Analysis of FreeRTOS Source Code:
Implementation of the Ready List in the Scheduler

By:

S.No	Name	Roll No.
1	AdwaitaSathrukkan	CB.EN.U4CSE21403
2	Bapathi Archana	CB.EN.U4CSE21410
3	GollaThanuja	CB.EN.U4CSE21419
4	Harish S	CB.EN.U4CSE21422
5	Sivakami V	CB.EN.U4CSE21456

Introduction

An RTOS (Real-Time Operating System) runs multiple sequential programs, called tasks, concurrently. It controls, or schedules, the execution of the tasks based on their priorities so that required work associated with input events to the system will be performed within specified periods. A handler is a special class of task that runs on an event, such as an external signal or timer expiration. We call tasks and handles kernel objects. Here are the key aspects and significance of FreeRTOS in real-time embedded systems:

1. Real-Time Capabilities: FreeRTOS is designed to handle real-time requirements, where tasks must meet strict timing constraints. It provides a preemptive scheduler that allows tasks to have fixed priorities and guarantees that higher-priority tasks can interrupt lower-priority tasks when necessary. This ensures that critical tasks are executed in a timely manner.
2. Small Footprint: FreeRTOS is designed to be highly efficient in terms of memory usage and processing overhead. It has a small code footprint, typically a few kilobytes, making it suitable for resource-constrained embedded systems with limited RAM (Random Access Memory) and ROM.
3. Task and Resource Management: FreeRTOS provides a task management mechanism that allows developers to create and schedule tasks with different priorities. It also offers synchronization primitives like semaphores, mutexes, and queues to facilitate inter-task communication and coordination. These features enable efficient multitasking and resource sharing in embedded systems.

Overall, FreeRTOS plays a significant role in real-time embedded systems by providing an efficient and portable operating system that enables developers to build responsive, deterministic, and reliable applications for resource-constrained devices.

The purpose of this report was to analyze the FreeRTOS Source code, understand its implementation of it in real-time embedded systems and understand the significance of ready list data structure for task scheduling. eventually presenting an overview of its adeptness in real-time applications and further evaluate potential areas for improvement in FreeRTOS's performance output.

This report discusses about:

Overview of FreeRTOS Scheduler

The FreeRTOS scheduler is a fundamental component of the FreeRTOS operating system, responsible for managing and scheduling tasks in real-time embedded systems. It ensures that tasks are executed efficiently and by their priorities and timing requirements. Here is an overview of the FreeRTOS scheduler:

1. Preemptive Scheduling: FreeRTOS uses a preemptive scheduling algorithm, allowing higher-priority tasks to interrupt lower-priority tasks. This ensures that critical tasks are executed promptly, even if lower-priority tasks are currently running. Preemptive scheduling is essential in real-time systems to meet strict timing constraints.
2. Task Priorities: Each task in FreeRTOS is assigned a priority value, typically ranging from 0 to the maximum priority supported by the system. The scheduler uses these priorities to decide which task should run when multiple tasks are ready to execute. Higher-priority tasks preempt lower-priority tasks.
3. Task States: FreeRTOS tasks can exist in different states, including "Running," "Ready," "Blocked," and "Suspended." The scheduler supports a ready list of tasks that are ready to run but are waiting for their turn. Tasks can block themselves, waiting for events or resources, and the scheduler moves them to the blocked state until the blocking condition is satisfied.
4. Context Switching: When the scheduler determines that a higher-priority task needs to run, it performs a context switch. A context switch involves saving the state of the currently running task, including its program counter and register values, and restoring the state of the higher-priority task. This allows the higher-priority task to resume execution seamlessly.
5. Task Scheduling Algorithm: FreeRTOS uses a priority-based preemptive scheduling algorithm. When multiple tasks with the same priority are ready to run, FreeRTOS employs a round-robin scheduling approach, where tasks of equal priority are given equal time slices. This ensures fairness among tasks of the same priority level.
6. Idle Task: FreeRTOS includes an idle task that runs when no other tasks are ready to execute. The idle task has the lowest priority and performs minimal processing, allowing the system to conserve power when there is no useful work to be done.

Overall, the FreeRTOS scheduler is responsible for managing task execution, prioritization, and context switching in real-time embedded systems. Its efficient scheduling algorithm and preemptive nature ensure that critical tasks meet their timing requirements, making it a valuable component in the development of responsive and deterministic applications.

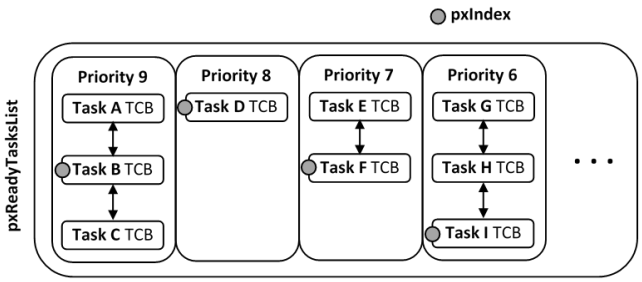
Understanding the Ready List

The Ready List is an essential component of the FreeRTOS scheduler, which maintains a list of tasks that are ready to execute. This list plays a crucial role in task scheduling by allowing tasks in the "ready" state to organize based on their priority levels. By doing so. The scheduler can select the next task for execution efficiently and optimize resource allocation and task scheduling. The primary purpose of the Ready List is to ensure efficient task scheduling. Whenever a task completes its execution or becomes ready to run due to an event or available resource it is added to the Ready List according to its priority. The scheduler then selects the highest priority task from this list for execution ensuring that important tasks are executed promptly. The Ready List allows quick identification and dispatch of critical tasks. Improving overall system performance and responsiveness. Moreover the Ready List manages different states of tasks such as "ready " "blocked," and "running." When a task is in a "ready" condition it waits for CPU time assigned by the scheduler for execution. In this case the Ready List helps manage transitions between these states efficiently. In summary. Through maintaining data about each tasked stored in order like this makes sure only urgent works are done first hence optimizing system performance with efficiency got from Centralised structure that holds all executed work available for prompt dispatch version 1 describes what an essential aspect ready list has been on FreeRTOS designed system aiding efficient Task management ensuring prompt response rate. When referring to computer language, running state signifies that specific moment in time when CPU executes a particular task. To facilitate smooth transitioning of these tasks between running and ready states, computational systems use Ready Lists. Whenever an operation finishes working or deliberately surrenders control over CPU, it re-enters into Ready List to be executed at any available opportunity in future.

Analyzing the Implementation of the Ready List

In FreeRTOS's operation system, there exists an imperative component known as Ready List - responsible for regulating task scheduling processes using various data structures and functions discussed below: vTaskSwitchContext, prvAddTaskToReadyList, prvRemoveTaskFromReadyList, uxTopReadyPriority, and pxCurrentTCB.
The paramount data structure utilized to develop this indispensable feature in Freertos stems from a linked list concept comprising numerous Task Control Blocks(TCBs). Each TCB holds essential details concerning individual tasks-like priorities, state or context-which then stitched together into an unidirectional chain sequence- forming what is called singly-linked-list or READY LIST; where order is maintained based on priority; with the most significant task holding position at the list's helm. In this essential configuration lies a highly efficient task selection channel for the scheduler to function optimally.

In FreeRTOS, the ready list is implemented using an array of doubly-linked lists. The array is called **pxReadyTasksLists** and is defined in tasks.c. Each element in the array represents a priority level, with the highest priority at index 0 and the lowest priority at the highest index.



When the scheduler is running, the method selects the first TCB that is located at the highest index (i.e., highest priority). The first element is picked when there are many elements in the circular doubly-linked list of highest priority. A round-robin system is used for this. The highest priority task in the ready list is tracked using the uxTopReadyPriority variable. When a job is ready to run, the prvAddTaskToReadyList function adds it to the appropriate linked list based on its priority. The job with the highest priority is then selected by the scheduler from the list of tasks that are available for switching. The ready list implementation in FreeRTOS is designed to be efficient and scalable, allowing for the management of large numbers of tasks with varying priorities. The use of linked lists allows for constant-time insertion and deletion of tasks in the ready list, while the array allows for constant-time lookup of the highest priority task.
Overall, the ready list implementation in FreeRTOS is a key component of the operating system's task scheduling mechanism and is critical to the efficient management of tasks in real-time applications.

The choice of an circular doubly-linked lists to implement the ready list in FreeRTOS has several advantages in terms of performance and memory usage.

=>Firstly, it offers flexibility that allows tasks to be easily added or removed without complicated reorganization. This feature is particularly helpful in efficiently handling tasks transitioning between different states and ensures that the Ready List can dynamically adjust its size according to the number of tasks without any hassle.

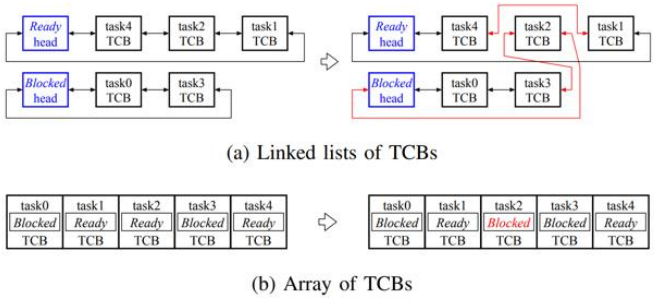
=>Secondly, the linked list system enables efficient insertion and removal of tasks in the Ready List where functions like prvAddTaskToReadyList and prvRemoveTaskFromReadyList are performed in constant time resulting in fast task insertion and removal with minimum overhead.

=>Thirdly, Fast task dispatching is possible due to the linked list structure of the Ready List. As a result, response time is reduced, which improves system performance since uxTopReadyPriority function retrieves highest priority tasks from Ready List by accessing the first task based on priority order.

=>Fourthly, memory usage is efficient because each TCB in the ready list only occupies space required for necessary task related information which reduces memory wastage. This feature is especially helpful in resource constrained embedded systems where efficient memory utilization is crucial.

=>Lastly, A linked list data structure combined with specific well defined individual functions like vTaskSwitchContext has made it easy to achieve simplicity and modularity while implementing Ready Lists since each function can be extended or modified independently for specific purposes.Great advantages come with using the linked list implementation of the Ready List function in conjunction with important functions like vTaskSwitchContext, prvAddTaskToReadyList, prvRemoveTaskFromReadyList, uxTopReadyPriority, pxCurrentTCB. An optimal arrangement for task scheduling as well as resource management is achieved due to its modular design which simplifies understanding of bugs and maintenance. Flexible options for memory utilization are provided at high speeds for task dispatching alongside insertion/removal operations that are efficient resulting in performance optimization. This means that FreeRTOS users can be certain on Fast management effectively because of these attributes uniquely inherent in this approach.

Detailed Examination of Ready List Operations



(a) Linked lists of TCBs

(b) Array of TCBs

(Data structure for task management)

FreeRTOSs' Ready List plays a vital role in managing tasks through operations such as task insertion, removal, and prioritization. These operations are carried out by functions like `vTaskSwitchContext`. `prvAddTaskToReadyList`. `prvRemoveTaskFromReadyList`. `uxTopReadyPriority`. And `pxCurrentTCB`.

Lets explore these functions in detail. Evaluating their pseudocode implementation, time complexity and impact on the efficiency of the scheduler.

1) The function responsible for initiating the context switch between tasks is called `vTaskSwitchContext`. It saves the context of the current task and selects the highest priority task from the Ready List to be scheduled next. A simplified pseudocode representation of this operation would be as follows:

   vTaskSwitchContext()
   Save current task using pxCurrentTCB
   Update current task state
   Select highest priority task from Ready List using uxTopReadyPriority
   Set selected task as the new current task
   Restore context of selected task and resume execution

2) prvAddTaskToReadyList` is responsible for inserting tasks into the Ready List based on priority when it becomes ready to run. The simplified pseudocode below exemplifies how this operation operates:

   prvAddTaskToReadyList(task)
   if Ready List is empty set 'task' as head of list

else
Iterate through list
Find correct position based on priority order
Insert 'task' at correct position within list

3) To remove a task from the Ready List when it changes to a blocked or running state we use `prvRemoveTaskFromReadyList` function. This function iterates through the Ready List and removes any matching tasks encountered. The pseudocode for this process is shown below.
prvRemoveTaskFromReadyList(task)
Iterate through the tasks in the list
If there is a match for task being removed
Eliminate that task from the list
Adjust pointers to maintain linked list structure
Break out of loop
4) Like `prvAddTaskToReadyList`. For retrieving highest priority tasks priority we use `uxTopReadyPriority`. It retrieves priority from head pointer directly without iterating which gives constant or predictable response times with respect to increase in number of tasks present.
uxTopReadyPriority()
Retrieve the head of the Ready List
Return the priority of the task at the head of the list
5) pxCurrentTCB: Finally current executing tasks control block (TCB) is accessed via using pointer `pxCurrentTCB`. Central to any successful operating system is the ability to switch between running tasks without significant delay or errors that can impact system stability. Within FreeRTOS this functionality is supported through use of the internal pointer known as `pxCurrentTCB` which tracks each tasks' status as it runs. Although specifics may vary based on underlying design choices, careful management of this reference point enables smooth task switching within the scheduler.

To ensure that the system can react quickly to changes in task states, these operations, which are essential to the management of tasks in real-time applications, must be carried out effectively. As tasks are ready or blocked for resources, operations for task insertion and removal are frequently carried out. Due to the current implementation's constant time complexity of O (1) for these operations, their negative effects on the scheduler's performance are kept to a minimum. Work prioritisation determines which jobs should be completed first, which affects the effectiveness of the scheduler. Due to O (1)'s constant time complexity in the existing implementation, the scheduler can quickly identify which job has the greatest priority. The quantity and frequency of the jobs in the system must be taken into accountto fully understand the effect these activities have on the scheduler's efficiency.

The effectiveness of task insertion and removal operations becomes more crucial in a system with lots of tasks. The system would become slower and less responsive as the number of tasks rose if these operations had a time complexity of O(n), where n is the total number of tasks. The current ready list implementation in FreeRTOS ensures that task insertion and removal operations remain effective even as the number of tasks in the system increases because it has a constant time complexity of O (1). With more tasks in the system, task prioritisation becomes even more crucial. The scheduler would take longer as the number of tasks increased to choose the task with the highest priority if task prioritisation had an O(n) time complexity. Overall, the performance of the scheduler is directly impacted by how efficiently FreeRTOS has optimised the time complexity of task insertion, removal, and prioritisation operations. FreeRTOS is able to deliver timely and effective task management for real-time applications by making sure that these operations continue to be effective even as the number of tasks in the system increases.

Integration of the Ready List with the Scheduler:

The Ready List is an important feature of the scheduler in a multi tasking operating system.The Ready List operates by keeping an accurate record of all executable tasks. This information is useful for the scheduler when selecting which task should take precedence over others during execution. By ensuring that tasks are completed quickly and effectively, the ready list interacts with other elements of the scheduler, such as context switching with **vTaskSwitchContext** and task preemption. The ready list prioritises tasks according to their priority level, chooses the next task based on priority, and makes sure the highest priority task is always in progress. For real-time applications where timely task execution is critical for meeting performance requirements, the interaction between the ready list and other scheduler components is crucial. FreeRTOS provides synchronization mechanisms such as **mutexes** and semaphores to ensure thread safety and prevent race conditions in real-time embedded

systems. Mutexes provide mutual exclusion to shared resources, allowing tasks to acquire exclusive access to a resource while blocking other tasks. **Semaphores** control access to shared resources by allowing tasks to wait for resource availability. FreeRTOS also offers other synchronization mechanisms like queues, event groups, and task notifications. Queues enable thread-safe communication between tasks, while event groups synchronize tasks based on events. Task notifications allow tasks to signal events to each other.
To prevent priority inversion, FreeRTOS incorporates algorithms such as priority inheritance and priority ceiling. These synchronization mechanisms and algorithms ensure that tasks can safely access shared resources and communicate with each other without interfering or causing race conditions. By providing thread safety and preventing priority inversion, FreeRTOS effectively manages tasks in real-time applications, enabling timely and efficient execution of critical operations.

Performance Evaluation:

For any schedule there is nothing more pertinent than a highly optimized Ready List mechanism to help improve overall system performance significantly. Therefore, evaluating performance characteristics within this article focuses on comprehensively capturing any potential limitations associated with implementing an effective Ready List implementation approach.

Efficiency plays a critical role in setting up an evaluation criterion as it includes assessing how well various operations get conducted without depleting resources at our disposal fast. Adding new tasks or removing them through conceptually arranging them in-respect-to Priority shapes a large part of efficiency assessment parameters. Nevertheless, the Glad List's efficacy primarily depends on specific data structure selections made to support underlying operations in maintaining and managing different tasks. For example, linked lists offer optional robust search tools through updating point-based mechanisms for repeating items that aren't as scalable when dealing with extensive lists of high priority assignments. Schedulers rely on using efficient data structures like Priority Queues or Heaps, enhancing performance by providing speedy ways of generating Nodes particularly important in cases where there is a need to furnish swift insertion or removal assistance, especially with items having higher-ranking priorities. On the flip side optimizing performance using these structures comes with increased memory requirements; hence they're not always suited for all systems preferences.
The term Scalability refers to how relevant a ready list mechanism can be improved without suffering significant strains on its current capabilities and functionalities over time while retaining its original efficiency levels. The scalability performance evaluation employs some of the same parameters already used when developing efficiencies analysis criteria but goes further identifying new tools that can optimize ongoing uprightness of mechanism effectiveness under ongoing future system environment e.g., number of new tasks loaded/minute.Linked listing solutions offer natural predictions for scalability outcomes. Using linked lists makes it possible to add various nodes (inputs) without experiencing delays typically related to slower search mechanisms while concurrently leveraging fast returns concerning high-priority assignments; however, accessing items having the highest priority will typically become hard as amounts assign increases over time.

To mitigate linked lists' given scalability limitations, multi-level comments combined with multiple ready-list applications provide critical pathways towards better growing assigned tasks management performances optimally. One effective way of handling a large number of tasks is using techniques that categorize them into multiple lists based on their importance or features. For optimal performance of other scheduler components Resource consumption by the Ready List and its effect should be kept at miminal levels. The amount consumed depends mainly on synchronization mechanisms used with spinlocks often incurring significant overhead during heavy task conflicts- so schedulers are now turning towards lock-free data implementations reader-writer locks to minimize this.
To perform optimally without compromise, there are still potential limitations that could affect a scheduling systems efficiency in relation to ready list implemention.Priority inversion- experienced when low-priority tasks hold locks required by high-priority tasks, causing needless delays-could affect how a system prioritizes its tasks. Schedulers have included priority inheritance or priority ceiling protocols to tackle this issue. Also, starving low-priority jobs of Ready List resources should be avoided so that all jobs can finish in good time- to avoid starvation, schedulers use techniques such as aging or priority boosting. Cache coherency affects performance when different cores access the Ready List - it is crucial to manage this by using techniques such as cache line padding or cache colouring.

Conclusion

After applying DSA techniques to the FreeRTOS source code. Many significant insights and discovery have emerged. These findings offer a comprehensive understanding of its capabilities and potential applications while illuminating the

underlying design principles and implementation strategies employed in FreeRTOS. The following summary summarizes these insights effectively:

=>Efficient task scheduling
=>Synchronization mechanisms
=>Resource management
=>Interrupt handling
=>Algorithmic optimizations
=>Modularity and Extensibility

In summary, the analysis provides a comprehensive overview of the implementation of the Ready List in FreeRTOS, emphasizing its critical role in managing tasks and their states. The analysis also highlights the importance of thread safety, preventing race conditions, and task prioritization in FreeRTOS. Overall, the implementation of the Ready List in FreeRTOS is optimized for efficiency and provides timely and efficient task management for real-time applications.

Overall, the Ready List is a critical component of the scheduler in FreeRTOS and plays a significant role in achieving efficient task scheduling and management. By prioritizing tasks based on their priority level, ensuring fairness in task scheduling, and using synchronization mechanisms to ensure thread safety, the Ready List enables the system to provide timely and efficient task management for real-time applications.

Studying the FreeRTOS source code from a DSA perspective reveals valuable insights into its design choices, efficient utilization of data structures and algorithms as well as optimization techniques. Therefore. FreeRTOS is an essential resources for developers and engineers working on real time applications

References

=> https://www.freertos.org

=> A comprehensive article on FreeRTOS titled "FreeRTOS: An Open Source Real Time Operating System for Embedded Systems" written by Richard Barry is published on IEEE Xplore. The article can be accessed at [https://ieeexplore.ieee.org/document/7058882]

=>Full Hardware Implementation of FreeRTOS-Based Real-Time Systems | IEEE Conference Publication | IEEE Xplore

=>What is the complexity of the scheduler of the FreeRTOS? - Stack Overflow

=> “Performance Evaluation of the FreeRTOS Real Time Operating System” authored by N. Gökçe et al.

=> "A Comparative Study on Scheduling Algorithms in FreeRTOS. " V. Shanthi and M. Murugan compare multiple scheduling algorithms on FreeRTOS, including round robin, earliest deadline first (EDF). And deadline monotonic (DM). This study is available on IEEE Xplore.

=> "Performance Analysis of Real Time Operating Systems: A Comparative Study" by H. Özkan and B. Özkan.

=> Richard Barry has written an informative tutorial guide titled "Mastering the FreeRTOS Real Time Kernel: A Hands On Tutorial Guide".