Counting Semaphores Introduction

We can define counting semaphores just like a binary semaphore. For instance, consider counting semaphores as a queue of length more than one. Tasks that want to use counting semaphores will track the number of items available in the queue. Whenever a task takes a counting semaphore, the number of items available in the queue reduces. In other words, the number of items available in the queue defines the count of semaphore.

Resource =5, Counting semaphore =5;

T1 CS=3, CS=2

T2 CS=2, CS=1

T3 Cs=5 -→ CS=4

T4 CS=0;

T5 CS=4 → CS=3

T6....T7, T8, T9,...T100 CS=1 -→ CS=0

Types

It has two types. It is typically used for two things such as counting events and for resource Management.

Counting Events

In case of events counting, we can consider it as a counter which defines the maximum value and also keeps track of the number of times the event has occurred. Usually an event handler is used to give semaphore access and keep track of count value. Whenever a task takes a semaphore value, the count value is decremented by one.

For example, we created a counting semaphore of maximum value of 4 and initial value set to zero. In this case, an event handler will 'give' a semaphore each time an event occurs but the maximum events it can handle are upto four.

Resource Management

In case of resource management, counting semaphore defines the count of the number of shared resources available and that can be used by tasks at the same time.

For example, there are two FreeRTOS tasks that want to write data to a serial monitor of Arduino through the UART communication module of Arduino. But as you know that Arduino IDE has only one serial monitor. Also, Arduino uno has only one on-board UART module. Therefore, we must manage UART and Arduino IDE serial monitor resources between two tasks by using counting semaphore for resource management. We must first initialize the counting semaphore equal to the number of

resources. After that, a task which wants to use a resource must take a semaphore by decrementing the semaphore's count value.

Counting Semaphore Resource Management Example

For example, if two resources are available that can be shared among tasks. Hence, we must first set the counting semaphore value to 2. Therefore, only two tasks can acquire semaphore value at the same time. Also, whenever a task takes a semaphore, count value will be decremented by one. When a count value reaches zero that means no other resource is available. When count value reaches zero, any other task tries to access semaphore, it will enter the blocking state until currently executing tasks release the semaphore (or release the resource).

FreeRTOS API for Counting Semaphores

Before using a counting semaphore, we must first create it using FreeRTOS API. Like xSemaphoreCreateBinary(), xSemaphoreCreateCounting() API function is used to create counting semaphores. Also, the same function is used for event counting and resource management. Only the way we use input arguments with xSemaphoreCreateCounting() initialize it for event counting or resource management.

This is a function prototype:

```
SemaphoreHandle_t xSemaphoreCreateCounting
( UBaseType_t uxMaxCount, UBaseType_t uxInitialCount );
10          1      0→ no semaphore is available
```

This function takes two input arguments. The first argument is known as the uxMaxCount and the second argument is known as the uxInitialCount. The uxMaxCount is the maximum value to which the semaphore will count.

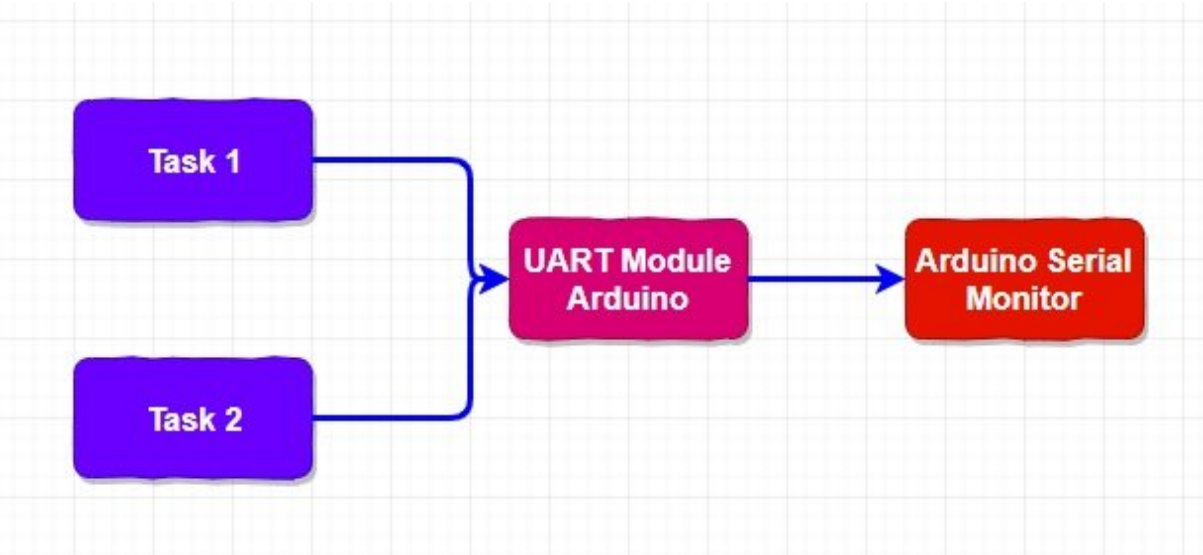
when the counting semaphore is to be used to count events, the uxMaxCount is the maximum number of events that can be counted. On the other hand, when the semaphore is used to manage access to a

collection of resources the uxMaxCount should be set to the total number of resources that are available.

The next argument the uxInitialCount is the initial count value of the semaphore after it has been created. When the semaphore is to be used for counting events the uxInitialCount should be set to zero this means when we create the semaphore, no event has occurred yet. Therefore set it to zero. When the counting semaphore is used to manage access to a collection of resources the uxInitialCount should be set equal to uxMaxCount. This means when the semaphore is created, all resources are available. Therefore, we set the uxInitialCount equal to the uxMaxCount.

FreeRTOS Counting Semaphores Example with Arduino

For demo, we will create an example using counting semaphore and Arduino. We will create two tasks such as Task1 and Task2. Both these tasks use serial communication of Arduino to send string to Arduino serial monitor. This picture depicts the functionality of this example.



Both tasks will use the same UART module of Arduino to write data on the serial monitor. If we consider the Arduino serial monitor and Uart communication module as a resource that will be utilized by both tasks. Then, we can use counting semaphore for this resource management

for Task1 and Task2. Because only one task can hold these Arduino resources at a time.

Program:

```
#include <Arduino_FreeRTOS.h>
#include <semphr.h>
SemaphoreHandle_t xCountingSemaphore;

void setup()
{
  Serial.begin(9600); // Enable serial communication library.
  pinMode(LED_BUILTIN, OUTPUT);

 // Create task for Arduino led
  xTaskCreate(Task1, // Task function
              "Ledon", // Task name
              128, // Stack size
              NULL,
              0 ,// Priority
              NULL );
   xTaskCreate(Task2, // Task function
              "Ledoff", // Task name
              128, // Stack size
              NULL,
              0, // Priority
              NULL );
   xCountingSemaphore = xSemaphoreCreateCounting(1,1);
   xSemaphoreGive(xCountingSemaphore);

}

void loop() {}


void Task1(void *pvParameters)
{
  (void) pvParameters;

  for (;;)
    {
      xSemaphoreTake(xCountingSemaphore, portMAX_DELAY);
      Serial.println("Inside Task1 and Serial monitor Resource Taken");
      digitalWrite(LED_BUILTIN, HIGH);
      xSemaphoreGive(xCountingSemaphore);
      vTaskDelay(1);
    }
  }
```

```
void Task2(void *pvParameters)
{
  (void) pvParameters;
  for (;;)
    {
      xSemaphoreTake(xCountingSemaphore, portMAX_DELAY);
      Serial.println("Inside Task2 and Serial monitor Resource Taken");
      digitalWrite(LED_BUILTIN, LOW);
      xSemaphoreGive(xCountingSemaphore);
      vTaskDelay(1);

  }
}
```