

```
#include "stm32f4xx.h"                  // Device header
#include "FreeRTOSConfig.h"             // ARM.FreeRTOS::RTOS:Config
#include "FreeRTOS.h"                   // ARM.FreeRTOS::RTOS:Core
#include "task.h"                       // ARM.FreeRTOS::RTOS:Core
#include "semphr.h"                     // ARM.FreeRTOS::RTOS:Core

void Blinking(void* xSemaphore)
{
        while(1)
        {
                if(xSemaphoreTake(xSemaphore, 0))
                {                       // blinking 4 sec
                        for(int count = 0; count < 4 ; count++)
                        {
                                GPIOA -> ODR |= GPIO_ODR_ODR_5; // turning LED on
                                vTaskDelay(500);
                                GPIOA -> ODR &= ~GPIO_ODR_ODR_5;
                                vTask-
Delay(500);
                        }
                        xSemaphoreGive(xSemaphore);
                }
                vTaskDelay(1); // time need to give chance    for check Mutex for SOS task
        }
}

void SOS(void* xSemaphore)
{
                int n = pdMS_TO_TICKS(50);
                int m = 0;
                while(1)
                {
                        if(xSemaphoreTake(xSemaphore, 0))
                        {
                                for(int count = 0; count < 18 ; count++)
                                { // blinking 4 sec
                                        if (m == 3)
                                                n = pdMS_TO_TICKS(233);
                                        if (m == 6)
                                                n =pdMS_TO_TICKS(50);
                                        if (m == 9){
                                                m = 0;
                                                n = pdMS_TO_TICKS(50);}
                                        GPIOA -> ODR |= GPIO_ODR_ODR_5;
                                        vTaskDelay(n);
```

```
46                                          GPIOA -> ODR &= ~GPIO_ODR_ODR_5;
47                                          vTaskDelay(n);
48                                          m++;
49                                  }
50                                  xSemaphoreGive(xSemaphore);
51                          }
52                          vTaskDelay(1); // time need to give chance    for
  check Mutex for Blinking task
53                  }
54 }
55
56 int main(void)
57 {
58
59         RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN; // turning on GPIOA
60         GPIOA->MODER |= GPIO_MODER_MODER5_0; // setting A5 to output
61
62         SemaphoreHandle_t xSemaphore = NULL; // initializing semaphore's
  handle
63         xSemaphore = xSemaphoreCreateMutex(); // creating Mutex
64
65         xTaskCreate(Blinking, "Task1", configMINIMAL_STACK_SIZE, xSema-
  phore, 8, NULL);
66         xTaskCreate(SOS, "Task2", configMINIMAL_STACK_SIZE, xSemaphore, 8,
  NULL);
67         vTaskStartScheduler();
68
69         while(1)
70         {
71         }
   }
```

Результаты работы.

В данной лабораторной работе для автоматизирования процесса предоставления доступа к памяти при debugging используется файл инициализации. Таким образом при запуске debug сначала выполняется файл инициализации, который разрешает доступ к требуемой области памяти. Для включения этого файла открываем Options for Target => debug =>

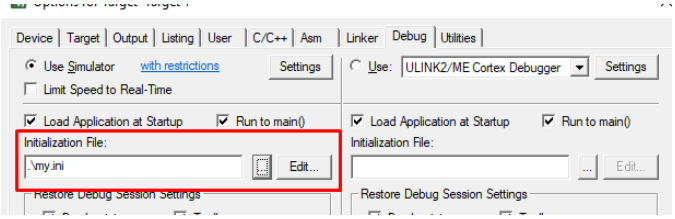
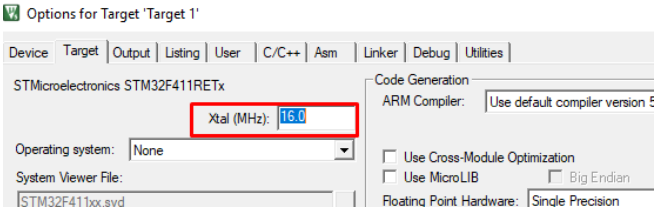


Рисунок 1

Код файла инициализации:

MAP 0x40000000, 0x47FFFFFF READ WRITE // allow R/W access to IO space

Также необходимо настроить частоту в Options for Target для правильной работы.



В результате работы программы получаем мигание диода в течение 4 секунд с частотой 1 Гц. После чего в течение последующих также 4 секунд мигание по сигналу SOS. Задачи по очереди обращаются к ресурсу, забирая друг у друга мьютекс.

Намеренно закомментируем строчку кода с отдачей мьютекса у первой задачи *Blinking*.

```
//xSemaphoreGive(xSemaphore);
```

В результате получим сначала мигание диода в режиме SOS, после освобождения мьютекса планировщик вызовет 1 задачу, она захватит мьютекс, произойдет мигание диода с частотой 1 Гц в течении 4 секунд.

Далее из за того, что первая задача не освободила мьютекс она больше не сможет произвести работу с ресурсом, соответственно 2 задача тоже не сможет поработать с ресурсом, т.о. ресурс станет бесконечно недоступным.

Вывод: в ходе работы было изучено использование мьютексов для решения совместного доступа к ресурсу нескольких задач. При правильном осуществлении доступа к ресурсам задачи работают последовательно и то количество времени, сколько мы указываем.