INTERNET OF THINGS
DEPARTMENT OF COMPUTER ENGINEERING
SANTA CLARA UNIVERSITY, CALIFORNIA
BEHNAM DEZFOULI, PHD

Santa Clara University
Internet of Things
RESEARCH LAB
SIOTLAB

Internet of Things

Lab 7
RTOS OS Features

Threads

Threads

- By using an RTOS you can separate the system functions into separate tasks (called **threads**) and develop them in a somewhat independent fashion

- In this section we focus on RTOS facilities such as **threads**, **semaphores**, **mutex**, and **queues**

Threads

- Threads are at the heart of an RTOS
- We can **create a new thread** by calling **`wiced_rtos_create_thread()`**
- Arguments are as follows:
  - **`wiced_thread_t* thread`**: A pointer to a **thread handle data structure**
    - This handle is used to identify the thread for other thread functions
    - You must first create the handle data structure before providing the pointer to the create thread function
  - **`uint8_t priority`**: This is the priority of the thread
    - **Priorities can be from 0 to 31 where 0 is the highest priority**
    - If the scheduler knows that two threads are eligible to run, it will run the thread with the higher priority
    - **The WICED Wi-Fi Driver (WWD) runs at priority 3**
  - **`char *name`**: A name for the thread
    - This name is only used by the debugger
    - You can give it any name or just use NULL if you don't want a specific name

Threads

- **`wiced_thread_function_t *thread`**: A function pointer to the function that is the thread
- **`uint32_t stack size`**: How many bytes should be in the thread's stack (you should be careful here as running out of stack can cause erratic, difficult to debug behavior)
  - Using 10000 is overkill but will work for any of the exercises we do in this class
- **`void *arg`**: A generic argument which will be passed to the thread
  - If you don't need to pass an argument to the thread, just use NULL

Threads

- Assume you want to create a thread that runs the function `mySpecialThread()`

```
#define THREAD_PRIORITY      (10)
#define THREAD_STACK_SIZE    (10000)
.
.
wiced_thread_t mySpecialThreadHandle;
.
.
wiced_rtos_create_thread(&mySpecialThreadHandle, THREAD_PRIORITY,
"mySpecialThreadName", mySpecialThread, THREAD_STACK_SIZE, NULL);
```

Pointer to a thread handle data structure

A function pointer to the function that is the thread

Threads

- **The thread function must take a single argument of type `wiced_thread_arg_t` and must have a `void` return**
- The body of a thread looks just like the “main” function of your application
- In fact, the main function (`application_start()`) is really just a thread that is initialized automatically!
- Typically a thread will run forever (just like ‘main”) so it will have an initialization section and a `while(1)` loop that repeats forever

```
void mySpecialThread(wiced_thread_arg_t arg)
{
    const int delay=100;
    while(1)
    {
        processData();
        wiced_rtos_delay_milliseconds(delay);
    }
}
```

Threads

- You should (almost) always put a `wiced_rtos_delay_milliseconds()` or `wiced_rtos_delay_microseconds()` of some amount in every thread **so that other threads get a chance to run**
- This applies to the main application `while(1)` loop as well since **the main application is just another thread**
- The **exception** is if you have some other thread control function such as a **semaphore** or **queue** which will cause the thread to periodically pause

- Note that if the main application thread (`application_start`) only does initialization and starts other threads, then you can eliminate the `while(1)` loop completely from that function
- In this case, after the other threads have started, the `application_start` function will just exist and will not take up any more CPU cycles

❑ **Example:** We are interested in implementing the following application:

- Use a thread to blink an LED
- The blinking interval is received from the user as a 3 digit number

Threads

Example: Thread-LED-UART (`thread_led_uart`)
Part: 1/3

```
#include "wiced.h"
#define RX_BUFFER_SIZE (5)
/* Thread parameters */
#define THREAD_PRIORITY   (10)
#define THREAD_STACK_SIZE (1024)

volatile uint16_t interval; //LED blinking interval

// Define the thread function that will blink the LED on/off every “interval” ms
void ledThread(wiced_thread_arg_t arg) {
        wiced_bool_t led1 = WICED_FALSE;

        while(1) {
                /* Toggle LED1 */
                if ( led1 == WICED_TRUE ) {
                        wiced_gpio_output_low( WICED_SH_LED1 );
                        led1 = WICED_FALSE;
                }
                else {
                        wiced_gpio_output_high( WICED_SH_LED1 );
                        led1 = WICED_TRUE;
                }
                wiced_rtos_delay_milliseconds( interval );
        }
}
```

The thread wakes up every “interval” ms and runs the loop

Example: Thread-LED-UART (`thread_led_uart`)
Part: 2/3

```
void application_start( )
{
    uint32_t expected_data_size = 3; //Receive 3 digits from user
    char     receiveString[expected_data_size];
    interval = 250;

    wiced_thread_t  ledThreadHandle;

    wiced_init();/* Initialize the WICED device */

    /* Configure and start the UART. */
    wiced_ring_buffer_t   rx_buffer;
    uint8_t               rx_data[RX_BUFFER_SIZE];
    ring_buffer_init(&rx_buffer, rx_data, RX_BUFFER_SIZE );

    wiced_uart_config_t uart_config =
    {
        .baud_rate    = 115200,
        .data_width   = DATA_WIDTH_8BIT,
        .parity       = NO_PARITY,
        .stop_bits    = STOP_BITS_1,
        .flow_control = FLOW_CONTROL_DISABLED,
    };

    wiced_uart_init( STDIO_UART, &uart_config, &rx_buffer); /* Setup UART */
```

For the configuration to work, we need to use `GLOBAL_DEFINES := WICED_DISABLE_STDIO` in the **make file**

Threads

Example: Thread-LED-UART (`thread_led_uart`)
Part: 3/3

```
    /* Initialize and start a new thread */
    wiced_rtos_create_thread(&ledThreadHandle, THREAD_PRIORITY,
                             "ledThread", ledThread, THREAD_STACK_SIZE, NULL);

     wiced_uart_transmit_bytes(WICED_UART_1,
                  "Enter a value between 000 and 999:\n", 35);

     while ( 1 ) {
         if ( wiced_uart_receive_bytes( STDIO_UART, &receiveString,
                   &expected_data_size, WICED_NEVER_TIMEOUT )
                                                    == WICED_SUCCESS )
         {
             interval = atoi(receiveString);      //ASCI to integer conversion

             wiced_uart_transmit_bytes(WICED_UART_1,
                                  "\nNew Value entered: " , 21);

             wiced_uart_transmit_bytes(WICED_UART_1, &receiveString , 3);
             wiced_uart_transmit_bytes(WICED_UART_1, "\n" , 1);
         }
     }
}
```

Semaphore

Semaphore

- A **semaphore** is **a signaling mechanism between threads**
- **You can use a semaphore to signal between threads that something is ready**

- In the WICED SDK, **semaphores are implemented as a simple unsigned integer**
- When you **“set” a semaphore it increments** the value of the semaphore
- When you **“get” a semaphore it decrements** the value, but if the value is 0, the thread will **suspend** itself until the semaphore is set

- **The get function requires a timeout parameter**
  - This sets **the time in milliseconds that the function waits before returning**
  - If you want the thread to wait indefinitely for the semaphore to be set, rather than continuing execution after a specific delay, then use `WICED_WAIT_FOREVER`

Semaphore

Initializes a semaphore

```
 * @param[in] semaphore : A pointer to the semaphore handle to be initialized
 * @return    WICED_SUCCESS : on success.
 * @return    WICED_ERROR   : if an error occurred

wiced_result_t wiced_rtos_init_semaphore( wiced_semaphore_t* semaphore );
```

Get (wait/decrement) a semaphore

```
 * Attempts to get (wait/decrement) a semaphore. If semaphore is at zero
 * already, then the calling thread will be suspended until another thread sets
 * the semaphore with @ref wiced_rtos_set_semaphore
 *
 * @param[in] semaphore : A pointer to the semaphore handle
 * @param[in] timeout_ms: The number of milliseconds to wait before returning
 *
 * @return    WICED_SUCCESS : on success.
 * @return    WICED_ERROR   : if an error occurred

wiced_result_t wiced_rtos_get_semaphore( wiced_semaphore_t* semaphore,
                                         uint32_t timeout_ms );
```

Semaphore

❑ **Example:** We are interested in implementing the following application:

- **The main thread looks for a button press, then uses a semaphore to communicate to the toggle LED thread**

- You can use a pin interrupt to detect the button press and set the semaphore
- Use `wiced_rtos_get_semaphore()` inside the LED thread so that it waits for the semaphore forever and then toggles the LED rather than blinking constantly

Semaphore

Example: Semaphore-LED (`semaphore_led`)
Part: 1/2

```
#include "wiced.h"
#define THREAD_PRIORITY   (10)
#define THREAD_STACK_SIZE (1024)
static wiced_thread_t ledThreadHandle;
static wiced_semaphore_t semaphoreHandle;

/* Interrupt service routine for the button */
void button_isr(void* arg) {
        wiced_rtos_set_semaphore(&semaphoreHandle); /* Set the semaphore */
}

void ledThread(wiced_thread_arg_t arg) // The thread function for LED toggle
{
    static wiced_bool_t led1 = WICED_FALSE;
    while(1)      {
     wiced_rtos_get_semaphore(&semaphoreHandle, WICED_WAIT_FOREVER);

        if ( led1 == WICED_TRUE ) {
                wiced_gpio_output_low( WICED_SH_LED1 );
                led1 = WICED_FALSE;
        }
        else {
                wiced_gpio_output_high( WICED_SH_LED1 );
                led1 = WICED_TRUE;
        }
        }
}
```

Check for the semaphore here
If it is not set, then this thread will suspend until the semaphore is set by the button thread

Semaphore

Example: Semaphore-LED (`semaphore_led`)
Part: 2/2

```
void application_start( )
{

    wiced_init();/* Initialize the WICED device */

    /* Setup the semaphore which will be set by the button interrupt */
    wiced_rtos_init_semaphore(&semaphoreHandle);

    /* Initialize and start LED thread */
    wiced_rtos_create_thread(&ledThreadHandle, THREAD_PRIORITY,
                             "ledThread", ledThread, THREAD_STACK_SIZE, NULL);

    /* Setup button interrupt */
    wiced_gpio_input_irq_enable(WICED_SH_MB1, IRQ_TRIGGER_FALLING_EDGE,
                                                    button_isr, NULL);

    /* No while(1) here since everything is done by the new thread. */

}
```

Mutex

Mutex

- **Mutex** is an abbreviation for "**Mutual Exclusion**"
- A mutex is **a lock on a specific resource**
- If you request a mutex on a resource that is already locked by another thread, **then your thread will go to sleep until the lock is released**
- In the exercises for this chapter you will create a mutex for the `WPRINT_APP_INFO` function

Mutex

❑ **Example:** We are interested in implementing the following application:

- An LED may behave strangely if two threads try to blink it at the same time
- Create two threads that do the following:
  - Thread 1 will blink LED0 with an ON and OFF delay of 150ms while Button 1 is being pressed
  - Thread 2 will blink the same LED (LED0) with an ON and OFF delay of 100ms while Button 2 is being pressed
  - Make sure you yield control in the thread when the button is not being pressed

- **Press button 1 and button 2 separately to observe the blink rates; then press both buttons simultaneously; Do you see issues with the blinking?**
- **Add a mutex to the project so that when you press button 1 it will ignore button 2 and vice versa**
- That is, the LED blink rate will follow the first button that was pressed

Mutex

Example: MUTEX-LED (`mutex_led`)
Part: 1/4

```
#include "wiced.h"

// Comment out the following line to see what happens
// without the mutex
#define USE_MUTEX

/* Thread parameters */
#define THREAD_PRIORITY      (10)
#define THREAD_STACK_SIZE    (1024)

static wiced_thread_t ledThread1Handle;
static wiced_thread_t ledThread2Handle;


#ifdef USE_MUTEX
static wiced_mutex_t MutexHandle;
#endif
```

Comment out this line to see what happens without mutex

Mutex

Example: MUTEX-LED (`mutex_led`)
Part: 2/4

```
/* Define the thread function that will blink LED1 if B1 is pressed */
void ledThread1(wiced_thread_arg_t arg)
{
    while(1)
    {
        #ifdef USE_MUTEX
        wiced_rtos_lock_mutex(&MutexHandle);
        #endif

        // Loop while button is pressed
        while(!wiced_gpio_input_get( WICED_SH_MB0 )) {
            wiced_gpio_output_high( WICED_SH_LED0 );
            wiced_rtos_delay_milliseconds(100);
            wiced_gpio_output_low( WICED_SH_LED0 );
            wiced_rtos_delay_milliseconds(100);
        }

        #ifdef USE_MUTEX
        wiced_rtos_unlock_mutex(&MutexHandle);
        #endif

        wiced_rtos_delay_milliseconds(1);
        // Yield control when button is not pressed
    }
}
```

Mutex

Example: MUTEX-LED (`mutex_led`)
Part: 3/4

```
/* Define the thread function that will blink LED1 if B2 is pressed */
void ledThread2(wiced_thread_arg_t arg)
{
    while(1)
    {
        #ifdef USE_MUTEX
        wiced_rtos_lock_mutex(&MutexHandle);
        #endif

        // Loop while button is pressed
        while(!wiced_gpio_input_get( WICED_SH_MB1 )) {
            wiced_gpio_output_low( WICED_SH_LED0 );
            wiced_rtos_delay_milliseconds(150);
            wiced_gpio_output_high( WICED_SH_LED0 );
            wiced_rtos_delay_milliseconds(150);
        }

        #ifdef USE_MUTEX
        wiced_rtos_unlock_mutex(&MutexHandle);
        #endif

        // Yield control when button is not pressed
        wiced_rtos_delay_milliseconds(1);
    }
}
```

Mutex

Example: MUTEX-LED (`mutex_led`)
Part: 4/4

```
void application_start( )
{
    wiced_init();   /* Initialize the WICED device */

    /* Initialize the Mutex */
    #ifdef USE_MUTEX
    wiced_rtos_init_mutex(&MutexHandle);
    #endif

    WPRINT_APP_INFO(("Threads created! \n"));


    /* Initialize and start threads */
    wiced_rtos_create_thread(&ledThread1Handle, THREAD_PRIORITY, "ledThread1",
                ledThread1, THREAD_STACK_SIZE, NULL);

    wiced_rtos_create_thread(&ledThread2Handle, THREAD_PRIORITY, "ledThread2",
                ledThread2, THREAD_STACK_SIZE, NULL);


    /* No while(1) here since everything is done by the new threads. */
}
```

Deliverable

Student Work

- Task 7-1.
  - Demo the code for the TA

- Task 7-2.
  - What happens if you forget to unlock the mutex in one of the threads? Why?

Queue

Queue

- A **queue** is **a thread-safe mechanism** to send data to another thread
- The queue is a **FIFO** - you read from the front and you write to the back
- **If you try to read a queue that is empty, your thread will suspend until something is written into it**
- The payload in a queue (size of each entry) and the size of the queue (number of entries) is user configurable at queue creation time

Queue

- The `wiced_rtos_push_to_queue()` requires a **timeout parameter**
  - **This sets the time in milliseconds that the function waits before returning if the queue is full**
  - If you want the thread to wait indefinitely for space in the queue rather than continuing execution after a specific delay then use `WICED_WAIT_FOREVER`
  - If you want the project to continue on immediately if there isn't room in the queue, then use `WICED_NO_WAIT`
- Likewise, the `wiced_rtos_pop_from_queue()` function requires a timeout parameter to specify how long the thread should wait if the queue is empty

Pushes an object onto a queue

```
 * @param[in] queue   : A pointer to the queue handle
 * @param[in] message: The object to be added to the queue
 * @param[in] timeout_ms : The number of milliseconds to wait before returning
 *
 * @return    WICED_SUCCESS : on success.
 * @return    WICED_ERROR   : if an error or timeout occurred
 */
wiced_result_t wiced_rtos_push_to_queue( wiced_queue_t* queue,
                                         void* message, uint32_t timeout_ms );
```

Queue

- You should **always initialize a queue before starting any threads that use it**
- The message size in a queue must be a multiple of 4 bytes
  - Specifying a message size that is not a multiple of 4 bytes will result in an unpredictable behavior
  - It is good practice to use uint32_t as the minimum size variable (this is true for all variables since the ARM processor is 32-bits)
- If you are using a queue push or pop function **inside of an ISR or a timer function**, **you MUST use `WICED_NO_WAIT` as the timeout; using a non-zero timeout is not supported in those cases**

Initializes a FIFO queue

```
 * @param[in] queue  : A pointer to the queue handle to be initialized
 * @param[in] name   : A text string name for the queue (NULL is allowed)
 * @param[in] message_size : Size in bytes of objects held in the queue
 * @param[in] number_of_messages : max number of objects in the queue
 * @return    WICED_SUCCESS : on success.
 * @return    WICED_ERROR   : if an error occurred

wiced_result_t wiced_rtos_init_queue( wiced_queue_t* queue, const char* name,
                                      uint32_t message_size, uint32_t number_of_messages );
```

❑ **Example:** We are interested in implementing the following application:

- **Use a queue to send a message to indicate the number of times to blink an LED**

- Add a static variable to the ISR that increments each time the button is pressed
- Push the value onto the queue to give the LED thread access to it
- Remember to use `WICED_NO_WAIT` for the timeout parameter in the ISR; Otherwise the push function will not work
- In the LED thread, pop the value from the queue to determine how many times to blink the LED

Queue

Example: Queue-LED (`queue_led`)
Part: 1/3

```
#include "wiced.h"

/* Thread parameters */
#define THREAD_PRIORITY   (10)
#define THREAD_STACK_SIZE (1024)
/* The queue messages will be 4 bytes each (a 32 bit integer) */
#define MESSAGE_SIZE     (4)
#define QUEUE_SIZE      (10)

static wiced_queue_t  queueHandle;
static wiced_thread_t ledThreadHandle;

wiced_bool_t buttonFlag = WICED_FALSE;

/* Interrupt service routine for the button */
void button_isr(void* arg)
{
        static uint32_t blinks = 0;

        blinks++;


        wiced_rtos_push_to_queue(&queueHandle, &blinks, WICED_NO_WAIT);
}
```

- **We will use `WICED_NO_WAIT` here so that the ISR won't lock execution if the queue is full**
- Note: `WICED_WAIT_FOREVER` is not allowed inside an ISR

Example: Queue-LED (`queue_led`) **Part:** 2/3

```
/* Define the thread function that will toggle the LED */
void ledThread(wiced_thread_arg_t arg)
{
    static uint32_t message;

    while(1)
    {
        uint32_t i; /* Loop Counter */


        wiced_rtos_pop_from_queue(&queueHandle, &message, WICED_WAIT_FOREVER);

        /* Blink LED1 the specified number of times */
        for(i=0; i < message; i++)
        {
                wiced_gpio_output_high( WICED_SH_LED1 );
                wiced_rtos_delay_milliseconds(150);
                wiced_gpio_output_low( WICED_SH_LED1 );
                wiced_rtos_delay_milliseconds(150);
        }

        // Wait 1 second between sets of blinks
        wiced_rtos_delay_milliseconds(1000);
    }
}
```

- If not empty, pull the value off the queue
- If empty, this will suspend the thread until a value is available

Example: Queue-LED (`queue_led`) **Part:** 3/3

```
void application_start( )
{

    wiced_init();/* Initialize the WICED device */


    /* Setup button interrupt */
    wiced_gpio_input_irq_enable(WICED_SH_MB1, IRQ_TRIGGER_FALLING_EDGE,
                                              button_isr, NULL);



    /* Initialize the queue */
    wiced_rtos_init_queue(&queueHandle, "blinkQueue", MESSAGE_SIZE, QUEUE_SIZE);



    /* Initialize and start LED thread */
    wiced_rtos_create_thread(&ledThreadHandle, THREAD_PRIORITY, "ledThread",
                                          ledThread, THREAD_STACK_SIZE, NULL);

    /* No while(1) here since everything is done by the new thread. */
}
```

The ISR pushes into the queue

ledThread pops from the queue

Timer

Timer

- An RTOS **timer** allows you **to schedule a function to run at a specified interval**

❑ Example: To send your data to the cloud every 10 seconds

- When you setup the timer you specify the function you want to run and how often you want it run
- The function that the timer calls takes a single argument of `void* arg`
  - If the function doesn't require any arguments you can specify NULL in the timer initialization function
- Note that there is a single execution of the function every time the timer expires rather than a continually executing thread so **the function should typically not have a while(1) loop** – it should just run and exit each time the timer calls it
- **The timer is a function, not a thread**
  - Therefore, make sure you don't exit the main application thread if your project has no other active threads

Timer

Initializes a RTOS timer

```
 * Timer does not start running until @ref wiced_rtos_start_timer is called
 * @param[in] timer    : A pointer to the timer handle to be initialized
 * @param[in] time_ms  : Timer period in milliseconds
 * @param[in] function : The callback handler function that is called each
 *                       time the timer expires
 * @param[in] arg      : An argument that will be passed to the callback
 *                       function
 *
 * @return    WICED_SUCCESS : on success.
 * @return    WICED_ERROR   : if an error occurred

wiced_result_t wiced_rtos_init_timer( wiced_timer_t* timer,
                 uint32_t time_ms, timer_handler_t function, void* arg );
```

Starts a RTOS timer

```
 *
 * @param[in] timer    : A pointer to the timer handle to start
 *
 * @return    WICED_SUCCESS : on success.
 * @return    WICED_ERROR   : if an error occurred
 */
wiced_result_t wiced_rtos_start_timer( wiced_timer_t* timer );
```

Timer

❑ **Example:** We are interested in implementing the following application:

- Make an LED blink using a timer

- The variable to remember the state of the LED must be static since the function will exit each time it completes rather than running infinitely like the thread
- Set up an RTOS timer that will call the LED function every 250ms

Timer

Example: Timer-LED (`timer_led`)
Part: 1/2

```
#include "wiced.h"

#define  TIMER_TIME  (250)

/* Define the  function that will blink the LED on/off */
void ledBlink(void* arg)
{
        static wiced_bool_t led1 = WICED_FALSE;

        /* Toggle LED1 */
        if ( led1 == WICED_TRUE )
        {
                wiced_gpio_output_low( WICED_SH_LED1 );
                led1 = WICED_FALSE;
        }
        else
        {
                wiced_gpio_output_high( WICED_SH_LED1 );
                led1 = WICED_TRUE;
        }
}
```

Timer

Example: Timer-LED (`timer_led`)
Part: 2/2

```
void application_start( )
{
    wiced_timer_t timerHandle;

    wiced_init();      /* Initialize the WICED device */

    /* Initialize and start a timer */
    wiced_rtos_init_timer(&timerHandle, TIMER_TIME, ledBlink, NULL);
    wiced_rtos_start_timer(&timerHandle);

    while ( 1 )
    {
        /* Nothing needed here since we only have one thread. */
    }
}
```

Deliverables

Student Work

- Task 7-3. Please develop the following code:
  - The application uses a timer and two threads
  - When button MB1 on the shield is pressed, a 2 second timer is started
  - Pressing the button also unblocks two threads (using two semaphores):
    - The first thread blinks shield's LED 1 as long as the timer has not fired (i.e., for 2 seconds)
    - The second thread samples temperature and sends it over UART as long as the timer has not fired (i.e., for 2 seconds)
  - While the two threads are active, pressing MB1 should not extend their 2 second activity duration