**Introduction:**

This lab demonstrates the proper use of RTOS concepts such as semaphores and mutexes. The source code is provided in order of the lab\_8 header file, the main routine source code, and the threads in ascending order (as documented). The Serial.c, Serial.h, and RTOS-specific files have been omitted due to no modification deviation from the vendor-provided code.

**Appendix: Source code**

#ifndef example\_lab\_8

#define example\_lab\_8

#include "stdint.h"

#include "Board\_Joystick.h"

#include "Board\_GLCD.h"

#include "cmsis\_os2.h"

#include "rtx\_os.h"

/\*\*

\* Modified from assignment provided sources:

\* Author of modifications: Peter A. Dranishnikov

\* Lab #: 8

\* Course: EEL4685C

\* Due date: April 23, Spring 2019

\*/

// shared variables

extern uint32\_t treceiveChar;

extern uint32\_t ttimerSec;

extern uint32\_t tgear;

extern uint32\_t joychar;

// mutexes

// Note, actual variable declarations are in main.c, since exactly one file must create the space.

// The extern statements tell other files that the variable already exists, and they are allowed to access it.

extern osMutexId\_t mut1Display;

extern osMutexId\_t mut2Ser; //mutex for serial output

// flag semaphores

extern osSemaphoreId\_t semTick;

extern osSemaphoreId\_t semChar;

extern osSemaphoreId\_t semSM;

//extern osSemaphoreId\_t semDisp; //not needed, semTick handles display thread

//threads

extern osThreadId\_t tid\_thdDisplay;

int Init\_thdDisplay (void);

extern osThreadId\_t tid\_thdTick;

int Init\_thdTick (void);

extern osThreadId\_t tid\_thdChar;

int Init\_thdChar (void);

extern osThreadId\_t tid\_thdJoystick;

int Init\_thdJoystick (void);

//New: sm thread declaration

extern osThreadId\_t tid\_thdStateMachine;

int Init\_thdStateMachine (void);

#endif

/\*\*

\* Modified from assignment provided sources:

\* Author of modifications: Peter A. Dranishnikov

\* Lab #: 8

\* Course: EEL4685C

\* Due date: April 23, Spring 2019

\*/

/\*----------------------------------------------------------------------------

\* CMSIS-RTOS 'main' function

\* Contains all thread initialization (with support function), mutexes,

\* semaphores, and an IRQHandler for serial input

\*---------------------------------------------------------------------------\*/

#include "RTE\_Components.h"

#include CMSIS\_device\_header

#include "serial.h"

#include "GLCD\_Config.h"

#include "stm32f2xx\_hal.h"

#include "lab\_8.h" // specific to this project

extern GLCD\_FONT GLCD\_Font\_16x24;

// Note that the main file declares the space for all the system's variables (someone has to),

// and "extern" declarations are in the lab\_8.h file so other files can find them

osMutexId\_t mut1Display;

osMutexId\_t mut2Ser;

osSemaphoreId\_t semTick;

osSemaphoreId\_t semChar;

osSemaphoreId\_t semSM;

//osSemaphoreId\_t semDisp;

uint32\_t treceiveChar;

void serial\_first\_message(void);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// The RTOS and HAL need the SysTick for timing. The RTOS wins and gets control

// of SysTick, so we need to route the HAL's tick call to the RTOS's tick.

// Don't mess with this code.

uint32\_t HAL\_GetTick(void) {

return osKernelGetTickCount();

}

/\*----------------------------------------------------------------------------

\* HW Init - since the HAL depends on a periodic timer, we need the RTOS

\* in order for several HW devices to initialize correctly, like the GLCD

\*---------------------------------------------------------------------------\*/

void app\_hw\_init (void \*argument) {

GLCD\_Initialize();

GLCD\_SetBackgroundColor(GLCD\_COLOR\_PURPLE);

GLCD\_SetForegroundColor(GLCD\_COLOR\_WHITE);

GLCD\_ClearScreen();

GLCD\_SetFont(&GLCD\_Font\_16x24);

Joystick\_Initialize(); // Note: Joystick now uses HAL functions

SER\_Init(115200); // 115200 baud, 8 data bits, 1 stop bits, no flow control

// configures and enables the interrupt for the USART 3 serial port.

USART3->CR1 |= USART\_CR1\_RXNEIE;

NVIC->ISER[ USART3\_IRQn/32] = (1UL << (USART3\_IRQn%32));

NVIC->IP[USART3\_IRQn] = 0x80;

serial\_first\_message();//print first serial message

// Create other threads here so that all initialization is done before others get scheduled.

Init\_thdStateMachine();

Init\_thdDisplay();

Init\_thdTick();

Init\_thdChar();

Init\_thdJoystick();

osThreadExit(); // job is done, thread suicide. There better be other threads created above...

}

int main (void) {

SystemCoreClockUpdate(); // always first, make sure the clock freq. is current

osKernelInitialize(); // Initialize CMSIS-RTOS

HAL\_Init();

mut1Display = osMutexNew(NULL);

if (mut1Display==NULL) while(1){} // failed, scream and die

mut2Ser = osMutexNew(NULL);

if (mut2Ser == NULL) while(1){} //surrender on failure

semTick = osSemaphoreNew(1000, 1, NULL);

if (semTick==NULL) while(1){} // failed, scream and die

semChar = osSemaphoreNew(1000, 0, NULL);

if (semChar==NULL) while(1){} // failed, scream and die

semSM = osSemaphoreNew(1000, 1, NULL);

if (semSM == NULL) while(1){} // surrender on failure

osThreadNew(app\_hw\_init, NULL, NULL); // Create application's main thread to init HW now that HAL is running

osKernelStart(); // Start thread execution

for (;;) {} // should never get here

}

// Interrupt serv routine for the serial port. Triggered upon receiving any character or space

void USART3\_IRQHandler(void)

{

treceiveChar = SER\_GetChar();

if (treceiveChar == 'D' || treceiveChar == 'd' || treceiveChar == 'U' || treceiveChar == 'u')

{

osSemaphoreRelease(semSM);

}

else; //do nothing

}

void serial\_first\_message(void)

{

//"Time\tGear\r\n"; equivalent null-terminated string

const uint32\_t init\_message[] = {'T', 'i', 'm', 'e', '\t', 'G', 'e', 'a', 'r', '\r', '\n', '\0'};

osMutexAcquire(mut2Ser, osWaitForever);

for(uint32\_t i = 0; init\_message[i] != '\0'; i++)

{

SER\_PutChar(init\_message[i]);

}

osMutexRelease(mut2Ser);

}

#include "lab\_8.h"

/\*\*

\* Modified from assignment provided sources:

\* Author of modifications: Peter A. Dranishnikov

\* Lab #: 8

\* Course: EEL4685C

\* Due date: April 23, Spring 2019

\*/

/\*----------------------------------------------------------------------------

\* Thread 1 'Thread\_Char': Display character thread

\* Displays the current gear to the main onboard display. Initializes on boot

\* to display the current gear (at initialization, 0)

\* Triggers on the character semaphore

\*---------------------------------------------------------------------------\*/

void thdChar (void \*argument); // thread function

osThreadId\_t tid\_thdChar; // thread id

int Init\_thdChar (void) {

tid\_thdChar = osThreadNew (thdChar, NULL, NULL);

if (!tid\_thdChar) return(-1);

osMutexAcquire(mut1Display, osWaitForever);

GLCD\_DrawChar(100,100,(tgear + 48));

osMutexRelease(mut1Display);

return(0);

}

void thdChar (void \*argument)

{

while (1)

{

osSemaphoreAcquire(semChar, osWaitForever);

osMutexAcquire(mut1Display, osWaitForever);

GLCD\_DrawChar(100,100,(tgear + 48));

osMutexRelease(mut1Display);

}

}

#include "lab\_8.h"

#include "serial.h"

/\*\*

\* Modified from assignment provided sources:

\* Author of modifications: Peter A. Dranishnikov

\* Lab #: 8

\* Course: EEL4685C

\* Due date: April 23, Spring 2019

\*/

/\*----------------------------------------------------------------------------

\* Thread 2 'Thread\_Display': Serial log thread

\* This may sound like a misnomer, but it indeed "displays" to the serial port.

\* Data displayed include the thread timer's value and the current gear

\* Triggers on the tick semaphore

\*---------------------------------------------------------------------------\*/

void thdDisplay (void \*argument); // thread function

osThreadId\_t tid\_thdDisplay; // thread id

int Init\_thdDisplay (void) {

tid\_thdDisplay = osThreadNew(thdDisplay, NULL, NULL);

if (!tid\_thdDisplay) return(-1);

return(0);

}

void thdDisplay (void \*argument)

{

//placeholder text, first 8 characters overritten

uint32\_t text[11] = {'0','0','0','.','0','0',' ','g','\r','\n','\0'};

uint32\_t timerDecomp[6];

while (1)

{

osSemaphoreAcquire(semTick, osWaitForever);

//isolate decimal digits for display

timerDecomp[5] = ttimerSec % 10;

timerDecomp[4] = (ttimerSec % 100) / 10;

timerDecomp[2] = (ttimerSec % 1000) / 100;

timerDecomp[1] = (ttimerSec % 10000) / 1000;

timerDecomp[0] = (ttimerSec % 100000) / 10000; //don't know of a better way

//convert to ascii

for(uint32\_t i = 0; i < 6; i++)

{

if(i != 3)

{

text[i] = timerDecomp[i] + 48;

}

}

text[7] = tgear + 48;

//dump to serial

osMutexAcquire(mut2Ser, osWaitForever);

for(uint32\_t i = 0; text[i] != '\0'; i++)

SER\_PutChar(text[i]);

osMutexRelease(mut2Ser);

}

}

#include "lab\_8.h"

/\*\*

\* Modified from assignment provided sources:

\* Author of modifications: Peter A. Dranishnikov

\* Lab #: 8

\* Course: EEL4685C

\* Due date: April 23, Spring 2019

\*/

/\*----------------------------------------------------------------------------

\* Thread 3 'Thread\_Joystick': Joystick thread

\* This thread is on a perpetual poll for new joystick positions

\* This thread does not trigger on any semaphore

\* This file used to have a drunk indentation format, which should be corrected

\* However, there may be a remaining combination of tabs and spaces

\* I apologize for any inconvenience this may cause when parsing

\*---------------------------------------------------------------------------\*/

void thdJoystick (void \*argument); // thread function

osThreadId\_t tid\_thdJoystick; // thread id

uint32\_t joychar = ' ';

int Init\_thdJoystick (void) {

tid\_thdJoystick = osThreadNew (thdJoystick, NULL, NULL);

if (!tid\_thdJoystick) return(-1);

return(0);

}

void thdJoystick (void \*argument) {

uint32\_t newjoystick, joystick;

while (1)

{

osDelay(osKernelGetTickFreq()/10);

newjoystick = Joystick\_GetState();

if (joystick != newjoystick)

{

switch (newjoystick)

{

case JOYSTICK\_UP:

{

joychar = 'U';

osSemaphoreRelease(semSM);

break;

}

case JOYSTICK\_DOWN:

{

joychar = 'D';

osSemaphoreRelease(semSM);

break;

}

default:

break;

}

}

joystick = newjoystick;

}

}

#include "lab\_8.h"

/\*\*

\* Modified from assignment provided sources:

\* Author of modifications: Peter A. Dranishnikov

\* Lab #: 8

\* Course: EEL4685C

\* Due date: April 23, Spring 2019

\*/

/\*----------------------------------------------------------------------------

\* Thread 4 'Thread\_StateMachine': State Machine thread

\* This thread is the primary state machine of a supposed gear shifter

\* Triggers on the statemachine semaphore

\*

\*---------------------------------------------------------------------------\*/

void thdStateMachine(void \*argument);

void fsm\_reaction(bool up, bool down);

osThreadId\_t tid\_thdStateMachine;

uint32\_t tgear;

int Init\_thdStateMachine(void)

{

tid\_thdStateMachine = osThreadNew(thdStateMachine, NULL, NULL);

if(!tid\_thdStateMachine) return(-1);

tgear = 0;

return(0);

}

void thdStateMachine(void \*argument)

{

while(1)

{

osSemaphoreAcquire(semSM, osWaitForever);

bool u = joychar == 'D' || (treceiveChar == 'U' || treceiveChar == 'u');

bool d = joychar == 'U' || (treceiveChar == 'D' || treceiveChar == 'd');

fsm\_reaction(u, d);

//IMPORTANT: these values must be set to zero for the state machine to transition correctly

treceiveChar = 0x00;

joychar = 0x00;

}

}

void fsm\_reaction(bool up, bool down)

{

switch (tgear)

{

case 0:

if (up && !down)

{

tgear = 1;

osSemaphoreRelease(semChar);

osSemaphoreRelease(semTick);

}

else if (up && down)

{

tgear = 0;

osSemaphoreRelease(semChar);

osSemaphoreRelease(semTick);

}

else if (!up && down)

{

tgear = 0;

osSemaphoreRelease(semChar);

osSemaphoreRelease(semTick);

}

else

{

osSemaphoreRelease(semChar);

osSemaphoreRelease(semTick);

}

break;

case 1:

if (up && !down)

{

tgear = 2;

osSemaphoreRelease(semChar);

osSemaphoreRelease(semTick);

}

else if (up && down)

{

osSemaphoreRelease(semChar);

osSemaphoreRelease(semTick);

tgear = 1;

osSemaphoreRelease(semChar);

osSemaphoreRelease(semTick);

}

else if (!up && down)

{

tgear = 0;

osSemaphoreRelease(semChar);

osSemaphoreRelease(semTick);

}

else;

break;

case 2:

if (up && !down)

{

tgear = 3;

osSemaphoreRelease(semChar);

osSemaphoreRelease(semTick);

}

else if (up && down)

{

tgear = 2;

osSemaphoreRelease(semChar);

osSemaphoreRelease(semTick);

}

else if (!up && down)

{

tgear = 1;

osSemaphoreRelease(semChar);

osSemaphoreRelease(semTick);

}

else;

break;

case 3:

if (up && !down)

{

tgear = 4;

osSemaphoreRelease(semChar);

osSemaphoreRelease(semTick);

}

else if (up && down)

{

tgear = 3;

osSemaphoreRelease(semChar);

osSemaphoreRelease(semTick);

}

else if (!up && down)

{

tgear = 2;

osSemaphoreRelease(semChar);

osSemaphoreRelease(semTick);

}

else;

break;

case 4:

if (up && !down)

{

tgear = 5;

osSemaphoreRelease(semChar);

osSemaphoreRelease(semTick);

}

else if (up && down)

{

tgear = 4;

osSemaphoreRelease(semChar);

osSemaphoreRelease(semTick);

}

else if (!up && down)

{

tgear = 3;

osSemaphoreRelease(semChar);

osSemaphoreRelease(semTick);

}

else;

break;

case 5:

if (up && !down)

{

tgear = 6;

osSemaphoreRelease(semChar);

osSemaphoreRelease(semTick);

}

else if (up && down)

{

tgear = 5;

osSemaphoreRelease(semChar);

osSemaphoreRelease(semTick);

}

else if (!up && down)

{

tgear = 4;

osSemaphoreRelease(semChar);

osSemaphoreRelease(semTick);

}

else;

break;

case 6:

if (up && !down)

{

tgear = 6;

osSemaphoreRelease(semChar);

osSemaphoreRelease(semTick);

}

else if (up && down)

{

tgear = 6;

osSemaphoreRelease(semChar);

osSemaphoreRelease(semTick);

}

else if (!up && down)

{

tgear = 5;

osSemaphoreRelease(semChar);

osSemaphoreRelease(semTick);

}

else;

break;

default:

for(;;){}

}

}

#include "cmsis\_os2.h" // CMSIS RTOS header file

#include "lab\_8.h"

/\*\*

\* Modified from assignment provided sources:

\* Author of modifications: Peter A. Dranishnikov

\* Lab #: 8

\* Course: EEL4685C

\* Due date: April 23, Spring 2019

\*/

/\*----------------------------------------------------------------------------

\* Thread 5 'Thread\_tick': Ticker thread

\* This thread ticks. Thats it.

\* This thread functions autonomously after init, but the delay can be adjusted

\* at compile time using the tickTime constant

\*---------------------------------------------------------------------------\*/

void thdTick (void \*argument); // thread function

osThreadId\_t tid\_thdTick; // thread id

const uint32\_t tickTime = 100;

uint32\_t ttimerSec;

int Init\_thdTick (void) {

tid\_thdTick = osThreadNew (thdTick, NULL, NULL);

if (!tid\_thdTick) return(-1);

ttimerSec = 0;

return(0);

}

void thdTick (void \*argument) {

while (1)

{

osSemaphoreRelease(semTick);

ttimerSec++;

osDelay(osKernelGetTickFreq()/tickTime);

}

}