|  |
| --- |
| ICBRAMR  **School of Electronics Engineering (SENSE)**  **B. Tech – Electronics & Computer Engineering**  **BECE403E – EMBEDDED SYSTEM DESIGN**  **LAB RECORD**  **(L15+L16)**  **Submitted By**  **21BLC1228 – Mayukh Ray**    **Submitted To**  **Dr. Manoj Kumar Rajagopal**  **DATE: 14/04/2024**  **Slot:** L15+L16  **Date: 14/4/2024**  **LAB – 13: Working with Mbed RTOS**  **AIM:**  Implement and verify the logic on the STM32 Nucleo-F401RE board using Keil Studio Cloud IDE.  **Software Required:** ARM Keil Studio (Mbed Online Compiler)  **Hardware Required:** Micro USB cable, NUCLEO-F401RE Board, LEDs, Jumper Wires (M-F and M-M), Breadboard  **Procedure:**   1. Go to ARM Keil Studio (<https://studio.keil.arm.com>) and log in 2. Select File → New → Mbed Project 3. Click the Example project drop-down list and select “mbed2-example-blinky” 4. In Project name field, provide the name of the new project and click Add project 5. Double click on the “main.cpp” file from the newly created project folder 6. Modify the code in the editor window as per the logic of your application 7. Check for any errors in the program under the “Problems” tab of the panels window 8. If no errors, connect the Nucleo Board to the computer using Micro USB Cable 9. Click Play icon (Run project) to upload and start the code execution on the board.   **PROGRAM:**  **Lab Task 1:**  **Multitasking application using LEDs**  Write a mbed RTOS program to demonstrate a simple multitasking application. Create a two threads named Thread1 and Thread2. Perform blinking of LED1 connected to PC\_10 for every one second in Thread1. Terminate the Thread1 operation after 10 seconds. Perform blinking of LED2 connected to PC\_11 for every 0.5 second in Thread2. Design and implement this logic on the STM 32 Nucleo L152RE board using Keil studio could platform.  **Code:**  #include “mbed.h”  #include “rtos.h”  DigitalOut led1(PC\_10);  DigitalOut led2(PC\_11);  Thread Thread1, Thread2;  void led1\_thread()  {  while(1)  {  led1=!led1;  Thread::wait(1000);  }  }  void led2\_Thread()  {  while(1)  {  led2=!led2;  Thread::wait(500);  }  }  int main()  {  Thread1.start(led1\_thread);  Thread2.start(led2\_thread);  Thread::wait(10000);  Thread1.terminate();  while(1);  }  **Output:**    **Output Verification:**    **Lab Task 2:**  **Interprocess communication (synchronization) - Mutex and Semaphore**  Write a mbed RTOS program to demonstrate the use of Mutex and Semaphore functionality by creating two threads named Thread1 and Thread2 respectively. Create a common function named count that print the value of the variable “i” on serial monitor with 1 second delay apart. Assign initial value of variable of “i” as 10. Assume Thread1 perform increment by 1 operation on “i” and pass the incremented value to “count” function. Thread2 performs decrement by 1 on “i” and pass the decremented value to “count” function. Use Mutex/Semaphore to protect the common function “count” for Thread1 and Thread2. Design and implement this logic on the STM 32 Nucleo L152RE board using Keil studio could platform.  **Code:**  #include “mbed.h”  #include “rtos.h”  Serial PC(USBTX,USBRX);  Thread Thread1, Thread2;  Mutex mutex1;  int i=0;  void count(int i)  {  mutex1.lock();  PC.printf(“Count value before delay: %d\n\r”, i);  Thread::wait(1000);  PC.printf(“Count value after delay:%d\n\r”,i);  mutex1.unlock();  }  void increment ()  {  i=i+1;  count(i);  }  void decrement()  {  i=i-1;  count(i);  }  int main()  {  Thread1.start(increment);  Thread2.start(decrement);  Thread1.join();  Thread2.join();  }  **Output:**    **Output Verification:**    **Lab Task 3:**  **Interprocess communication (synchronization) – Signal(Event flag)**  Write a mbed RTOS program to demonstrate the use of Signal (event flag) by using LED (PC\_11) and button (PC\_10). Create a thread called Flash which waits for event flag 1 to be set by calling function signal\_wait(0x1). Event flag 1 is set when the button is pressed by calling function signal\_set(0x1). After this point thread active and flashes the LED every second. Design and implement this logic on the STM 32 Nucleo L152RE board using Keil studio could platform.  **Code:**  #include “mbed.h”  #include “rtos.h”  DigitalOut LED(PC\_11);  DigitalOut SW(PC\_10);  Thread Thread1;  void Flash()  {  Thread::signal\_wait(0x1);  while(1)  {  LED=!LED;  Thread::wait(1000);  }  }  int main()  {  Thread1.start(Flash);  while(SW==0);  Thread1.signal\_set(0x1);  while(1);  }  **Output:**      **Output Verification:**    **INFERENCE:**  **Lab-1:**  **Multitasking with mbed RTOS:** The program demonstrates the use of mbed Real-Time Operating System (RTOS) to implement multitasking on the STM32 Nucleo L152RE board. By creating multiple threads, the program enables concurrent execution of different tasks, allowing for efficient utilization of the microcontroller's resources.  **Thread Creation and Management:** Two threads, named Thread1 and Thread2, are created using the Thread class provided by mbed RTOS. Each thread is associated with a specific function (Thread1\_func and Thread2\_func) which defines its behavior.  **LED Blinking:** The program controls the blinking of two LEDs, LED1 connected to pin PC\_10 and LED2 connected to pin PC\_11. Each LED is toggled at a specific interval determined by the respective threads (Thread1 and Thread2). LED1 blinks every 1 second for a duration of 10 seconds, while LED2 blinks every 0.5 seconds continuously.  **Thread Synchronization:** Although not explicitly demonstrated in this example, mbed RTOS provides mechanisms for thread synchronization and communication, such as mutexes, semaphores, and message queues, which can be utilized for coordination between threads if needed in more complex applications.  **Main Function:** The main function serves as the entry point of the program. It starts both threads (Thread1 and Thread2) and then waits for Thread1 to finish execution using the Thread1.join() method. This ensures that the program does not exit until Thread1 has completed its task, providing a synchronized termination.  **Lab-2:**  **Multithreading with Mutex and Semaphore:** This program demonstrates the concurrent execution of two threads, Thread1 and Thread2, utilizing Mutex and Semaphore for synchronization. Mutex count\_mutex is used to protect the critical section of the count function, ensuring that only one thread can access it at a time. Semaphore count\_semaphore is initialized with a count of 1, allowing only one thread to acquire it at a time, providing mutual exclusion.  **Common Function:** The count function prints the current value of the variable i on the serial monitor. It is called by both Thread1 and Thread2 after modifying the value of i.  **Thread1 and Thread2 Operations:** Thread1 performs an increment operation on the variable i, while Thread2 performs a decrement operation. Both threads acquire the count\_semaphore before accessing the count function to ensure mutual exclusion.  **Thread Execution:** Both Thread1 and Thread2 execute their respective functions indefinitely in a loop. They acquire the semaphore, perform the operation on i, call the count function, and then release the semaphore before sleeping for 1 second.  **Main Function:** The main function serves as the entry point of the program. It starts both threads (Thread1 and Thread2) and then waits for them to finish execution using the join() method, ensuring that the program does not exit prematurely.  **Lab-3:**  **Event Flags:** This program demonstrates the use of event flags (Signal) to synchronize the operation of threads. Thread Flash waits for event flag 0x1 to be set, indicating that the button has been pressed.  **Thread Creation:** The Flash thread is created, which waits for the event flag to be set and then toggles the LED every second.  **Button and LED:** The program uses a button connected to pin PC\_10 and a LED connected to pin PC\_11. When the button is pressed, event flag 0x1 is set, signaling the Flash thread to start flashing the LED.  **Main Function:** In the main function, the program continuously checks the state of the button. If the button is pressed (button is true), it sets event flag 0x1 for the Flash thread using Thread::signal\_set.  **Thread Synchronization:** The Flash thread waits for the event flag to be set by calling ThisThread::flags\_wait\_any(0x1). Once the flag is set, it enters the flashing loop, toggling the LED every second.  **RESULT:**  **Lab-1:**  Hence, we were able to demonstrate a simple multitasking application.  **Lab-2:**  Hence, we were able to demonstrate the use of Mutex and Semaphore functionality by creating two threads named Thread1 and Thread2 respectively.  **Lab-3:**  Hence, we were able to demonstrate the use of Signal (event flag) by using LED (PC\_11) and button (PC\_10). |