Exploring FreeRTOS

Scheduler's Ready List

Implementation

Team Name : Algosaints

Team Members :

CB.EN.U4CSE21413 – Charan Kumar A

CB.EN.U4CSE21416 – Dhanush S

CB.EN.U4CSE21418 – Gautham Suresh

CB.EN.U4CSE21425 – Joseph Jeffrey J

CB.EN.U4CSE21432 – Krishna Kumar S

Introduction :

FreeRTOS is an open-source real-time operating system. Designed for embedded systems. It has become a widely used kernel in industries such as automotive, aerospace, consumer electronics, and IoT devices. Due to its lightweight and portable nature, it offers an efficient task-scheduling system with synchronization primitives and memory management features suitable for resource-constrained microcontrollers and processors.

This report focuses on the implementation of the Ready List in the FreeRTOS scheduler which plays a vital role in task management and scheduling. The Ready List is a data structure responsible for holding references to tasks that are ready to execute. It organizes tasks based

on their priorities. Making it easier to select the highest-priority task for execution. This report aims to provide an in-depth analysis of how the Ready List is implemented in the FreeRTOS source code.

Our goal is to gain insights into how the FreeRTOS scheduler works internally by examining this specific aspect of its operation.

Our analysis will delve into the data structures and algorithms utilized to represent and manipulate the Ready List data structure. We will explore handling of various task states such as insertion and removal of tasks different scheduling policies support, among other aspects. Additionally. We will discuss how Ready List interacts with other essential components in the FreeRTOS kernel including context switching and synchronization primitives affecting overall performance.

Overview of FreeRTOS Scheduler :

In a real-time operating system like FreeRTOS, managing and scheduling tasks effectively plays a decisive role in its smooth operation. This responsibility falls squarely on its scheduler- an essential component responsible for determining which task executes on its processor at any given moment.

By use of pre-emption and priority-based scheduling algorithms, each task within this system boasts assigned values depending on importance levels such that whatever duty meets readiness criteria gets executed first according to its value. To make selections in multitasking scenarios within strict time limits, context switching between different duties occurs regularly thanks to periodic interruptions performed by this Scheduler.

One crucial piece of data structure prioritizing tasks efficiently available within this responsibility lies in what we call "the Ready List." In essence: It's an updated compilation containing all current duties set up for running whenever they fall due seemingly based on designated levels of importance per time while reflecting fairness by cycling through duties with similar values equally distributed before selecting only one among them as higher prioritized than others if allowed via their designations ensure getting a rounded opportunity to execute their roles in a fair and timely manner.

The Ready List plays a pivotal role within the FreeRTOS scheduler, making way for smooth task selection based on priorities. Furthermore, it prioritizes high-priority tasks to ensure their timely execution, thereby fostering deterministic behaviour in real-time systems.

Understanding the Ready List :

Efficiently managing tasks in an operating system depends significantly upon utilizing a data structure called a ready list, which tracks available and "ready" processes to run during scheduling cycles. Nodes within this linked list depict individual processes sorted by their designated priority that enables accessibility and orderliness for proper functioning at best capacity- starting with high priority first- by engaging whichever program proceeds itself at the 'top' of the queue. If interrupted by another program, then existing performing programs return to their initial place within said queue before continuing its cycle once more.

Regular use of such tasks enhances execution speed causing improvement across CPU performance whilst achieving seamless multi-tasking efforts- ultimately benefiting overall functionality.

One handy feature offered by modern-day computing systems involves using a tool known as "the ready list." Notably useful in preventing deadlocks, this technique tracks which resources different processes are currently employing - avoiding potential issues where two different operations require one shared resource simultaneously.

Through this method of organization comes increased performance, and better reliability in task scheduling - allowing efficiently managed operations to run seamlessly over extended periods with no hitch whatsoever.

Ultimately adopting "the ready list" assists in optimizing computer operation standards by enhancing technological excellence across multiple industries through greater proficiency - benefiting users in numerous ways.

Analyzing the Implementation of the Ready List :

In FreeRTOS, two main implementations are employed when creating its Ready List depending upon the configuration and optimization choices made during compilation. These are either using an array of task lists or a bitmap-based data structure.

The implementation using arrays with linked lists becomes ideal when there are limited numbers of known priority levels from which multiple tasks share similar priorities. It offers ease in accessing these tasks within their corresponding linked list owing to O(1) complexity leading to efficiency in insertions or deletions from those specific vertices.

The data structure used to implement the Ready List in FreeRTOS is typically an array of task lists.

In the implementation of an array of task lists, an array is used, where each index represents a priority level. Each priority level contains a linked list of tasks that have the same priority. This data structure allows for quick access to tasks of a particular priority and enables efficient insertion and removal operations.

Now, a Bitmap-Based Structure could also be used. In some configurations, FreeRTOS employs a bitmap-based structure to represent the Ready List. Here, a bitmap is used. If a bit is set, it indicates that there are tasks ready at that priority level. This approach reduces memory consumption. However, it requires additional processing to identify the highest-priority task during scheduling.

The choice of data structure depends on factors such as the number of priority levels, available memory, and desired performance.

It has several advantages. The main advantage is that the array-based implementation allows efficient insertion. It also takes care of its removal. It provides constant-time complexity for these operations, making it suitable for systems with a small number of priority levels.

Apart from this, the bitmap-based structure optimizes memory consumption. It results in reduced memory overhead compared to the array-based implementation.

In summary, the choice of data structure in FreeRTOS's Ready List implementation enhances performance and memory usage. The array-based implementation provides efficient task manipulation, while the bitmap-based structure optimizes memory consumption, especially for systems with a large number of priority levels.

Detailed Examination of Ready List Operations :

1. **vTaskSwitchContext:**

In operating systems or real-time kernels, the vTaskSwitchContext is used as it ensures that tasks are adequately switched during context changes. With this code, we check for the suspension of the scheduler. Following this, appropriate action is taken by declaring a respective flag.

1. **prvAddTaskToReadyList:**

This code defines a macro called `prvAddTaskToReadyList`. It is used to add a task to a ready list in a multitasking system. It includes tracing the task's transition to the ready state, recording the task's priority, and inserting the task into the appropriate ready list based on its priority.

1. **vTaskDelete:**

As part of its extensive range of offerings, FreeRTOS introduces vTaskDelete which is an essential feature designed for dynamic task handling during runtime while maintaining high performance. This incredible API allows developers to stop the execution of a given task in real time while ensuring that all associated processes are efficiently completed. Using vTaskDelete frees up essential system resources such as control blocks.

1. **uxTopReadyPriority:**

In FreeRTOS, `uxTopReadyPriority()` is a function that is used to determine the highest priority level of all the tasks that are in the ready state. Tasks with higher priority levels are given precedence over the rest. Thus, it uses a priority-based pre-emptive scheduling algorithm. Each task is initially assigned a priority level, a ready list is created and organized based on priority. The uxTopReadyPriority() scans through and identifies the task with the highest priority and returns it.

1. **pxCurrentTCB:**

The pxCurrentTCB represents the current thread’s context or thread control block. It is ideally used to track and manage the state of the current thread. It may include information about the current thread’s execution. In a multithreaded system, it can be used to manage concurrency and synchronize the threads.

Integration of the Ready List with the Scheduler :

For rendering easy coordination among interacting FreeRTOS platform's tasks/resources, synchronization techniques are fabricated with FreeRTOS innovation. There is an array of such techniques that enable task executables for exchanging materials with one another, easy coordination&exchange/generation/modification/fabrication/altering information amongst each other.

1. Mutexes, which are of key importance in this regard, are utilized to stop concurrent entry by various tasks into shared resources. Typically, Mutex is Got & released thereby ensuring exclusive control over shared resources.
2. Semaphores act as signalling/synchronization instrumentations between a myriad of threads/tasks. It can be counted as a binary and counting semaphore scenario that highlights availability (checking safety status available or not )&useful in Mutual exclusion type cases to render control over a specific operation
3. Queues which offer effective ways of message sharing help fuel holistic communication across varied processes/threads/tasks. They form integrated Communication patterns like Inter-Process Communication (IPC) & Inter Thread Communication (ITC).
4. Event flags work complementary with several Task Notifications. It provides a means for concrete event/conditions-based inter-task synchronization/communication. Tasks wait while one/multiple event flags are set/cleared for further communicative purposes.
5. Task Notifications ensure lightweight sync and communicative interaction. The notifications correspondingly render easy transmission of different stats among diverse processes/tasks/signalling events/sync.

Conclusion :

After analysing this report we can come to a comprehensive understanding of a Ready Task List including how it stores tasks and which Data Structure has been chosen for its implementation. Additionally, we can evaluate the time complexity involved in processing tasks with this system. To operate correctly certain members must be included in the task list such as pxTopOfStack, xEventListItem, UX priority and pxStack. These aspects rely heavily on particular functions like vTaskSwitchContext and prvAddTaskToReadyList while others like prvRemoveTaskFromReadyList, uxTop Readypriority and pxCurrentTCB contribute to making these processes more efficient. Through careful analysis, we gain insight into how these various components work together to add or remove items from the Ready Task list along with recognition towards which high-priority task should be actioned first.