# Changed Priority

/\* USER CODE BEGIN Header \*/

/\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* @file : main.c

\* @brief : Main program body

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* @attention

\*

\* <h2><center>&copy; Copyright (c) 2020 STMicroelectronics.

\* All rights reserved.</center></h2>

\*

\* This software component is licensed by ST under Ultimate Liberty license

\* SLA0044, the "License"; You may not use this file except in compliance with

\* the License. You may obtain a copy of the License at:

\* www.st.com/SLA0044

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

/\* USER CODE END Header \*/

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

#include "main.h"

#include "cmsis\_os.h"

#include "string.h"

#include "semphr.h"

#include "queue.h"

#include "FreeRTOS.h"

#include "task.h"

/\* Private includes ----------------------------------------------------------\*/

/\* USER CODE BEGIN Includes \*/

/\* USER CODE END Includes \*/

/\* Private typedef -----------------------------------------------------------\*/

/\* USER CODE BEGIN PTD \*/

/\* USER CODE END PTD \*/

/\* Private define ------------------------------------------------------------\*/

/\* USER CODE BEGIN PD \*/

uint8\_t myTxData[17] = "Mutex Created\r\n";

/\* USER CODE END PD \*/

/\* Private macro -------------------------------------------------------------\*/

/\* USER CODE BEGIN PM \*/

/\* USER CODE END PM \*/

/\* Private variables ---------------------------------------------------------\*/

I2C\_HandleTypeDef hi2c1;

SPI\_HandleTypeDef hspi1;

UART\_HandleTypeDef huart1;

UART\_HandleTypeDef huart2;

///\* Definitions for defaultTask \*/

//osThreadId\_t defaultTaskHandle;

//const osThreadAttr\_t defaultTask\_attributes = {

// .name = "defaultTask",

// .priority = (osPriority\_t) osPriorityNormal,

// .stack\_size = 128

//};

///\* Definitions for myTask02 \*/

//osThreadId\_t myTask02Handle;

//const osThreadAttr\_t myTask02\_attributes = {

// .name = "myTask02",

// .priority = (osPriority\_t) osPriorityNormal,

// .stack\_size = 128

//};

///\* Definitions for myTask03 \*/

//osThreadId\_t myTask03Handle;

//const osThreadAttr\_t myTask03\_attributes = {

// .name = "myTask03",

// .priority = (osPriority\_t) osPriorityLow,

// .stack\_size = 128

//};

///\* Definitions for myTask04 \*/

//osThreadId\_t myTask04Handle;

//const osThreadAttr\_t myTask04\_attributes = {

// .name = "myTask04",

// .priority = (osPriority\_t) osPriorityNormal,

// .stack\_size = 128

//};

/\* Definitions for mutex01 \*/

osMutexId\_t mutex01Handle;

const osMutexAttr\_t mutex01\_attributes = {

.name = "mutex01"

};

/\* Definitions for semaphore01 \*/

osSemaphoreId\_t semaphore01Handle;

const osSemaphoreAttr\_t semaphore01\_attributes = {

.name = "semaphore01"

};

/\* USER CODE BEGIN PV \*/

/\* USER CODE END PV \*/

/\* Private function prototypes -----------------------------------------------\*/

void SystemClock\_Config(void);

static void MX\_GPIO\_Init(void);

static void MX\_I2C1\_Init(void);

static void MX\_SPI1\_Init(void);

static void MX\_USART1\_UART\_Init(void);

static void MX\_USART2\_UART\_Init(void);

/\* USER CODE BEGIN PFP \*/

/\* USER CODE END PFP \*/

/\* Private user code ---------------------------------------------------------\*/

/\* USER CODE BEGIN 0 \*/

SemaphoreHandle\_t SimpleMutex; // define the semaphore handler

TaskHandle\_t First\_Task\_Handler; // defined three tasks

TaskHandle\_t MiddlePT\_Handler; // using task Handler

TaskHandle\_t LessPT\_Handler; //

TaskHandle\_t Fourth\_Handler;

TaskHandle\_t Fifth\_Handler;

void First\_PT (void \*argument);

void MiddlePT\_Task (void \*argument);

void LessPT\_Task (void \*argument);

void Fourth\_Task (void \*argument);

void Fifth\_Task (void \*argument);

void Send\_Uart (char \*str)

{

xSemaphoreTake(SimpleMutex, portMAX\_DELAY);

HAL\_UART\_Transmit(&huart2,str, strlen(str), HAL\_MAX\_DELAY);

HAL\_Delay(2000);

xSemaphoreGive(SimpleMutex);

}

/\* USER CODE END 0 \*/

/\*\*

\* @brief The application entry point.

\* @retval int

\*/

int main(void)

{

/\* USER CODE BEGIN 1 \*/

/\* USER CODE END 1 \*/

/\* MCU Configuration--------------------------------------------------------\*/

/\* Reset of all peripherals, Initializes the Flash interface and the Systick. \*/

HAL\_Init();

/\* USER CODE BEGIN Init \*/

/\* USER CODE END Init \*/

/\* Configure the system clock \*/

SystemClock\_Config();

/\* USER CODE BEGIN SysInit \*/

/\* USER CODE END SysInit \*/

/\* Initialize all configured peripherals \*/

MX\_GPIO\_Init();

MX\_I2C1\_Init();

MX\_SPI1\_Init();

MX\_USART1\_UART\_Init();

MX\_USART2\_UART\_Init();

/\* USER CODE BEGIN 2 \*/

/\* USER CODE END 2 \*/

/\* Init scheduler \*/

osKernelInitialize();

/\* Create the mutex(es) \*/

/\* creation of mutex01 \*/

mutex01Handle = osMutexNew(&mutex01\_attributes);

/\* USER CODE BEGIN RTOS\_MUTEX \*/

/\* add mutexes, ... \*/

/\* USER CODE END RTOS\_MUTEX \*/

/\* Create the semaphores(s) \*/

/\* creation of semaphore01 \*/

semaphore01Handle = osSemaphoreNew(2, 2, &semaphore01\_attributes);

/\* USER CODE BEGIN RTOS\_SEMAPHORES \*/

/\* add semaphores, ... \*/

/\* USER CODE END RTOS\_SEMAPHORES \*/

/\* USER CODE BEGIN RTOS\_TIMERS \*/

/\* start timers, add new ones, ... \*/

/\* USER CODE END RTOS\_TIMERS \*/

/\* USER CODE BEGIN RTOS\_QUEUES \*/

/\* add queues, ... \*/

/\* USER CODE END RTOS\_QUEUES \*/

/\* Create the thread(s) \*/

/\* creation of defaultTask \*/

/\* Start scheduler \*/

/\* We should never get here as control is now taken by the scheduler \*/

/\* Infinite loop \*/

/\* USER CODE BEGIN WHILE \*/

SimpleMutex = xSemaphoreCreateMutex();

if (SimpleMutex != NULL)

{

HAL\_UART\_Transmit(&huart2,myTxData,15 ,1000);

}

// priority changed

xTaskCreate(First\_PT,"First\_Task",128,NULL,2,&First\_Task\_Handler); // priority changed from 3 to 2

xTaskCreate(MiddlePT\_Task,"MiddlePT",128,NULL,1,&MiddlePT\_Handler); // priority changed from 2 to 1

xTaskCreate(LessPT\_Task,"LessPT",128,NULL,0,&LessPT\_Handler); // priority changed from 1 to 0

xTaskCreate (Fourth\_Task,"Fourth\_task",128,NULL,3,&First\_Task\_Handler); // priority changed from 0 to 3

vTaskStartScheduler();

while (1)

{

/\* USER CODE END WHILE \*/

/\* USER CODE BEGIN 3 \*/

}

/\* USER CODE END 3 \*/

}

/\*\*

\* @brief System Clock Configuration

\* @retval None

\*/

void SystemClock\_Config(void)

{

RCC\_OscInitTypeDef RCC\_OscInitStruct = {0};

RCC\_ClkInitTypeDef RCC\_ClkInitStruct = {0};

RCC\_PeriphCLKInitTypeDef PeriphClkInit = {0};

/\*\* Initializes the CPU, AHB and APB busses clocks

\*/

RCC\_OscInitStruct.OscillatorType = RCC\_OSCILLATORTYPE\_HSE;

RCC\_OscInitStruct.HSEState = RCC\_HSE\_BYPASS;

RCC\_OscInitStruct.PLL.PLLState = RCC\_PLL\_NONE;

if (HAL\_RCC\_OscConfig(&RCC\_OscInitStruct) != HAL\_OK)

{

Error\_Handler();

}

/\*\* Initializes the CPU, AHB and APB busses clocks

\*/

RCC\_ClkInitStruct.ClockType = RCC\_CLOCKTYPE\_HCLK|RCC\_CLOCKTYPE\_SYSCLK

|RCC\_CLOCKTYPE\_PCLK1|RCC\_CLOCKTYPE\_PCLK2;

RCC\_ClkInitStruct.SYSCLKSource = RCC\_SYSCLKSOURCE\_HSE;

RCC\_ClkInitStruct.AHBCLKDivider = RCC\_SYSCLK\_DIV1;

RCC\_ClkInitStruct.APB1CLKDivider = RCC\_HCLK\_DIV1;

RCC\_ClkInitStruct.APB2CLKDivider = RCC\_HCLK\_DIV1;

if (HAL\_RCC\_ClockConfig(&RCC\_ClkInitStruct, FLASH\_LATENCY\_0) != HAL\_OK)

{

Error\_Handler();

}

PeriphClkInit.PeriphClockSelection = RCC\_PERIPHCLK\_USART1|RCC\_PERIPHCLK\_USART2

|RCC\_PERIPHCLK\_I2C1;

PeriphClkInit.Usart1ClockSelection = RCC\_USART1CLKSOURCE\_PCLK2;

PeriphClkInit.Usart2ClockSelection = RCC\_USART2CLKSOURCE\_PCLK1;

PeriphClkInit.I2c1ClockSelection = RCC\_I2C1CLKSOURCE\_PCLK1;

if (HAL\_RCCEx\_PeriphCLKConfig(&PeriphClkInit) != HAL\_OK)

{

Error\_Handler();

}

/\*\* Configure the main internal regulator output voltage

\*/

if (HAL\_PWREx\_ControlVoltageScaling(PWR\_REGULATOR\_VOLTAGE\_SCALE1) != HAL\_OK)

{

Error\_Handler();

}

}

/\*\*

\* @brief I2C1 Initialization Function

\* @param None

\* @retval None

\*/

static void MX\_I2C1\_Init(void)

{

/\* USER CODE BEGIN I2C1\_Init 0 \*/

/\* USER CODE END I2C1\_Init 0 \*/

/\* USER CODE BEGIN I2C1\_Init 1 \*/

/\* USER CODE END I2C1\_Init 1 \*/

hi2c1.Instance = I2C1;

hi2c1.Init.Timing = 0x2000090E;

hi2c1.Init.OwnAddress1 = 0;

hi2c1.Init.AddressingMode = I2C\_ADDRESSINGMODE\_7BIT;

hi2c1.Init.DualAddressMode = I2C\_DUALADDRESS\_DISABLE;

hi2c1.Init.OwnAddress2 = 0;

hi2c1.Init.OwnAddress2Masks = I2C\_OA2\_NOMASK;

hi2c1.Init.GeneralCallMode = I2C\_GENERALCALL\_DISABLE;

hi2c1.Init.NoStretchMode = I2C\_NOSTRETCH\_DISABLE;

if (HAL\_I2C\_Init(&hi2c1) != HAL\_OK)

{

Error\_Handler();

}

/\*\* Configure Analogue filter

\*/

if (HAL\_I2CEx\_ConfigAnalogFilter(&hi2c1, I2C\_ANALOGFILTER\_ENABLE) != HAL\_OK)

{

Error\_Handler();

}

/\*\* Configure Digital filter

\*/

if (HAL\_I2CEx\_ConfigDigitalFilter(&hi2c1, 0) != HAL\_OK)

{

Error\_Handler();

}

/\* USER CODE BEGIN I2C1\_Init 2 \*/

/\* USER CODE END I2C1\_Init 2 \*/

}

/\*\*

\* @brief SPI1 Initialization Function

\* @param None

\* @retval None

\*/

static void MX\_SPI1\_Init(void)

{

/\* USER CODE BEGIN SPI1\_Init 0 \*/

/\* USER CODE END SPI1\_Init 0 \*/

/\* USER CODE BEGIN SPI1\_Init 1 \*/

/\* USER CODE END SPI1\_Init 1 \*/

/\* SPI1 parameter configuration\*/

hspi1.Instance = SPI1;

hspi1.Init.Mode = SPI\_MODE\_MASTER;

hspi1.Init.Direction = SPI\_DIRECTION\_2LINES;

hspi1.Init.DataSize = SPI\_DATASIZE\_4BIT;

hspi1.Init.CLKPolarity = SPI\_POLARITY\_LOW;

hspi1.Init.CLKPhase = SPI\_PHASE\_1EDGE;

hspi1.Init.NSS = SPI\_NSS\_SOFT;

hspi1.Init.BaudRatePrescaler = SPI\_BAUDRATEPRESCALER\_2;

hspi1.Init.FirstBit = SPI\_FIRSTBIT\_MSB;

hspi1.Init.TIMode = SPI\_TIMODE\_DISABLE;

hspi1.Init.CRCCalculation = SPI\_CRCCALCULATION\_DISABLE;

hspi1.Init.CRCPolynomial = 7;

hspi1.Init.CRCLength = SPI\_CRC\_LENGTH\_DATASIZE;

hspi1.Init.NSSPMode = SPI\_NSS\_PULSE\_ENABLE;

if (HAL\_SPI\_Init(&hspi1) != HAL\_OK)

{

Error\_Handler();

}

/\* USER CODE BEGIN SPI1\_Init 2 \*/

/\* USER CODE END SPI1\_Init 2 \*/

}

/\*\*

\* @brief USART1 Initialization Function

\* @param None

\* @retval None

\*/

static void MX\_USART1\_UART\_Init(void)

{

/\* USER CODE BEGIN USART1\_Init 0 \*/

/\* USER CODE END USART1\_Init 0 \*/

/\* USER CODE BEGIN USART1\_Init 1 \*/

/\* USER CODE END USART1\_Init 1 \*/

huart1.Instance = USART1;

huart1.Init.BaudRate = 115200;

huart1.Init.WordLength = UART\_WORDLENGTH\_8B;

huart1.Init.StopBits = UART\_STOPBITS\_1;

huart1.Init.Parity = UART\_PARITY\_NONE;

huart1.Init.Mode = UART\_MODE\_TX\_RX;

huart1.Init.HwFlowCtl = UART\_HWCONTROL\_NONE;

huart1.Init.OverSampling = UART\_OVERSAMPLING\_16;

huart1.Init.OneBitSampling = UART\_ONE\_BIT\_SAMPLE\_DISABLE;

huart1.AdvancedInit.AdvFeatureInit = UART\_ADVFEATURE\_NO\_INIT;

if (HAL\_UART\_Init(&huart1) != HAL\_OK)

{

Error\_Handler();

}

/\* USER CODE BEGIN USART1\_Init 2 \*/

/\* USER CODE END USART1\_Init 2 \*/

}

/\*\*

\* @brief USART2 Initialization Function

\* @param None

\* @retval None

\*/

static void MX\_USART2\_UART\_Init(void)

{

/\* USER CODE BEGIN USART2\_Init 0 \*/

/\* USER CODE END USART2\_Init 0 \*/

/\* USER CODE BEGIN USART2\_Init 1 \*/

/\* USER CODE END USART2\_Init 1 \*/

huart2.Instance = USART2;

huart2.Init.BaudRate = 115200;

huart2.Init.WordLength = UART\_WORDLENGTH\_8B;

huart2.Init.StopBits = UART\_STOPBITS\_1;

huart2.Init.Parity = UART\_PARITY\_NONE;

huart2.Init.Mode = UART\_MODE\_TX\_RX;

huart2.Init.HwFlowCtl = UART\_HWCONTROL\_NONE;

huart2.Init.OverSampling = UART\_OVERSAMPLING\_16;

huart2.Init.OneBitSampling = UART\_ONE\_BIT\_SAMPLE\_DISABLE;

huart2.AdvancedInit.AdvFeatureInit = UART\_ADVFEATURE\_NO\_INIT;

if (HAL\_UART\_Init(&huart2) != HAL\_OK)

{

Error\_Handler();

}

/\* USER CODE BEGIN USART2\_Init 2 \*/

/\* USER CODE END USART2\_Init 2 \*/

}

/\*\*

\* @brief GPIO Initialization Function

\* @param None

\* @retval None

\*/

static void MX\_GPIO\_Init(void)

{

GPIO\_InitTypeDef GPIO\_InitStruct = {0};

/\* GPIO Ports Clock Enable \*/

\_\_HAL\_RCC\_GPIOC\_CLK\_ENABLE();

\_\_HAL\_RCC\_GPIOA\_CLK\_ENABLE();

\_\_HAL\_RCC\_GPIOB\_CLK\_ENABLE();

/\*Configure GPIO pin Output Level \*/

HAL\_GPIO\_WritePin(GPIOB, LD3\_Pin|GPIO\_PIN\_4|GPIO\_PIN\_5, GPIO\_PIN\_RESET);

/\*Configure GPIO pins : LD3\_Pin PB4 PB5 \*/

GPIO\_InitStruct.Pin = LD3\_Pin|GPIO\_PIN\_4|GPIO\_PIN\_5;

GPIO\_InitStruct.Mode = GPIO\_MODE\_OUTPUT\_PP;

GPIO\_InitStruct.Pull = GPIO\_NOPULL;

GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_LOW;

HAL\_GPIO\_Init(GPIOB, &GPIO\_InitStruct);

}

/\* USER CODE BEGIN 4 \*/

void First\_PT (void \*argument)

{ char \*strtosend = "In First\_Task=====================\r\n";

while (1)

{

char \*str = "Enter First\_Task and about to enter in to mutex \r\n";

HAL\_UART\_Transmit(&huart2, str, strlen(str),HAL\_MAX\_DELAY);

Send\_Uart(strtosend);

char \*str2 = "Leaving First\_Task=====================\r\n ";

HAL\_UART\_Transmit(&huart2, str2, strlen(str2),HAL\_MAX\_DELAY);

vTaskDelay(1500);

}

}

void MiddlePT\_Task (void \*argument)

{ char \*strtosend = "In MiddlePT ...............................\r\n";

while (1)

{

char \*str = "\n Enter MiddlePT and about to enter in to mutex \r\n";

HAL\_UART\_Transmit(&huart2, str, strlen(str),HAL\_MAX\_DELAY);

Send\_Uart(strtosend);

char \*str2 = "\n Leaving MiddlePT ............................\r\n";

HAL\_UART\_Transmit(&huart2, str2, strlen(str2),HAL\_MAX\_DELAY);

vTaskDelay(2000);

}

}

void LessPT\_Task (void \*argument)

{ char \*strtosend = "\n In LessPT \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\r\n";

while (1)

{

char \*str = "\n Enter LessPT and about to enter in to mutex \r\n";

HAL\_UART\_Transmit(&huart2, str, strlen(str),HAL\_MAX\_DELAY);

Send\_Uart(strtosend);

char \*str2 = "\n Leaving LessPT \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\r\n";

HAL\_UART\_Transmit(&huart2, str2, strlen(str2),HAL\_MAX\_DELAY);

vTaskDelay(3000);

}

}

void Fourth\_Task (void \*argument)

{ char \*strtosend = "\n In FourthPT %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%\r\n";

while (1)

{

char \*str = "\n Enter FourthPT and about to enter in to mutex \r\n ";

HAL\_UART\_Transmit(&huart2, str, strlen(str),HAL\_MAX\_DELAY);

Send\_Uart(strtosend);

char \*str2 = "\n Leaving FourthPT %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%\r\n";

HAL\_UART\_Transmit(&huart2, str2, strlen(str2),HAL\_MAX\_DELAY);

vTaskDelay(3000);

}

}

/\* USER CODE BEGIN 4 \*/

// Callback functions used as interrupt handlers for each serial

// peripheral. Should create buffers for each one and keep interrupt

// handlers short.

void HAL\_UART\_RxCpltCallback(UART\_HandleTypeDef \*huart) {

}

void HAL\_SPI\_RxCpltCallback(SPI\_HandleTypeDef \*hspi) {

}

void HAL\_I2C\_MasterRxCpltCallback(I2C\_HandleTypeDef \*hi2c) {

}

/\* USER CODE END 4 \*/

/\*\*

\* @brief Period elapsed callback in non blocking mode

\* @note This function is called when TIM1 interrupt took place, inside

\* HAL\_TIM\_IRQHandler(). It makes a direct call to HAL\_IncTick() to increment

\* a global variable "uwTick" used as application time base.

\* @param htim : TIM handle

\* @retval None

\*/

void HAL\_TIM\_PeriodElapsedCallback(TIM\_HandleTypeDef \*htim)

{

/\* USER CODE BEGIN Callback 0 \*/

/\* USER CODE END Callback 0 \*/

if (htim->Instance == TIM1) {

HAL\_IncTick();

}

/\* USER CODE BEGIN Callback 1 \*/

/\* USER CODE END Callback 1 \*/

}

/\*\*

\* @brief This function is executed in case of error occurrence.

\* @retval None

\*/

void Error\_Handler(void)

{

/\* USER CODE BEGIN Error\_Handler\_Debug \*/

/\* User can add his own implementation to report the HAL error return state \*/

/\* USER CODE END Error\_Handler\_Debug \*/

}

#ifdef USE\_FULL\_ASSERT

/\*\*

\* @brief Reports the name of the source file and the source line number

\* where the assert\_param error has occurred.

\* @param file: pointer to the source file name

\* @param line: assert\_param error line source number

\* @retval None

\*/

void assert\_failed(char \*file, uint32\_t line)

{

/\* USER CODE BEGIN 6 \*/

/\* User can add his own implementation to report the file name and line number,

tex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) \*/

/\* USER CODE END 6 \*/

}

#endif /\* USE\_FULL\_ASSERT \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* (C) COPYRIGHT STMicroelectronics \*\*\*\*\*END OF FILE\*\*\*\*/