UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

IMARIO ALMEIDA BARBOSA

RELATÓRIO

Projeto de Sistema de Tempo-Real I

Introdução

Esse relatório descreverá como foi implementado o projeto de RTOS que foi solicitado. O projeto posui as seguintes tasks:

- LEDTask
- SendData
- KeypadScanningTask
- PlantControlTask
- createAperiodcJob
- WebServerTask
- RS232Task

E algumas funções auxiliares, que são:

- UpdateDisplay
- KeyPressed
- xGetUsageCPU

Implementação do escalonador

Configuração do FreeRTOSConfig.h

A biblioteca de escalonamento utilizada foi a do ES Free. Primeiramente é necessário fazer os downloads dos arquivos scheduler.c e scheduler.h presentes em https://github.com/ESFree/ESFree. Após o download é preciso transferir os arquivos para a pasta do projeto. Agora temos que configurar o FreeRTOSConfig.h, como eu implementei o projeto tendo como o base do Demo do posix então algumas definições não foram necessárias pois já estavam ativadas por padrão. A primeira definição a se fazer no FreeRTOSConfig.h é definir o número de índices na matriz de armazenamento local do thread de cada tarefa, representando por **configNUM_THREAD_LOCAL_STORAGE_POINTERS**, que foi definido como 5. Essa foi a única definição realizada no FreeRTOSConfig.h para o funcionamento do escalonador, lembrando que existem outras definições necessárias mas como o projeto teve como base o Demo então não foi necessário defini-las.

Configuração do scheduler.h

No scheduler.h existe um define schedSCHEDULING_POLICY, ele vai definir qual política de escalonamento será utilizada, no meu projeto foi utilizado o EDF, então o schedSCHEDULING_POLICY foi definido como schedSCHEDULING_POLICY_EDF.

Implementação das Tasks e funções auxiliares

Nesse tópico será descrito as task implementadas no projeto e suas funções auxiliares.

- **PlantControlTask**: É a task que vai receber os dados do sensor e processar o algoritmo de controle. Para o recebimento dos valores dos sensores foi configurado uma fila, que caso ambos os sensores tenham conseguido enviar os dados com sucesso o algoritmo de controle é executado, esse algoritmo é um loop que percorre uma constante CTRL_ALG_LOAD que foi definida como 0x100000
- **LEDTask**: É a task que pisca determinado LED baseado na condição do processador. A implementação dela foi bem simples, se o xGetUsageCPU for maior que 85 a função vToggleLED com o LED vermelho como parâmetro é chamada, se não a mesma função é chamada mas o parâmetro dela é o LED verde. A função xGetUsageCPU é a função criada para calcular o tempo em que as tasks passam executando, ela calcula esse tempo baseado na fórmula ((configTICK_RATE_HZ - time_idle)/configTICK_RATE_HZ)*100), onde o time_idle é igual a função ulTaskGetIdleRunTimeCounter e a constante configTICK_RATE_HZ é igual a 1000. Um detalhe importante é que nessa task existe uma chamada da função ulTaskClearIdleRunTimeCounter, que é uma função criada por mim para zerar o contador de ticks da idle task, essa função foi implementada no task.c.
- **SendData**: É a task responsável por enviar os dados dos sensores para a PlantControlTask, os valores dos sensores são gerados usando a fórmula rand() % 100, que gera números aleatórios de 0 a 100. Para dar uma ideia de interferência no envio uma condição foi criada, o dado só é enviado se o número gerado pelo sensor for menor ou igual a 90.
- **KeypadScanningTask**: É a task que ler o teclado e atualiza o display, se a tecla for pressionada a tela é atualizada. Como eu não consegui configurar um scanf para receber os dados de teclado, eu simulei um teclado utilizando a função KeyPressed, criada por mim, ela foi baseada no teclado da imagem do projeto, que possui teclas de 0 a 9 e uma seta para a direita e outra pra esquerda. Para a implementação dela foi criado um vetor de char com todos esses caracteres e foi criada uma condição para a detecção, um número aleatório de 0 a 100 é gerado, se ele for maior que 98 uma tecla aleatória é enviada para a task e o retorno é pdTRUE caso o número gerado seja menor que 98 então o retorno é pdFALSE.
- **createAperiodcJob**: É a task que foi criada para simular a interrupção que chamaria as tasks WebServerTask e RS232Task, na implementação são gerados dois números aleatórios de 0 a 100, caso o primeiro seja maior que 70 uma task aperiódica chamada RS232Task é criada, e se o segundo número for maior que 85 a task WebServerTask também aperiódica é criada.
- **WebServerTask**: É a task que simula um servidor web, a implementação dela é um laço que percorre até a constante HTTP_REQUEST_PROC_LOAD, que foi definida como 0xf000

- **RS232Task**: Segue o mesmo princípio da WebServerTask a única diferença é a constante, que nesse caso é RS232_CHAR_PROC_LOAD, que tem o valor 0x1000

Código

O código será disponibilizado no seguinte Github:

https://github.com/imarioa/RTOS/tree/main/Projeto/FreeRTOS/Demo/Posix_GCC