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: