Fr. CONCEICAO RODRIGUES COLLEGE OF ENGINEERING (FrCRCE)

Department of Electronics and Computer Science (ECS)

9. Task-related functions using FreeRTOs

Course, Subject & Experiment Details

Academic year	2022 – 2023	Estimated Time	02 Hours
Course	T.E. (ECS)	Subject Name	Embedded systems and RTOS
Semester	VI	Chapter Title	FreeRTOS
Experiment Type	Coding	Subject Code	ECC 601

Aim & Objective of Experiment

To demonstrate the usage of vTaskSuspend, vTaskResume and vTaskDelete functions of FreeRTOS

Theory:

vTaskSuspend(): This function is used to Suspend a task, the suspended remains in the same state util it is resumed. For this, we need to pass the handle of the tasks that needs to be suspended. Passing NULL will suspend own task.

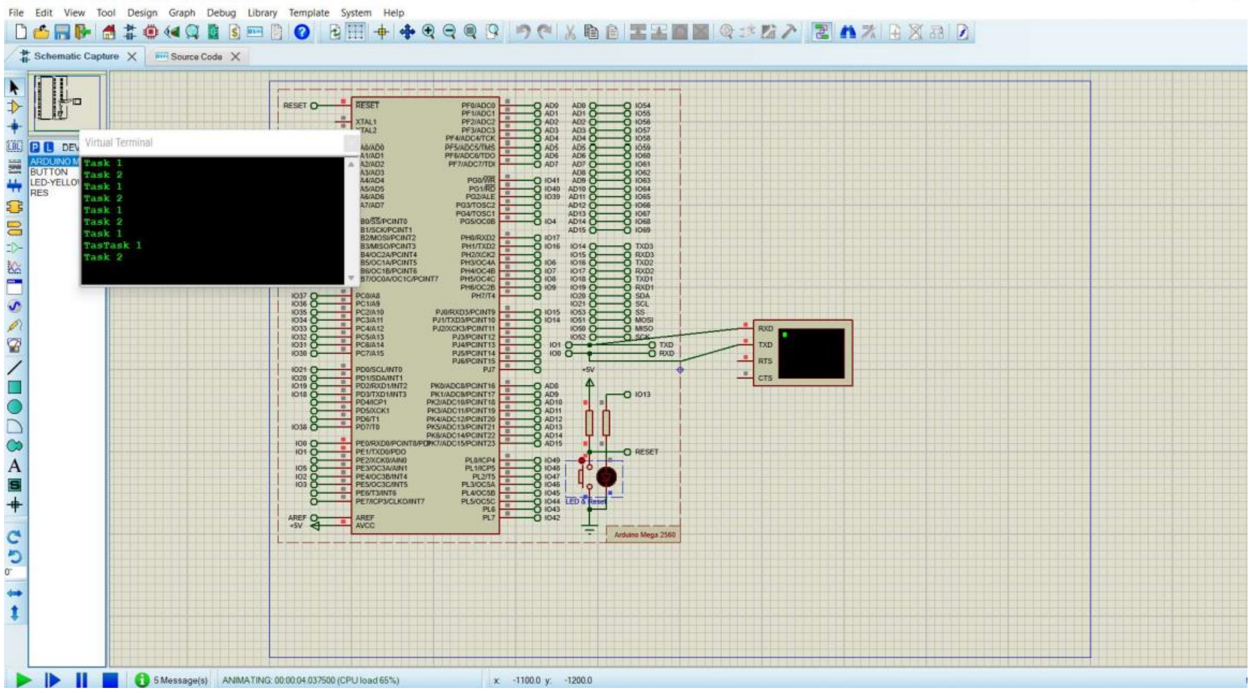
vTaskResume(): This function is used to resume a suspended task. If the Resumed task has higher priority than the running task then it will preempt the running task or else stays in ready state

vTaskDelete(): This function is used to delete as task. We need to pass the taskHandle of the task to be deleted.
To delete the own task we should pass NULL as the parameter.

Code:

```
#include <Arduino_FreeRTOS.h>

void Task_Print1(void *param);

void Task_Print2(void *param);

TaskHandle_t Task_Handle1;

TaskHandle_t Task_Handle2;
```

```
int counter = 0;
void setup() {
 // put your setup code here, to run once:
 Serial.begin(9600);
 xTaskCreate(Task_Print1,"Task1",100,NULL,1,&Task_Handle1);
 xTaskCreate(Task_Print2,"Task2",100,NULL,1,&Task_Handle2);
}
void loop() {
 // put your main code here, to run repeatedly:
}
void Task_Print1(void *param){
 (void) param;
 TickType_t getTick;
 getTick = xTaskGetTickCount(); //the getTick will get time from systick of OS
 while(1){
 Serial.println("Task 1");
 counter++;
 if(counter == 15){
 vTaskResume(Task_Handle2);
 }
// vTaskDelayUntil(&getTick,1000 / portTICK_PERIOD_MS);
 vTaskDelay(1000/portTICK_PERIOD_MS);
 }
}
void Task_Print2(void *param){
 (void) param;
 while(1){
 counter++;
 Serial.println("Task 2");
 if(counter == 10){
vTaskDelete(Task_Handle2);
 vTaskSuspend(Task_Handle2);
 }
 vTaskDelay(1000/portTICK_PERIOD_MS);
 }
 }
```



```
Task 1
Task 2
Task 1
Task 2
Task 1
Task 2
Task 1
TasTask 1
Task 2
```

File Edit View Tool Design Graph Debug Library Template System Help
Schematic Capture
Source Code
Virtual Terminal
Arduino Mega 2560
5 Message(s)
ANIMATING: 00:00:04.037500 (CPU load 65%)

Postlab questions

1. Explain the instances when tasks may need to be temporarily suspended and resumed later.
2. What is the possible disadvantage of deleting tasks?

Fr. CONCEICAO RODRIGUES COLLEGE OF ENGINEERING (FrCRCE)
Department of Electronics and Computer Science (ECS)

10. Inter-process communication using FreeRTOS

Course, Subject & Experiment Details

Academic year	2022 – 2023	Estimated Time	02 Hours
Course	T.E. (ECS)	Subject Name	Embedded systems and RTOS
Semester	VI	Chapter Title	FreeRTOS
Experiment Type	Coding	Subject Code	ECC 601

Aim & Objective of Experiment

To demonstrate the usage of a Mutex using FreeRTOS

Theory:

Inter Process Communication (IPC) refers to a mechanism, where the operating systems allow various processes to communicate with each other. This involves synchronizing their actions and managing shared data.

Mutexes are binary semaphores that include a priority inheritance mechanism. Whereas binary semaphores are the better choice for implementing synchronisation (between tasks or between tasks and an interrupt), mutexes are the better choice for implementing simple mutual exclusion (hence 'MUT'ual 'EX'clusion).

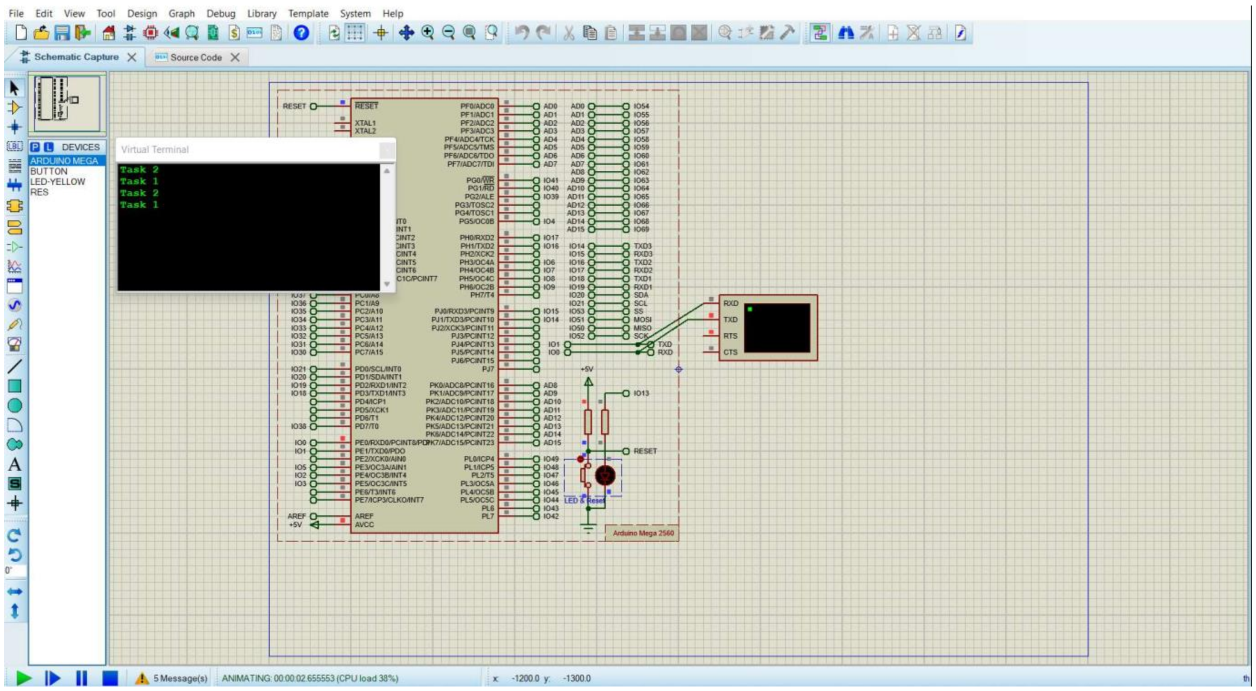
When used for mutual exclusion the mutex acts like a token that is used to guard a resource. When a task wishes to access the resource it must first obtain ('take') the token. When it has finished with the resource it must 'give' the token back - allowing other tasks the opportunity to access the same resource.

Note that Binary semaphores and mutexes are very similar but have some subtle differences: Mutexes include a priority inheritance mechanism, binary semaphores do not. This makes binary semaphores the better choice for implementing synchronisation (between tasks or between tasks and an interrupt), and mutexes the better choice for implementing simple mutual exclusion.

A semaphore can be used in order to control the access to a particular resource that consists of a finite number of instances

Code:

```
#include <Arduino_FreeRTOS.h>
void Task_Print1(void *param);
void Task_Print2(void *param);
TaskHandle_t Task_Handle1;
TaskHandle_t Task_Handle2;
volatile boolean myMutex = false;
void setup() {
 Serial.begin(9600);
 xTaskCreate(Task_Print1, "Task 1", 100, NULL, 1, &Task_Handle1);
 xTaskCreate(Task_Print2, "Task 2", 100, NULL, 1, &Task_Handle2);
 // put your setup code here, to run once:
}
void loop() {
 // put your main code here, to run repeatedly:
}
void Task_Print1(void *param)
{
(void) param;
 while(1)
 {
 while(myMutex == true);
 for(int i = 0; i < 5; i++)
 {
 Serial.println(i);
 vTaskDelay(1000/portTICK_PERIOD_MS);
 }
 myMutex = true;
 }
}
void Task_Print2(void *param)
{
 (void) param;
 while(1)
 {
 while(myMutex == false);
 for(int i = 0; i < 5; i++)
 {
 Serial.println(i);
 vTaskDelay(1000/portTICK_PERIOD_MS);
 }
 myMutex = false;
 }
```



```
Task 2
Task 1
Task 2
Task 1
```

```
5 Message(s)   ANIMATING: 00:00:02.655553 (CPU load 38%)   x: -1200.0 y: -1300.0
```

Postlab questions:

1. Explain the various IPC techniques supported by FreeRTOS
2. Clearly describe the usage of a queue IPC