RTOS

SESSION 3

AMIT

Content

- OPERATING SYSTEM MODELS
- Types of Systems.
- Software architecture concepts.
- Type of RTS.
- What is RTOS?
- Difference between GPOS and RTOS.
- Software architecture concepts.
- Why do we need RTOS?
- Let’s Create the Scheduler APIs

NAMING CONVENTION AT FREERTOS :

- **Popular Data Type that is used in freeRTOS :**

BaseType_t

- This is always defined as the most efficient data type for the architecture.
- Typically, this is a 32-bit type on a 32-bit architecture, a 16-bit type on a 16-bit
  architecture, and an 8-bit type on an 8-bit architecture.
- BaseType_t is generally used for return types that can take only a very limited range of values, and for pdTRUE/pdFALSE type Booleans.

Macro	Value
pdTRUE	1
pdFALSE	0
pdPASS 1	1
pdFAIL 0	0

NAMING CONVENTION AT FREERTOS :

1. **pdTRUE**
   This indicates that the task was created successfully.
2. **errCOULD_NOT_ALLOCATE_REQUIRED_MEMORY**
   This indicates that the task could not be created because there was insufficient heap memory available for FreeRTOS to allocate enough RAM to hold the task data structures and stack.
3. **pdPASS**
   This indicates that the task has been created successfully.
4. **pdFAIL**
   This indicates that the task has not been created because there is insufficient heap memory available for FreeRTOS to allocate enough RAM to hold the task data structures and stack.

** To use the Macros of “**pdPASS** or **pdFAIL**”, include “**projdefs.h**” library. **

NAMING CONVENTION AT FREERTOS :

- **Function Names:**
  - Functions are prefixed with return type and file defined in.
  - File scope (private) functions are prefixed with ‘prv’.

For example:

- vTaskPrioritySet() returns a void and is defined within task.c .
- xQueueReceive() returns a variable of type BaseType_t and is defined within queue.c.
- pvTimerGetTimerID() returns a pointer to void and is defined within timers.c.

- **Macro Names:**

  - Most macros are written in upper case,and prefixed with lower case letters that indicate where the macro is defined.

For example:

- portMAX_DELAY → portable.h or portmacro.h
- taskENTER_CRITICAL() → task.h

NAMING CONVENTION AT FREERTOS :

- **Variable Names:**
  - ‘c’ for char
  - ‘s’ for int16_t (short)
  - ‘l’ int32_t (long)
  - ‘x’ for BaseType_t or any other non-standard types (task handles, queue handles,..etc).
  - ‘u’ for unsigned variable.
  - ‘p’ for pointer.

For example:

a variable of type uint8_t will be prefixed with ‘uc’,

```
usStackDepth
```

a variable of type pointer to char will be prefixed with ‘pc’.

```
pcQueueGetName();
```

CONFIGURATION PARAMETERS AT

- ✓ System frequency.
- ✓ RTOS tick frequency.
- ✓ Enable / disable preemption.
- ✓ Enable / disable time slicing.
- ✓ Enable / disable Tick and IDLE hooks.
- ✓ Enable / disable using certain APIs.
- ✓ MAX PRIORITIES
- ✓ Heap size.
- ✓ MINIMAL size of Stack.
- ✓ Max task name length

```
#define configUSE_PREEMPTION            1
#define configCPU_CLOCK_HZ              ( ( unsigned long ) 16000000ul )
#define configTICK_RATE_HZ              ( ( portTickType ) 500ul )
#define configMAX_PRIORITIES            ( ( unsigned portBASE_TYPE ) 3 )
#define configMINIMAL_STACK_SIZE        ( ( unsigned short ) 250 )
#define configTOTAL_HEAP_SIZE           ( (size_t ) ( 1200 ) )
#define configMAX_TASK_NAME_LEN         ( 8 )
#define configUSE_COUNTING_SEMAPHORES     1
#define configUSE_MUTEXES                 1
```

define configUSE_TIME_SLICING

By default configUSE_TIME_SLICINGis defined as 1

So,FreeRTOS uses prioritized preemptive scheduling with time slicing. That means the RTOS scheduler will always run the highest priority task that is in the Ready state, and will switch between tasks of equal priority on every RTOS tick .

If configUSE_TIME_SLICING is set to 0

then the RTOS scheduler will still run the highest priority task that is in the Ready state, but will not switch between tasks of equal priority just because a tick interrupt has occurred.

- **configUSE_PREEMPTION**

  -activate the “Co-operative /Non-pre-emptive” kernel if it is set by “**0**”
  or activate the “Pre-emptive” kernel if it is set by “**1**”.

- **configCPU_CLOCK_HZ**

  - It is used to set your system frequency that the CPU will be run.

- **configTICK_RATE HZ**

  - It is used to determine how many ticks that the scheduler will be run per second, so the tick time or “OS Time” will be “$\text{OS Tick} = \frac{1}{TICK\ RATE}$” .

- **configMAX_PRIORITIES**

  - It is used to determine the maximum priority that will be used in the system, and the highest priority will be “MAX_PRIORITY – 1”.

- **configMINIMAL_STACK_SIZE**

  - It is used to determine the minimum stack size of the idle task

- **configTOTAL_HEAP_SIZE**

  -It is used to determine how many words that the freeRTOS will dynamiclly allocate into the heap memory during runtime for TCBs and tasks execution.

- **configMAX_TASK_NAME_LEN**

  -It is used to determine the maximum characters that task name will take including the NULL terminator ‘\0’.

- **configUSE_COUNTING_SEMAPHORES**

  -It is used to determine the if you need to use the counting semaphore “1” or not “0”, it will be declared later.

- **configUSE_MUTEXES**

  -It is used to determine the if you need to use the mutex semaphore “1” or not “0”, it will be declared later.

For more configurations, you can visit:

freertos.org/a00110.html

The Idle Task in FREERTOS

The idle task is created automatically when the RTOS scheduler is started to ensure there is always at least one task that is able to run. It is created at the lowest possible priority to ensure it does not use any CPU time if there are higher-priority application tasks in the ready state.

The idle task is responsible for freeing memory allocated by the RTOS to tasks that have since been deleted. It is therefore important in applications that make use of the vTaskDelete() function to ensure the idle task is not starved of processing time. The idle task has no other active functions so can legitimately be starved of microcontroller time under all other conditions.

It is possible for application tasks to share the idle task priority (tskIDLE_PRIORITY).

Idle Task hook in FREERTOS

An idle task hook is a function that is called during each cycle of the idle task. If you want application functionality to run at the idle priority then there are two options:

- Implement the functionality in an idle task hook.

There must always be at least one task that is ready to run. It is therefore imperative that the hook function does not call any API functions that might cause the idle task to block (vTaskDelay(), or a queue or semaphore function with a block time, for example). It is ok for co-routines to block within the hook function.

- Create an idle priority task to implement the functionality.

This is a more flexible solution but has a higher RAM usage overhead.

Idle Task hook in FREERTOS

To create an idle hook:

Set configUSE_IDLE_HOOK to 1 in FreeRTOSConfig.h.

Define a function that has the following name and prototype:

void vApplicationIdleHook(void);

* It is common to use the idle hook function to place the microcontroller CPU into a power-saving mode.

TASK CONTROL APIS AT FREERTOS

- ✓ vTaskDelay()
- ✓ vTaskDelayUntil()
- ✓ uxTaskPriorityGet()
- ✓ vTaskPrioritySet()
- ✓ vTaskSuspend()
- ✓ vTaskResume()
- ✓ xTaskResumeFromISR()
- ✓ xTaskAbortDelay()

UBaseType_t uxTaskPriorityGet(TaskHandle_t xTask);

- INCLUDE_uxTaskPriorityGet
  must be defined as 1 for this function to be available. See the RTOS Configuration documentation for more information.
  Obtain the priority of any task.

- Parameters:
  xTask Handle of the task to be queried. Passing a NULL handle results in the priority of the calling task being returned.
- Returns:
  The priority of xTask.

void vTaskPrioritySet(TaskHandle_t xTask,
UBaseType_t uxNewPriority);

- INCLUDE_vTaskPrioritySet
  must be defined as 1 for this function to be available. See the RTOS Configuration documentation for more information.
  Set the priority of any task. A context switch will occur before the function returns if the priority being set is higher than the currently executing task.

- Parameters:
  xTask Handle of the task whose priority is being set. A NULL handle sets the priority of the calling task. uxNewPriority The priority to which the task will be set.Priorities are asserted to be less than configMAX_PRIORITIES. If configASSERT is undefined, priorities are silently capped at (configMAX_PRIORITIES - 1).

void vTaskSuspend(TaskHandle_t xTaskToSuspend);

- INCLUDE_vTaskSuspend

must be defined as 1 for this function to be available. See the RTOS Configuration documentation for more information. Suspend any task. When suspended a task will never get any microcontroller processing time, no matter what its priority. Calls to vTaskSuspend are not accumulative - i.e. calling vTaskSuspend () twice on the same task still only requires one call to vTaskResume () to ready the suspended task.

- Parameters:

xTaskToSuspend Handle to the task being suspended. Passing a NULL handle will cause the calling task to be suspended.

void vTaskResume(TaskHandle_t xTaskToResume);

- INCLUDE_vTaskSuspend
  must be defined as 1 for this function to be available. See the RTOS Configuration documentation for more information.Resumes a suspended task.A task that has been suspended by one or more calls to vTaskSuspend () will be made available for running again by a single call to vTaskResume ().

- Parameters:
  xTaskToResume Handle to the task being readied.

void vTaskDelete(TaskHandle_t xTask);//its not Task control it's Task Creation

- INCLUDE_vTaskDelete
  must be defined as 1 for this function to be available. See the RTOS Configuration documentation for more information.
  Remove a task from the RTOS kernels management. The task being deleted will be removed from all ready, blocked, suspended and event lists.
  NOTE: The idle task is responsible for freeing the RTOS kernel allocated memory from tasks that have been deleted. It is therefore important that the idle task is not starved of microcontroller processing time if your application makes any calls to vTaskDelete (). Memory allocated by the task code is not automatically freed, and should be freed before the task is deleted.

- Parameters:
  xTask The handle of the task to be deleted. Passing NULL will cause the calling task to be deleted.

FREERTOS APIs EXERCISE :

Now try to implement programs using thos APIs

THANK YOU