**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**19CSE212- DATA STRUCTURES AND ALGORITHMS**

**CASE STUDY: FreeRTOS ( Free Real Time Operating System)**

**ANALYSIS OF FREERTOS SOURCE CODE**

**ANALYSING THE IMPORTANCE OF READY LIST IN EFFICIENT EXECUTION OF TASKS**

**GROUP NAME : CAPTAIN SPARROW**

**GROUP MEMBERS :**

|  |  |  |
| --- | --- | --- |
| **Sl. No** | **Roll No.** | **Name** |
| 1. | **CB.EN.U4CSE21412** | BOYA ABHIRAM |
| 2. | **CB.EN.U4CSE21431** | 1. ADITHYA REDDY |
| 3. | **CB.EN.U4CSE21434** | LOKESH CHOWDARY MADALA |
| 4. | **CB.EN.U4CSE21444** | PAKALA VENUTEJA |
| 5. | **CB.EN.U4CSE21465** | UNNAM VIGNESH |

**TABLE OF CONTENTS: Pg No.**

1. Introduction 3
2. Introduction to freeRTOS SCHEDULER 3
3. Significance of the Ready List in FreeRTOS Task Management 4
4. Analysis of Ready List Implementation 5
5. Detailed Examination Of Ready List Operations 6
6. Integration of Ready List with the Scheduler 8
7. Performance Characteristics of Ready List Implementation and

its Potential Limitations 8

1. Conclusion 9
2. References 9

**OVERVIEW OF THE REPORT:**

The report provides a comprehensive analysis of the Ready List in FreeRTOS, focusing on its significance in efficient task management and scheduling for real-time embedded systems. It starts with an introduction to FreeRTOS and its importance. An overview of the FreeRTOS scheduler highlights the critical role of the Ready List. The report explains the Ready List's purpose in managing task states and explores its implementation details, including the chosen data structure and its advantages. It examines operations such as insertion, removal, and prioritization, analyzing their impact on the scheduler's efficiency. Integration with other components and synchronization mechanisms are also explored. The report concludes with a summary of findings, emphasizing the Ready List's crucial role in achieving efficient task scheduling and management in FreeRTOS.

**INTRODUCTION:**

Free Real Time Operating System (RTOS) is an open-source real-time operating system designed for embedded systems. It has a wide range of microcontrollers and microprocessors. FreeRTOS features a real time kernel that provides scheduling functionality, inter-task communication, timing analysis, synchronization primitives, resource efficiency, and portability in real time applications. It is used in a wide range of applications, including automobiles, aerospace, consumer electronics, medical devices like fitness trackers etc., for real time monitoring and analytics.

**FreeRTOS has the following standard features:**

* Flexible task priority assignment
* Task run-time statistics
* Preemptive Priority Scheduling
* Mutexes and Semaphores
* Software Timers
* Binary and Counting Semaphores

**INTRODUCTION TO FreeRTOS SCHEDULER:**

The FreeRTOS scheduler is a core component of RTOS operating system. It is responsible for managing and coordinating the execution of tasks or threads in an embedded system. The scheduler ensures that tasks run in a timely and efficient manner, providing real-time responsiveness and multitasking capabilities.

The FreeRTOS scheduler employs a fixed priority preemptive scheduling policy, with round-robin time slicing of equal priority tasks. Each task is assigned a priority, and the scheduler determines the order in which tasks are executed based on these priorities. Higher priority tasks preempt lower priority tasks, allowing critical tasks to run promptly and meet timing deadlines.

Context switching is a vital mechanism used by the scheduler. When a task is preempted, the scheduler saves the current task's context (registers and other relevant data), switches to the next task, and restores its context. This allows for seamless transitions between tasks and efficient utilization of system resources.

FreeRTOS incorporates time-slicing, allowing tasks of equal priority to share the CPU's execution time. This ensures fairness among tasks and prevents a single high-priority task from monopolizing the processor.

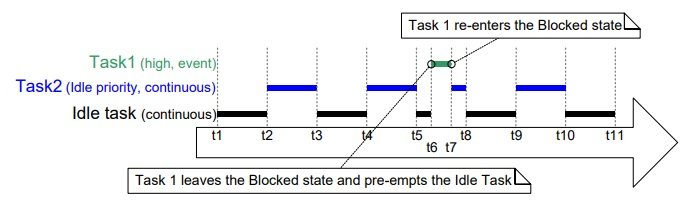
The scheduler manages task delays, handles interrupts, and coordinates task synchronization using mechanisms like semaphores and mutexes. The Ready List is a crucial component that schedules tasks based on priorities, maximizing CPU resource utilization.

**CONFIGURING THE FreeRTOS SCHEDULING POLICY**

The following FreeRTOSConfig.h settings change the default scheduling behaviour:

**If (configUSE\_PREEMPTION ==0)** the Running state task enters the Blocked or Suspended state and calls taskYIELD(), or an interrupt service routine (ISR) manually requests a context switch.

**If (configUSE\_TIME\_SLICING == 0)** the scheduler will not switch between equal priority tasks on each interrupt



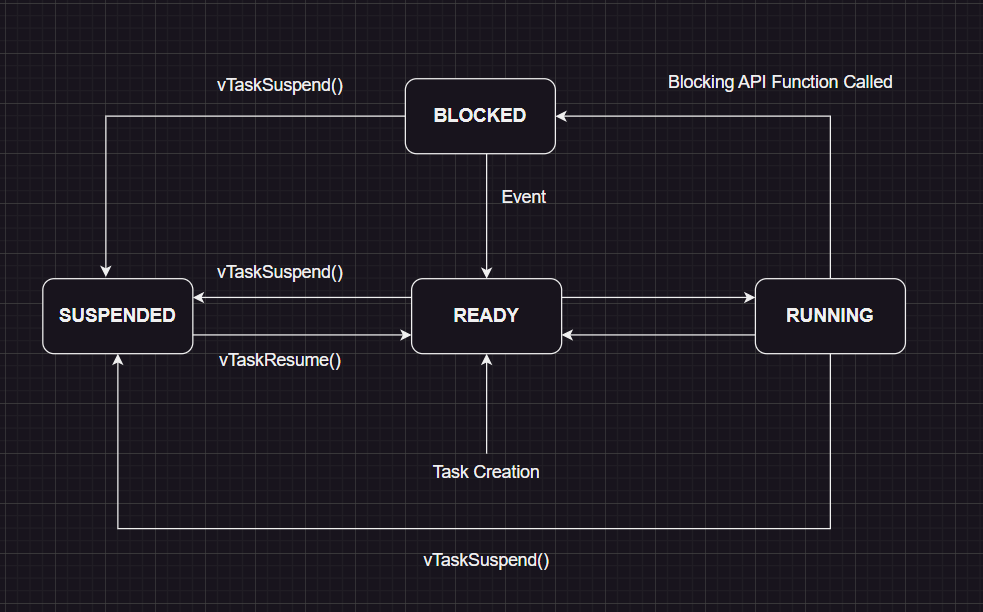
**SIGNIFICANCE OF READY LIST IN FreeRTOS TASK MANAGEMENT**

In FreeRTOS, the Ready List is a fundamental data structure deployed by the scheduler for managing the execution of tasks. The Ready List is a crucial mechanism for achieving real-time responsiveness and effective multitasking in FreeRTOS-based systems. It serves as a repository for all the tasks that are ready to be executed by the CPU (Central Processing Unit) but are waiting for their turn to run. The purpose of the Ready List is to efficiently organize and prioritize tasks, ensuring the fair and orderly execution of processes in a multitasking environment.

The Ready List contains references or pointers to the tasks that are in the "ready" state (**pxTCB, pxCurrentTCB**) pointing to the task control blocks (TCB) where each TCB contains information such as the task's priority, state, stack pointer, and other relevant data. The Ready List organizes these TCBs in an orderly manner allowing the scheduler to select the highest priority task and assigning the CPU for execution.

If a task is in **ready state**, its TCB is inserted into the ready list at the appropriate position based on its priority. During the scheduling process, the scheduler scans the ready list for the highest priority task. When a higher-priority task is inserted into the Ready List, it preempts the currently running lower-priority task performs a context switch from the currently running task to the selected task. The context switch involves saving the current task's context (registers and other relevant data) and restoring the context of the selected task to resume for execution. The selected task is marked as **running state** currently executing in the CPU processors.

The Ready List plays a role in coordinating task synchronization mechanisms, such as semaphores and mutexes. When a task is in **blocked or suspended state** waiting for a synchronization object, its TCB is removed from the Ready List until the synchronization condition is met. Once the task becomes ready again, its TCB is reinserted into the Ready List, allowing the scheduler to consider it for execution.



**ANALYSIS OF READY LIST IMPLEMENTATION:**

The Ready List in FreeRTOS is implemented using an **array of doubly circular linked lists.** Each element of the array represents a priority level, and within each element, a doubly circular linked list is used to store the TCBs (task control blocks) of tasks with the same priority as preemptive scheduling is being followed. The suspended task after the time slice is finished, is preempted, and added to end of circular linked list using the **xGenericListItem** and new task with highest priority in the ready state is assigned for CPU execution. The tasks holding the same priority levels follow the round robin scheduling policy.

The Ready List implements **configMAX\_PRIORITIES** constant to define the maximum number of priority levels available for the tasks in the system. It determines the range of priority levels in the system, and they directly correspond to the indices of the array. Each element in the array maintains a circular doubly linked list of TCB’s that handles the tasks with the same priorities. The pointers (**pxTCB, pxCurrentTCB**) point to the linked list of TCB’s for accessing its priority, state, memory region, stack pointers and program counters.

The Ready List in FreeRTOS provides a flexible way for the scheduler, allowing constant-time access to the highest priority task and efficient handling of tasks with the same priority. The array structure enables fast access to the highest priority task by accessing its index, while the linked list within each array element facilitates efficient insertion and removal operations for tasks with the same priority.

The **pxTCB** points to the TCB’s priority level and is stored in **uxPriority** variable. The **uxPriority** is compared with the **uxTopReadyPriority** which is the internal variable used by the scheduler. If (**uxPriority** > **uxTopReadyPriority)** then the **uxTopReadyPriority** is assigned with **uxPriority** value and the scheduler preempts the current running task and assigns the CPU for the corresponding highest priority task for execution. After the execution of the highest priority task is done, the scheduler reassigns the preempted task with CPU for execution.

The choice of using an array of circular doubly linked lists for the ready list in FreeRTOS is driven to balance performance and memory usage in task scheduling. This approach offers several advantages in terms of performance and memory efficiency:

1. **Constant-Time Access to Highest Priority Task:** The array allows for direct access to the highest priority task in the ready list. Since the index of the array corresponds to the priority level, accessing the highest priority task becomes a constant-time operation (O (1)).
2. **Efficient Insertion and Removal of Tasks:** Each element of the array maintains a linked list for tasks with the corresponding priority level. Linked lists provide efficient insertion and removal operations for tasks with the same priority, typically with constant time complexity (O(1)).
3. **Scalability:** The array of linked lists can handle varying numbers of tasks and priority levels. As the number of tasks increases or decreases, memory allocation scales accordingly based on the actual number of tasks in each priority level. This scalability allows efficient memory management, especially in systems with many tasks or varying task loads.
4. **Dynamic Memory Management:** Linked lists within the array allow for dynamic memory management. As tasks become ready or complete their execution, TCBs can be dynamically allocated or deallocated as needed. This dynamic memory allocation ensures that memory resources are utilized efficiently and avoids unnecessary memory consumption.

The array of circular doubly linked lists strikes a balance between performance and memory usage. It provides constant-time access to the highest priority task, efficient handling of tasks with the same priority, flexible prioritization, and memory optimization. This data structure choice is particularly advantageous when the number of priority levels is small, and memory efficiency is a concern.

**DETAILED EXAMINATION OF READY LIST OPERATIONS**

**1. vTaskSwitchContext():**

The function **vTaskSwitchContext()** is a key component of the task scheduler in FreeRTOS. It is responsible for performing a context switch between tasks, updating run-time statistics (if enabled), and performing stack overflow checks. It ensures that tasks are switched efficiently and that the appropriate tasks are selected for execution based on their priorities.

The function checks if the scheduler is currently suspended, then it sets the **xYieldPending flag** to **pdTrue** to indicate that a context switch should occur. If the scheduler is not suspended the flag is set to **pdFalse** to indicate no context switch request is pending.

If (**configGenerate\_RUN\_TIME\_STATS==1),** the current run-time counter value and value stored in **ulTaskSwitchedInTime** is added to **ulRunTimeCounter.** Finally, **ulTaskSwitchedInTime** is updated with current run-time counter value for the next context switch.

**taskFIRST\_CHECK\_FOR\_STACKOVERFLOW ():** Function call to compare current stack pointer with predefined stack overflow limits to detect stack overflow conditions. Time complexity- O(1)

**taskSELECT\_HIGHEST\_PRIORITY\_TASK():** Function call to schedule next ready task for execution based on task priorities and scheduling algorithms. Time complexity- O(n)

**traceTASK\_SWITCHED\_IN():** Function call to record new task is being switched in. Time complexity- O(1)

The time complexity for **vTaskSwitchContext()** is O(n) and space complexity is O(1).

**2. prvAddTaskToReadyList():**

The **prvAddTaskToReadyList()** is used to add a task to ready list in FreeRTOS. The pxTCB parameter is a pointer to the TCB which must be added to the ready list.

**traceMOVED\_TASK\_TO\_READY\_STATE( pxTCB ):** Function call to update **pxTCB->xGenericListItem** from blocked, suspended or running state to ready state. Time complexity – O(1)

**taskRECORD\_READY\_PRIORITY( ( pxTCB )->uxPriority ):** Function call to compare the newly added task priority with the currently running task priority level. If **uxPriority > uxTopReadyPriority ,** then **uxTopReadyPriority** is assigned with **uxPriority.** This function enables the scheduler to keep track of highest priority task among the other tasks present in the ready list. Time complexity – O(1)

**vListInsertEnd( &( pxReadyTasksLists[ ( pxTCB )->uxPriority ] ), &( ( pxTCB )->xGenericListItem ) ):** Function call to insert newly added TCB at the end of linked list. The **pxReadyTasksLists** is an array holding the linked lists of TCB’s based on priority levels. The **pxTCB->xGenericListItem** is member of TCB which is used to link the node with other nodes of the linked list. The **vListInsertEnd()** inserts the newly added task at the end of the linked list corresponding to its **uxPriority** level. Time Complexity – O(1)

The time complexity for prvAddTaskToReadyList() is O(1) and space complexity is O(1).

**3. uxListRemove():**

The **uxListRemove()** removes a given item (**pxItemToRemove**) from a linked list by adjusting the pointers of the previous and next items in the list.

The **pxList** is a pointer reference to the complete list structure containing the items. The **pvContainer** is a member of **pxItemToRemove** allowing the item to store the reference of the list.

The pointers of the previous and next items are updated to remove the **pxItemToRemove** from the list. The (**pxList->pxIndex**) is updated to point the previous item of **pxItemToRemove** and the **pvContainer** member is set to **NULL.** The number of items in the list (**pxList-> uxNumberofItems)** is decremented and the updated number of items in the list are returned.

The time complexity for **uxListRemove()** is O(1) and space complexity is O(1).

**4. uxTopReadyPriority:**

In FreeRTOS, **uxTopReadyPriority** is a static volatile variable of UBaseType\_t (unsigned integer) datatype which is initially assigned with **tskIDLE\_PRIORITY.** The **tskIDLE\_PRIORITY** is a predefined constant that represents the priority level assigned to the idle task. The **tskIDLE\_PRIORITY** constant is typically defined as the lowest priority level in the system.

The **uxTopReadyPriority** variable is used to keep track of the highest priority task among the tasks which are ready to run. While inserting or deleting tasks from the ready list, the scheduler updates **uxTopReadyPriority** value accordingly. If a new task is inserted into the ready list and its **uxPriority** is greater than the **uxTopReadyPriority,** the variable is assigned with **uxPriority,** allowing the scheduler to preempt the currently running task and allocate CPU to the newly assigned task for execution. In the same way, **uxTopReadyPriority** is updated if any task is removed from the ready list by searching for the highest priority value in the remaining tasks.

The time complexity for **uxTopReadyPriority** is O(1) and space complexity is O(1) (primitive operation)

**5. pxCurrentTCB:**

The **pxCurrentTCB** is a pointer to the currently executing task in the system. It is a static volatile variable of type TCB\_t and is initialized to NULL value.

The control block, also known as the Task Control Block (TCB), is a data structure that contains information about a task, such as its state, priority, stack, program counters and many more. It points to the TCB of currently executing tasks accessing every detail allowing the kernal to manage and schedule tasks effectively. By dereferencing **pxCurrentTCB**, the scheduler can access fields within the TCB structure, allowing it to retrieve or modify task-specific information.

The **pxCurrentTCB** pointer gets updated by scheduler during task context switches. The latest scheduled tasks are referenced by this pointer to collect or update any information of the scheduled task and save its contexts during context switching.

The time complexity of **pxCurrentTCB** is O(1) and space complexity is O(1) as it is a primitive operation.

**INTEGRATION OF READY LIST WITH THE SCHEDULER:**

The Ready List is a critical component of the FreeRTOS task scheduler, playing a vital role in managing the ready tasks. It seamlessly interacts with other scheduler elements like context switching and task preemption to ensure effective task scheduling and execution.

Context switching occurs when a higher-priority task becomes ready, and the Ready List aids the scheduler in determining the most critical task to run, enabling smooth transitions between tasks. Task preemption is made possible by the Ready List's ability to identify the highest-priority task, allowing the scheduler to interrupt lower-priority tasks.

To ensure thread safety and prevent race conditions, FreeRTOS incorporates synchronization mechanisms such as interrupt disabling during critical sections and uses appropriate locking mechanisms in multi-processor systems. Moreover, scheduling algorithms optimize the Ready List by determining the execution order of tasks. The Ready List's seamless integration, synchronization mechanisms, and efficient algorithms contribute to the superior task scheduling and execution capabilities of FreeRTOS.

**PERFORMANCE CHARACTERISTICS OF READY LIST IMPLEMENTATION AND ITS POTENTIAL LIMITATIONS:**

1. **Insertion and Removal Time:** The efficiency of inserting tasks into and removing tasks from the Ready List directly impacts the responsiveness of the task scheduler. Evaluating the time complexity of these operations can indicate how well the implementation scales with a growing number of tasks.

**Limitation:** Any significant increase in insertion or removal time as the task count increases may indicate limitation in managing many tasks effectively.

2. **Priority Ordering:** The Ready List should maintain tasks in priority order, ensuring the highest-priority task is always selected for execution. Evaluating how efficiently the Ready List maintains this ordering is crucial for optimal scheduling.

**Limitation:** The priority ordering mechanism may introduce additional computational overhead, potentially slowing down context switching and task preemption.

3. **Memory Usage:** Assessing the memory footprint of the Ready List implementation is important in resource-constrained systems. The efficient utilization of memory for storing task-related data structures and metadata is crucial.

**Limitation:** The memory usage of the Ready List implementation may be high, posing challenges in memory-constrained environments and restricting the number of manageable tasks.

**CONCLUSION:**

The Ready List implementation in FreeRTOS is a critical and fundamental component that plays a pivotal role in achieving efficient task scheduling and management. It seamlessly interacts with context switching and task preemption, enabling the identification of the highest-priority task for execution and ensuring optimal scheduling and smooth transitions between tasks.

The implementation of the Ready List is designed with efficiency in mind, ensuring the responsiveness and fairness of the task scheduler. Thorough performance evaluations are conducted to assess its strengths and identify any potential limitations, providing valuable insights for further optimization. By studying the FreeRTOS source code, a deeper understanding of the internal mechanisms and algorithms used by the Ready List is gained, contributing to a comprehensive understanding of the task scheduler's efficiency.

The Ready List in FreeRTOS, provides synchronization mechanisms, and optimization algorithms, enhancing the task scheduling capabilities and presents opportunities for ongoing improvements, enabling highly efficient task execution and effective task management.

**REFERENCES:**

1. <https://coed.asee.org/wp-content/uploads/2020/08/2-Use-of-FreeRTOs-in-Teaching-a-Real-Time-Embedded-Systems-Design-Course.pdf>
2. <https://www.freertos.org/about-RTOS.html>
3. <https://docs.espressif.com/projects/esp-idf/en/v4.2.1/esp32/api-guides/freertos-smp.html>
4. <https://www.freertos.org/fr-content-src/uploads/2018/07/FreeRTOS_Reference_Manual_V10.0.0.pdf>