

Priority Inversion

Following code is a example of 3 tasks with different priorities and resource shared between lowest and highest priority task. Since the mutex is defined for shared resource, priority inheritance raises priority of lowest task and converts unbounded priority inversion to bounded priority inversion

If we use binary semaphore it will be unbounded priority inversion.

Code:

```
#include <Arduino_FreeRTOS.h>
#include <semphr.h>
#include <task.h>

// Globals
static SemaphoreHandle_t lock;
TaskHandle_t myTaskH=NULL;
TaskHandle_t myTaskM=NULL;
TaskHandle_t myTaskL=NULL;
TickType_t cs_wait = 750; // Time spent in critical section (ms)
TickType_t med_wait = 750;
//*****
***
// Tasks

// Task L (low priority)
void doTaskL(void *parameters) {

    TickType_t timestamp;

    // Do forever
    while (1) {
        Serial.print("The priority of task L is ");
        Serial.println(uxTaskPriorityGet(myTaskL));
        // Take lock
        Serial.println("Task L trying to take lock...");
        if (xSemaphoreTake(lock, 100)==pdTRUE)
        { // Say how long we spend waiting for a lock
            Serial.print("Task L got lock. ");
            Serial.println(" Task L Doing some work...");
            Serial.println("Mutex Value at start of L");
            Serial.println(uxSemaphoreGetCount(lock));
            xTaskCreate(doTaskH,
                        "Task H",
                        128,
                        NULL,
                        3,
                        &myTaskH);
            xTaskCreate(doTaskM,
```

```

    "Task M",
    128,
    NULL,
    2,
    &myTaskM);

```

```

    // Hog the processor for a while doing nothing
    timestamp = xTaskGetTickCount() * portTICK_PERIOD_MS;
    while ( (xTaskGetTickCount() * portTICK_PERIOD_MS) - timestamp < cs_wait);
    // Release lock
    Serial.println("Task L releasing lock.");
    Serial.print("The priority of task L is ");
    Serial.println(uxTaskPriorityGet(myTaskL));
    xSemaphoreGive(lock);
    Serial.println("Mutex Value in Task L ");
    Serial.println(uxSemaphoreGetCount(lock));
    }
    // Go to sleep
    vTaskDelay(500 / portTICK_PERIOD_MS);
}
}

```

```

// Task M (medium priority)
void doTaskM(void *parameters) {

```

```

    TickType_t timestamp;
    while (1) {

        // Hog the processor for a while doing nothing
        Serial.println("Task M doing some work...");
        timestamp = xTaskGetTickCount() * portTICK_PERIOD_MS;
        while ( (xTaskGetTickCount() * portTICK_PERIOD_MS) - timestamp < med_wait);
        // Go to sleep
        Serial.println("Task M done!");
        vTaskDelay(500 / portTICK_PERIOD_MS);
    }
}

```

```

// Task H (high priority)
void doTaskH(void *parameters) {

```

```

    TickType_t timestamp;
    Serial.println("Task H STARTS");
    // Do forever
    while (1) {

        // Take lock
        Serial.println("Task H trying to take lock...");
        if (xSemaphoreTake(lock, portMAX_DELAY)==pdTRUE)

```

```

{
// Say how long we spend waiting for a lock
Serial.print("Task H got lock. Spent ");
// Hog the processor for a while doing nothing
timestamp = xTaskGetTickCount() * portTICK_PERIOD_MS;
while ( (xTaskGetTickCount() * portTICK_PERIOD_MS) - timestamp < med_wait);
// Release lock
Serial.println("Task H releasing lock.");
xSemaphoreGive(lock);
Serial.println("Mutex Value in Task H");
Serial.println(uxSemaphoreGetCount(lock));
}
else
{
Serial.println("Task H is blocked");
}

// Go to sleep
vTaskDelay(500 );
}
}

//*****
***
// Main (runs as its own task with priority 1 on core 1)

void setup() {

// Configure Serial
Serial.begin(9600);
Serial.println();
Serial.println("---FreeRTOS Priority Inversion Demo---");

// Create semaphores and mutexes before starting tasks
lock = xSemaphoreCreateMutex();
Serial.print("Mutex Value at start");
Serial.println(uxSemaphoreGetCount(lock));
// xSemaphoreGive(lock); // Make sure binary semaphore starts at 1

// Start Task L (low priority)
xTaskCreate(doTaskL, "Task L",
           128,
           NULL,
           1,
           &myTaskL);

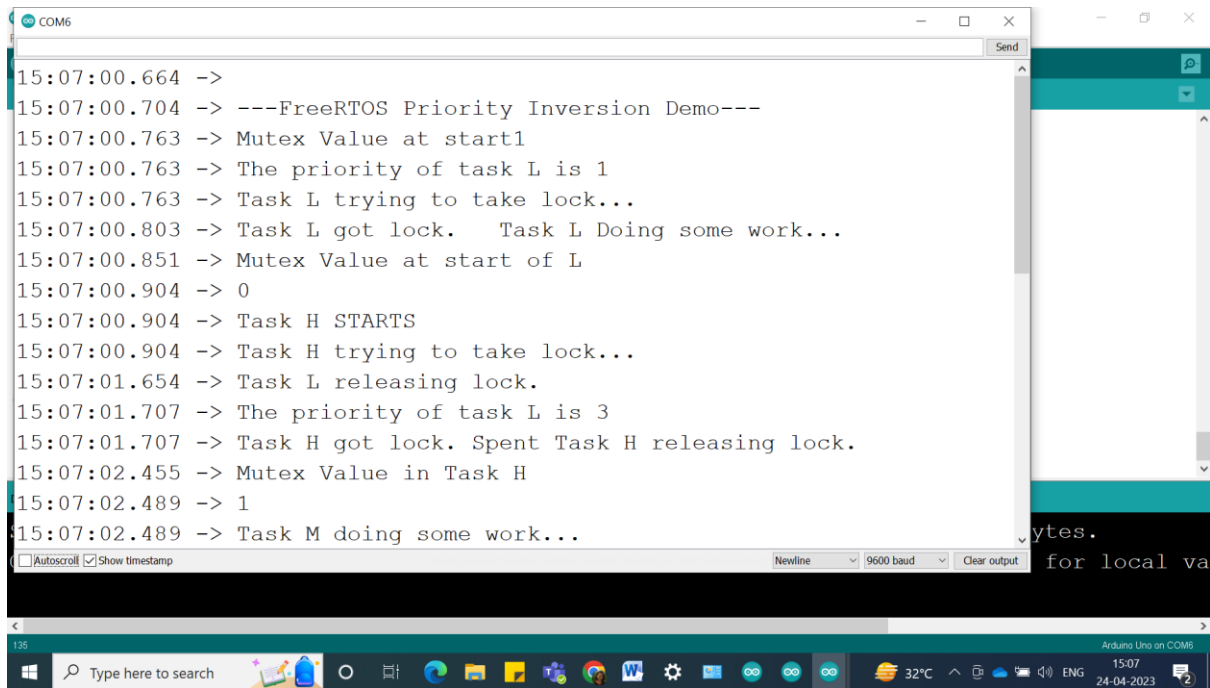
}

void loop() {

```

```
// Execution should never get here  
}
```

Output:



The screenshot shows a serial monitor window titled 'COM6' with a 'Send' button. The output text is as follows:

```
15:07:00.664 ->  
15:07:00.704 -> ---FreeRTOS Priority Inversion Demo---  
15:07:00.763 -> Mutex Value at start1  
15:07:00.763 -> The priority of task L is 1  
15:07:00.763 -> Task L trying to take lock...  
15:07:00.803 -> Task L got lock. Task L Doing some work...  
15:07:00.851 -> Mutex Value at start of L  
15:07:00.904 -> 0  
15:07:00.904 -> Task H STARTS  
15:07:00.904 -> Task H trying to take lock...  
15:07:01.654 -> Task L releasing lock.  
15:07:01.707 -> The priority of task L is 3  
15:07:01.707 -> Task H got lock. Spent Task H releasing lock.  
15:07:02.455 -> Mutex Value in Task H  
15:07:02.489 -> 1  
15:07:02.489 -> Task M doing some work...
```

At the bottom of the window, there are checkboxes for 'Autoscroll' (unchecked) and 'Show timestamp' (checked). To the right of these are dropdown menus for 'Newline' and '9600 baud', and a 'Clear output' button. The Windows taskbar is visible at the bottom, showing the search bar, taskbar icons, and system tray with temperature (32°C), time (15:07), and date (24-04-2023).