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: «УПРАВЛЕНИЕ ЗАДАЧАМИ»

Вариант 2

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

Преподаватель _____ Гречухин М. Н

Санкт-Петербург
2020

Цель: Разобраться с различными функциями управления задачами, с алгоритмами планирования, с приоритетами задач и их влиянием на работу.

Ход работы.

Настройка проекта производится по методике первой лабораторной работы.

Задание 1 (для варианта 2): Создать две функции-задачи. Одна из них будет мигать диодом на плате с частотой 10Гц, вторая – реализовывать бегущий огонь на внешней диодной линейке справа налево по два с частотой 20Гц. Приоритет задачам поставить равный. Частоту мигания диода и задержку переключения бегущего огня реализовать приближённо, с помощью цикла for, подобрав подходящее число итераций.

Настроить использование вытесняющей многозадачности без разделения времени, а затем с разделением времени.

Для реализации вытесняющей многозадачности без разделения времени в конфигурации FreeRTOS установить configUSE_PREEMPTION = 1 и configUSE_TIME_SLICING = 0.

С разделением времени изменяем configUSE_PREEMPTION = 1 и configUSE_TIME_SLICING = 1.

В задаче для мигания одного диода и диодной ленты между переключениями состояния задержка реализуется с помощью цикла. Количество итераций цикла рассчитывается из расчета, что 1 итерация это 1 такт процессора, то есть 1000/16 [МГц] [мс]. Далее вручную происходит сверка с реальным временами в debuger и время корректируется. Так в первом случае при количестве итераций , соответствующей 1000 мс, подсчитанной исходя из логики выше, реальная задержка составила 20000 мс, что в 20 раз больше требуемой. Поэтому полученной число при расчете по логике выше нужно разделить на 20. Таким образом макрос для расчета количества итераций будет:

#define delay_ms(time_in_ms) time_in_ms*800

Для бегущего огня время частоте соответствует время между изменениями состояния, то есть переходу к следующему порядку включения огней.

Код

```
#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

#define delay_ms(time_in_ms) time_in_ms*800

void task_one_diod_blinking(void* pvParameters)
{
        for(;;)
        {
                GPIOA -> ODR ^= GPIO_ODR_ODR_10;
                for(int i =0; i!=delay_ms(50); i++);
        }
```

```
}

void task_running_fire(void* pvParameters)
{
        int OD_R = GPIO_ODR_ODR_8|GPIO_ODR_ODR_9;
        for(;;)
        {
                GPIOA -> ODR |= OD_R;
                for(int i =0; i!=delay_ms(50); i++);
                GPIOA -> ODR &= ~OD_R;
                OD_R = OD_R >> 1;
                if (OD_R == 0)
                          OD_R =
GPIO_ODR_ODR_8|GPIO_ODR_ODR_9;

        }
}

int main(void)
{
        RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN; //Turning on GPIOA
        GPIOA->MODER |= GPIO_MODER_MODER0_0; //turning A0 to output
        GPIOA->MODER |= GPIO_MODER_MODER1_0; //turning A1 to output
        GPIOA->MODER |= GPIO_MODER_MODER2_0; //turning A2 to output
        GPIOA->MODER |= GPIO_MODER_MODER3_0; //turning A3 to output
        GPIOA->MODER |= GPIO_MODER_MODER4_0; //turning A4 to output
        GPIOA->MODER |= GPIO_MODER_MODER5_0; //turning A5 to output
        GPIOA->MODER |= GPIO_MODER_MODER8_0; //turning A8 to output
        GPIOA->MODER |= GPIO_MODER_MODER9_0; //turning A9 to output
        GPIOA->MODER |= GPIO_MODER_MODER10_0; //turning A10 to output for
one diod

        xTaskCreate(task_one_diod_blinking, "task_one_diod_blinking",
configMINIMAL_STACK_SIZE, NULL, 10, NULL);
        xTaskCreate(task_running_fire, "task_running_fire",
configMINIMAL_STACK_SIZE, NULL, 10, NULL);
        vTaskStartScheduler();
        while(1)
        {

        }
  }
```

При использовании многозадачности без разделения времени задача по миганию одиночного светодиода вообще не выполняется, а бегущий огонь хоть и выполняется, но выполняется некорректно. После переключения в режим с разделением времени на каждую задачу уделяется одинаковое время и задачи выполняются в нормальном режиме, не мешая друг другу.

Задание 2.

Изменить поведение задач. Первая задача делает 2 включения и выключения диода на плате с периодом 1 с, после чего приостанавливается на 100 тиков. Вторая – пробегает по линейке бегущим огнём 2 раза, после чего приостанавливается на 5 тиков. Приоритеты оставить равные. Затем изменить приоритеты задачи. Повысить приоритет второй задачи относительно первой.

При одинаковых приоритетах (они равны 10) приостановка второй задачи на 5 тиков не заметна, так как это очень малое время, а второй задачи на 100 тиков заметно, так как 100 тиков соответствует 100 мс, что составляет значительную долю от полупериода 500 мс.

Затем понижаем приоритет первой задачи до 3. Диод, управляемый первой задачей, загорелся и долгое время не потухает.

Объяснить это можно так: теперь постоянно выполняется более приоритетная задача – вторая. Первая задача выполняется только во время приостановки второй задачи на 5 тиков, так как вторая задача в это время находится в режиме блокировки. После окончания блокировки вторая задача опять забирает на себя все выполнение и вытесняет первую задачу. То есть к задержке (циклу for) первой задачи прибавляется время выполнения 2 задачи: итого 500 мс + 100 с = 100,5 сек, через это время диод потухнет.

Код

```
#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

#define delay_ms(time_in_ms) time_in_ms*800

void task_one_diod_blinking(void* pvParameters)
{
        char m = 0;
        for(;;)
        {
                GPIOA -> ODR ^= GPIO_ODR_ODR_10;
                for(int i =0; i!=delay_ms(500); i++);
                m++;
                if (m == 2*2)
                        {
                                m = 0;
                                vTaskDelay(100);
                        }
        }
}

void task_running_fire(void* pvParameters)
{
        char m = 0;
        int OD_R = GPIO_ODR_ODR_8|GPIO_ODR_ODR_9;
        for(;;)
        {
                GPIOA -> ODR |= OD_R;
```

```
                for(int i =0; i!=delay_ms(50); i++);
                GPIOA -> ODR &= ~OD_R;
                OD_R = OD_R >> 1;
                if (OD_R == 0)
                {
                        OD_R = GPIO_ODR_ODR_8|GPIO_ODR_ODR_9;
                        m++;
                        if (m == 2)
                        {
                                m = 0;
                                vTaskDelay(5);
                        }
                }
        }
}

int main(void)
{
        RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN; //Turning on GPIOA
        GPIOA->MODER |= GPIO_MODER_MODER0_0; //turning A0 to output
        GPIOA->MODER |= GPIO_MODER_MODER1_0; //turning A1 to output
        GPIOA->MODER |= GPIO_MODER_MODER2_0; //turning A2 to output
        GPIOA->MODER |= GPIO_MODER_MODER3_0; //turning A3 to output
        GPIOA->MODER |= GPIO_MODER_MODER4_0; //turning A4 to output
        GPIOA->MODER |= GPIO_MODER_MODER5_0; //turning A5 to output
        GPIOA->MODER |= GPIO_MODER_MODER8_0; //turning A8 to output
        GPIOA->MODER |= GPIO_MODER_MODER9_0; //turning A9 to output
        GPIOA->MODER |= GPIO_MODER_MODER10_0; //turning A10 to output for
one diod

        xTaskCreate(task_one_diod_blinking, "task_one_diod_blinking",
configMINIMAL_STACK_SIZE, NULL, 3, NULL);
        xTaskCreate(task_running_fire, "task_running_fire",
configMINIMAL_STACK_SIZE, NULL, 5, NULL);
        vTaskStartScheduler();
        while(1)
        {

        }
  }
```

Задание 3.

Изменить задачи, реализовав точную задержку с помощью API FreeRTOS вместо приближённой, реализованной с помощью for.

При выполнении обе задачи выполняются корректно. Это происходит благодаря тому, что теперь задержка между изменениями состояния диодной ленты реализована в виде функции vTaskDelay(), которая вводит задачу в режим блокировки и позволяет первой (менее приоритетной) задаче выполниться.

Код

```
#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
void task_one_diod_blinking(void* pvParameters)
{
        char m = 0;
        for(;;)
        {
                GPIOA -> ODR ^= GPIO_ODR_ODR_10;
                vTaskDelay(500);
                m++;
                if (m == 2*2)
                        {
                                m = 0;
                                vTaskDelay(100);
                        }
        }
}
void task_running_fire(void* pvParameters)
{
        char m = 0;
        int OD_R = GPIO_ODR_ODR_8|GPIO_ODR_ODR_9;
        for(;;)
        {
                GPIOA -> ODR |= OD_R;
                vTaskDelay(50);
                GPIOA -> ODR &= ~OD_R;
                OD_R = OD_R >> 1;
                if (OD_R == 0)
                {
                        OD_R = GPIO_ODR_ODR_8|GPIO_ODR_ODR_9;
                        m++;
                        if (m == 2)
                        {
                                m = 0;
                                vTaskDelay(5);
                        }
                }
        }
}
int main(void)
{
        RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN; //Turning on GPIOA
        GPIOA->MODER |= GPIO_MODER_MODER0_0; //turning A0 to output
```

```
46          GPIOA->MODER |= GPIO_MODER_MODER1_0; //turning A1 to output
47          GPIOA->MODER |= GPIO_MODER_MODER2_0; //turning A2 to output
48          GPIOA->MODER |= GPIO_MODER_MODER3_0; //turning A3 to output
49          GPIOA->MODER |= GPIO_MODER_MODER4_0; //turning A4 to output
50          GPIOA->MODER |= GPIO_MODER_MODER5_0; //turning A5 to output
51          GPIOA->MODER |= GPIO_MODER_MODER8_0; //turning A8 to output
52          GPIOA->MODER |= GPIO_MODER_MODER9_0; //turning A9 to output
53          GPIOA->MODER |= GPIO_MODER_MODER10_0; //turning A10 to output for
54 one diod
55
56          xTaskCreate(task_one_diod_blinking, "task_one_diod_blinking",
57 configMINIMAL_STACK_SIZE, NULL, 3, NULL);
58          xTaskCreate(task_running_fire, "task_running_fire",
59 configMINIMAL_STACK_SIZE, NULL, 5, NULL);
            vTaskStartScheduler();
            while(1);
    }
```

Задание 4.

Изменить алгоритм задач. Первая задача в начале цикла приостанавливает вторую задачу, далее выполняет *4* цикла включения/выключения диода на плате, после чего возобновляет вторую задачу и блокируется на две секунды. Время приостановки задачи изменить по необходимости.

Важно!!! Для остановки и возобновления работы 2 задачи использована функция xTaskGetHandle, но у нее есть ограничения по длине имени задачи в 10 знаков. Поэтому пришлось изменить имя задачи.

В данном задании задачи имеют одинаковый приоритет равный 5. В первой задаче реализована приостановка второй задачи и ее возобновление. Первая задача приостанавливает вторую, затем выполняет 4 мигания светодиодов, после чего возобновляет работу второй задачи и уходит в блокировку на 2 секунды, позволяя второй задаче выполняться в течении этого времени. Через две секунды она выходит из состояния блокировки, приостанавливает вторую задачу и все происходит заново и так по циклу.

Код

```
#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
void task_one_diod_blinking(void* pvParameters)
{
        char m = 0;
        for(;;)
        {
                vTaskSuspend(xTaskGetHandle("task2"));
                GPIOA -> ODR ^= GPIO_ODR_ODR_10;
                vTaskDelay(500);
                m++;
                if (m == 2*4)
                        {
                                m = 0;
                                vTaskResume(xTaskGetHandle("task2"));
                                vTaskDelay(2000);
                        }
        }
}
void task2(void* pvParameters)
{
        int OD_R = GPIO_ODR_ODR_8|GPIO_ODR_ODR_9;
        for(;;)
        {
                GPIOA -> ODR |= OD_R;
                vTaskDelay(50);
                GPIOA -> ODR &= ~OD_R;
                OD_R = OD_R >> 1;
                if (OD_R == 0)
                {
                        OD_R = GPIO_ODR_ODR_8|GPIO_ODR_ODR_9;
                }
```

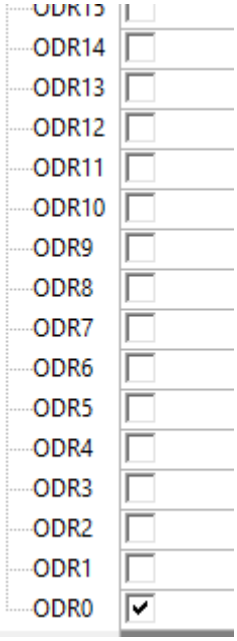
```
        }
}
int main(void)
{
        RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN; //Turning on GPIOA
        GPIOA->MODER |= GPIO_MODER_MODER0_0; //turning A0 to output
        GPIOA->MODER |= GPIO_MODER_MODER1_0; //turning A1 to output
        GPIOA->MODER |= GPIO_MODER_MODER2_0; //turning A2 to output
        GPIOA->MODER |= GPIO_MODER_MODER3_0; //turning A3 to output
        GPIOA->MODER |= GPIO_MODER_MODER4_0; //turning A4 to output
        GPIOA->MODER |= GPIO_MODER_MODER5_0; //turning A5 to output
        GPIOA->MODER |= GPIO_MODER_MODER8_0; //turning A8 to output
        GPIOA->MODER |= GPIO_MODER_MODER9_0; //turning A9 to output
        GPIOA->MODER |= GPIO_MODER_MODER10_0; //turning A10 to output for
one diod

        xTaskCreate(task_one_diod_blinking, "task_one_diod_blinking",
configMINIMAL_STACK_SIZE, NULL, 5, NULL);
        xTaskCreate(task2, "task2", configMINIMAL_STACK_SIZE, NULL, 5,
NULL);
        vTaskStartScheduler();
        while(1);
  }
```

Задание 5.

Изменить алгоритм задач. В начале работы создаётся только первая задача. Первая задача выполняет *4* цикла включения\выключения диода, после чего создаёт вторую задачу с приоритетом, равным своему. После того, как первая задача выполнит в общей сложности *6* циклов включения\выключения диода, она удаляет вторую задачу. Далее процесс повторяется циклически.

Важно!!! Для выполнения данного необходимо заменить модель памяти Heap_1 на Heap_2, так как у Heap_1 нет возможности очистки памяти, которая необходима для реализации функции vTaskDelete(). Если оставить Heap_1, то происходит зависание программы в теле конфигурации файла Heap_1.c на попытке принудительной очистки памяти.

В данном задании вторая задача создается и по истечению определенного времени удаляется из первой задачи. Таким образом вторая задача уже не является независимой, так как она управляется первой задачей.



В результате выполнения диод из первой задачи мигает 4 раза и далее начинается работать задача 2. После достижения первым диодом 6 циклов мигания вторая задача удаляется. Причем, выходные порты остаются в последнем состоянии. То есть диоды не гаснут, так как на портах остается высокий уровень.

После создания 2 задачи, выполнения алгоритма бегущего огня начинается заново.

Рис.1

Код

```
#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
void task2(void* pvParameters)
{
        int OD_R = GPIO_ODR_ODR_8|GPIO_ODR_ODR_9;
        for(;;)
        {
                GPIOA -> ODR |= OD_R;
                vTaskDelay(50);
                GPIOA -> ODR &= ~OD_R;
                OD_R = OD_R >> 1;
```

```
14                if (OD_R == 0)
15                {
16                        OD_R = GPIO_ODR_ODR_8|GPIO_ODR_ODR_9;
17                }
18        }
19 }
20 void task_one_diod_blinking(void* pvParameters)
21 {
22        char m = 0;
23        for(;;)
24        {
25                GPIOA -> ODR ^= GPIO_ODR_ODR_10;
26                vTaskDelay(500);
27                m++;
28                if(m==2*4)
29                {
30                                xTaskCreate(task2, "task2",
31 configMINIMAL_STACK_SIZE, NULL, 5, NULL);
32                }
33                if (m == 2*6)
34                        {
35                                m = 0;
36                                vTaskDelete(xTaskGetHandle("task2"));
37                                vTaskDelay(2000);
38                        }
39        }
40 }
41 int main(void)
42 {
43        RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN; //Turning on GPIOA
44        GPIOA->MODER |= GPIO_MODER_MODER0_0; //turning A0 to output
45        GPIOA->MODER |= GPIO_MODER_MODER1_0; //turning A1 to output
46        GPIOA->MODER |= GPIO_MODER_MODER2_0; //turning A2 to output
47        GPIOA->MODER |= GPIO_MODER_MODER3_0; //turning A3 to output
48        GPIOA->MODER |= GPIO_MODER_MODER4_0; //turning A4 to output
49        GPIOA->MODER |= GPIO_MODER_MODER5_0; //turning A5 to output
50        GPIOA->MODER |= GPIO_MODER_MODER8_0; //turning A8 to output
51        GPIOA->MODER |= GPIO_MODER_MODER9_0; //turning A9 to output
52        GPIOA->MODER |= GPIO_MODER_MODER10_0; //turning A10 to output for
53 one diod
54
55        xTaskCreate(task_one_diod_blinking, "task_one_diod_blinking",
56 configMINIMAL_STACK_SIZE, NULL, 5, NULL);
          vTaskStartScheduler();
          while(1);
   }
```

Выводы.

1) Во время выполнения лабораторной работы прошло ознакомление с режимом многозадачности с разделением времени и без разделения. Отличие между ними состоит в том, что разделение времени позволяет гарантирует равное время выполнения для равноприоритетных задач.
2) Исследовано влияние приоритета задач на работу программы. Когда задача с меньшим приоритетом практически не выполняется из-за того, что вторая задача практически полностью занимает процессорное время. Чтобы она все таки выполнялась, нужно реализовывать задержки в функции-задаче с помощью перевода задачи в режим блокировки функцией vTaskDelay(). Так менее приоритетные задачи получают «шанс» на выполнение.
3) С помощью функций приостановки и возобновления задач можно при выполнении одной задачи прямо в ней приостановить другую на какое-то время, затем возобновить. Точно по такой же логике можно создавать и удалять задачи.