DSA REPORT
Review- 1
(Analysis of Free RTOS source code)

By SORTING SPARTANS

AMRITA VISHWA VIDYAPEETHAM
DEEMED TO BE UNIVERSITY

1.	Harecharan S	CB.EN.U4CSE21450
2.	Hari Kesav D	CB.EN.U4CSE21421
3.	M.K.ENIYAN	CB.EN.U4CSE21435
4.	Mitesh.V	CB.EN.U4CSE21438
5.	Rajkumar R	CB.EN.U4CSE21446

What is a Ready list ?

In RTOS (Real time operating system) , A ready list or ready queue is a Data structure which stores the threads or tasks which are ready to be executed.

Ready list or ready queue is used to manage the tasks or threads which are waiting for the CPU to execute them (i,e) the tasks which are ready to run.

The tasks stored in ready list are usually sorted by RTOS based on their priority , The task with the highest priority is placed at the front of the list.

RTOS scheduler executes these tasks in the same order as it was sorted in the list , once a task is completed or it is no longer ready to run , it is removed from the ready list.

The ready list is DYNAMIC as it is constantly changing because of new addition of tasks or removal of tasks.

Ready list determines the order of execution in RTOS , hence it plays a key role.

Underlying data structure in Ready list :

The ready list in the RTOS is made up of array of doubly linked lists , where each element of the array is a doubly linked list .

Each doubly linked list are TCBs (Task Control Block).

The TCB holds regarding state of task , priority , stack pointer , program counter & other data required for context switching & task scheduling

Array :

It is of length 10 , with index ranging from 0 to 9 , the index zero holds the least priority & the index 9 holds the maximum priority. So basically the array consists of 10 doubly linked lists which are TCBs of RTOS.

Inside each index there exists a doubly linked list of infinite length , which means there can be N no. of list items.

Doubly linked list :

Each element of array is a doubly linked list , Each node of linked list is a task control block (TCB) , it contains pointer to the next node & pointer the previous node.

TIME COMPLEXITY :

For accessing an element , The time complexity is O(1). Because we can access the elements by their index , accessing the elements from array of doubly linked list takes constant time.

For insertion of an element at beginning or at the end , the time complexity will be O(1) , and for insertion at a specific position , the time complexity will be O(n).

For searching of an element , the time complexity will be O(n) , Because we have to traverse all the elements in the list from head to tail

For deletion of an element , the time complexity will be O(1) , since accessing the element takes constant time , we delete the element easily .

The traversing operation will have time complexity of O(N) , because there are N no. of nodes.

MAIN FUNCTIONS TO BE DISCUSSED:

BEFORE ANALYSING AND UNDERSTANDING WHAT THESE CODES DO , WE MUST HAVE A BASIC UNDERSTANDING OF THE STRUCTURE DATATYPE AND ITS MEMBERS:

The code snippet you provided defines a structure named **tskTaskControlBlock** or **tskTCB**. This structure represents the control block or control structure for a task in a multitasking environment, such as the FreeRTOS real-time operating system.

Let's go through the members of the **tskTaskControlBlock** structure:

1. **pxTopOfStack**: It is a pointer to the location of the last item placed on the task's stack. It is typically used during context switching.
2. **xEventListItem**: It is another **ListItem_t** structure used to reference the task from an event list.
3. **uxPriority**: It represents the priority of the task. A lower value indicates a lower priority, and 0 is typically the lowest priority.
4. **pxStack**: It is a pointer to the start of the task's stack. The stack is where the task's context, including local variables and return addresses, is stored.

1) vTaskSwitchContext :

ARGUMENTS: VOID

1. MAINLY USED IN vTaskSuspend function used to suspend a task. This function handles the context switching and adjustment of **pxCurrentTCB** (current task) based on whether the task being suspended is the currently running task or a different task.

- If the currently running task is being suspended (**pxTCB == pxCurrentTCB**), it checks if the scheduler is running (**xSchedulerRunning**). If it is running, it yields within the API (allowing other tasks to run), and if it is not running, it adjusts **pxCurrentTCB** to point to another task.

- If a different task is being suspended (**pxTCB != pxCurrentTCB**), it checks if the scheduler is running and resets the next expected unblock time if necessary.

Overall, **vTaskSuspend** suspends a specified task or the calling task and performs the necessary operations to update the task lists and manage task switching if required.

2) prvAddTaskToReadyList :

ARGUMENTS: pxTCB – pointer to the member of the structure tskTCB.

prvAddTaskToReadyList is a utility for adding a task control block (TCB) to a ready list in a task scheduling system. It likely performs some additional actions such as tracing, recording priority, and inserting the TCB into the appropriate list. It has sub functions like

1. **traceMOVED_TASK_TO_READY_STATE**: Traces or logs the transition of a task to the ready state.
2. **taskRECORD_READY_PRIORITY**: Records or updates the priority of a task that has become ready.
3. **vListInsertEnd**: Inserts an item (such as a task control block) at the end of a linked list.

3) prvRemoveTaskFromReadylist :

There is no separate function to remove tasks from the ready list like **prvRemoveTaskToReadyList(pxTCB)** instead, uses **uxListRemove**

To remove a process from ready queue, so we have to mention the structure and a pointer to the address that we want to remove as parameters

4) uxTopReadyPriority :

In the free RTOS code it is a built-in variable and an unsigned integer which is used in storing the highest priority of any task that is ready to run where the next task to be executed is also determined by uxTopReadyPriority and used by the function “vTaskSwitchContext()”. When a task is created or preempted due to higher priority task this variable gets updated when added to ready list. It plays a crucial part in scheduling because it has to ensure highest priority tasks are executed first and should prevent lower priority task from running when higher priority task is running. Its value typically ranges from 0 to (configMAX_PRIORITIES – 1). The scheduler can efficiently identify highest-priority task without the need to iterate through all the tasks in system which improves performance and responsiveness of task scheduling process.

5) pxCurrentTCB :

This variable in free RTOS represents/acts like a pointer to currently executing task in Task Control Block(TCB). It is typically used by the kernel to keep track of currently executing task in other words provides access to TCB of currently running task on the CPU. The function vTaskSwitchContext() uses it to switch from one task to another, to get the stack pointer of currently running task, uses the stack pointer to update the context of the next task to run. It is also used by the function vTaskSuspend() and vTaskResume() which are responsible for the tasks to be suspended or resumed. They use this variable to get the TCB of the task to be suspended or resumed, using the same to update the status of the task’s state. This is a critical part of the task scheduler which allows to keep track of the currently running task.

Conclusion :

Finally in the end of this report we can understand what is a ready list, how the Ready Task List is stores the task, what kind of Data Structure is used to implement it and the time complexity of this task list. The members required for this task list are pxTopOfStack, xEventListItem, uxpriority, pxStack to work accordingly and the functions which help in add the functionalities to this task list are vTaskSwitchContext, prvAddTaskToReadyList, prvRemoveTaskFromReadyList, uxTopReadyPriority, pxCurrentTCB.

We get to know how the task is inserted, deleted, how they are switched, and recognize the high priority task and execute accordingly.