ANALYSIS OF FREERTOS SOURCECODE AND THE IMPLEMENTATION OF READY LIST

TEAM:ERROR 404

CB.EN.U4CSE21407-ASHVABA

CB.EN.U4CSE21417-ANANYA ELA

CB.EN.U4CSE21429-KARTHIK D

CB.EN.U4CSE21452-SAKTHI B

CB.EN.U4CSE21457-SIVARAJKUMAR.S

INTRODUCTION

This report documents the understanding of the FREERTOS,which is an operating system(os) for low energy and resource microcontrollers.This plays a significant role in embedded systems as it acts as a real time operating system for these systems and microcontrollers.This plays a major role in scheduling,resource management,interrupt handling etc in these embedded systems.

The purpose of this report is to analyse the FREERTOS source code and understand the implementation of the ready list in the FREERTOS scheduler and its importance in the scheduling process .

This report comprises of several sub topics such as overview of freertos which would provide a high level explanation about the FREERTOS,following that is the concept of what a ready list is and its functions,adding to that is the explanation about the implementation of the ready list in FREERTOS and the data structures used to implement the same.Then,a detailed explanation of ready list operation such as prioritization,insertion,deletion etc in terms of scheduling and the efficiency of the scheduling is discussed about.Finally,the integration of the ready list with the scheduler is explained with how the ready list interacts with other components of the scheduler and its synchronization mechanism.

OVERVIEW OF FREE RTOS SCHEDULER

The FreeRTOS scheduler is a software component that manages the execution of tasks in a multitasking system. It is responsible for:

Determining which task should run next.

Scheduling the execution of tasks.

Suspending and resuming tasks.

Deleting tasks.

The scheduler is implemented as a kernel-level component, meaning that it runs in kernel mode and has well-constructed tenancy over the CPU. This allows the scheduler to preempt lower priority tasks with higher priority tasks, ensuring that the highest priority tasks unchangingly run first.

The Ready List is a data structure that contains all of the tasks that are ready to run. The scheduler maintains the Ready List in priority order, with the highest priority tasks at the top of the list. When the scheduler needs to segregate a task to run, it simply selects the task at the top of the Ready List.

The FreeRTOS scheduler is a powerful and flexible tool that can be used to manage the execution of tasks in a multitasking system. It is a hair-trigger component of any embedded system that uses FreeRTOS.

Here are some spare details well-nigh the FreeRTOS scheduler:

The scheduler is preemptive, meaning that it can interrupt a running task to run a higher priority task.

The scheduler supports a variety of scheduling policies, including round-robin, priority-based, and pearly scheduling.

The scheduler can be configured to run in a variety of modes, including cooperative, preemptive, and hybrid.

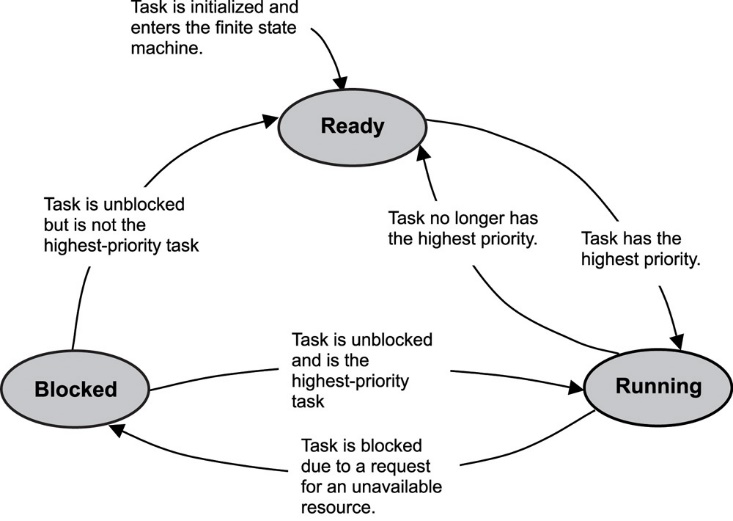
The scheduler provides a number of features for managing tasks, including task creation, task deletion, task suspension, and task resumption.

The FreeRTOS scheduler is a well-tested and reliable component that has been used in a wide variety of embedded systems. It is a valuable tool for developers who need to manage the execution of tasks in a multitasking system.

UNDERSTANDING THE READY LIST

Tasks that are ready to be executed are held in a data structure which is known as the ready list.These tasks are waiting for their turn to get executed by the CPU,which is scheduled and managed by the scheduler.This ready list plays a very significant role in the process of task scheduling. FreeRTOS allows us to set priorities for tasks, which allows the scheduler to preempt lower priority tasks with higher priority tasks. The ready list is a manager inside the operating system in charge of figuring out which task should run at each tick.

Every process gets executed in a cycle of different states which is namely the ready state,running state,blocked state.



READY STATE:The tasks which are all ready to be executed are said to be in the ready state. These tasks are waiting for their execution based on their priority.The task with the highest priority next moves into the running state.

RUNNING STATE:The process currently being executed by the processor is known to be in running state.This tends to stay in running state as long as it executes or moves to blocked state or back to ready state if it is waiting for any resource to proceed execution or if the next process is assigned the highest priority,this process will move to the ready state again.

BLOCKED STATE:The process is said to be in blocked state when it is waiting for another action for the current process to proceed execution,for example if a process requires a input from the user to be given to proceed further execution it has to wait until it receives it from the user,at this point of time the process will be moved from the running state to the blocked state.As soon as the process receives the necessary resource it will either move to ready state or the running state based on its priority of execution.

ANALYSING THE IMPLEMENTATION OF READY LIST

The Ready List in FreeRTOS is implemented as a doubly linked list.Thia data structure is used for some particular reasons for its performance and memory usage characteristics.

Doubly-linked lists are efficient for tasks that require frequent insertion and removal of elements. This is because the elements in a doubly-linked list can be inserted or removed at any point in the list without the need to shift any other elements. This makes them well-suited for implementing the Ready List, which is constantly being updated as tasks are created, deleted, and suspended.

Doubly-linked lists uses less memory than other data structures, such as arrays. This is because each element in a doubly-linked list only needs to store a pointer to the previous and next elements in the list. This can be a significant savings in memory, especially on embedded systems with limited memory resources

This data structure is chosen for a number of reasons, including:

Speed: Adding and removing tasks from a doubly linked list is a relatively fast operation. This is important for the scheduler, which needs to be able to quickly switch between tasks.

Determinism: The order of tasks in a doubly linked list is deterministic. This means that the scheduler will always choose the next task to run in the same order, regardless of how many times the scheduler is called. This is important for applications that require predictable behavior.

Memory usage: Doubly linked lists are relatively efficient in terms of memory usage. This is important for embedded systems, which often have limited memory resources.

Simple: The linked list is a simple data structure to understand and implement.

Efficient: The linked list is an efficient data structure for storing a list of elements that are ordered by priority.

Dynamic: The linked list is a dynamic data structure, which means that it can be easily resized as needed

There are some of the disadvantages in using this data structure lets see some of them:

More memory usage: A linked list uses more memory than some other data structures, such as an array. However, the additional memory usage is not significant for most applications.

Time: Insertion and deletion of tasks in a linked list is slower than in some other data structures, such as an array. However, this is not a significant issue for FreeRTOS because tasks are rarely inserted or deleted from the Ready List.

Space: The linked list requires more space than some other data structures, such as an array.

DETAILED EXAMINATION OF READY LIST OPERATIONS

**VTaskSwitchContext**

vTaskSwitchContext is a function that switches context from the current task to the next most important task. The time complexity of the vTaskSwitchContext is constant, O(1), because it involves updating some data structures and switching the processor context to the new task.

**prvAddTaskToReadyList**

The prvAddTaskToReadyList function adds a task to the ready list based on priority. Its implementation and time complexity are usually efficient because it takes O(1) time to perform a simple insertion in an ordered linked list. However, in the worst-case scenario where many jobs have the same priority, it may take O(n) time to find the correct position in the list, where n is the total number of jobs with the same rank which provides efficient work input for most events.

**prvRemoveTaskFromReadyList**

The prvRemoveTaskFromReadyList function removes a task from the ready list. Its implementation and time complexity are generally more efficient because they require updating of doubly linked lists, except O(1) time. However, in the worst-case scenario where the job to be removed is at the end of the list, it may take O(n) time to truncate the list, where n is the total number of jobs in the list for job removal leave well for most situations

**UxTopReadyPriority**

The first object uxTopReadyPriority is a variable that holds the priority of each ready task. Its effort is high because it is updated dynamically by the scheduler, and it takes a constant time (O(1)). This allows for quick identification of the most important task, facilitating efficient context switching. Flexibility is useful in a system with a large number of tasks, and avoids the need to redo all the work to determine the highest priority.

**pxCurrentTCB**

pxCurrentTCB is a pointer to the task control block (TCB) of the currently executing task. Its effectiveness is great because it provides direct information about the current work without the need for further analysis. It is a global variable in the kernel, making it easier to access project-specific data and properties. The use of pxCurrentTCB is usually limited to kernel-level code or low-level functionality. Modifying the pointer directly in application level code should be avoided.

INTEGRATION WITH THE SCHEDULER

The Ready List is a data structure that maintains a list of all the threads that are ready to run. The scheduler interacts with the Ready List in a number of ways, including:

Context switching: When the scheduler decides to switch to a new thread, it removes the current thread from the Ready List and adds the new thread to the Ready List.

Task preemption: If a higher-priority thread becomes ready to run, the scheduler may preempt the current thread and switch to the higher-priority thread.

To prevent race conditions, the scheduler must use synchronization mechanisms such as mutexes or semaphores.

Here are some of the synchronization mechanisms or algorithms used to ensure thread safety and prevent race conditions:

Mutexes: A mutex is a lock that can be acquired by one thread at a time. When a thread acquires a mutex, it has exclusive access to the data structure that the mutex protects.

Semaphores: A semaphore is a counter that can be incremented or decremented by threads. A semaphore can be used to control access to a shared resource. For example, a semaphore can be used to ensure that only a certain number of threads can access a database at the same time.

The choice of synchronization mechanism or algorithm depends on the specific requirements of the application. For example, if the application is multithreaded and performance is critical, then a lock-free algorithm may be the best choice. However, if the application is not multithreaded or performance is not critical, then a mutex or semaphore may be a better choice

Here are some additional considerations when choosing a synchronization mechanism or algorithm:

Complexity: Some synchronization mechanisms are more complex than others. For example, lock-free algorithms are more complex than algorithms that use locks.

Portability: Some synchronization mechanisms are more portable than others. For example, mutexes are more portable than semaphores.

Correctness: Some synchronization mechanisms are more correct than others. For example, lock-free algorithms are more correct than algorithms that use locks.

The choice of synchronization mechanism or algorithm should be made carefully to ensure that the application is thread safe and that race conditions are prevented.

CONCLUSION

The Ready List is a hair-trigger component of the FreeRTOS real-time operating system (RTOS). It is used to store tasks that are ready to run, and it is the primary data structure used by the scheduler to determine which task should run next.

The Ready List is implemented as a linked list of task tenancy blocks (TCBs). Each TCB contains information well-nigh a task, such as its priority, its state, and its stack pointer.

The scheduler maintains a separate Ready List for each priority level. When a task is created, it is widow to the Ready List for its priority level. When a task is suspended, it is removed from the Ready List. When a task is unsuspended, it is widow when to the Ready List.

When the scheduler is ready to select a task to run, it simply walks through the Ready List for the highest priority level that contains any tasks. The first task in the list is then scheduled to run.

The Ready List is a simple but constructive data structure that allows the scheduler to efficiently select tasks to run. It is a hair-trigger component of FreeRTOS, and it is essential for ensuring that tasks are scheduled in a timely and deterministic manner.

In wing to its role in task scheduling, the Ready List moreover plays a role in other aspects of task management, such as task deletion and task suspension. When a task is deleted, it is removed from all of the Ready Lists. When a task is suspended, it is removed from the Ready List for its priority level.

The Ready List is a well-designed and implemented data structure that is essential for the efficient operation of FreeRTOS. It is a key component of the RTOS scheduler, and it plays a role in many other aspects of task management.

Some concluding remarks or insights gained from studying the FreeRTOS source lawmaking include:

The Ready List is a simple but constructive data structure that is well-suited for the task of storing tasks that are ready to run.

The scheduler uses the Ready List to efficiently select tasks to run.

The Ready List is moreover used for other aspects of task management, such as task deletion and task suspension.

The Ready List is a key component of the FreeRTOS..