Mennat-Allah Ashraf Fouad
17P3051

AIN SHAMS UNIVERSITY
FACULTY OF ENGINEERING
ICHEP; Mechatronics Engineering Program

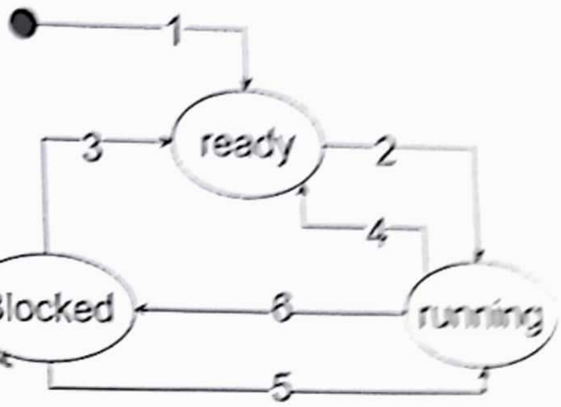
Course Code: CSE 347/345 *Time: 1 Hour*

Mid Term: Embedded System Design

The Exam Consists of **4** Questions in **5** Pages *Total Marks: 25 Marks*

Question 1: match a transition-number with the sentence that is suitable to change the task state as it shown:

A. Task is unblocked but is not the highest priority task 3
B. Task has the highest priority 2
C. Task no longer has the highest priority 4
D. Task is unblocked and its the highest priority task 5
E. Task is waiting for an event 6
F. Task is initialized and activated 1



Question 2: Complete the following sentences:

A. The figure below represents the code of a print_task_name.........task

Show_Name_of_Task

```
void Task( void *pvParameters )
{
char *pcTaskName;


    pcTaskName = ( char * ) pvParameters;

    for( ;; )
    {
        vPrintString( pcTaskName );


    }
}
```

B. The figure below represents the code of a delay task

```
void Task( void *pvParameters )
{
portTickType xLastWakeTime;

    xLastWakeTime = xTaskGetTickCount();

    for( ;; )
    {

        vPrintString( pcTaskName );

        vTaskDelayUntil( &xLastWakeTime, ( 10 / portTICK_RATE_MS ) );
    }
}
```

Question 3: Choose the correct answer:

1) context switching is:
   A. forcing the program counter to a specific address line to excute
   B. restoring saved context including its program counter value

Prof. Dr. Sherif Hammad

C. forcing link register to another return address

D. forcing the stack pointer to another context table address (TCB)

E. A&C

[✓ F.] B&D

2) The .... decides which thread should be executing by examining the priority assigned to each thread by the application designer

A. Systick handler

B. Kernel

[✓ C.] TCB

3) what is the register that its value should change to change the task TCB and do the context switching:

[✓ A.] Stack Pointer

B. Program Counter

C. Link Register

D. PSR

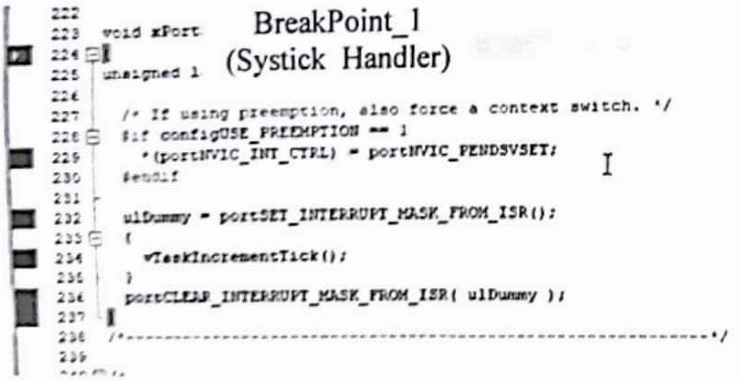
4) If you put a break points into Kernel and Systick handler as shown below, calling VtaskDelay() API would make the processor hit:

[✓ A.] BreakPoint_1 then another ready task

B. BreakPoint_2 then another ready task

C. BreakPoint_2 then BreakPoint_1 then another ready task

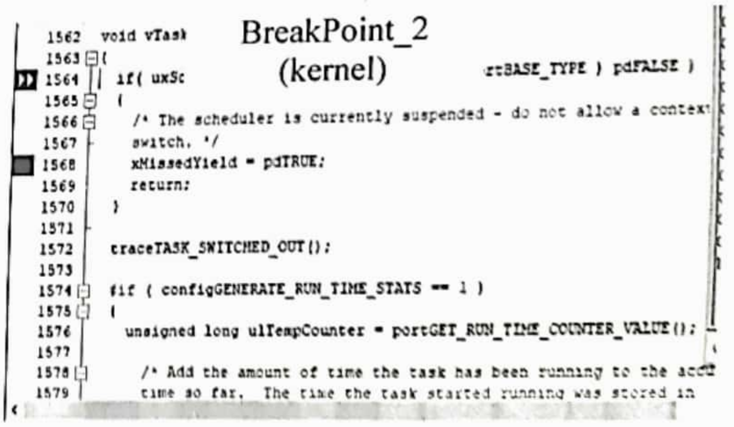
D. BreakPoint_1 then BreakPoint_2 then another ready task



```
void xPort
{
unsigned l

  /* If using preemption, also force a context switch. */
  #if configUSE_PREEMPTION == 1
    *(portNVIC_INT_CTRL) = portNVIC_PENDSVSET;
  #endif

  ulDummy = portSET_INTERRUPT_MASK_FROM_ISR();
  {
    vTaskIncrementTick();
  }
  portCLEAR_INTERRUPT_MASK_FROM_ISR( ulDummy );
}
/*-----------------------------------------------------------*/
```



```
void vTask
{
  if( uxSc                                    rtBASE_TYPE ) pdFALSE )
  {
    /* The scheduler is currently suspended - do not allow a context
    switch. */
    xMissedYield = pdTRUE;
    return;
  }

  traceTASK_SWITCHED_OUT();

  #if ( configGENERATE_RUN_TIME_STATS == 1 )
  {
    unsigned long ulTempCounter = portGET_RUN_TIME_COUNTER_VALUE();

      /* Add the amount of time the task has been running to the accu
      time so far.  The time the task started running was stored in
```

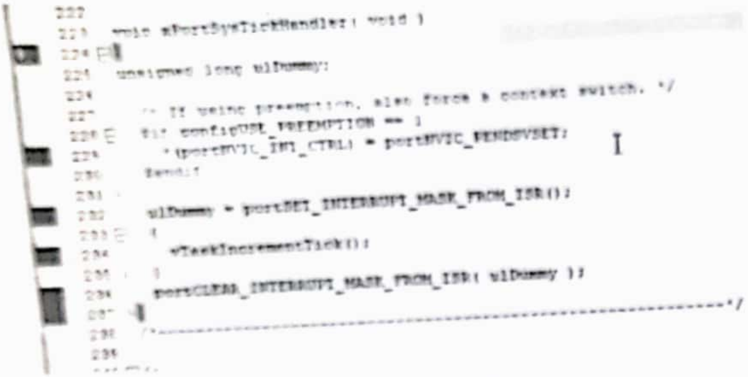
5) If you put a break points into Kernel and Systick handler as shown below, after preempting a continuous task at the end of its time slice the processor will hit:

A. BreakPoint_1 then BreakPoint_2 then another ready task

[✓ B.] BreakPoint_2 then BreakPoint_1 then another ready task

C. BreakPoint_1 then another ready task

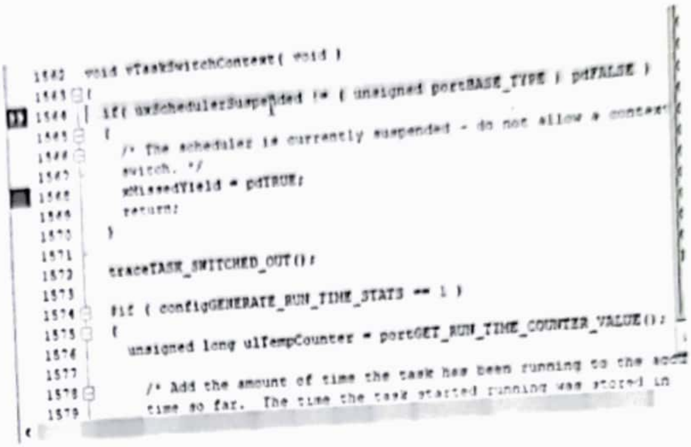
D. BreakPoint_2 then another ready task



```
void xPortSysTickHandler( void )
{
unsigned long ulDummy;

    /* If using preemption, also force a context switch. */
    #if configUSE_PREEMPTION == 1
        *(portNVIC_INT_CTRL) = portNVIC_PENDSVSET;
    #endif

    ulDummy = portSET_INTERRUPT_MASK_FROM_ISR();
    {
        vTaskIncrementTick();
    }
    portCLEAR_INTERRUPT_MASK_FROM_ISR( ulDummy );
}
/*-----------------------------------------------------------*/
```



```
void vTaskSwitchContext( void )
{
    if( uxSchedulerSuspended != ( unsigned portBASE_TYPE ) pdFALSE )
    {
        /* The scheduler is currently suspended - do not allow a contex
        switch. */
        xMissedYield = pdTRUE;
        return;
    }

    traceTASK_SWITCHED_OUT();

    #if ( configGENERATE_RUN_TIME_STATS == 1 )
    {
        unsigned long ulTempCounter = portGET_RUN_TIME_COUNTER_VALUE();

            /* Add the amount of time the task has been running to the acc
            time so far.  The time the task started running was stored in
```

BreakPoint_1
(Systick Handler)

BreakPoint_2
(kernel)

Question 4:

In a FreeRtos Project two short periodic tasks having the same priority was created:

- Task_1 toggles the Red LED every 1 seconds
- Task_2 toggles the Blue LED every 2 seconds
- Task_3 toggles the ~~Blue~~ LED every 3 seconds Green.

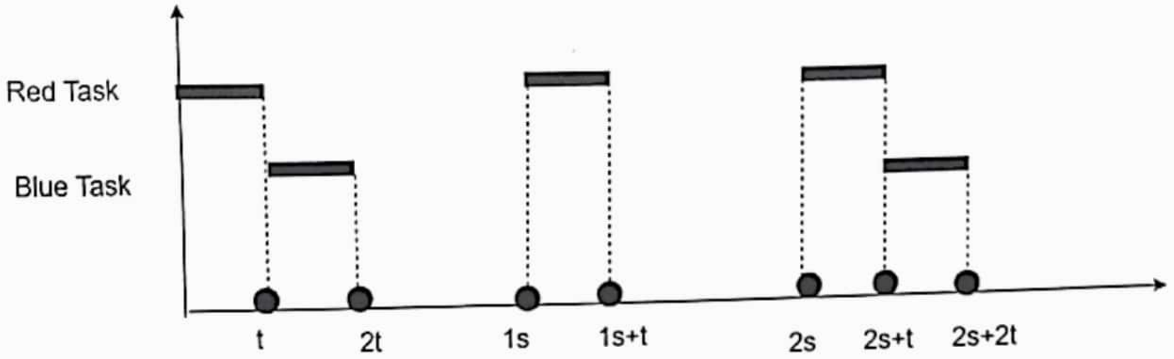
Draw its timing Diagram (for the first 4 seconds) for each of these cases:

- if the periodicity of each task was achieved using **vTaskDelay();**
- if the periodicity of each task was achieved using **vTaskDelayUntil();**

Knowing that t = execution time for each of the tasks

Question 4:

Write the code that achieves this timing diagram



t < time slice

You can use these functions to toggle the LEDS:

```
InitializeLeds();
ToggleRedLed();
ToggelBlueLed();
ToggleGreenLed();
```

```
void Task_1( void *pvParameters );
void Task_2( void *pvParameters );
void Task_3(void * pv_Parameters);

const char *pcTextForTask1 = "Red Task is running\n";
const char *pcTextForTask2 = "Blue Task is running\n";
const char *pcTextForTask3 = "~~InitTask~~ Green is running\n";

int main( void )
{ xTaskCreat(InitTask(), "Task_0", 240(void *) pcTextforTask0, 5, Null);
xTaskCreate( .RedTask(), "Task_1", 240, (void*)pcTextForTask1, ....1., NULL );
xTaskCreate( ..BlueTask(), "Task_2", 240, (void*)pcTextForTask2, ...2, NULL );
xTaskCreate( ..GreenTask() "Task_3", 240, (void*)pcTextForTask3, ...3.., NULL );
}
```

```
void InitTask(void)
{   PortF_initialization();                    portB_initialization();
    ~~Initializeleds();~~                      //Red LED in port B
                                               // Blue LED in Port B
                                               // Green LED in port B
```

```
        // Suspend ourselves.
      vTaskSuspend( NULL );
}


void BlueTask(void)
{
    TickType_t xLastWakeTime;

    xLastWakeTime = xTaskGetTickCount();

    while(1)
    {                                                        //oR use
      GPIO_PORTB_DATA_R ^= 0X04;  //Toggle Blue Led();
      vTaskDelayuntil( &lastWakeTime, 2/Port TICK_RATE_Ms);
    }
}


void RedTask(void)
{
      TickType_t xLastWakeTime;

      xLastWakeTime = xTaskGetTickCount();

      while(1)
      {                                                     //oR use
          GPIO_PORTB_DATA_R ^= 0xo2;  // ToggleRedLed();
          vTaskDelayUntil(&LastwakeTime, 1/ portTIck_RATE_Ms);
      }

}
void GreenTask (void) { TickType_t xLastWakeTime;
xLastWakeTime = xTaskGetTick Count ();
while (1){                                                   //oR use
      GPIo_PoRTB_DATA_R ^= 0x08;  //ToggleGreenLed
```

```
      vTaskDelayUntil(&LastWakeTime, 3/portTIck_RATE_Ms
      }
}
```