Experiment 1: Implement semaphore between 2 tasks

Approach: Create 2 tasks, one producer (Prio: 1) and another consumer (Prio: 2)
Make sure producer gets the semaphoref first
Get semaphore in producer and then release
Get semaphore in consumer and then release.

```
void prod(void *arg)
{
        xSemaphoreTake(sem,portMAX_DELAY);
        TickType_t delay = pdMS_TO_TICKS(8);
        while(1)
        {
                vTaskDelay(delay);
                xSemaphoreGive(sem);
                HAL_GPIO_TogglePin(GPIOD, LD1_Pin);

                vTaskDelete(NULL);
        }


        vTaskDelay(delay);
}
```

2) Semaphore is taken, performed a task, semaphore is given

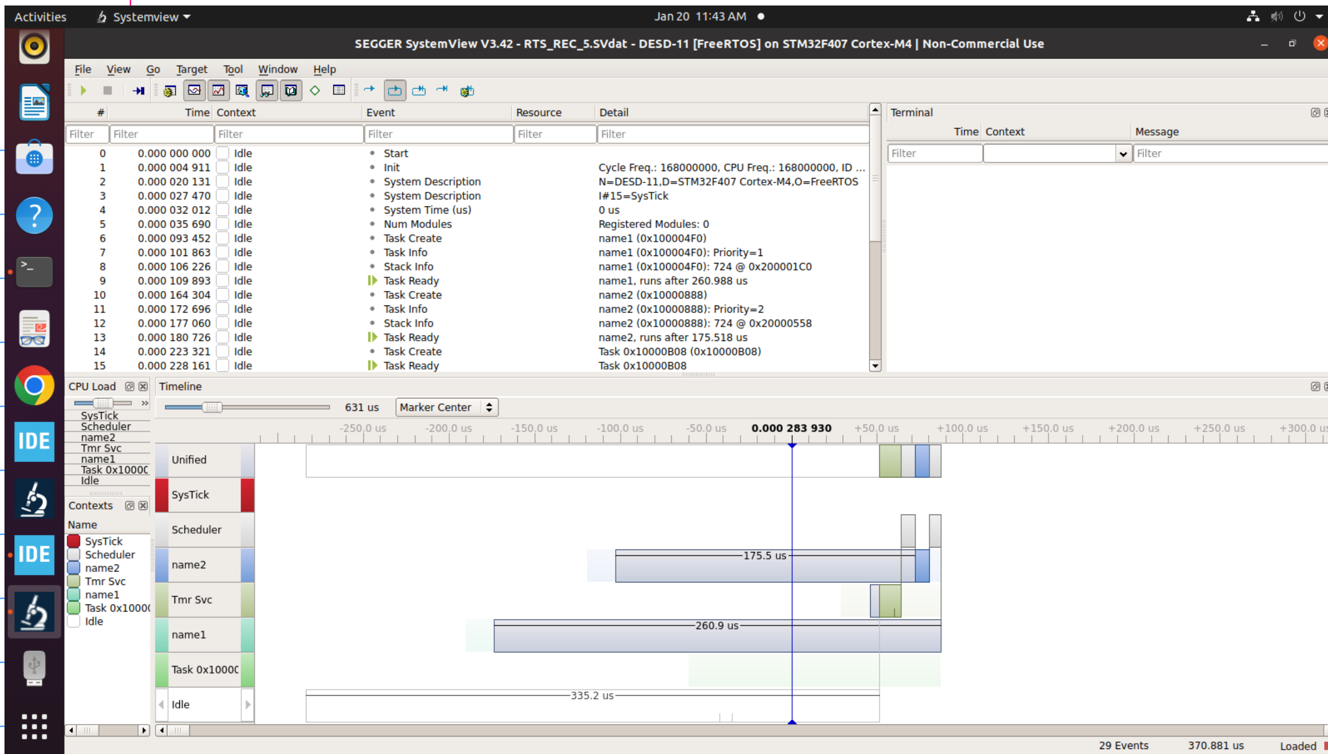
```
/*TASK2*/
void cons(void *arg)
{

                TickType_t delay = pdMS_TO_TICKS(1);
                vTaskDelay(delay);
                /*Above delay is to make sure that producer runs first*/
                while(1)
                {
                        xSemaphoreTake(sem,portMAX_DELAY);
                        HAL_GPIO_TogglePin(GPIOD, LD1_Pin);
                        xSemaphoreGive(sem);
                        vTaskDelete(NULL);
                }

}
```

1) This task will run first, programmer have to implement vTaskDelay() to make sure producer gets the semaphore first

3) After producer releases the seamphore, consumer takes it, perform a task and releases it.

```
Inside int main(void)

*DWT_CYCCNT |= (1 << 0);

    SEGGER_SYSVIEW_Conf();
    //SEGGER_UART_init(200000);
    SEGGER_SYSVIEW_Start();

   vSemaphoreCreateBinary(sem);
    xTaskCreate(prod, "name1", 200, NULL, 1, NULL);
    xTaskCreate(cons, "name2", 200, NULL, 2, NULL);

    vTaskStartScheduler();
```

Creating task & setting the Priority is done using xTaskCreate()


Activities
Systemview
Jan 20 11:43 AM
SEGGER SystemView V3.42 - RTS_REC_5.SVdat - DESD-11 [FreeRTOS] on STM32F407 Cortex-M4 | Non-Commercial Use
File View Go Target Tool Window Help
Time Context Event Resource Detail
0 0.000 000 000 Idle Start
1 0.000 004 911 Idle Init Cycle Freq.: 168000000, CPU Freq.: 168000000, ID ...
2 0.000 020 131 Idle System Description N=DESD-11,D=STM32F407 Cortex-M4,O=FreeRTOS
3 0.000 027 470 Idle System Description I#15=SysTick
4 0.000 032 012 Idle System Time (us) 0 us
5 0.000 035 690 Idle Num Modules Registered Modules: 0
6 0.000 093 452 Idle Task Create name1 (0x100004F0)
7 0.000 101 863 Idle Task Info name1 (0x100004F0): Priority=1
8 0.000 106 226 Idle Stack Info name1 (0x100004F0): 724 @ 0x200001C0
9 0.000 109 893 Idle Task Ready name1, runs after 260.988 us
10 0.000 164 304 Idle Task Create name2 (0x10000888)
11 0.000 172 696 Idle Task Info name2 (0x10000888): Priority=2
12 0.000 177 060 Idle Stack Info name2 (0x10000888): 724 @ 0x20000558
13 0.000 180 726 Idle Task Ready name2, runs after 175.518 us
14 0.000 223 321 Idle Task Create Task 0x10000B08 (0x10000B08)
15 0.000 228 161 Idle Task Ready Task 0x10000B08
Terminal
Time Context Message
CPU Load
Timeline
631 us
Marker Center
0.000 283 930
Unified
SysTick
Scheduler
name2
175.5 us
Tmr Svc
name1
260.9 us
Task 0x1000C
Idle
335.2 us
Contexts
Name
SysTick
Scheduler
name2
Tmr Svc
name1
Task 0x1000(
Idle
29 Events
370.881 us
Loaded

2) Implement queue send and recieve by creating 2 tasks.

Approach: Create a Queue and store it's handle in a variable
Create 2 tasks using xTaskCreate and pass the handle as a parameter.
Send an element in Queue from task 1
Recieve it in another task.

```
TaskHandle_t task1_hand, task2_hand;
QueueHandle_t Queue_Hand;
UBaseType_t len=50;

Queue_Hand=xQueueCreate(len,5);
if( Queue_Hand == NULL )
    {
            HAL_GPIO_WritePin(GPIOD, LD3_Pin, 1);          /* The queue could not be created. */
    }
/*arg1: length of data, arg2: no of elements in queue*/
if( xTaskCreate(task1, "task1", 200, Queue_Hand, 1, &task1_hand) != pdPASS )
    {
            HAL_GPIO_WritePin(GPIOD, LD1_Pin, 1);          /* The task could not be created. */
    }

if( xTaskCreate(task2, "task2", 200, Queue_Hand, 2, &task2_hand) != pdPASS )
    {
            HAL_GPIO_WritePin(GPIOD, LD2_Pin, 1);          /* The task could not be created. */
    }
vTaskStartScheduler();
```

In main function- Create a Queue using xQueueCreate(). This api returns the handle of the queue created. Store this handle in a variable.
Create 2 tasks using xTaskCreate & pass the queue handle in both the xTaskCreate calls.

```
/*Sender*/
void task1(void *arg)
{
        TickType_t delay = pdMS_TO_TICKS(5);
        while(1)
        {
                xQueueSendToBack(arg,"Hello", portMAX_DELAY);
                vTaskDelay(delay);
        }
}
```

Fetch the handle in void *arg. Send element in Queue using api xQueueSendToBack().

```
/*Reciever*/
void task2(void *arg)
{
//      char rbuff[20];
        TickType_t delay = pdMS_TO_TICKS(5);
        while(1)
        {
                xQueueReceive(arg,rbuff,portMAX_DELAY);
                vTaskDelay(delay);
        }
}
```

Fetch the queue handle in void *arg.
Task this will be in blocking state till the point the queue is empty. As soon as task1 fills some elements in queue, task2 gets unblocked.

Experiment 3) Implement the concept of notification between 2 tasks
Approach: Create 2 tasks

From task 1, notify task2 & delete itself
In task 2, recieve notification sent from task1 & delete itself.

```
inside int main(void)

 *DWT_CYCCNT |= (1 << 0);

    SEGGER_SYSVIEW_Conf();
    //SEGGER_UART_init(200000);
    SEGGER_SYSVIEW_Start();

  //  vSemaphoreCreateBinary(sem);
    xTaskCreate(prod, "producer", 200, NULL, 1, &prod_hand);
    xTaskCreate(cons, "consumer", 200, NULL, 1, &cons_hand);

    vTaskStartScheduler();vTaskDelete(NULL);
```

Create 2 tasks in main function keeping the priority alike.

```
TaskHandle_t prod_hand;
TaskHandle_t cons_hand;
xSemaphoreHandle sem;

void prod(void *arg)
{
        TickType_t delay = pdMS_TO_TICKS(2);
        while(1)
        {
                HAL_GPIO_WritePin(GPIOD, LD1_Pin,1);
                xTaskNotifyGive(cons_hand);
                vTaskDelay(delay);
         //ulTaskNotifyTake( pdTRUE, portMAX_DELAY );

                vTaskDelete(NULL);
        }
        //vTaskDelay(delay);
}
```

Use xTaskNotifyGive() api and pass the handle of destination task as parameter.

After that, delete the running task by vTaskDelete(NULL)

```
/*TASK2*/
void cons(void *arg)
{

        TickType_t delay = pdMS_TO_TICKS(3);
                //
                /*Above delay is to make sure that producer runs first*/
                while(1)
                {
                        ulTaskNotifyTake(pdTRUE, portMAX_DELAY );
                        vTaskDelay(delay);
                        HAL_GPIO_WritePin(GPIOD, LD2_Pin,1);
                        //xSemaphoreGive(sem);
                        //xTaskNotifyGive(prod_hand);
                        vTaskDelete(NULL);

                }

}
```

Use ulTaskNotifyTake() api to recieve the notificat

Experiment 4)
Implement a mechanism where an external interrupt (a button click) comes, the ISR send an element in queue to task1. Every 5th time the button ispressed, the task1 should unblock another task (task2) by sending an element in another queue & glow an LED.

Approach:

1) Create 2 tasks, task1 (priority 1)& task2 (priority 2)
2) Wrie an ISR (which is triggered by an exterbal interrupt) which sends an element in a queue (queue1). Count the no of button clicks done (external interrupt triggers) and send the count queue1, which in turn is recieved by task1.
3) After recieving queue element (count) in task1, check if the count is divisible by 5 compleetely. If the (count%5==0), send the count in another queue (queue2) to task2.
4) Glow LED in task2.

Desired output: LED should be glown after every 5th button click.

```
Inside int main(void)

  *DWT_CYCCNT |= (1 << 0);

    SEGGER_SYSVIEW_Conf();
    //SEGGER_UART_init(200000);
    SEGGER_SYSVIEW_Start();
    TaskHandle_t task1_hand, task2_hand;

    UBaseType_t len=50;

    Queue_Hand1=xQueueCreate(len,5);

    Queue_Hand2=xQueueCreate(len,5);
    /*arg1: length of data, arg2: no of elements in queue*/

    xTaskCreate(task1, "task1", 200, Queue_Hand1, 1, &task1_hand);

     xTaskCreate(task2, "task2", 200, Queue_Hand2, 2, &task2_hand);

    vTaskStartScheduler();
```

Create 2 Queues using xQueueCreate api, store the respective handles using variables
Create 2 tasks using xTaskCreate and pass the Queue handles as a parameter.

```c
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{

    uint32_t delay=200;
    static uint32_t last_tick=0;
    BaseType_t xHigherPriorityTaskWoken = pdFALSE;
    uint32_t tick_start = HAL_GetTick();
      if(tick_start-last_tick > delay)
      {
        count++;
        xQueueSendFromISR(Queue_Hand1,&count, xHigherPriorityTaskWoken);
        portYIELD_FROM_ISR(xHigherPriorityTaskWoken);

      }
      last_tick = tick_start;
}
```

In ISR (triggered after an external interrupt at pin PA0, user button), get the system tick count using HAL_GetTick() api and store it in tick_start variable of type uint32_t. Initialise a static int variable tick_start (which is initialised with value 0).

To implement a mechanism to check debouncing,check if last called interrupt with the current called interruot has a difference of atleast 200 ticks. This way, we are able to implement a debouncing avoidance mechanism witout implementing delay. We are just making sure that difference between 2 truggers have a substantial tick difference of 200 ticks.

After the condition becomes true, increamente the count and send it in queue using ISR safe API xQueueSendFromISR().

The Interrupt Safe API

Often it is necessary to use the functionality provided by a FreeRTOS API function from an interrupt service routine (ISR), but many FreeRTOS API functions perform actions that are not valid inside an ISR—the most notable of which is placing the task that called the API function into the Blocked state; if an API function is called from an ISR, then it is not being called from a task, so there is no calling task that can be placed into the Blocked state. FreeRTOS solves this problem by providing two versions of some API functions; one version for use from tasks, and one version for use from ISRs. Functions intended for use from ISRs have “FromISR” appended to their name.

Note: Never call a FreeRTOS API function that does not have “FromISR” in its name from an ISR.

The ARM Cortex cores, and ARM Generic Interrupt Controllers (GICs), use numerically *low* priority numbers to represent logically *high* priority interrupts. This can seem counter-intuitive, and is easy to forget. If you wish to assign an interrupt a logically low priority, then it must be assigned a numerically high value. If you wish to assign an interrupt a logically high priority, then it must be assigned a numerically low value.

For above mentioned conflict (lack of proper handshaking between FreeRTOS & ARM Cortex M4 NVIC convention), we have to make the priority of external interrupt highest (i.e, 15) in NVIC.

NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
Non maskable interrupt	☑	0	0
Hard fault interrupt	☑	0	0
Memory management fault	☑	0	0
Pre-fetch fault, memory access fault	☑	0	0
Undefined instruction or illegal state	☑	0	0
System service call via SWI instruction	☑	0	0
Debug monitor	☑	0	0
Pendable request for system service	☑	0	0
System tick timer	☑	15	0
PVD interrupt through EXTI line 16	☐	0	0
Flash global interrupt	☐	0	0
RCC global interrupt	☐	0	0
EXTI line0 interrupt	☑	15	0
Time base: TIM3 global interrupt	☑	15	0
FPU global interrupt	☐	0	0

```
void task1(void *arg)
{
    int count;
    char msg[20];
    TickType_t delay = pdMS_TO_TICKS(5);
    while(1)
    {

        xQueueReceive(Queue_Hand1,&count,NULL);

        if(count%5==0)
        {
        strcpy(msg,"LED_ON");
        xQueueSendToBack(Queue_Hand2,msg, portMAX_DELAY);

        strcpy(msg,"LED_OFF");

        }
        vTaskDelay(delay);
    }
}
```

Note: It seems that xQueueRecieveFromISR is a right fit to recieve Queue from ISR in task2, these ISRs which are appended with "FromISR" are only valid to be used inside ISR.

Recive the count inside Queue1 and check if count is in multiple of 5. If the condition is true, send the message LED_ON in another Queue2 and send it to task2.

```c
void task2(void *arg)
{
	char rbuff[20];
	TickType_t delay = pdMS_TO_TICKS(5);
	while(1)
	{
		xQueueReceive(Queue_Hand2,rbuff,portMAX_DELAY);
		if(strcmp(rbuff,"LED_ON"))
		{
			HAL_GPIO_TogglePin(GPIOD, LD1_Pin);
		}
		vTaskDelay(delay);
	}
}
```

Recieve Queue2 from task1 in task2.
Just check if the recieved element
(a buffer) have message LED_ON.
If yes, then Toggle the LED.