**Санкт-Петербургский государственный политехнический университет Петра Великого**

**Кафедра компьютерных интеллектуальных технологий**

**ОТЧЕТ**

**По лабораторной работе №7**

**Тема: Использование мьютексов и семафоров для управления большим количеством потоков**

|  |  |
| --- | --- |
| Преподаватель:  Вербова Н.М.  Подпись\_\_\_\_\_\_\_\_\_\_\_\_ | Выполнил:  Студент группы 23546/1  Клеверов М.А.  Подпись\_\_\_\_\_\_\_\_\_\_\_ |
|  |  |

# Постановка задачи

Цель работы: Научиться использовать мьютексы для ограничения доступа большого числа потоков к светодиодам, а также реализовать барьер на основе семафоров для доступа потоков к периферии.

Постановка задачи: создать новый проект в Keil μVision5 и разработать программу для микроконтроллера (МК) STM32F200, которая будет выполнять поставленную цель.

# Текст программы 1

#include "stm32f2xx.h"

#include <cmsis\_os.h>

#include <stdlib.h>

int thread\_count=8;

osThreadId\* tid\_arr;

void delay(int n)

{

unsigned int i;

for ( i = 0; i < n\*40000; i++) {}

}

void LEDfuncG(int ODRnum)

{

GPIOG->ODR |= 1 << ODRnum;

delay(ODRnum);

GPIOG->ODR &= ~1 << ODRnum;

delay(ODRnum);

}

void LEDfuncH(int ODRnum)

{

GPIOH->ODR |= 1 << ODRnum;

delay(ODRnum);

GPIOH->ODR &= ~1 << ODRnum;

delay(ODRnum);

}

void LEDfuncI(int ODRnum)

{

GPIOI->ODR |= 1 << ODRnum;

delay(ODRnum);

GPIOI->ODR &= ~1 << ODRnum;

delay(ODRnum);

}

osMutexId mutex;

osMutexDef (mutex);

void threadG (void const \*argument) {

unsigned int i;

for (;;) {

osMutexWait(mutex, osWaitForever);

for( i = 0; i < 5; i++)

LEDfuncG((int) argument);

osMutexRelease(mutex);

}

}

void threadH (void const \*argument) {

unsigned int i;

for (;;) {

osMutexWait(mutex, osWaitForever);

for( i = 0; i < 5; i++)

LEDfuncH((int) argument);

osMutexRelease(mutex);

}

}

void threadI (void const \*argument) {

unsigned int i;

for (;;) {

osMutexWait(mutex, osWaitForever);

for( i = 0; i < 5; i++)

LEDfuncI((int) argument);

osMutexRelease(mutex);

}

}

osThreadDef (threadG, osPriorityNormal, 5, 0);

osThreadDef (threadH, osPriorityNormal, 5, 0);

osThreadDef (threadI, osPriorityNormal, 5, 0);

int main ()

{

osKernelInitialize();

RCC->AHB1ENR |= RCC\_AHB1ENR\_GPIOGEN | RCC\_AHB1ENR\_GPIOHEN | RCC\_AHB1ENR\_GPIOIEN;

GPIOG->MODER = 0;

GPIOH->MODER = 0;

GPIOI->MODER = 0;

GPIOG->MODER = GPIO\_MODER\_MODER6\_0 | GPIO\_MODER\_MODER7\_0 | GPIO\_MODER\_MODER8\_0;

GPIOH->MODER = GPIO\_MODER\_MODER2\_0 |GPIO\_MODER\_MODER6\_0 | GPIO\_MODER\_MODER7\_0 | GPIO\_MODER\_MODER8\_0;

GPIOI->MODER = GPIO\_MODER\_MODER10\_0 ;

tid\_arr = (osThreadId\*) malloc((thread\_count +1) \* sizeof(osThreadId));

tid\_arr[0] = osThreadCreate (osThread(threadH), (void\*)3);

tid\_arr[1] = osThreadCreate (osThread(threadH), (void\*)6);

tid\_arr[2] = osThreadCreate (osThread(threadH), (void\*)7);

tid\_arr[3] = osThreadCreate (osThread(threadI), (void\*)10);

tid\_arr[4] = osThreadCreate (osThread(threadG), (void\*)6);

tid\_arr[5] = osThreadCreate (osThread(threadG), (void\*)7);

tid\_arr[6] = osThreadCreate (osThread(threadG), (void\*)8);

tid\_arr[7] = osThreadCreate (osThread(threadH), (void\*)2);

mutex = osMutexCreate(osMutex(mutex));

osKernelStart();

}

# Текст программы 2

#include "stm32f2xx.h"

#include <cmsis\_os.h>

#include <stdlib.h>

int thread\_count=8;

osThreadId\* tid\_arr;

int on\_diods=9;

void delay (int n)

{

unsigned int i;

for (i=0; i< n \* 50000; i++) {}

}

void LEDfunc (int\* arg)

{

if (arg[0]==0) {

GPIOH->ODR |= 1 << (arg[1] );

delay(4);

GPIOH->ODR &= ~1 << (arg[1]);

}

if (arg[0]==1) {

GPIOG->ODR |= 1 << (arg[1]);

delay(4);

GPIOG->ODR &= ~1 << (arg[1]);

}

if (arg[0]==2) {

GPIOI->ODR |= 1 << (arg[1]);

delay(4);

GPIOI->ODR &= ~1 << (arg[1]);

}

}

void LEDfuncON (int\* arg)

{

if (arg[0]==0) {

GPIOH->ODR |= 1 << (arg[1] );

}

if (arg[0]==1) {

GPIOG->ODR |= 1 << (arg[1]);

}

if (arg[0]==2) {

GPIOI->ODR |= 1 << (arg[1]);

}

}

void LEDfuncOF (int\* arg)

{

if (arg[0]==0) {

GPIOH->ODR &= ~1 << (arg[1]);

}

if (arg[0]==1) {

GPIOG->ODR &= ~1 << (arg[1]);

}

if (arg[0]==2) {

GPIOI->ODR &= ~1 << (arg[1]);

}

}

osSemaphoreId mutex;

osSemaphoreDef(mutex);

unsigned int count = 0;

void thread(void const \*argument)

{

int\* arg= (int\*) argument;

while(1)

{

osSignalWait(0x01,osWaitForever);

if (arg[2]!= (on\_diods-1)){

LEDfunc((int\*)argument);

osSignalSet(tid\_arr[arg[2]+1],0x01);

}

else{

LEDfuncON(arg);

osSemaphoreWait(mutex, 0xffff); // allow only one task to increment the counter and access this turnstile

on\_diods--;

if (on\_diods == 0){

on\_diods=8;

delay(4);

GPIOI->ODR = 0;

GPIOH->ODR = 0;

GPIOG->ODR = 0;

}

osSignalSet(tid\_arr[0],0x01);

osSemaphoreRelease(mutex);

}

}

}

osThreadDef (thread, osPriorityNormal, 5, 0);

int main ()

{

int \*\*mas\_arg;

int i =0;

osKernelInitialize ();

RCC->AHB1ENR |= RCC\_AHB1ENR\_GPIOGEN | RCC\_AHB1ENR\_GPIOHEN| RCC\_AHB1ENR\_GPIOIEN;

GPIOG->MODER = 0;

GPIOH->MODER = 0;

GPIOI->MODER = 0;

GPIOG->MODER = GPIO\_MODER\_MODER6\_0 | GPIO\_MODER\_MODER7\_0 | GPIO\_MODER\_MODER8\_0;

GPIOH->MODER = GPIO\_MODER\_MODER2\_0 | GPIO\_MODER\_MODER3\_0|GPIO\_MODER\_MODER6\_0 | GPIO\_MODER\_MODER7\_0 | GPIO\_MODER\_MODER10\_0;

GPIOI->MODER = GPIO\_MODER\_MODER10\_0 ;

tid\_arr = (osThreadId\*) malloc((thread\_count +1) \* sizeof(osThreadId));

mutex = osSemaphoreCreate(osSemaphore(mutex), 1);

mas\_arg= (int\*\*) malloc((8) \* sizeof(int\*));

for( i=0; i<thread\_count+1; i++){

mas\_arg[i] = (int\*) malloc((3) \* sizeof(int));

}

mas\_arg[0][0]= 0;

mas\_arg[0][1]= 3;

mas\_arg[0][2]= 0;

mas\_arg[1][0]= 0;

mas\_arg[1][1]= 6;

mas\_arg[1][2]= 1;

mas\_arg[2][0]= 0;

mas\_arg[2][1]= 7;

mas\_arg[2][2]= 2;

mas\_arg[3][0]= 2;

mas\_arg[3][1]= 10;

mas\_arg[3][2]= 3;

mas\_arg[4][0]= 1;

mas\_arg[4][1]= 6;

mas\_arg[4][2]= 4;

mas\_arg[5][0]= 1;

mas\_arg[5][1]= 7;

mas\_arg[5][2]= 5;

mas\_arg[6][0]= 1;

mas\_arg[6][1]= 8;

mas\_arg[6][2]= 6;

mas\_arg[7][0]= 0;

mas\_arg[7][1]= 2;

mas\_arg[7][2]= 7;

mas\_arg[8][0]= 0;

mas\_arg[8][1]= 2;

mas\_arg[8][2]= 8;

for( i=0; i<thread\_count+1; i++){

tid\_arr[i] = osThreadCreate (osThread(thread), (void\*)mas\_arg[i]);

osSignalClear(tid\_arr[i],0x01);

}

osKernelStart ();

osSignalSet(tid\_arr[0],0x01);

}

# Выводы

В данной работе был реализован барьер на основе семафоров для синхронизации неограниченного числа потоков. С помощью данного барьера можно имлпементировать синхронизационные точки, в которых потоки будут дожидаться все остальных, после чего продолжать свое дальнейшее выполнение.