

Semaphores

This article provides examples of various different Semaphores.

Binary Semaphore

A binary semaphore is a signal. In FreeRTOS, it is nothing but a queue of 1 item.

```
#include "semphr.h"
#include "queue.h"
// Queue of 1 item
static SemaphoreHandle_t binary_semaphore;
static void lcd_update_task(void *p) {
    while(1) {
        // CONSUME the signal - xQueueReceive()
        puts("lcd_update_task(): Attempting to take the semaphore");
        if (xSemaphoreTake(binary_semaphore, portMAX_DELAY)) {
            // go update the LCD
            printf("%3d: Okay, I will update the LCD\n", (unsigned)xTaskGetTickCount());
        }
    }
}
static void producer(void *p) {
    while (1) {
        if (!xSemaphoreGive(binary_semaphore)) { // xQueueReceive()
            puts("ERROR: Line: 33: Could not give the Binary Semaphore");
        }
        if (!xSemaphoreGive(binary_semaphore)) {
            puts("ERROR: Line 36: Could not give the Binary Semaphore");
        }
        vTaskDelay(1000);
    }
}
```

```

int main(void) {
    binary_semaphore = xSemaphoreCreateBinary(); // xQueueCreate(1, 0-byte-payload);
    xTaskCreate(lcd_update_task, "lcd",          1024 * 8, NULL, 2, NULL); // High priority
    xTaskCreate(producer, "producer", 1024 * 8, NULL, 1, NULL); // Low priority

    vTaskStartScheduler(); // this function doesn't return
    return 0;}

```

Counting Semaphore

```

#include "semphr.h"
#include "queue.h"
// Queue of 1 item
static SemaphoreHandle_t counting_semaphore;
static void lcd_update_task(void *p) {
    while(1) {
        // CONSUME the signal - similar to: xQueueReceive()
        //puts("lcd_update_task(): Attempting to take the semaphore");
        if (xSemaphoreTake(counting_semaphore, portMAX_DELAY)) {
            // go update the LCD
            printf("%3d: Okay, I will update the LCD\n", (unsigned)xTaskGetTickCount());
        }
    }
}
static void producer(void *p) {
    while (1) {
        xSemaphoreGive(counting_semaphore);
        // ...
        xSemaphoreGive(counting_semaphore);
        vTaskDelay(3000);
    }
}
int main(void) {
    //binary_semaphore = xSemaphoreCreateBinary(); // xQueueCreate(1, 0-byte-payload);
    counting_semaphore = xSemaphoreCreateCounting(1, 0);
    xTaskCreate(lcd_update_task, "lcd",          1024 * 8, NULL, 2, NULL); // High priority

```

```
xTaskCreate(producer,          "producer", 1024 * 8, NULL, 1, NULL); // Low priority

vTaskStartScheduler(); // this function doesn't return
return 0;}
```

Mutex

A mutex stands for "**M**utual **E**xclusion".

```
#include "semphr.h"
#include "queue.h"

static SemaphoreHandle_t mutex; // Mut = Mutual, and ex = Exclusion
static int global_counter;

static void write_file(const char * file, const char * data) {
    ++global_counter;
}

static void log_write_task(void *p) {
    int counter = 0;
    while (counter++ < 5 * 1000 * 1000) {
        if (xSemaphoreTake(mutex, 1000)) {
            write_file("log.txt", "... data to write ...");
            xSemaphoreGive(mutex);
        }
    }
    printf("Global counter value: %d\n", global_counter);
    vTaskSuspend(NULL);
}

static void file_write_task(void *p) {
    int counter = 0;
    while (counter++ < 5 * 1000 * 1000) {
        if (xSemaphoreTake(mutex, 1000)) {
            write_file("config_file.txt", "... data to write ...");
            xSemaphoreGive(mutex);
        }
    }
    printf("Global counter value: %d\n", global_counter);
}
```

```

    vTaskSuspend(NULL);
}
// Interesting fact:
// "Safe" RTOS variants do not have Mutexes
// So what do they want us to do???
void file_write_in_safe_rtos_applications(void *p) {
    while (1) {
        xQueueReceive(file_write_request);
        write_file(file_write_request.name, file_write_request.data);
    }
}
static void file_write_task(void *p) {
    while(1) {
        file_write_request;
        file_write_request.name = "config.txt";
        file_write_request.data = "data";
        file_write_request.queue_handle_to_respond_back_with_status;
        xQueueSend(file_write_request);
        xQueueReceive(queue_handle_to_respond_back_with_status);
    }
}
static void log_write_task(void *p) {
    while(1) {
        file_write_request;
        file_write_request.name = "log.txt";
        file_write_request.data = "data";
        file_write_request.another_queue;
        xQueueSend(file_write_request);
        xQueueReceive(another_queue);
    }
}
int main(void) {
    // binary_semaphore = xSemaphoreCreateBinary(); // xQueueCreate(1, 0-byte-payload);
    // counting_semaphore = xSemaphoreCreateCounting(1, 0);
    // mutex = xSemaphoreCreateCounting(1, 1);
    mutex = xSemaphoreCreateMutex();
}

```

```

xTaskCreate(log_write_task, "lcd",      1024 * 8, NULL, 2, NULL); // High priority
xTaskCreate(file_write_task,"producer", 1024 * 8, NULL, 2, NULL); // Low priority

vTaskStartScheduler(); // this function doesn't return
return 0;
}

```

Recursive Mutex

```

// RECURSIVE mutex
// Popular TCP/IP library: LwIP
int x;
void some_nested_code() {
    if (xSemaphoreTake(mutex)) {
        // No other task will get the mutex
        x++;
        xSemaphoreGive(mutex); // will not return the mutex
    }
}
void task(void *p) {
    while (1) {
        if (xSemaphoreTake(mutex)) {
            // No other task will get the mutex
            some_nested_code();
            x++;
            xSemaphoreGive(mutex);
        }
    }
}

```

Revision #3

Created 4 years ago by [Preet Kang](#)

Updated 7 months ago by [Preet Kang](#)