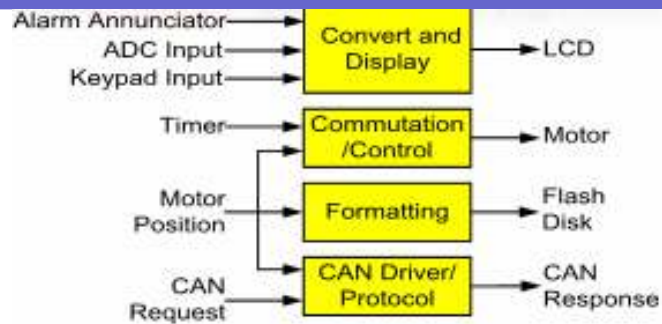


ARM PROGRAMMING

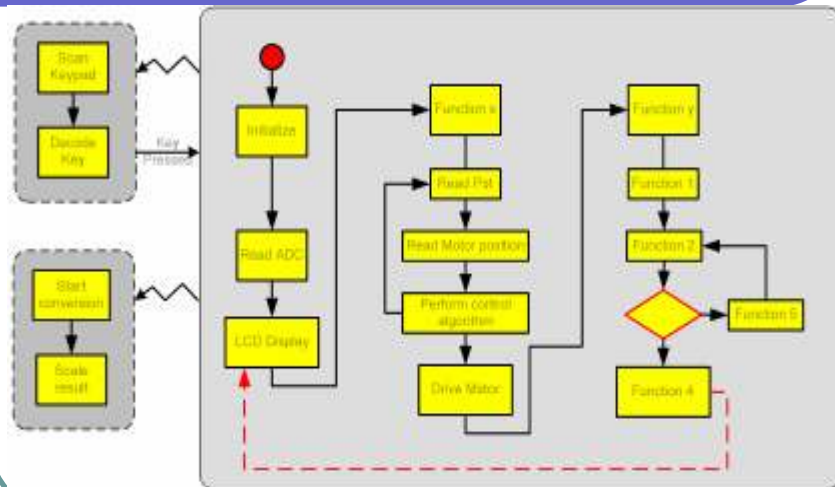
Bùi Quốc Bảo

Why use RTOS

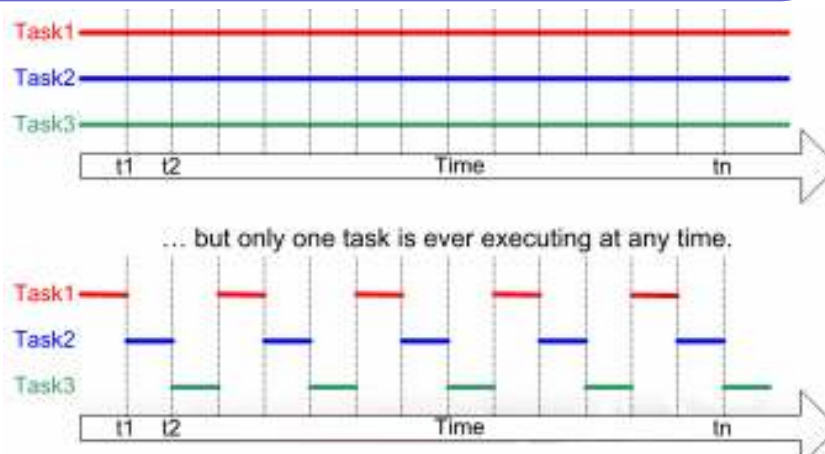


- Three sources of LCD messages
- Mix of fast and slow hardware
- Mix of continuous, periodic, event driven, high priority and low priority requirements

Why use RTOS



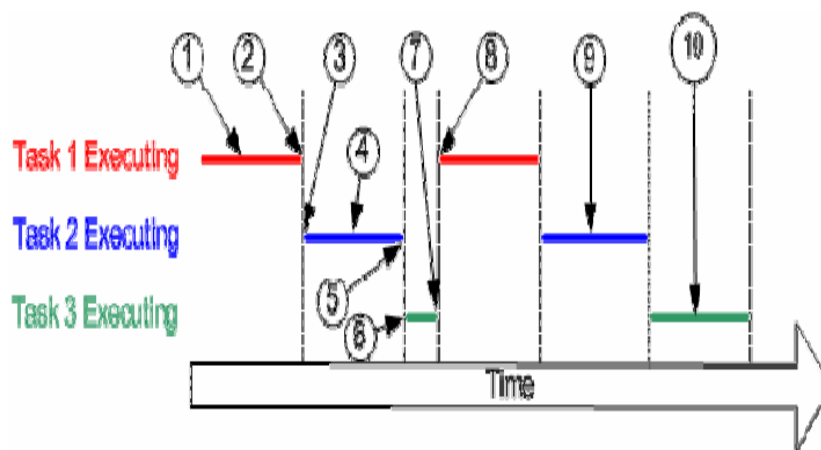
Concurrent processing



Scheduler

- Scheduler là 1 phần của kernel dùng để quyết định tác vụ nào được chạy tại mỗi thời điểm.
- Kernel có thể dừng và phục hồi hoạt động của 1 tác vụ trong khi tác vụ đang chạy.
- Một tác vụ có thể dừng chính nó bằng cách **delay (sleep)** một khoảng thời gian, hoặc **chờ (block)** để đợi một sự kiện (vd: keypressed) hay 1 tài nguyên (vd: serialport)

Scheduler



Scheduler

- At (1) task 1 is executing.
- At (2) the kernel suspends task 1 ...
- and at (3) resumes task 2.
- While task 2 is executing (4), it locks a processor peripheral for it's own exclusive access.
- At (5) the kernel suspends task 2 ...
- ... and at (6) resumes task 3.

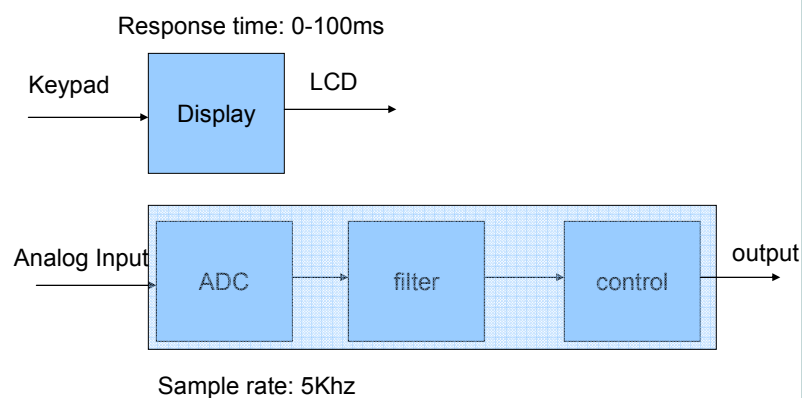
Scheduler

- Task 3 tries to access the same processor peripheral, finding it locked, task 3 cannot continue so suspends itself at (7).
- At (8) the kernel resumes task 1.
- Etc.
- The next time task 2 is executing (9) it finishes with the processor peripheral and unlocks it.
- The next time task 3 is executing (10) it finds it can now access the
- processor peripheral and this time executes until suspended by the kernel.

Realtime scheduler

- Mỗi tác vụ phải đáp ứng (response) trong thời gian qui định (deadline).
- Mỗi tác vụ có 1 mức ưu tiên riêng
- Scheduler sẽ đảm bảo tác vụ có quyền ưu tiên cao luôn được thực thi bằng cách tạm dừng tác vụ có quyền ưu tiên thấp.

Realtime OS



Keypad handler

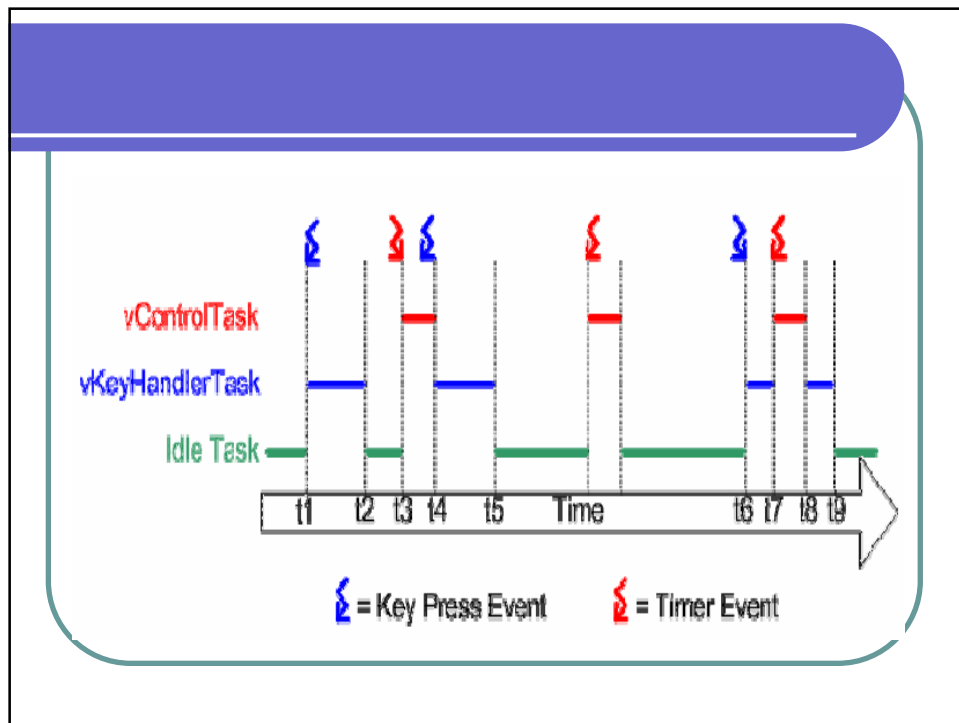
```
void vKeyHandlerTask( void *pvParameters )
{
    // Key handling is a continuous process and as such the task
    // is implemented using an infinite loop (as most real time
    // tasks are).
    for( ;; )
    {
        [Suspend waiting for a key press]
        [Process the key press]
    }
}
```

control

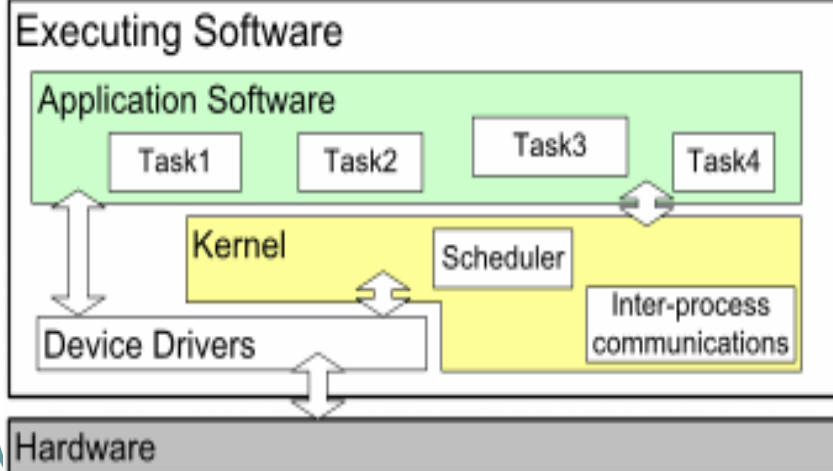
```
void vControlTask( void *pvParameters )
{
    for( ;; )
    {
        [Suspend waiting for 0.5ms since the start of the previous
        cycle]
        [Sample the input]
        [Filter the sampled input]
        [Perform control algorithm]
        [Output result]
    }
}
```

Mức ưu tiên

- Deadline của tác vụ control thì nghiêm ngặt hơn của tác vụ keypad.
- Hậu quả của việc trễ deadline của tác vụ control thì lớn hơn so với keypad



FreeRTOS realtime kernel



Example using FreeRTOS

```
int main( void ){
    /* Create the 2 task in exactly the same way. */
    xTaskCreate( vTask1, "Task 1", 1000, NULL, 1, NULL );
    xTaskCreate( vTask2, "Task 2", 1000, NULL, 1, NULL );
    /* Start the scheduler so our tasks start executing. */
    vTaskStartScheduler();
    /* If all is well we will never reach here as the
       scheduler will now be running. If we do reach
       here then it is likely that there was insufficient
       heap available for the idle task to be created. */
    for( ;; );
    return 0;
}
```


Example using FreeRTOS

```
void vTask1( void *pvParameters )
{
    const char *pcTaskName = "Task 1 is running\r\n";
    volatile unsigned long ul;

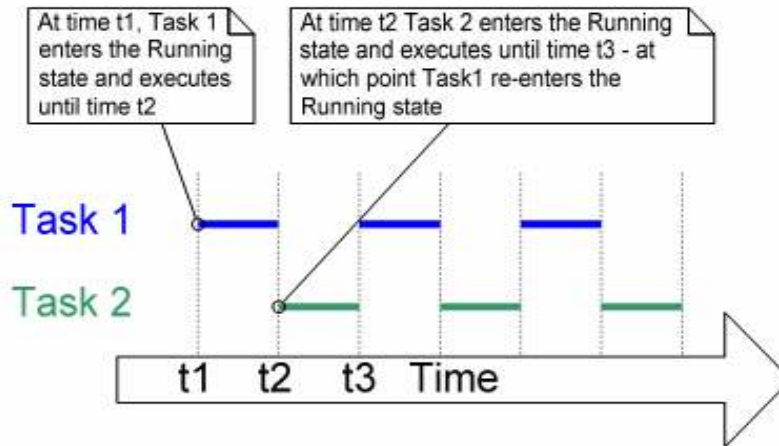
    /* As per most tasks, this task is implemented in an infinite
    loop. */
    for( ;; )
    {
        /* Print out the name of this task. */
        vPrintString( pcTaskName );
        /* Delay for a period. */
        for( ul = 0; ul < mainDELAY_LOOP_COUNT; ul++ )
        {
        }
    }
}
```

Example using FreeRTOS

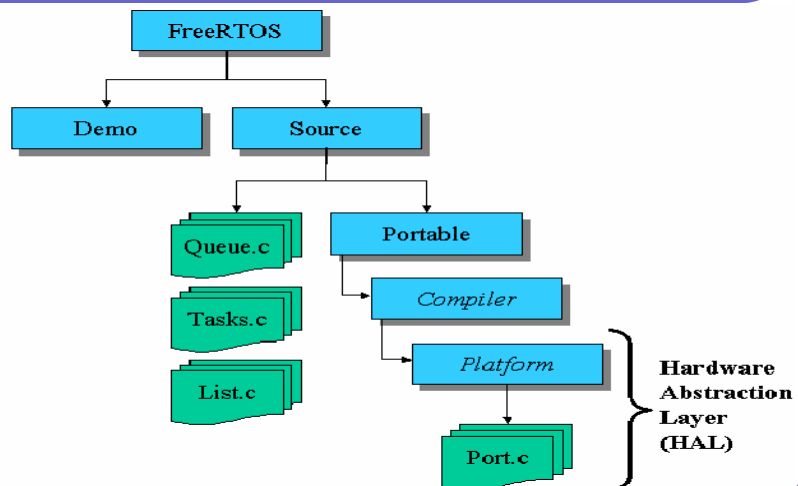
```
void vTask2( void *pvParameters )
{
    const char *pcTaskName = "Task 2 is running\r\n";
    volatile unsigned long ul;

    /* As per most tasks, this task is implemented in an infinite
    loop. */
    for( ;; )
    {
        /* Print out the name of this task. */
        vPrintString( pcTaskName );
        /* Delay for a period. */
        for( ul = 0; ul < mainDELAY_LOOP_COUNT; ul++ )
        {
        }
    }
}
```

The actual execution pattern of the two tasks



FreeRTOS source code structure



freeRTOSconfig.h

```
#define configUSE_PREEMPTION            1
#define configUSE_IDLE_HOOK            0
#define configUSE_TICK_HOOK            1
#define configCPU_CLOCK_HZ              ( ( unsigned long ) 50000000 )
#define configTICK_RATE_HZ              ( ( portTickType ) 1000 )
#define configMINIMAL_STACK_SIZE        ( ( unsigned short ) 70 )
#define configTOTAL_HEAP_SIZE           ( ( size_t ) ( 24000 ) )
#define configMAX_TASK_NAME_LEN         ( 12 )
#define configUSE_TRACE_FACILITY        1
#define configUSE_16_BIT_TICKS          0
#define configIDLE_SHOULD_YIELD         0
#define configUSE_CO_ROUTINES           0
#define configUSE_MUTEXES                1
#define configUSE_RECURSIVE_MUTEXES    1
#define configCHECK_FOR_STACK_OVERFLOW  2
```

freeRTOSconfig.h

```
#define configMAX_PRIORITIES    ( ( unsigned portBASE_TYPE ) 5 )
#define configMAX_CO_ROUTINE_PRIORITIES ( 2 )
#define configQUEUE_REGISTRY_SIZE    10

/* Set the following definitions to 1 to include the API function, or zero
to exclude the API function. */

#define INCLUDE_vTaskPrioritySet    1
#define INCLUDE_uxTaskPriorityGet   1
#define INCLUDE_vTaskDelete        1
#define INCLUDE_vTaskCleanUpResources 0
#define INCLUDE_vTaskSuspend       1
#define INCLUDE_vTaskDelayUntil    1
#define INCLUDE_vTaskDelay         1
#define INCLUDE_uxTaskGetStackHighWaterMark 1
```

Task management

Tác vụ là 1 hàm với một vòng lặp vô tận.

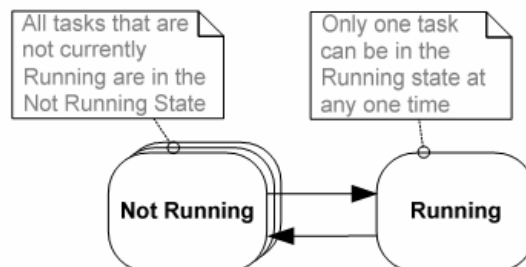
```
void ATaskFunction( void *pvParameters )
{
    int iVariableExample = 0;

    /* A task will normally be implemented as in infinite loop. */
    for( ;; )
    {
        [suspend waiting for a key press]
        [process the key]
    }

    vTaskDelete( NULL );
}
```

Task state

Tác vụ chuyển từ trạng thái dừng sang chạy gọi là “swapped in”,
“switched in”
Tác vụ chuyển từ trạng thái chạy sang dừng gọi là “swapped out”,
“switched out”



Task management function

Creation

xTaskCreate
xTaskDelete

Control

vTaskDelay
vTaskDelayUntil
uxTaskPriorityGet
vTaskPrioritySet
vTaskSuspend
vTaskResume
xTaskResumeFromIS

Utilities

tskIDLE_PRIORITY
xTaskGetTickCount
uxTaskGetNumberOfTasks
vTaskList
vTaskGetRunTimeStats
vTaskStartTrace
ulTaskEndTrace
uxTaskGetStackHighWaterMark
vTaskSetApplicationTaskTag
xTaskSetApplicationTaskTag
xTaskCallApplicationTaskHook

Naming convention

FreeRTOS name	Equivalent C
portCHAR	char
portFLOAT	float
portDOUBLE	double
portLONG	int
portSHORT	short
portSTACK_TYPE	unsigned int
portBASE_TYPE	int
portTickType	unsigned int

Naming convention

- Variables of type char are prefixed c
- Variables of type short are prefixed s
- Variables of type long are prefixed l
- Variables of type float are prefixed f
- Variables of type double are prefixed d
- Enumerated variables are prefixed e
- Other types (e.g. structs) are prefixed x
- Pointers have an additional prefixed p, for example a pointer to a short will have prefix ps
- Unsigned variables have an additional prefixed u, for example an unsigned short will have prefix us

Naming convention

- File private functions are prefixed with prv
- API functions are prefixed with their return type, as per the convention defined for variables
- Function names start with the file in which they are defined. For example vTaskDelete is defined in Task.c
- x is used both for structures and for RTOS types such as portTickType (in the ARM port this is equivalent to unsigned int)

Creating a Task, xTaskCreate

```
portBASE_TYPE xTaskCreate(  
    pdTASK_CODE          pvTaskCode,  
    const signed portCHAR * const pcName,  
    unsigned portSHORT    usStackDepth,  
    void                  *pvParameters,  
    unsigned portBASE_TYPE uxPriority,  
    xTaskHandle           *pxCreatedTask  
);
```

Creating a Task, xTaskCreate

- pvTaskCode: con trỏ chỉ đến tên tác vụ
- pcName: tên mô tả tác vụ (chiều dài lớn nhất mặc định là 16)
- usStackDepth: độ rộng stack của tác vụ.
- pvParameters: con trỏ chỉ đến tham số đưa vào tác vụ.

Creating a Task, xTaskCreate

- **uxPriority:** mức ưu tiên của tác vụ.
 - Có giá trị từ 0 đến (`configMAX_PRIORITIES - 1`)
 - 0 là mức ưu tiên thấp nhất
- **pxCreatedTask:** dùng để trả về handle của tác vụ

Creating a Task, xTaskCreate

- **Giá trị trả về:**
 - `pdPass` nếu thành công
 - `errCOULD_NOT_ALLOCATE_REQUIRED_MEMORY` nếu tác vụ không thể được khởi tạo

Creating a Task, xTaskCreate

```
xTaskCreate( vTask1,  
/* Pointer to the function that implements the task. */  
    "Task 1",  
/* Text name for the task. This is to facilitate debugging  
   only. */  
    1000,  
/* Stack depth - most small microcontrollers will use much  
   less stack than this. */  
    NULL,      /* We are not using the task parameter. */  
    1,         /* This task will run at priority 1. */  
    NULL );    /* We are not going to use the task handle. */
```

Using the task parameter

```
void vTaskFunction( void *pvParameters )  
{  
    char *pcTaskName;  
    volatile unsigned long ul;  
    pcTaskName = ( char * ) pvParameters;  
  
    for( ;; )  
    {  
        vPrintString( pcTaskName );  
  
        /* Delay for a period. */  
        for( ul = 0; ul < mainDELAY_LOOP_COUNT; ul++ );  
    }  
}
```

Using the task parameter

```
static const char *pcTextForTask1 = "Task 1 is running\r\n";
static const char *pcTextForTask2 = "Task 2 is running\t\n";
int main( void )
{
    /* Create one of the two tasks. */
    xTaskCreate( vTaskFunction, "Task 1", 1000,
                (void*)pcTextForTask1,
    /* Pass the text to be printed into the task
       using the task parameter. */
                1, NULL );
    xTaskCreate( vTaskFunction, "Task 2", 1000,
                (void*)pcTextForTask2, 1, NULL );
    vTaskStartScheduler();
    for( ;; );
}
```

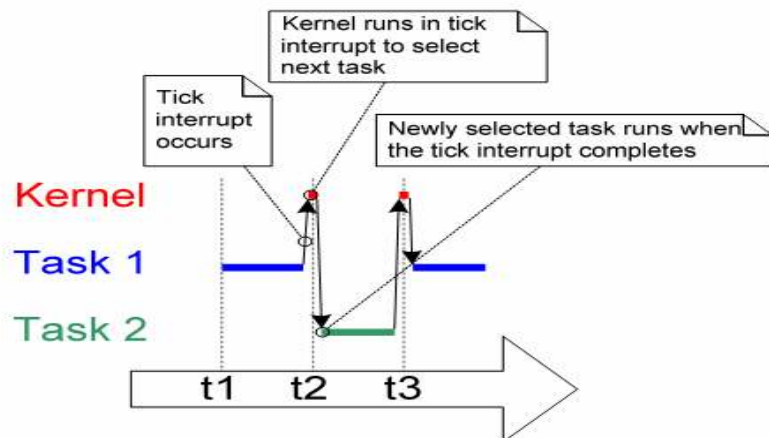
Task priorities

- Tham số configMAX_PRIORITIES qui định số mức ưu tiên có thể có.
- Nhiều tác vụ có thể có cùng mức ưu tiên
- Scheduler chạy sau mỗi khoảng thời gian xác định (system tick).
- Tham số configTICK_RATE_HZ qui định khoảng thời gian giữa 2 lần systemTick.

Task priorities

- Các API của FreeRTOS luôn dùng số lần systemTick xảy ra để đếm thời gian
- Tham số portTICK_RATE_MS dùng để chuyển đổi từ số lần sysTick sang thời gian ở ms.

Task priorities



Two task with different priorities

```
static const char *pcTextForTask1 = "Task 1 is running\r\n";
static const char *pcTextForTask2 = "Task 2 is running\t\n";
int main( void )
{
    /* Create one of the two tasks. */
    xTaskCreate( vTaskFunction, "Task 1", 1000,
                (void*)pcTextForTask1,
    /* Pass the text to be printed into the task
       using the task parameter. */
                1, NULL );
    xTaskCreate( vTaskFunction, "Task 2", 1000,
                (void*)pcTextForTask2, 2, NULL );
    vTaskStartScheduler();
    for( ;; );
}
```

Continuous processing

- Trong các ví dụ trên, các tác vụ luôn chạy mà không cần đợi bất kỳ sự kiện nào.
- Vì vậy, các tác vụ luôn ở trạng thái sẵn sàng chuyển sang Running

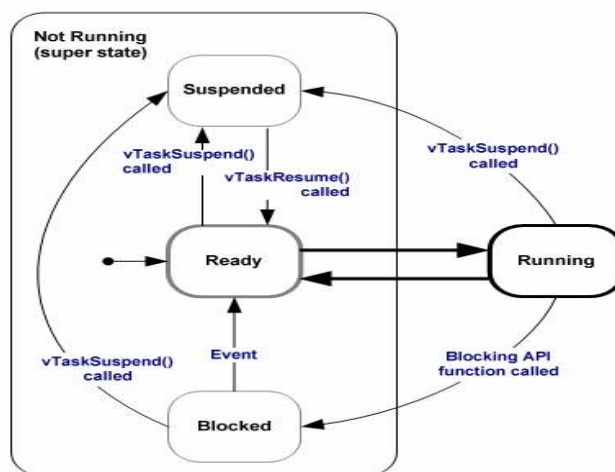


Chuyện gì xảy ra khi tác vụ dạng này có độ ưu tiên cao?

Event driven task

- Trong hệ thống nhúng, các tác vụ thường được thiết kế theo dạng “event-driven”
- Chúng phải chờ 1 sự kiện xảy ra để có thể đi vào trạng thái “Running”
- Scheduler sẽ chọn tác vụ có độ ưu tiên cao nhất mà có thể chuyển sang trạng thái Running.

Task state



The Blocked State

- Tác vụ đi vào trạng thái Block để chờ 2 dạng event sau:
- Thời gian:
 - một khoảng thời gian delay hoặc một thời điểm xác định kể từ khi scheduler bắt đầu chạy.
- Synchronization event: (sự kiện đồng bộ)
 - Khi sự kiện được tạo từ 1 tác vụ khác hay interrupt.
 - **FreeRTOS queues, binary semaphores, counting semaphores, recursive semaphores, mutexes** có thể dùng để tạo synchronization event.

The Suspended State

- Tác vụ bị trong trạng thái Suspended sẽ bị scheduler bỏ qua.
- Hàm `vTaskSuspend()` được gọi để đưa tác vụ vào trạng thái Suspended
- Hàm `vTaskResume()` hoặc `xvTaskResumeFromISR` dùng để đưa tác vụ ra khỏi trạng thái Suspended.

The Ready State

- Tác vụ sẵn sàng để chạy (vẫn nằm ở trạng thái NotRunning) thì ở trạng thái Ready

vTaskDelay()

```
void vTaskDelay( portTickType  
xTicksToDelay );
```

- Hàm vTaskDelay() đưa tác vụ vào trạng thái Blocked.
- Tác vụ trở lại trạng thái Ready sau xTicksToDelay lần SysTick.

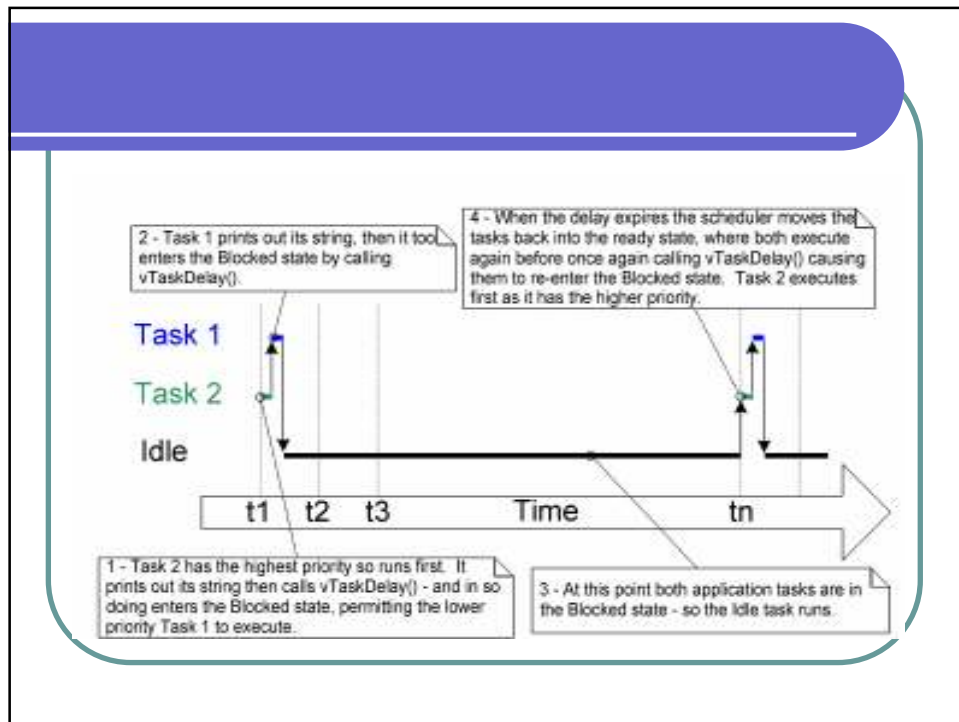
Example using vTaskDelay

```
void vTaskFunction( void *pvParameters )
{
    char *pcTaskName;
    pcTaskName = ( char * ) pvParameters;

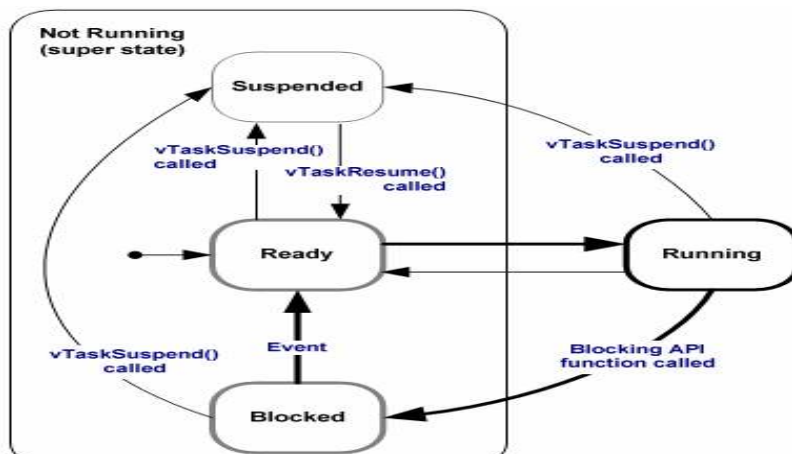
    /* As per most tasks, this task is implemented in an infinite loop. */
    for( ;; )
    {
        vPrintString( pcTaskName );

        /* In this case a period of 250 milliseconds is being specified. */
        vTaskDelay( 250 / portTICK_RATE_MS );
    }
}
```

```
static const char *pcTextForTask1 = "Task 1 is running\r\n";
static const char *pcTextForTask2 = "Task 2 is running\t\n";
int main( void )
{
    /* Create one of the two tasks. */
    xTaskCreate( vTaskFunction, "Task 1", 1000,
                (void*)pcTextForTask1,
                /* Pass the text to be printed into the task
                 using the task parameter. */
                1, NULL );
    xTaskCreate( vTaskFunction, "Task 2", 1000,
                (void*)pcTextForTask2, 2, NULL );
    vTaskStartScheduler();
    for( ;; );
}
```

Task transition



vTaskDelayUntil() API function

- void vTaskDelayUntil(portTickType *pxPreviousWakeTime, portTickType xTimeIncrement);
- vTaskDelayUntil chỉ chính xác số lần SysTick kể từ khi tác vụ chuyển sang trạng thái Running trước đó đến lần chuyển sang trạng thái Running kế tiếp

```
void vTaskFunction( void *pvParameters )
{
    char    *pcTaskName;
    portTickType    xLastWakeTime;
    pcTaskName = ( char * ) pvParameters;

    xLastWakeTime = xTaskGetTickCount();

    /* As per most tasks, this task is implemented in an infinite loop. */
    for( ;; )
    {
        /* Print out the name of this task. */
        vPrintString( pcTaskName );
        vTaskDelayUntil( &xLastWakeTime, ( 250 / portTICK_RATE_MS ) );
    }
}
```

Blocking and unblocking task

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

    pcTaskName = ( char * ) pvParameters;
    for( ;; )
    {
        vPrintString( pcTaskName );
    }
}
```

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

    xLastWakeTime = xTaskGetTickCount();

    /* As per most tasks, this task is implemented in an infinite loop. */
    for( ;; )
    {
        /* Print out the name of this task. */
        vPrintString( "Periodic task is running\r\n" );

        /* The task should execute every 10 milliseconds exactly. */
        vTaskDelayUntil( &xLastWakeTime, ( 10 / portTICK_RATE_MS ) );
    }
}
```

```

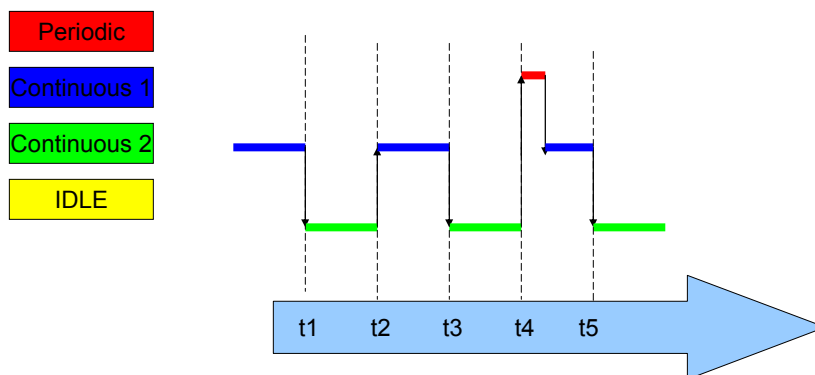
int main( void )
{
    xTaskCreate( vContinuousProcessingTask, "Task 1", 1000,
        (void*)pcTextForTask1, 1, NULL );
    xTaskCreate( vContinuousProcessingTask, "Task 2", 1000,
        (void*)pcTextForTask2, 1, NULL );

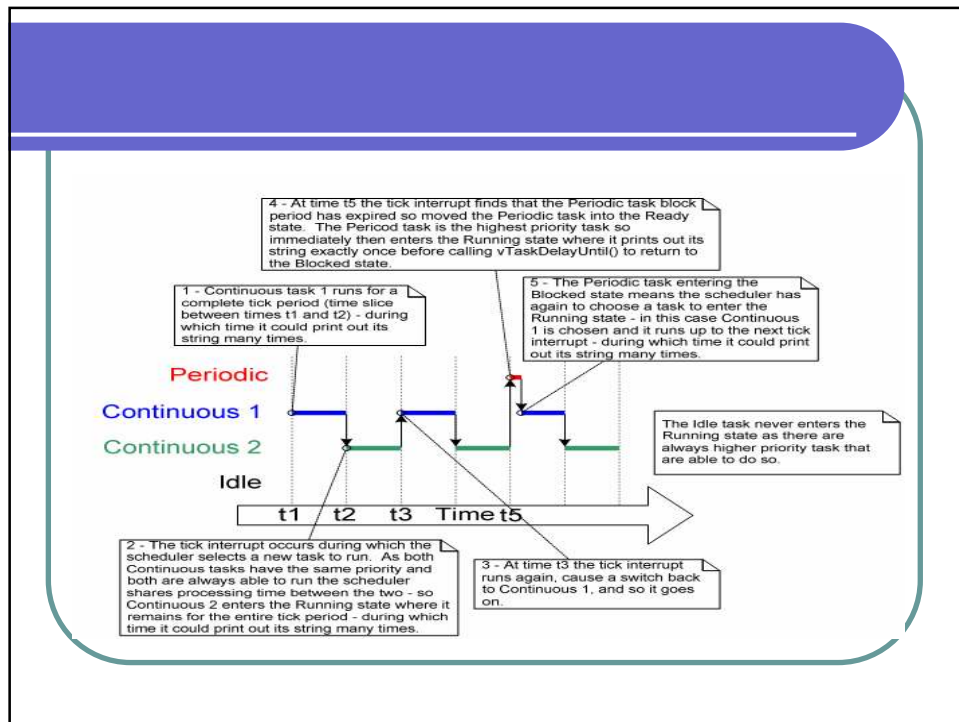
    /* Create one instance of the periodic task at priority 2. */
    xTaskCreate( vPeriodicTask, "Task 3", 1000,
        (void*)pcTextForPeriodicTask, 2, NULL );

    /* Start the scheduler so our tasks start executing. */
    vTaskStartScheduler();

    for( ;; );
    return 0;
}

```





IDLE task

- Idle task được tạo ra tự động, có độ ưu tiên 0.
- Nó được chạy khi không có bất kỳ tác vụ nào có mức ưu tiên cao hơn sẵn sàng chạy.

IDLE task hook

- Idle task hook là 1 hàm được idle task gọi trong mỗi vòng lặp của nó.
- Idle task hook thường dùng để:
 - Thực hiện các xử lý có ưu tiên thấp
 - Đo thời gian rảnh của hệ thống
 - Đưa CPU vào trạng thái công suất thấp

IDLE task hook

```
/* Declare a variable that will be incremented by the hook
function. */
unsigned long ulIdleCycleCount = 0UL;

/* Idle hook functions MUST be called vApplicationIdleHook(),
take no parameters,
and return void. */
void vApplicationIdleHook( void )
{
    /* This hook function does nothing but increment a counter.
    */
    ulIdleCycleCount++;
}
```

IDLE task hook

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

    pcTaskName = ( char * ) pvParameters;

    /* As per most tasks, this task is implemented in an infinite loop. */
    for( ;; )
    {
        vPrintStringAndNumber( pcTaskName, ulIdleCycleCount );

        /* Delay for a period for 250 milliseconds. */
        vTaskDelay( 250 / portTICK_RATE_MS );
    }
}
```

IDLE task hook

- IDLE task hook không được block hoặc suspend.
- Nếu có 1 tác vụ gọi hàm vTaskDelete(), IDLE task hook phải quay về idle task sau 1 khoảng thời gian vì idle task có nhiệm vụ tổ chức lại tài nguyên của kernel sau khi có 1 tác vụ bị xóa bỏ.

Change task priority

```
void vTaskPrioritySet( xTaskHandle pxTask,  
    unsigned portBASE_TYPE uxNewPriority );
```

(Tham số INCLUDE_vTaskPrioritySet phải bằng 1 để có thể dùng được hàm này.)

- **Parameters:**

- pxTask: Handle của tác vụ muốn được thay đổi mức ưu tiên. Nếu handle là NULL mức ưu tiên của chính tác vụ đang gọi hàm này sẽ được set.
- uxNewPriority: Mức ưu tiên mới

Query the task's priority

```
unsigned portBASE_TYPE uxTaskPriorityGet(  
    xTaskHandle    pxTask );
```

pxTask: task handle

Nếu pxTask là NULL, giá trị trả về sẽ là mức ưu tiên của chính tác vụ đang gọi hàm này.


```

void vTask1( void *pvParameters )
{
    unsigned portBASE_TYPE uxPriority;
    uxPriority = uxTaskPriorityGet( NULL );

    for( ;; )
    {
        /* Print out the name of this task. */
        vPrintString( "Task1 is running\r\n" );

        vPrintString( "About to raise the Task2 priority\r\n" );
        vTaskPrioritySet( xTask2Handle, ( uxPriority + 1 ) );
    }
}

```

```

void vTask2( void *pvParameters )
{
    unsigned portBASE_TYPE uxPriority;

    uxPriority = uxTaskPriorityGet( NULL );

    for( ;; )
    {
        vPrintString( "Task2 is running\r\n" );

        vTaskPrioritySet( NULL, ( uxPriority - 2 ) );
    }
}

```

```
/* Declare a variable that is used to hold the handle of
   Task2. */
xTaskHandle xTask2Handle;

int main( void )
{
    xTaskCreate( vTask1, "Task 1", 1000, NULL, 2, NULL );
    xTaskCreate( vTask2, "Task 2", 1000, NULL, 1,
                &xTask2Handle );
    vTaskStartScheduler();

    for( ;; );
}
```

Delete a task

```
void vTaskDelete( xTaskHandle
                  pxTaskToDelete );
```

Tác vụ có thể xóa 1 tác vụ khác hay chính nó.

Ide task sẽ giải phóng vùng nhớ đã cấp phát cho tác vụ được xóa.

```
void vTask1( void *pvParameters )
{
    const portTickType xDelay100ms = 100 / portTICK_RATE_MS;

    for( ;; )
    {
        /* Print out the name of this task. */
        vPrintString( "Task1 is running\r\n" );

        xTaskCreate( vTask2, "Task 2", 1000, NULL, 2, &xTask2Handle );
        vTaskDelay( xDelay100ms );
    }
}
```

```
void vTask2( void *pvParameters )
{
    vPrintString( "Task2 is running and about  
to delete itself\r\n" );
    vTaskDelete( xTask2Handle );
}
```

```

int main( void )
{
    xTaskCreate( vTask1, "Task 1", 1000, NULL, 1, NULL );

    /* Start the scheduler so the tasks start executing. */
    vTaskStartScheduler();

    /* main() should never reach here as the scheduler
    has been started. */
    for( ;; );
}

```

