

Team 6 - QuickTech

Name:

1. Iman Aidi Elham
2. Lai Chee Yee
3. Tie Sing Hao
4. Amin Haikal

Problem Solving 4: RTOS

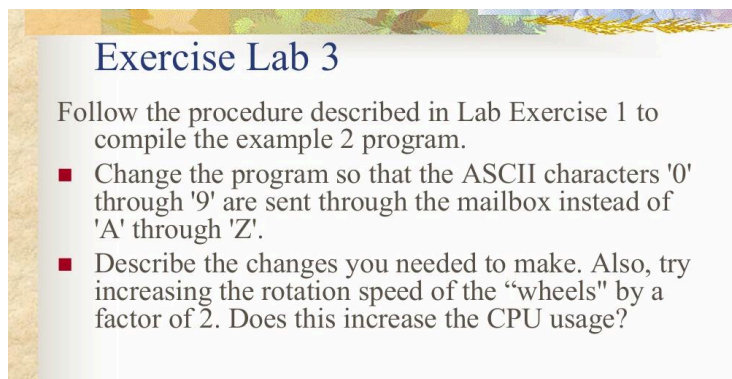
Instruction 1

Refer to the lecture material on RTOS and the lab material on uC/OS-II RTOS (or any latest version). Based on Lab Material uC/OS-II RTOS with 3 questions (Lab1 – Lab3), below are the assigned task to each group. Each group needs to write a C concurrent programming solution using the uC/OS-II RTOS.

Lab 1: AttoCode, Devin

Lab 2: RedBull Tech, TechTitans

Lab 3: Realme, QuickTech, Nas Co.



Exercise Lab 3

Follow the procedure described in Lab Exercise 1 to compile the example 2 program.

- Change the program so that the ASCII characters '0' through '9' are sent through the mailbox instead of 'A' through 'Z'.
- Describe the changes you needed to make. Also, try increasing the rotation speed of the “wheels” by a factor of 2. Does this increase the CPU usage?

Instruction 2

Select another RTOS, which you can compare to the uC/OS-II RTOS (or any latest version) solution in Instruction 1. Compare your selected RTOS features with uC/OS-II RTOS, and based on the features' comparison, discuss the differences between your concurrent solution in Instruction 1 and the selected RTOS possible solution.

Instruction 3

Using FreeRTOS as a reference point, imagine you're developing a real-time system for a traffic light controller. This system needs to manage the sequence and timing of traffic signals (red, yellow, green) for an intersection.

- a. Identify two core functionalities of FreeRTOS that would be essential for implementing the traffic light controller. Explain how these features would be used in this scenario.

- b. Research and explore an additional advanced feature of FreeRTOS that might be useful for this application (e.g., Semaphores, Mutexes). Briefly explain the chosen feature and its potential benefit in the traffic light controller system.

Answers:

Instruction 1

Part 1: Modify the Program to Send ASCII Characters '0' through '9' Through the Mailbox

To modify the example program so that ASCII characters '0' through '9' are sent through the mailbox instead of 'A' through 'Z', we need to change the initialization of txmsg in Task 4 and adjust the condition that resets txmsg.

Original Task 4 Code:

```
void Task4 (void *data) {
    txmsg = 'A';
    for (;;) {
        OSMboxPost(TxMbox, (void *)&txmsg);
        OSMboxPend(AckMbox, 0, &err);
        txmsg++;
        if (txmsg == 'Z') {
            txmsg = 'A';
        }
    }
}
```

Modified Task 4 Code:

```
void Task4 (void *data) {
    txmsg = '0'; // Start with ASCII '0'
    for (;;) {
        OSMboxPost(TxMbox, (void *)&txmsg);
        OSMboxPend(AckMbox, 0, &err);
        txmsg++;
        if (txmsg > '9') { // Reset after '9'
            txmsg = '0';
        }
    }
}
```

This change ensures that the characters '0' through '9' are cycled through and sent via the mailbox.

Part 2: Increase the Rotation Speed of the "Wheels" by a Factor of 2

To increase the rotation speed of the wheels displayed by Task 2 and Task 3, we need to reduce the delay times in those tasks. The original delay for each step of the rotation is 10 ticks for Task 2 and 20 ticks for Task 3. Halving these values will double the rotation speed.

Original Task 2 Code:

```
void Task2 (void *data) {
    for (;;) {
        PC_Dispatch(70, 15, '|', DISP_FGND_YELLOW + DISP_BGND_BLUE);
        OSTimeDly(10);
        PC_Dispatch(70, 15, '/', DISP_FGND_YELLOW + DISP_BGND_BLUE);
        OSTimeDly(10);
        PC_Dispatch(70, 15, '-', DISP_FGND_YELLOW + DISP_BGND_BLUE);
        OSTimeDly(10);
        PC_Dispatch(70, 15, '\\', DISP_FGND_YELLOW + DISP_BGND_BLUE);
        OSTimeDly(10);
    }
}
```

Modified Task 2 Code:

```
void Task2 (void *data) {
    for (;;) {
        PC_Dispatch(70, 15, '|', DISP_FGND_YELLOW + DISP_BGND_BLUE);
        OSTimeDly(5); // Reduced delay
        PC_Dispatch(70, 15, '/', DISP_FGND_YELLOW + DISP_BGND_BLUE);
        OSTimeDly(5); // Reduced delay
        PC_Dispatch(70, 15, '-', DISP_FGND_YELLOW + DISP_BGND_BLUE);
        OSTimeDly(5); // Reduced delay
        PC_Dispatch(70, 15, '\\', DISP_FGND_YELLOW + DISP_BGND_BLUE);
        OSTimeDly(5); // Reduced delay
    }
}
```

Original Task 3 Code:

```
void Task3 (void *data) {
    for (i = 0; i < 499; i++) dummy[i] = '?';
    for (;;) {
        PC_Dispatch(70, 16, '|', DISP_FGND_YELLOW + DISP_BGND_BLUE);
        OSTimeDly(20);
        PC_Dispatch(70, 16, '\\', DISP_FGND_YELLOW + DISP_BGND_BLUE);
        OSTimeDly(20);
        PC_Dispatch(70, 16, '-', DISP_FGND_YELLOW + DISP_BGND_BLUE);
        OSTimeDly(20);
        PC_Dispatch(70, 16, '/', DISP_FGND_YELLOW + DISP_BGND_BLUE);
        OSTimeDly(20);
    }
}
```

Modified Task 3 Code:

```
void Task3 (void *data) {
    for (i = 0; i < 499; i++) dummy[i] = '?';
    for (;;) {
        PC_Dispatch(70, 16, '|', DISP_FGND_YELLOW + DISP_BGND_BLUE);
        OSTimeDly(10); // Reduced delay
        PC_Dispatch(70, 16, '\\', DISP_FGND_YELLOW + DISP_BGND_BLUE);
        OSTimeDly(10); // Reduced delay
        PC_Dispatch(70, 16, '-', DISP_FGND_YELLOW + DISP_BGND_BLUE);
        OSTimeDly(10); // Reduced delay
        PC_Dispatch(70, 16, '/', DISP_FGND_YELLOW + DISP_BGND_BLUE);
        OSTimeDly(10); // Reduced delay
    }
}
```

Observations:

1. Program Output

- The program now sends ASCII characters '0' through '9' instead of 'A' through 'Z'.
- The rotation speed of the wheels is doubled.

2. CPU Usage

- Increasing the rotation speed will likely increase CPU usage because the tasks will run more frequently with reduced delays.

3. Task Count and Performance

- Ensure that OS_MAX_TASKS and OS_LOWEST_PRIO in OS_CFG.H are appropriately configured to accommodate the number of tasks.
- Verify the stack sizes to ensure they are sufficient for the increased activity.

Instruction 2

Selected RTOS: FreeRTOS

Comparison of Features:

Feature	µC/OS-II	FreeRTOS
License	Proprietary (source code available with license)	MIT License (open source)
Task Management	Supports up to 64 tasks with unique priorities	Supports unlimited tasks (dependent on memory) with flexible priorities
Kernel Size	Minimal footprint (2-10 KB)	Small footprint (~6-12 KB)
API Complexity	Simple, consistent API	Rich API with advanced features
Scheduling	Preemptive, priority-based	Preemptive, priority-based with optional time slicing
Inter-task Communication	Semaphores, mailboxes, queues	Semaphores, queues, task notifications, event groups
Stack Management	Manual stack management	Automatic stack management with stack overflow checking
Portability	Highly portable	Highly portable, supports many architectures
Real-time Capabilities	Hard real-time capabilities	Soft and hard real-time capabilities

Discussion:

In Instruction 1, we used μ C/OS-II to implement a concurrent solution with multiple tasks displaying numbers on the screen. Each task had a unique priority, and we used semaphores to manage access to shared resources (random number generator). The solution demonstrated μ C/OS-II's capability to handle real-time tasks efficiently with a small memory footprint and a straightforward API.

FreeRTOS Possible Solution:

FreeRTOS offers a more flexible and feature-rich environment compared to μ C/OS-II. Here are some potential differences in the concurrent solution:

1. Task Creation

- **μ C/OS-II:** Uses **OSTaskCreate** to create tasks with a unique priority.
- **FreeRTOS:** Uses **xTaskCreate**, which allows tasks to have flexible priorities and can be adjusted dynamically. FreeRTOS also supports creating tasks with time-slicing, enabling fair CPU time distribution among tasks with the same priority.

2. Inter-task Communication

- **μ C/OS-II:** Utilizes semaphores and mailboxes for communication.
- **FreeRTOS:** Provides semaphores, queues, task notifications, and event groups, offering more advanced and varied methods for inter-task communication. For example, task notifications can be used for simple signaling, and queues can be used for passing data safely between tasks.

3. Stack Management

- **μ C/OS-II:** Requires manual stack management.
- **FreeRTOS:** Automatically manages stack allocation and provides stack overflow checking, which enhances reliability and makes the system easier to debug.

4. Advanced Features

- **μ C/OS-II:** Limited advanced features.
- **FreeRTOS:** Includes features like software timers, event groups, and task notifications which can simplify the implementation of complex real-time systems.

Instruction 3

Traffic Light Controller with FreeRTOS

Core Functionalities for Traffic Light Controller:

1. Task Scheduling and Management:

- **Feature:** Preemptive Scheduling with Priorities
- **Usage:** Create tasks for each traffic signal light (red, yellow, green) with appropriate priorities. For example, the task controlling the green light can have a higher priority than the task for the red light if we want the green light to have more responsiveness. Each task will handle the timing and state changes of the lights, ensuring smooth transitions.

2. Inter-task Communication:

- **Feature:** Queues
- **Usage:** Use queues to manage communication between tasks. For example, a queue can be used to signal when one light changes state, triggering the next light in sequence. This ensures coordinated timing across the traffic light system. Each task can send a message to the queue when it completes its timing, indicating it's time for the next light to take over.

Advanced Feature: Semaphores

Feature: Semaphores (Counting and Binary)

Potential Benefit:

- Semaphores can be used to manage shared resources and ensure mutual exclusion when tasks access shared hardware components like the LED lights controlling the traffic signals.
- **Usage Scenario:** A binary semaphore can be used to ensure that only one task is accessing the traffic light hardware at a time, preventing conflicts and ensuring safe updates to the light states. Counting semaphores can be used if there are multiple shared resources or multiple instances where tasks need to coordinate their activities.

Example Usage:

```
void vGreenLightTask(void *pvParameters) {
    for (;;) {
        // Wait for the semaphore to be available
        if (xSemaphoreTake(xSemaphore, portMAX_DELAY) == pdTRUE) {
            // Turn on the green light
            // Perform green light timing operations
            // ...
            // Release the semaphore
            xSemaphoreGive(xSemaphore);
        }
        // Delay for the green light duration
        vTaskDelay(pdMS_TO_TICKS(GREEN_LIGHT_DURATION_MS));
    }
}
```

In this example, the semaphore ensures that only one task can control the traffic light hardware at any given time, preventing race conditions and ensuring reliable operation.