PROJECT 1

Advanced Real Time Systems - ECE 5550G

Question 1

The Blink_AnalogRead program implements two periodic tasks :

- TaskBlink
    - Human Readable Name : Blink
    - Task Stack Size : 128
    - Task Priority : 2
    - Task Period : 1*s*
    - Code Snippet :

```
// Setting Up the Task : TaskBlink
xTaskCreate(
    TaskBlink
    ,  "Blink"        // Human Readable Name
    ,  128            // Stack Size
    ,  NULL
    ,  2              // Priority 2
    ,  NULL );
```

```
// TaskBlink Function Definition
void TaskBlink(void *pvParameters)  // This is a task.
{
  (void) pvParameters;

  // Initialize digital LED_BUILTIN on pin 13 as an output.
  pinMode(LED_BUILTIN, OUTPUT);

  for (;;)
  {
    digitalWrite(LED_BUILTIN, HIGH);
    vTaskDelay( 1000 / portTICK_PERIOD_MS ); // wait for one second
    digitalWrite(LED_BUILTIN, LOW);
    vTaskDelay( 1000 / portTICK_PERIOD_MS ); // wait for one second
  }
}
```

- TaskAnalogRead
    - Human Readable Name : AnalogRead
    - Task Stack Size : 128
    - Task Priority : 1
    - Task Period : 15ms
    - Code Snippet :

```
// Setting Up The Task : TaskAnalogRead
xTaskCreate(
    TaskAnalogRead
    ,  "AnalogRead"   // Human Readable Name
    ,  128            // Stack Size
    ,  NULL
    ,  1              // Priority 1
    ,  NULL );
```

```
// TaskAnalogRead Function Definition
void TaskAnalogRead(void *pvParameters)
{
  (void) pvParameters;

  for (;;)
  {
    // read the input on analog pin 0:
    int sensorValue = analogRead(A0);

    // print out the value you read:
    Serial.println(sensorValue);

    // one tick delay (15ms) in between reads for stability
    vTaskDelay(1);
  }
}
```

Question 2

The modified AnalogRead_Blink program :

- For each task we get and store the task handle.
- Using the task handle, we get the assigned task name.
- Every time the task is executed the task name is printed.

The task periods of the TaskBlink and TaskAnalogRead have been changed to 500 ms and 100 ms respectively to help in debugging and printing the task name in Serial Monitor.

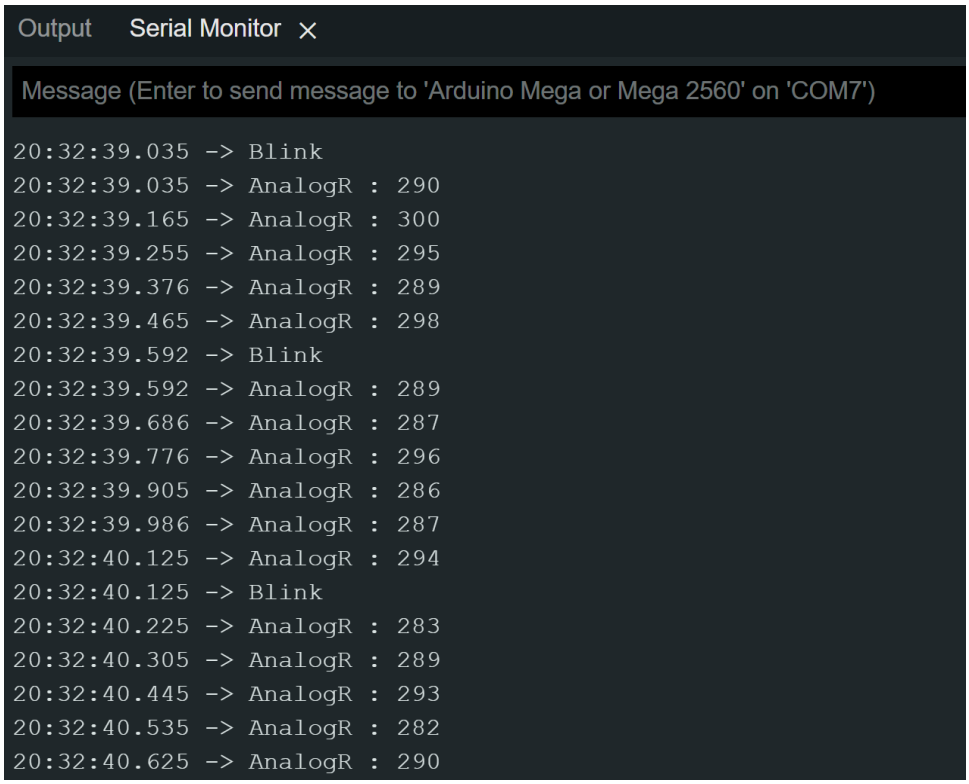
Code Snippet for TaskBlink :

```
void TaskBlink(void *pvParameters)
{
  (void) pvParameters;

  bool led_state                 = LOW;
  const TaskHandle_t task_handle = xTaskGetCurrentTaskHandle();
  const char *task_name          = pcTaskGetName(task_handle);

  // initialize digital LED_BUILTIN on pin 13 as an output.
  pinMode(LED_BUILTIN, OUTPUT);

  for (;;)
  {
    // Print the task name
    Serial.println(task_name);

    // Toggle the LED
    digitalWrite(LED_BUILTIN, led_state);
    led_state = !led_state;

    // Wait for 500 ms
    vTaskDelay(pdMS_TO_TICKS(500));
  }
}
```

Code Snippet for TaskAnalogRead :

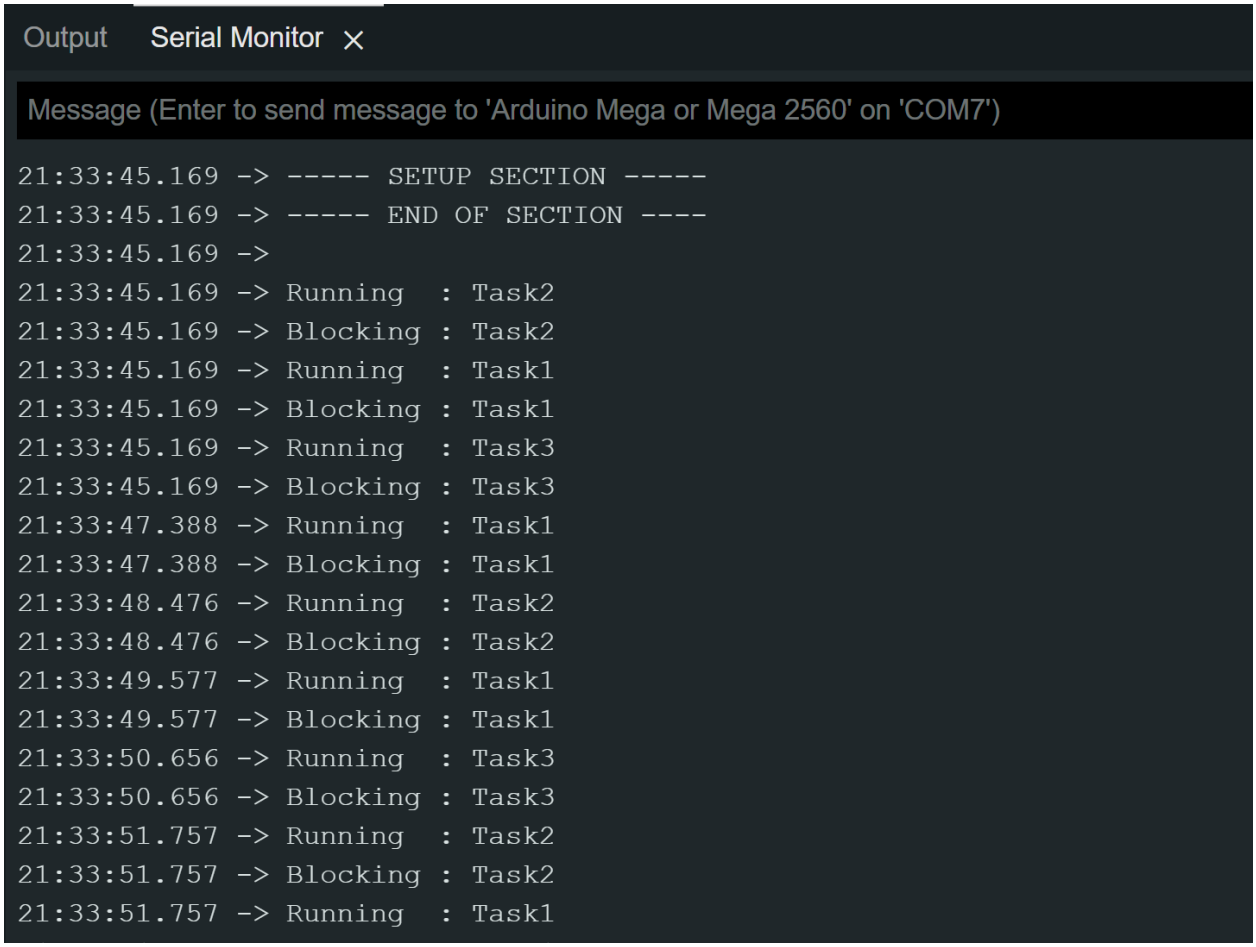
```
void TaskAnalogRead(void *pvParameters)
{
  (void) pvParameters;

  const TaskHandle_t task_handle = xTaskGetCurrentTaskHandle();
  const char *task_name          = pcTaskGetName(task_handle);

  for (;;)
  {
    // Print the task name
    Serial.print(task_name);
    Serial.print(" : ");

    // read the input on analog pin 0:
    int sensorValue = analogRead(A0);
    // print out the value you read:
    Serial.println(sensorValue);

    // Wait for 100 ms
    vTaskDelay(pdMS_TO_TICKS(100));
  }
}
```

Output of the Serial Monitor for Question 2 :



Question 3

Output of the Serial Monitor :



From the output we can observe :

- Each of the tasks are running periodically at the specified intervals.
- Whenever the tasks have the same arrival time, the highest priority task is executed first followed by the next highest priority task and so on. An example of such an instant is at the timestamp 21:33:45.169, where all three tasks have the same arrival time but are executed based on their priorities.
- If a lower priority task is already running, it is preempted by a task which has a higher priority.

So we can conclude that FreeRTOS is running a priority-based scheduler.

Question 4

The objective is to modify the code such that the lowest priority job is selected instead of the highest priority job.

This can be accomplished by inverting all the arithmetic, comparison and sorting operations, and changing the initial value of the variables associated with selecting a job for execution. By making these changes to the tasks.cpp file the lowest priority job will be treated as a highest priority job and vice-versa.

The list of changes made to the task.h :

- Defined two macro flags in the task.h :
  - **P_TEST** : When set to 1, the priority selection order is inverted by applying the changes described below to tasks.cpp file and task.h file. When set to 0, the original code behavior is restored.
  - **P_DEBUG** : When set to 1, Debug output is printed on the Serial Monitor. When set to 0, the debug is not displayed.
- The macro definition `tskIDLE_PRIORITY` is changed from `tskIDLE_PRIORITY ((UBaseType_t ) 0U)` to `tskIDLE_PRIORITY ((UBaseType_t) (configMAX_PRIORITIES - 1U))`

The list of changes made to the tasks.cpp file :

- Included the `"Arduino.h"` file to perform debugging via `Serial.println()`
- In the `taskRECORD_READY_PRIORITY(uxPriority)` macro definition, `(uxPriority) > uxTopReadyPriority` is changed to `(uxPriority) < uxTopReadyPriority`
- In the `taskSELECT_HIGHEST_PRIORITY_TASK()` macro definition, `++uxTopPriority` is changed to `--uxTopPriority`
- The initial value of the `uxTopReadyPriority` variable is changed to `uxTopReadyPriority = configMAX_PRIORITIES - 1U`
- The initial value of the `uxTopUsedPriority` variable is changed to `uxTopUsedPriority = 0`
- The macro invocation `listSET_LIST_ITEM_VALUE(&( pxNewTCB->xEventListItem ), ( TickType_t ) configMAX_PRIORITIES - ( TickType_t ) uxPriority)` is changed to `listSET_LIST_ITEM_VALUE(&( pxNewTCB->xEventListItem ), ( TickType_t ) uxPriority)` so that items are sorted in order with respect to inverted priority value.
- The comparison statement `(pxCurrentTCB->uxPriority >= pxNewTCB->uxPriority)` is changed to `(pxCurrentTCB->uxPriority <= pxNewTCB->uxPriority)`

- The comparison statement `(pxCurrentTCB->uxPriority > pxNewTCB->uxPriority)` is changed to `(pxCurrentTCB->uxPriority < pxNewTCB->uxPriority)`
- The comparison statement `(uxNewPriority > uxCurrentBasePriority)` is changed to `(uxNewPriority < uxCurrentBasePriority)`
- The comparison statement `(uxNewPriority >= pxCurrentTCB->uxPriority)` is changed to `(uxNewPriority <= pxCurrentTCB->uxPriority)`
- The comparison statement `(pxTCB->uxPriority >= pxCurrentTCB->uxPriority)` is changed to `(pxTCB->uxPriority <= pxCurrentTCB->uxPriority)`
- In the Idle task creation statement, the priority is changed from `portPRIVILEGE_BIT` to `tskIDLE_PRIORITY`
- Many comparison statements changes can be seen in the tasks.cpp file.

Summary of the changes :

- So we can observe that wherever priority is being compared we have to reverse or invert the logic so that the task selection gets inverted.
- Event Lists are always sorted based on the priority order, so these lines also have to be changed to maintain the order of the lists.
- The Idle task is given the lowest priority value when it is created, so the priority of the idle task needs to be made the highest value when the priority is inverted.

Observations in the output :

- The task which had lowest priority in Q3 (i.e Task3) has the highest priority now and the task which had the highest priority in Q3 (i.e Task2) has the lowest priority now, and so Task3 is executed first when the tasks arrive at the time.
- If a lower priority task is already running, it is preempted by a task which has a higher priority.