FreeRTOS: UMA FERRAMENTA PODEROSA PARA O ENSINO DE SISTEMAS DE TEMPO-REAL

DOI: 10.37702/2175-957X.COBENGE.2023.4331

Francisco Borges Carreiro - fborges@ifma.edu.br
Instituto Federal de Educação Ciência e Tecnologia do Maranhão

Resumo: *O artigo apresenta o FreeRTOS como uma ferramenta essencial para o ensino de sistemas de tempo real. Destaca-se sua facilidade de uso, ampla disponibilidade de hardware, recursos como tarefas, semáforos, filas e grupos de eventos, além de outros recursos avançados, como temporizadores e gerenciamento de energia. O FreeRTOS permite aos alunos compreenderem os fundamentos e desafios dos sistemas de tempo real, preparando-os para projetar e desenvolver aplicações embarcadas confiáveis*

Palavras-chave: *FreeRTOS, Real-time systems, Embedded systems, Task scheduling, Resource management*

FreeRTOS: UMA FERRAMENTA PODEROSA PARA O ENSINO DE SISTEMAS DE TEMPO-REAL

1 INTRODUÇÃO

Os sistemas operacionais de tempo-real são responsáveis por controlar e monitorar uma ampla variedade de processos e sistemas, que podem variar desde sistemas de transporte e comunicação até dispositivos médicos e sistemas de energia. Esses sistemas são projetados para responder a eventos e estímulos em tempo-real, garantindo a execução precisa e oportuna das tarefas. Eles são caracterizados pela necessidade de responder às solicitações em tempo hábil e garantir que a execução das tarefas sejam concluídas dentro de prazos pré-determinados.

Os sistemas de tempo-real possuem algumas características importantes, dentre elas podemos destacar:

Determinismo: o tempo de resposta e a ordem de execução das tarefas são previsíveis;
Restrições de tempo: as tarefas devem ser concluídas dentro de um prazo pré-determinado;
Confiabilidade: as tarefas devem ser executadas de forma correta e consistente;
Reatividade: as respostas do sistema devem ser rápidas atendento às necessidades de tempo da aplicação.

Os sistemas de tempo-real devem atender a restrições de tempo rígidas e portanto devem ser capazes de receber eventos externos, processá-los e fornecer uma resposta dentro de prazos específicos e predeterminados. A falha em cumprir esses prazos pode ter consequências sérias, como perda de dados, falha na execução de tarefas críticas ou até mesmo riscos à segurança. São amplamente utilizados em uma variedade de indústrias, incluindo automotiva, aviação, telecomunicações, automação industrial, dispositivos médicos e muitas outras aplicações críticas. Exemplos de sistemas de tempo real embarcados incluem sistemas de controle de vôo, sistemas de freios automotivos antitravamento (ABS), dispositivos médicos implantáveis e sistemas de automação industrial.

Ensinar sistemas operacionais embarcados em cursos de engenharia pode apresentar certas dificuldades devido à complexidade e natureza específica desses sistemas. No entanto, com o uso do FreeRTOS, que é um sistema operacionalde de tempo-real aberto e gratuito, pode ajudar a superar esses desafios.

As dificuldades comuns e como o FreeRTOS pode ajudar são:

Conhecimento prévio de programação de baixo nível: Os sistemas operacionais embarcados podem envolver na sua programação em nível de *hardware* pode exigir entendimento de linguagens de programação de baixo nível como C ou Assembly. Segundo D. E. Simon et al (pp. 336-344, 2018), o FreeRTOS oferece uma camada de abstração de hardware que facilita o desenvolvimento, permitindo que os alunos se concentrem mais na lógica da aplicação em vez de detalhes de baixo nível.

Gerenciamento de recursos e temporização: Em sistemas embarcados, o gerenciamento de recursos, como memória, processadores e periféricos, é crítico. Além disso, a temporização adequada das tarefas em tempo real é essencial para garantir o funcionamento correto do sistema. De acordo com A. Saeed et al (pp. 242-249, 2021, o FreeRTOS fornece recursos integrados para o gerenciamento eficiente de recursos e um mecanismo de agendamento de tarefas baseado em prioridades, permitindo que os alunos aprendam a lidar com essas questões de maneira organizada.
Depuração e análise de desempenho: Depurar sistemas embarcados pode ser desafiador, especialmente quando se lida com sistemas em tempo-real. O FreeRTOS oferece recursos de depuração e rastreamento, como pontos de interrupção e registros de eventos, que permitem aos alunos observar e analisar o comportamento do sistema em tempo-real. Segundo P. Merino et al (pp. 334-342, 2019) esses recursos auxiliam na identificação e correção eficiente de problemas. Isso ajuda a identificar e corrigir problemas de forma mais eficiente.
Disponibilidade de documentação e suporte: O FreeRTOS possui uma documentação abrangente, incluindo manuais de referência, exemplos de código e tutoriais, que auxiliam os alunos na compreensão e implementação dos conceitos. Além disso, a comunidade do FreeRTOS é ativa e pode fornecer suporte adicional por meio de fóruns e grupos de discussão (FreeRTOS Documentation).

Ao utilizar o FreeRTOS como plataforma de ensino nos cursos de engenharia, é possível superar as dificuldades no ensino de sistemas operacionais de tempo-real. De acordo com as referências citadas e a experiência prática deste autor no ensino desta disciplina, o FreeRTOS simplifica o desenvolvimento, o gerenciamento de recursos e a depuração, permitindo que os alunos se concentrem nos aspectos-chave do projeto de sistemas embarcados, liberando energia e tempo que seriam gastos na programação, caso não houvesse recursos de um sistema operacional eficiente e de fácil utilização.
Uma das vantagens do uso do FreeRTOS é a ampla disponibilidade de hardware e ambientes de programação suportados. O FreeRTOS é altamente portável e pode ser executado em uma ampla variedade de plataformas de hardware, incluindo microcontroladores populares como ARM Cortex-M, ESP32, STM32, PIC e AVR (FreeRTOS - Supported Devices). Isso oferece aos desenvolvedores flexibilidade para escolher o hardware mais adequado para suas aplicações. Além disso, o FreeRTOS é suportado por várias IDEs e frameworks de desenvolvimento gratuitos, como a IDE do Arduino, PlatformIO, ESP-IDF, Keil MDK e STM32CubeIDE (Arduino - FreeRTOS Library)(PlatformIO - FreeRTOS)(Espressif Systems). Esses ambientes de programação fornecem uma ampla gama de recursos e ferramentas para programar, depurar e testar sistemas embarcados em tempo real. Com a disponibilidade de hardware e ambientes de programação abrangentes, o FreeRTOS se destaca como uma escolha popular e acessível para desenvolvedores que desejam aproveitar os benefícios dos sistemas em tempo-real em uma variedade de projetos e plataformas.

O FreeRTOS está amplamente disponível na IDE do Arduino, sendo uma opção popular para o desenvolvimento de sistemas embarcados em tempo-real. A integração do FreeRTOS na IDE do Arduino é possível graças ao framework oficial para Arduino (Arduino Core), que inclui suporte nativo para o ESP32, entre outros dispositivos. Além disso, o framework ESP-IDF (Espressif IoT Development Framework), desenvolvido pela Espressif Systems, fornece uma camada de software completa para o ESP32, permitindo

uma integração perfeita com o FreeRTOS. Com essa combinação, os desenvolvedores podem aproveitar os recursos poderosos do FreeRTOS, como criação de tarefas, escalonamento e gerenciamento de tempo, juntamente com a facilidade de uso da IDE do Arduino e a compatibilidade com a plataforma ESP32. Isso torna o desenvolvimento de aplicações em tempo real no ESP32 mais acessível e conveniente, permitindo aos desenvolvedores criar sistemas embarcados complexos com eficiência e confiabilidade. Além disso, o FreeRTOS, a IDE do Arduino, o PlatformIO, o ESP-IDF e o Keil MDK são todos gratuitos, o que contribui ainda mais para a comunidade acadêmica, geralmente limitada em recursos. Somado a isso existem simuladores para ambientes Arduino que podem ser utilizados hardware embarcado como Arduinos e ESP32. Um exemplo de bom simulador na plataforma Arduino, disponível online, é o Wokwi. O Wokwi é gratuito e possui diversos componentes e bibliotecas que podem ser usados

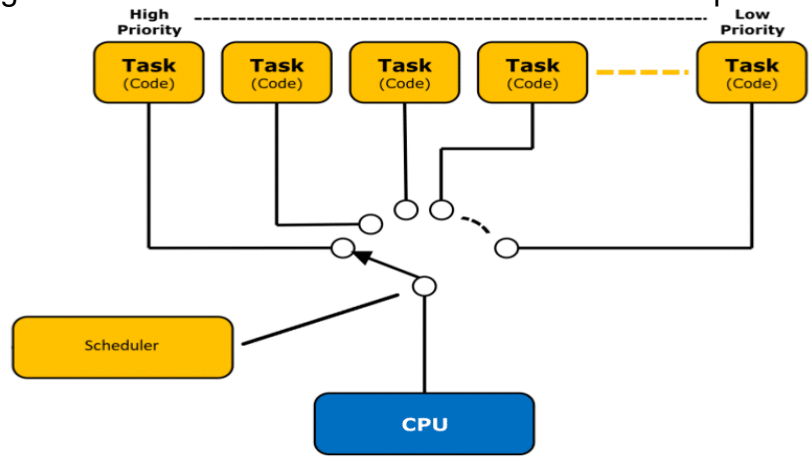
2 TAREFAS, ESCALONAMENTO E GERENCIAMENTO DE TEMPO

Uma tarefa é uma unidade básica de trabalho em um sistema computacional, e representa uma sequência de instruções que deve ser executada pelo processador. As tarefas são gerenciadas pelo sistema operacional, em nosso caso o FreeRTOS, que se encarrega de garantir que elas sejam executadas de forma correta e eficiente. No FreeRTOS, as tarefas são escalonadas por meio de um algoritmo de escalonamento preemptivo, que garante que as tarefas de maior prioridade sejam executadas primeiro.

Duas técnicas de escalonamento no FreeRTOS: *Round Robin* e Prioridade Real. No escalonamento *Round Robin*, todas as tarefas têm a mesma prioridade e são executadas por um período de tempo definido, antes de serem interrompidas e passarem a vez para a próxima tarefa. Já no escalonamento de Prioridade Real, as tarefas são executadas de acordo com sua prioridade, sendo que as tarefas de maior prioridade são executadas primeiro e as de menor prioridade são interrompidas apenas quando uma tarefa de prioridade mais alta se torna ativa.

Um ilustração de como as tarefas em um sistema de tempo-real são executadas pelo escalonador ou agendador é na Figura 1.

Figura 1- Escalonamento de tarefas em ssistemas operacionais de tempo-real



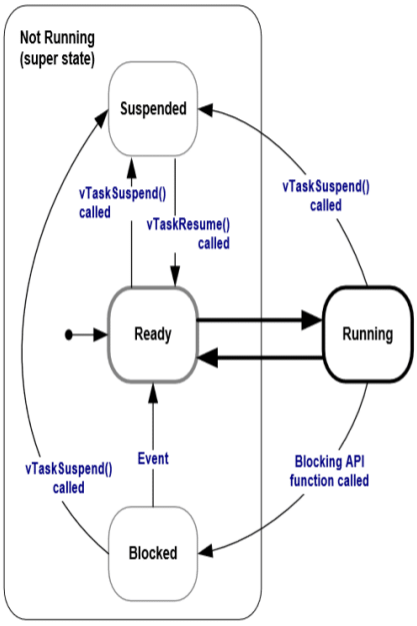
Fonte: https://shopz.off-75.ml/ProductDetail.aspx?iid=299631400&pr=42.88

Como se pode observar na Figura 1 as tarefas são executadas de acordo com a prioridade e o escalonador é responsável por fazer o controle da execução das tarefas.

O gerenciamento de tempo no FreeRTOS é feito por meio do uso de *ticks*, que são unidades de tempo definidas pelo usuário. O FreeRTOS utiliza um timer para gerar interrupções em intervalos regulares, que são usados para atualizar os ticks e executar algumas funções internas do sistema, como o escalonamento de tarefas. O gerenciamento de tempo no FreeRTOS é feito por meio de algumas funções principais, como vTaskDelay, que faz com que a tarefa corrente fique bloqueada por um determinado período de tempo, e vTaskSuspend, que suspende a execução de uma tarefa por um tempo indefinido.

No FreeRTOS, as tarefas podem estar em diferentes estados, dependendo de sua execução e prioridade. A Figura 2 ilustra os principais estados de uma tarefa no FreeRTOS a saber: Pronto (Ready), Executando (Running), Suspensa(Suspend), ou Bloqueada(blocked).

Figura 2 - Estados de uma tarefa no FreeRTOS



Fonte. https://www.freertos.org/RTOS-task-states.html

Pronta (Ready): Uma tarefa pronta está aguardando para ser executada. Ela foi criada e está pronta para ser escalonada pelo escalonador do FreeRTOS. No entanto, outras tarefas podem estar sendo executadas atualmente e, portanto, a tarefa pronta aguarda sua vez.

Executando (Running): Uma tarefa em estado de execução está atualmente sendo executada pelo escalonador. Somente uma tarefa pode estar nesse estado em um determinado momento, já que apenas uma tarefa pode executar em um núcleo/processador por vez.

Bloqueada (Blocked): Uma tarefa bloqueada está temporariamente impedida de executar, aguardando que algum evento ocorra ou uma condição seja atendida. Existem vários motivos pelos quais uma tarefa pode entrar em um estado bloqueado, como

aguardar a disponibilidade de um semáforo, esperar por uma mensagem em uma fila ou aguardar um período de tempo.

Suspensa (Suspended): Uma tarefa suspensa é desativada e não é mais escalonada pelo sistema operacional. Ela não é considerada para execução e permanecerá nesse estado até que seja explicitamente retomada. Tarefas suspensas não consomem recursos do sistema e podem ser úteis em cenários onde uma tarefa precisa ser temporariamente desativada.

Excluída (Deleted): Uma tarefa excluída não será mais executada. Isso geralmente ocorre quando a tarefa completa sua execução ou quando é explicitamente excluída por outra tarefa. Uma tarefa excluída não pode mais ser recuperada e os recursos alocados para ela são liberados.

Os estados das tarefas no FreeRTOS refletem a dinâmica da execução do sistema e permitem que o escalonador gerencie e coordene a execução de tarefas concorrentes. As tarefas podem transitar entre esses estados conforme o progresso do sistema, as operações de bloqueio/desbloqueio e outras ações específicas do programa.

É importante entender esses estados ao desenvolver e depurar aplicativos no FreeRTOS, pois eles influenciam diretamente o comportamento e a interação das tarefas dentro do sistema (Documentação oficial do FreeRTOS)(RICHARD BARRY).

Noramalmente discutir código não é uma tarefa agradável mas para mostrar a facilidade implementar tarefas no FreeRTOS pondo o leitor na condição de aluno que deseja comprrender como criar Tarefas, agendar a execução de tarefa e o gerenciamento da escala de execução de uma tarefa acredito que seja útil e não enfadonho explicar esses recursos no FreeRTOS.

```
//Inclusão das bibliotecas geral do Arduino e FreeRTOS para utilizar o serviço para
criação de tarefas
#include <Arduino.h>
#include <freertos/FreeRTOS.h>
#include <freertos/task.h>
//Definido o número de prioridades
#define MAX_PRIORITIES 10
// Pino do LED embutido na placa ESP32. Certifique em que pino o LED builtin está
conectado, a maioria é no PIN 2
#define LED_PIN 2
// Variável booleana para o estado do LED builtin inicialamente com valor zero
bool ledState = LOW;
// Função para a primeira tarefa
void task1(void *parameter) {
 while (true) {
  ledState = !ledState;     // Alterna o estado do LED
  digitalWrite(LED_PIN, ledState);
  vTaskDelay(pdMS_TO_TICKS(1000));// task1 será chamada 1 segundo depois
   }
}
// Tarefa2 simplesmente faz o print Tarefa 2 executando...
void task2(void *parameter) {
  while (true) {
  Serial.println("Tarefa 2 executando...");
  vTaskDelay(pdMS_TO_TICKS(1000)); //task2 será chamada 1 segundo depois
   }
 }
```

```
// incicialização da void setup()
void setup() {
 pinMode(LED_PIN, OUTPUT);
 Serial.begin(9600);
 // Criação das tarefas
 xTaskCreate(task1, "Tarefa 1", 10000, NULL, 1, NULL);
 xTaskCreate(task2, "Tarefa 2", 10000, NULL, 1, NULL);
}
void loop() {
 // Vazio, o loop não é usado no FreeRTOS
}
```

A função xTaskCreate() é usada no FreeRTOS para criar uma nova tarefa. Ela recebe vários parâmetros que definem o comportamento e as características da tarefa a ser criada. Vamos analisar os parâmetros da chamada `xTaskCreate(task1, "Tarefa 1", 10000, NULL, 3, NULL);`:
task1: É o nome da função que será executada como a tarefa. Nesse caso, é a função `task1` que foi definida anteriormente.
"Tarefa 1": É uma string que define um nome para a tarefa. Esse nome é usado internamente pelo FreeRTOS para fins de depuração e identificação da tarefa.
10000: É o tamanho da pilha alocada para a tarefa em bytes. A pilha é usada para armazenar as variáveis locais e informações de contexto da tarefa. Esse valor determina a quantidade de memória disponível para a tarefa.
NULL: É o parâmetro que pode ser usado para passar dados adicionais para a tarefa. Nesse caso, não está sendo passado nenhum parâmetro adicional, então `NULL` foi usado.
3: É a prioridade da tarefa1. O valor da prioridade varia de 0 (a mais baixa) a configMAX_PRIORITIES - 1 (a mais alta). Tarefas com prioridade mais alta são executadas antes das tarefas com prioridade mais baixa quando ambas estão prontas para serem executadas.
NULL: É um ponteiro para a variável que receberá o identificador da tarefa criada. Nesse caso, não estamos usando esse identificador, então usamos `NULL`.
Ao chamar xTaskCreate(task1, "Tarefa 1", 10000, NULL, 3, NULL);, estamos criando uma nova tarefa chamada "Tarefa 1" que executará a função `task1`, com uma pilha de 10000 bytes, prioridade 3 e sem passar parâmetros adicionais. A tarefa1 será adicionada ao escalonador do FreeRTOS e será executada conforme a prioridade atribuída e o escalonamento definido.

É importante ajustar adequadamente o tamanho da pilha, a prioridade e outros parâmetros ao criar tarefas no FreeRTOS, levando em consideração as necessidades e características específicas do sistema embarcado em questão.
Analisando xTaskCreate(task2, "Tarefa 2", 10000, NULL, 2, NULL);:

task2: É o nome da função que será executada como a tarefa que imprime a mensagem Tarefa 2 executando... ..
Tarefa 2: É uma string que define um nome para a tarefa. Esse nome é usado internamente pelo FreeRTOS para fins de depuração e identificação da tarefa.

10000: É o tamanho da pilha alocada para a tarefa em bytes. A pilha é usada para armazenar as variáveis locais e informações de contexto da tarefa. Esse valor determina a quantidade de memória disponível para a tarefa.
NULL: É o parâmetro que pode ser usado para passar dados adicionais para a tarefa. Nesse caso, não estamos passando nenhum parâmetro adicional, então usamos NULL.
2: É a prioridade da tarefa. O valor da prioridade varia de 0 (a mais baixa) a configMAX_PRIORITIES - 1 (a mais alta). Tarefas com prioridade mais alta são executadas antes das tarefas com prioridade mais baixa quando ambas estão prontas para serem executadas.
NULL: É um ponteiro para a variável que receberá o identificador da tarefa criada. Nesse caso, não estamos usando esse identificador, então usamos NULL.
Com a chamada xTaskCreate(task2, "Tarefa 2", 10000, NULL, 2, NULL);, estamos criando uma nova tarefa chamada "Tarefa 2" que executará a função task2. Essa tarefa terá uma pilha de 10000 bytes, prioridade 2 e não terá parâmetros adicionais.
Lembrando que a prioridade 2 indica que a tarefa terá uma prioridade menor do que a tarefa task1, que tem prioridade 3. Tarefas com prioridades mais altas serão executadas antes das tarefas com prioridades mais baixas quando estiverem prontas para serem executadas.

O escalonador utilizado no exemplo é o preemptivo baseado em prioridade (Rate Monotônic). Esta escolha é feita simplesmente definido prioridade diferente para cada tarefa. Se definirmos a mesma prioridade para ambas tarefas, digamos 3, o escalonador a ser considerado seria o Round Robin e o FreeRTOS agendaria a execução de cada tarefa dando uma fatia de tempo (*time slice*) igual para cada tarefa e executaria mudando de uma tarefa para outra a cada *time slice*. Seguindo o exemplo o aluno aprende a criar tarefas, definir prioridades, definir o período de execução da tarefa, compreender como o FreeRTOS escalona a execução das tarefas, definir o tamanho da pilha alocada para a tarefa em bytes.

3. SEMÁFOROS NO FreeRTOS

No contexto de sistemas operacionais em tempo-real, um semáforo é uma estrutura de dados utilizada para controlar o acesso concorrente a recursos compartilhados por múltiplas tarefas. Ele permite a sincronização e a comunicação entre as tarefas, evitando condições de corrida e garantindo a consistência dos dados.
No FreeRTOS, há três tipos principais de semáforos disponíveis:
Semáforo binário (Binary Semaphore): É um semáforo que pode ter dois estados: adquirido (ocupado) ou liberado (disponível). Esse tipo de semáforo é usado principalmente para sincronização básica entre tarefas. A função xSemaphoreCreateBinary() é usada para criar um semáforo binário.
Semáforo de contagem (Counting Semaphore): É um semáforo que pode ter um número inteiro positivo como contagem. Esse tipo de semáforo é útil quando se deseja controlar o acesso a um número limitado de instâncias de um recurso compartilhado, como buffers ou recursos de hardware. A função `xSemaphoreCreateCounting()` é usada para criar um semáforo de contagem.

O FreeRTOS utiliza o protocolo de herança de prioridade em sua impelementação e para isso somente precisa usar as funções para o semáforo considerado (Documentação oficial do FreeRTOS)(RICHARD BARRY). Este protocolo resolve o

problema conhecido como bloqueio por inversão de prioridade. Um problema de inversão de prioridade ocorreu com o Mars Pathfinder durante a missão da NASA em 1997, que envolveu o rover Sojourner e a sonda espacial Mars Pathfinder. Durante a missão, o Mars Pathfinder usava um sistema operacional de tempo-real chamado VxWorks para gerenciar suas operações. Uma das tarefas críticas no sistema era a comunicação entre o rover Sojourner e a sonda Pathfinder usando um protocolo chamado Synchronous Path Integrated Conductor (SPIN). Essa comunicação ocorria por meio de filas de mensagens (o recurso compartilhado nesse caso é a fila de mensagens). O problema ocorreu quando uma tarefa de menor prioridade, responsável por processar imagens capturadas pelo rover, ficou bloqueada aguardando o acesso à fila de mensagens compartilhada com a tarefa de maior prioridade, que controlava o movimento do rover. A tarefa de menor prioridade impediu a execução da tarefa de maior prioridade. A inversão de prioridade ocorreu porque o sistema operacional VxWorks utilizado na missão não possuía uma política de prioridade herdada. Novas técnicas e abordagens foram implementadas para mitigar esse tipo de problema e garantir a correta execução de tarefas com diferentes prioridades em sistemas críticos (IEEE Conference on Aerospace).

No FreeRTOS, os semáforos são implementados como objetos de sincronização que podem ser usados para garantir a exclusão mútua entre tarefas ou para controlar o acesso a recursos compartilhados. Os semáforos podem ser criados usando a função xSemaphoreCreateBinary() ou xSemaphoreCreateCounting(), dependendo das necessidades específicas.

A função xSemaphoreCreateBinary() cria um semáforo binário, que pode ter apenas dois estados: ocupado ou desocupado. É útil quando há a necessidade de sincronizar o acesso a um único recurso compartilhado entre duas ou mais tarefas.

A função xSemaphoreCreateCounting() cria um semáforo de contagem, que pode ter um número configurável de estados. É útil quando é necessário controlar o acesso a um recurso compartilhado com um número limitado de instâncias disponíveis, como pools de memória ou buffers.

Após criar um semáforo, ele pode ser usado pelas tarefas usando as funções xSemaphoreTake() e xSemaphoreGive(). A função xSemaphoreTake() é usada para adquirir (ou "pegar") o semáforo, bloqueando a tarefa se ele estiver ocupado. A função xSemaphoreGive() é usada para liberar (ou "dar") o semáforo, indicando que o recurso compartilhado está disponível novamente.

Existem outras funcionalidades e configurações disponíveis para semáforos, como temporizadores e ações em caso de timeout. É importante consultar a documentação oficial do FreeRTOS para obter informações mais detalhadas sobre o uso e as opções de configuração dos semáforos. A implementação de aplicação usando semáforo pode ser encontrada em FreeRTOS)(Richard Barry)(FreeRTOS github) não apresentado aqui para não discutir código e devido a limitação do número de páginas.

4. FILAS DE MENSAGENS E FILAS DE TAREFAS

No contexto dos sistemas operacionais de tempo-real, uma fila é uma estrutura de dados que permite a comunicação e troca de informações entre tarefas ou interrupções de forma assíncrona. As filas são úteis para compartilhar dados entre diferentes partes do sistema, permitindo que as tarefas transmitam informações de maneira eficiente e segura.

No FreeRTOS, as filas são suportadas como um recurso nativo e podem ser criadas e gerenciadas usando as funções e APIs fornecidas pelo sistema operacional. O FreeRTOS oferece dois tipos principais de filas: filas de mensagem e filas de tarefas.

Filas de mensagem (Message Queue): Uma fila de mensagem é uma estrutura que permite que as tarefas enviem mensagens de tamanho variável para outras tarefas. As mensagens podem conter qualquer tipo de dado, como números, estruturas ou ponteiros. As filas de mensagem no FreeRTOS são implementadas usando buffers de tamanho fixo. As funções `xQueueCreate()` e `xQueueSend()` são usadas para criar uma fila de mensagem e enviar uma mensagem, respectivamente. Da mesma forma, xQueueReceive() é usada para receber uma mensagem de uma fila.
Filas de tarefas (Task Queue): Uma fila de tarefas é uma estrutura que permite que as tarefas sejam colocadas em uma fila e posteriormente removidas para execução. Isso é útil quando é necessário controlar a ordem de execução das tarefas. O FreeRTOS permite que tarefas sejam colocadas em uma fila e agendadas para execução usando as funções `xQueueSendToBack()` ou `xQueueSendToFront()`, dependendo da posição desejada na fila. A função `xQueueReceive()` é usada para remover uma tarefa da fila e iniciar sua execução.

As filas no FreeRTOS são protegidas por mecanismos internos de exclusão mútua, garantindo a consistência e a integridade dos dados. Além disso, o FreeRTOS fornece opções para especificar timeouts na espera por uma fila e gerenciamento de prioridade no acesso à fila. O uso de filas no FreeRTOS facilita a troca de informações e o compartilhamento de dados entre tarefas em um ambiente de tempo real, fornecendo um mecanismo de comunicação eficiente e seguro. Como utilizar este recurso (FreeRTOS)(Richard Barry)(FreeRTOS github) são referências com exemplos de códigos que tratam desse assunto de formapática.

5. GRUPOS DE EVENTOS

Os grupos de eventos (event groups) são ua funcionalidade poderosa disponível no FreeRTOS para permitir a comunicação e sinmcronização entre tarefas através de eventos específicos. Esses eventos podem ser sinais, flags ou condições que as tarefas esperam ou ativam para coordenar suas ações.

O FreeRTOS define os grupos de eventos como objetos de sincronização que contêm um conjunto de bits, onde cada bit pode ser configurado como "ligado" ou "desligado". Cada bit representa um evento específico, e as tarefas podem esperar por um ou mais eventos ocorrerem antes de continuar sua execução.
A API de grupos de eventos no FreeRTOS inclui as seguintes funções principais:

xEventGroupCreate(): Cria um grupo de eventos vazio e retorna um identificador para esse grupo.
xEventGroupSetBits(): Ativa um ou mais bits dentro do grupo de eventos, indicando que esses eventos ocorreram. As tarefas que estão esperando por esses eventos serão notificadas.
xEventGroupClearBits(): Desativa um ou mais bits dentro do grupo de eventos.

xEventGroupWaitBits(): Bloqueia a tarefa até que um conjunto específico de bits dentro do grupo de eventos seja ativado. Essa função pode esperar por um padrão exato de bits ou apenas verificar se qualquer um dos bits necessários está ativo.
xEventGroupSync(): Sincroniza um grupo de eventos entre múltiplas tarefas, permitindo que todas as tarefas esperem por um conjunto específico de eventos antes de continuarem a execução.

O uso de grupos de eventos no FreeRTOS permite que as tarefas se comuniquem e coordenem suas ações de maneira eficiente e sincronizada. Por exemplo, uma tarefa pode esperar que um evento ocorra antes de prosseguir, enquanto outra tarefa pode sinalizar a ocorrência desse evento usando a função `xEventGroupSetBits()`.
Além disso, os grupos de eventos também podem ser combinados com outros recursos do FreeRTOS, como semáforos e filas, para criar estruturas de sincronização mais complexas e flexíveis.

É importante observar que os grupos de eventos consomem recursos, principalmente memória, para armazenar os bits de evento. Portanto, ao projetar um sistema usando grupos de eventos, é necessário considerar o número de eventos necessários e a alocação de memória correspondente.

6. OUTROS RECURSOS IMPORTANTES do FreeRTOS

Além dos recursos que já foram abordados, existem outros recursos do FreeRTOS que são igualmente importantes e relevantes para o ensino de sistemas de tempo-real. Aqui estão mais alguns recursos que se pode considerar no ensino de Sistemas de Tempo-Real usando o FreeRTOS:
Temporizadores (Timers): Os temporizadores no FreeRTOS permitem agendar a execução de tarefas ou ações em intervalos regulares de tempo. Eles são úteis para executar tarefas periódicas, atrasos programados ou operações que devem ocorrer em momentos específicos. Os temporizadores podem ser criados usando as funções `xTimerCreate()` e `xTimerStart()`, entre outras.
Gerenciamento de memória: O FreeRTOS oferece funcionalidades para gerenciamento de memória, como alocadores de memória dinâmica e pools de memória estática. Isso permite um uso eficiente da memória do sistema, garantindo a disponibilidade adequada de recursos para as tarefas.
Deadlock detection (Detecção de bloqueios): O FreeRTOS inclui uma funcionalidade de detecção de deadlock, que pode ajudar a identificar situações em que as tarefas estão bloqueadas e não podem progredir. Essa funcionalidade pode ser útil para a depuração e solução de problemas em sistemas complexos.
Introdução a interrupções: O FreeRTOS fornece mecanismos para lidar com interrupções de hardware, permitindo que tarefas e interrupções cooperem para processar eventos assíncronos. É importante abordar os conceitos básicos de interrupções e como o FreeRTOS lida com elas para garantir uma compreensão abrangente do sistema.

Gerenciamento de energia: O FreeRTOS oferece recursos de gerenciamento de energia que podem ser aplicados em sistemas embarcados com restrições de energia. Isso inclui a capacidade de colocar tarefas em suspensão para economizar energia quando não estão sendo utilizadas.

Cada um dos recursos do FreeRTOS desempenha um papel significativo no desenvolvimento de sistemas confiáveis e eficientes, e explorá-los junto aos alunos a obter uma compreensão mais aprofundada dos desafios e soluções relacionados aos sistemas de tempo-real o torna uma ferramenta de ensino muito útil para a formação de engenheiros.

7. EXPERIÊNCIAS DE ENSINO E APRENDIZADO EM SALA DE AULA

Ao utilizar o FreeRTOS como ferramenta de ensino em sistemas de tempo-real, foi possível observar uma série de benefícios e resultados positivos. Os alunos demonstraram maior facilidade na compreensão dos conceitos relacionados a sistemas operacionais embarcados, pois o FreeRTOS oferece APIs e recursos poderosos, como criação de tarefas, semáforos, filas e grupos de eventos. Além disso, a disponibilidade de uma ampla gama de *hardware* suportado e ambientes de programação integrados, como a IDE do Arduino e o ESP-IDF, proporcionou aos alunos a oportunidade de experimentar e desenvolver aplicações em tempo-real em diferentes plataformas. Ao trabalharem com o FreeRTOS, os alunos puderam se concentrar nos aspectos-chave do projeto de sistemas embarcados, como o design da lógica de controle e a comunicação entre tarefas, em vez de se preocuparem com a implementação de um sistema operacional completo do zero. Isso permitiu um aprendizado mais eficiente e uma maior dedicação aos conceitos teóricos e práticos essenciais para o desenvolvimento de sistemas de tempo-real.

8. CONCLUSÕES

Este trabalho apresentou o sistema operacional de tempo-real FreeRTOS e demonstrou ser uma ferramenta essencial para o ensino de sistemas de tempo real. Abordou conceitos introdutórios de sistemas de tempo-real e apresentou de forma detalhada a utilização do FreeRTOS, destacando a facilidade de uso, ampla disponibilidade de hardware, recursos como tarefas, semáforos, filas e grupos de eventos, além de outros recursos avançados, como temporizadores e gerenciamento de energia. Também apresentou a experência de sua utilização em sala de aula no ensino dos sistemas de tempo real embarcados no desenvolvimento de aplicações confiáveis.

AGRADECIMENTOS

Gostaria de expressar meu sincero agradecimento ao IFMA (Instituto Federal do Maranhão) pelo apoio valioso e pelo laboratórios de automação. Além disso, gostaria de estender meus agradecimentos à FAPEMA (Fundação de Amparo à Pesquisa e ao Desenvolvimento Científico e Tecnológico do Maranhão) pelo apoio financeiro por meio de bolsas de pesquisa. Sua assistência tem sido fundamental para o avanço de minhas atividades de pesquisa.

REFERÊNCIAS

A. Saeed, D. Kreutz, H. Guan, "Teaching Embedded Real-Time Systems with FreeRTOS," IEEE Transactions on Education, vol. 64, no. 3, pp. 242-249, 2021.
Arduino - FreeRTOS Library. Disponível em: https://www.arduino.cc/reference/en/libraries/freertos/. Consltado em 16 de Maio de 2023.
D. E. Simon, J. W. Bretl, J. B. Albus, "Using FreeRTOS in an Embedded Systems Course," IEEE Transactions on Education, vol. 61, no. 4, pp. 336-344, 2018.
Documentação oficial do FreeRTOS. Disponível em https://www.freertos.org/. Consultado em 16 de Maio de 2023.
Espressif Systems - ESP-IDF Programming Guide. Disponível em: https://docs.espressif.com/projects/esp-idf/en/latest/. Consltado em 16 de Maio de 2023.
FreeRTOS - Supported Devices. Disponível em: https://www.freertos.org/RTOS_ports.html. Consltado em 16 de Maio de 2023.
P. Merino, J. A. Fdez-Valdivia, J. A. López-Ramos, "Teaching Embedded Systems Using FreeRTOS and Real Hardware," IEEE Transactions on Education, vol. 62, no. 4, pp. 334-342, 2019.
PlatformIO - FreeRTOS. Disponível em: https://docs.platformio.org/en/latest/frameworks/freertos.html. Consltado em 16 de Maio de 2023.
Richard Barry, "Mastering the FreeRTOS Real Time Kernel: A Practical Guide". Disponível em https://www.freertos.org/. Consltado em 16 de Maio de 2023.
IEEE Conference on Aerospace. Ten Years After: Enduring Lessons Learned from Mars Pathfinder. Disponível em https://ieeexplore.ieee.org/document/4161713. Consltado em 16 de Maio de 2023.
wokwi. Diponível em https://wokwi.com/. Consltado em 16 de Maio de 2023.

FREERTOS: A POWERFUL TOOL FOR REAL-TIME SYSTEMS EDUCATION

Abstract: *This document introduces FreeRTOS as a fundamental tool for teaching real-time systems. It highlights its user-friendly nature, wide hardware availability, key features such as tasks, semaphores, queues, and event groups, as well as advanced functionalities like timers and power management. FreeRTOS enables students to grasp the fundamentals and challenges of real-time systems, equipping them with the skills to design and develop reliable embedded applications.*

Keywords: *FreeRTOS, Real-time systems, Embedded systems, Task scheduling, Resource management*