Aula 13
Internet das Coisas

FreeRTOS/ESP32

Cairo Mateus Neves Ribeiro/Bruno Manias Trazzi

Supervisão: Prof. Julio Cezar Estrella

jcezar@icmc.usp.br

Roteiro

- Introdução ao FreeRTOS
- FreeRTOS e ESP32
- Hello World ESP32 e FreeRTOS
- Criação de Tasks
- Troca de Mensagens
- Região Crítica

Introdução a FreeRTOS

- Sistemas operacional de tempo real para sistemas embarcados em microprocessadores.
- Fornece API para funcionalidades como:
  - criação e controle de tasks;
  - comunicação entre tasks;
  - proteção de região crítica;
- Proporciona maior portabilidade ao código.

FreeRTOS e ESP32

- Espressif IoT Development Framework (IDF) é o framework oficial para o desenvolvimento do ESP32.
- O ESP32 apresenta dois núcleos físicos XTENSE LX6 (dual-core).
- Cada núcleo roda a até 240MHz.
- A IDF oferece suporte a multiprocessamento de maneira transparente (Com o uso do FreeRTOS acoplado no código fonte).

FreeRTOS e ESP32

- O ESP32 apresenta suporte ao multiprocessamento simétrico.
- O núcleo principal chama: PRO_CPU (0).
- O núcleo secundário chama: APP_CPU (1).
- Por padrão a função loop é executada na APP_CPU.

Primeiro Exemplo - Hello World

Delegar e executar uma função para um outro núcleo do ESP32 usando IPC, usado para funções que irão retornar, ou seja, não executam laço infinito

- esp_ipc_call
- xPortGetCoreID

Primeiro Exemplo - Hello World

- *esp_err_t* esp_ipc_call (*uint32* **cpu_id**, *esp_ipc_func_t* **func**, *void* ***arg**)
  - **cpu_id**: informa qual CPU vai executar a tarefa.
  - **func**: informa qual função será executada.
  - **arg**: passa os argumentos para a função.

Primeiro Exemplo - Hello World

- *int* **xPortGetCoreID** ()
  - Retorna o núcleo que está sendo executado.
  - PRO_CPU = 1
  - APP_CPU = 0

Segundo Exemplo - Criação de Tasks

Criação de tarefas com base na função **xTaskCreatePinnedToCore** para criação de tarefas que não irão retornar, ou seja, irão executar laço infinito.

O *header* da função xTaskCreate é:

- *BaseType_t* xTaskCreatePinnedToCore (*TaskFunction_t* **pvTaskCode**, *const char * const* **pcName**, *configSTACK_DEPTH_TYPE* **usStackDepth**, *void* ***pvParameters**, *UBaseType_t* **uxPriority**, *TaskHandle_t* ***pxCreateTask**, const BaseType_t **xCoreID**)

Segundo Exemplo - Criação de Tasks

- **pvTaskCode:** ponteiro para a função a ser executada.
- **pcName:** nome da *task* para fins de debug e recuperação de seu *handle*.
- **usStackDepth:** porção de memória usada como stack da função a ser executada.
- **pvParameters:** parâmetros a serem passados a função que será executada.
- **uxPriority:** prioridade da *task* usada pelo *scheduler.*
- **pxCreateTask**: ponteiro em que o handle da *task* será armazenada.
- **xCoreID**: o *core* para o qual a *task* será delegada.

Terceiro Exemplo - Troca de Mensagens

Permite a comunicação entre tarefas e permite criar filas de funções.

- xQueueCreate
- xQueueSend
- xQueueReceive

Terceiro Exemplo - Troca de Mensagens

- Cria uma nova fila de mensagens.
- *QueueHandle_t* xQueueCreate (*UBaseType_t* **uxQueueLenght**, *UBaseType_t* **uxItemSize**)
  - **uxQueueLenght:** número máximo de itens na fila.
  - **uxItemSize:** tamanho em *bytes* dos elementos da fila.

Terceiro Exemplo - Troca de Mensagens

- Envio de mensagens.
- *BaseType_t* xQueueSend (*QueueHandle_t* **xQueue**, *const void* *__pvItemToQueue__, *TickType_t* **xTicksToWait**)
  - **xQueue:** fila em que o item será adicionado.
  - **pvItemToQueue:** o elemento que será adicionado a fila.
  - **xTicksToWait:** número de ticks de clock que a diretiva deve esperar por um espaço vago caso a fila esteja cheia.

Terceiro Exemplo - Troca de Mensagens

- Receber mensagens armazenadas no buffer.
- *BaseType_t* xQueueReceive (*QueueHandle_t* **xQueue**, *const void* *__pvItemToQueue__, *TickType_t* **xTicksToWait**)
  - **xQueue:** fila em que o item será adicionado.
  - **pvItemToQueue:** *buffer* em que o elemento recebido deve ser armazenado.
  - **xTicksToWait:** número de ticks de clock que a diretiva deve esperar por um espaço vago caso a fila esteja vazia.

Quarto Exemplo - Proteção de Região Crítica

As regiões críticas podem ser controladas com o uso de semáforos binários, semáforos contadores e mutex.

- Semáforos são melhor empregados para sincronização.
- Mutex é melhor usada para exclusão mútuo.

Quarto Exemplo - Proteção de Região Crítica

- A seguinte função cria um semáforo binário.
- *SemaphoreHandle_t* xSemaphoreCreateBinary()
- Este semáforo inicializa bloqueado, então para usá-lo é necessário desbloqueá-lo.

Quarto Exemplo - Proteção de Região Crítica

- Para criação de um semáforo contador é necessária a função *SemaphoreHandle_t* xSemaphoreCreateCounting (*UBaseType_t* **uxMaxCount**, *UBaseType_t* **uxInitialCount**)
- **uxMaxCount:** valor máximo para o contador.
- **uxInitialCount:** valor de inicialização do contador.

Quarto Exemplo - Proteção de Região Crítica

- Criar um mutex.
- *SemaphoreHandle_t* xSemaphoreCreateMutex ()

Quarto Exemplo - Proteção de Região Crítica

- Obter o semáforo criado pelos métodos **xSemaphoreCreateBinary, xSemaphoreCreateCounting** e **xSemaphoreCreateMutex.**
- *void* xSemaphoreTake (*SamaphoreHandle_t* **xSemaphore**, *TickType_t* **xTicksToWait**)
- **xSemaphore:** identificar para o semáforo obtido.
- **xTicksToWait:** o tempo de espera para o semáforo ficar disponível.

Quarto Exemplo - Proteção de Região Crítica

- Macro para liberar um semáforo.
- *void* xSemaphoreGive (*SamaphoreHandle_t* **xSemaphore**)
- **xSemaphore:** identificar para o semáforo obtido.

Conclusão

- FreeRTOS é um bom sistema operacional para dispositivos de baixa capacidade.
- ESP32 consegue executar tarefas paralelas com seus dois núcleos.
- O FreeRTOS disponibiliza as diretivas primitivas para o controle de troca de mensagens, proteção de região crítica e controle da função que deve ser executada em cada núcleo.

Referências

- Principais conceitos de RTOS para iniciantes com Arduino e FreeRTOS. Link: <https://www.embarcados.com.br/rtos-para-iniciantes-com-arduino-e-freertos/>
- ESP32 - Lidando com Multiprocessamento. Link: <https://www.embarcados.com.br/esp32-lidando-com-multiprocessamento-parte-i/>
- Projetos no Arduino com FreeRTOS - conceitos, tasks e consumo de memória por task. Link: <https://www.filipeflop.com/blog/tasks-no-freertos/>
- Documentação da API oferecida pelo FreeRTOS. Link: <https://www.freertos.org/a00106.html>
- Documentação da API oferecida pelo port da Espressif do FreeRTOS. Link: <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/freertos.html#overview>

Atividade

- Disponível no Moodle conforme consta no cronograma da disciplina

Próxima Aula

- Entrega e Apresentação do Projeto