



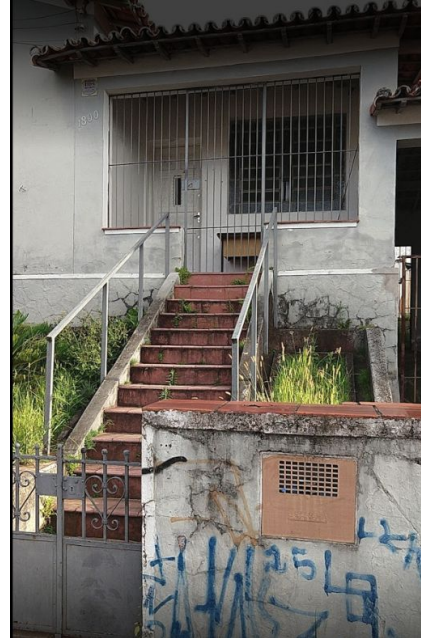
Aprendendo a programar microcontroladores ARM da ST com FreeRTOS

Jorge Guzman



0 Laboratório Hacker de Campinas

- Sala Central (Oficinas, Palestras)
- Sala Coringa (Biblioteca)
- Laboratorio de Eletronica
- Cozinha
- Marcenaria
- Area externa (Area de testes)
- Network





Yo

- Formado em engenharia da computação
- Especialização em Automação industrial
- Especialização em Engenharia de Software
- Membro do LHC
- Articulista do Portal Embarcados
- 9 anos trabalhando com desenvolvimento de Firmware



FERRAMENTAS ST

- IDEs
 - System WorkBench
 - Atollic
 - STM32CubeIDE
- CubeMX
 - Camada HAL drivers da familia ARM da ST.
 - Bibliotecas CMSYS-Math, CMSYS-NN.
 - FreeRTOS, FATFS, LWIP, USB, etc.
- Debuggers:
 - ST-Link
 - J-Link



STM32CubeIDE

- Usa a interface Eclipse
- Interface gráfica para configurar todos os periféricos
- Gera código de inicialização
- Debug da aplicação.



KITS DE DESENVOLVIMENTO



STM32 Nucleo
development boards

Flexible
prototyping



Discovery kits

Key feature
prototyping



Evaluation boards

Full feature
evaluation



STM32 Nucleo
expansion boards

Add-on
functionalities

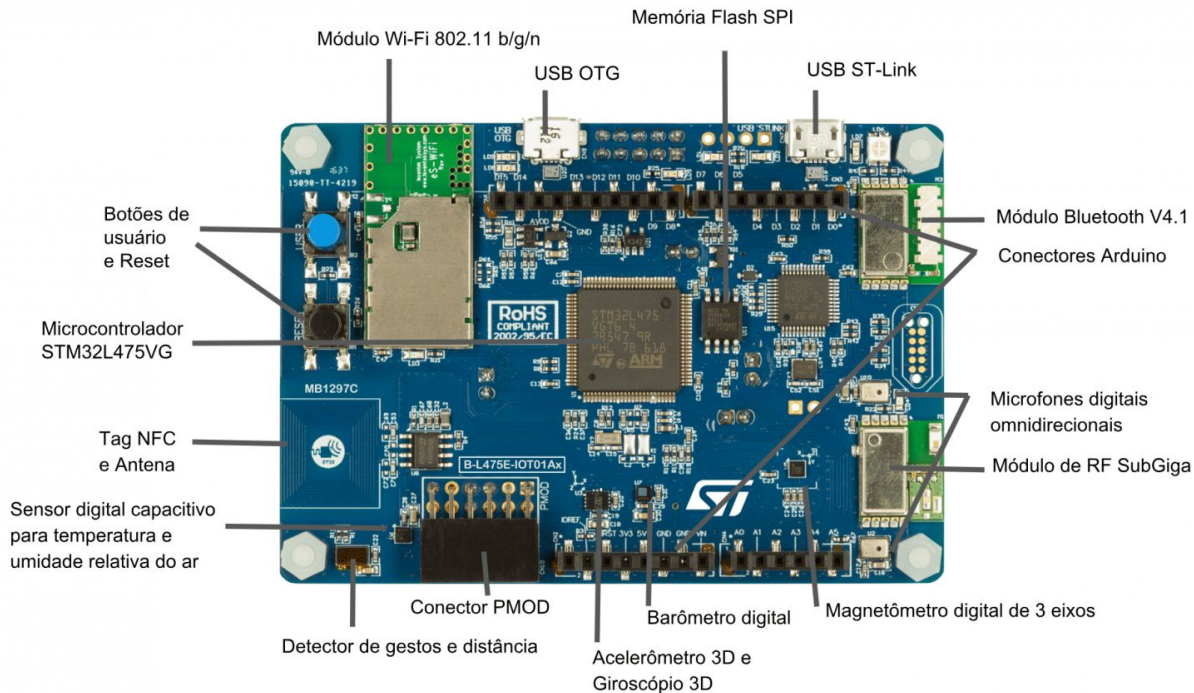


Third-party
boards

From full
evaluation to
open hardware

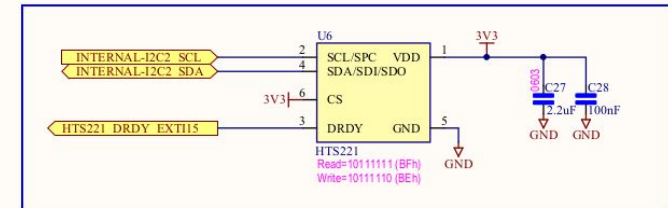
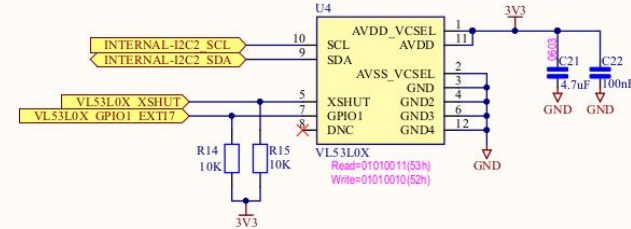
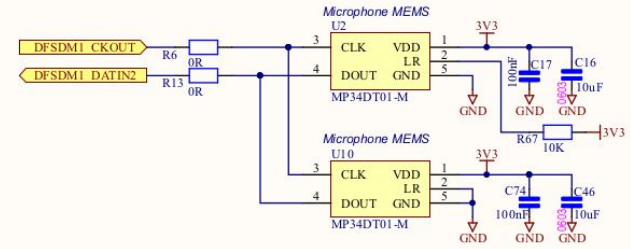
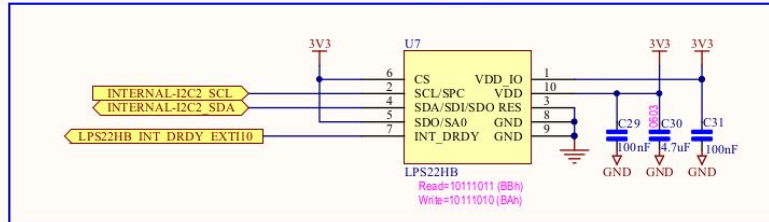
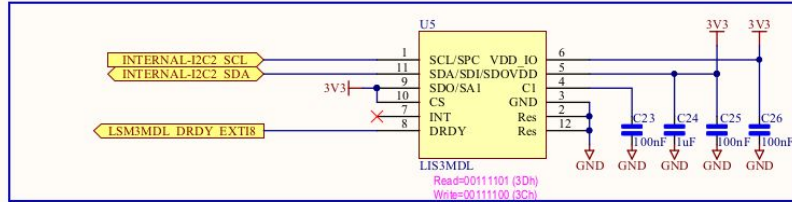
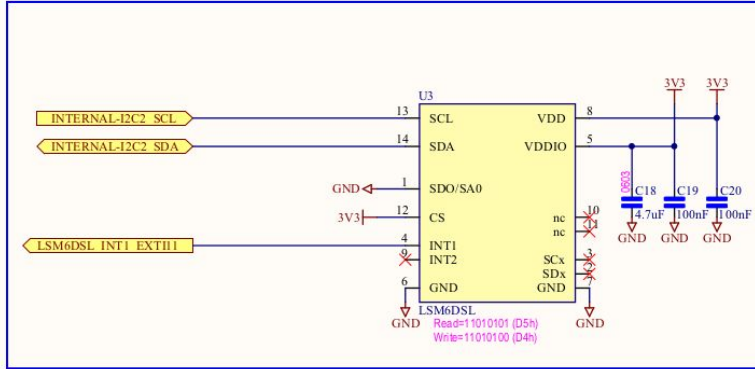


B-L475E-IOT01A



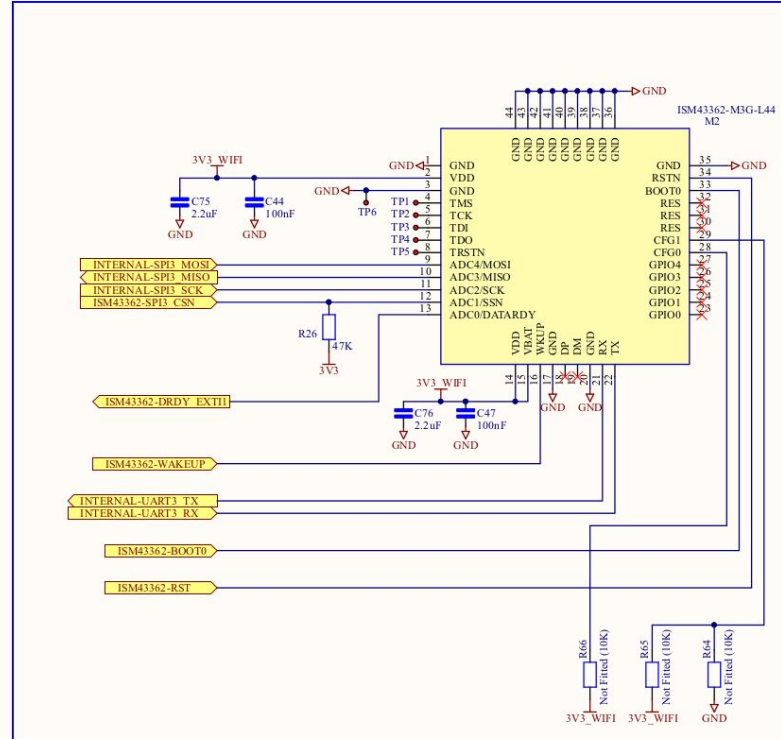
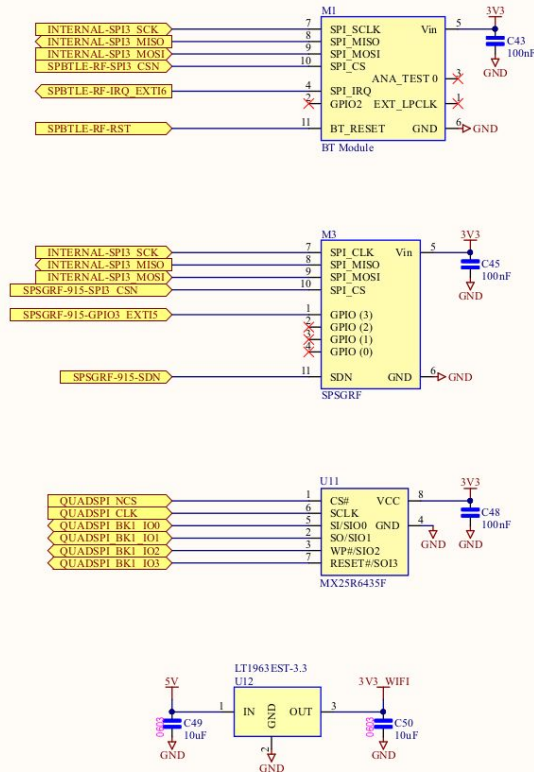


Conexão Elétrica Sensores (I2C)





Conexão Elétrica Módulo Wifi (SPI)





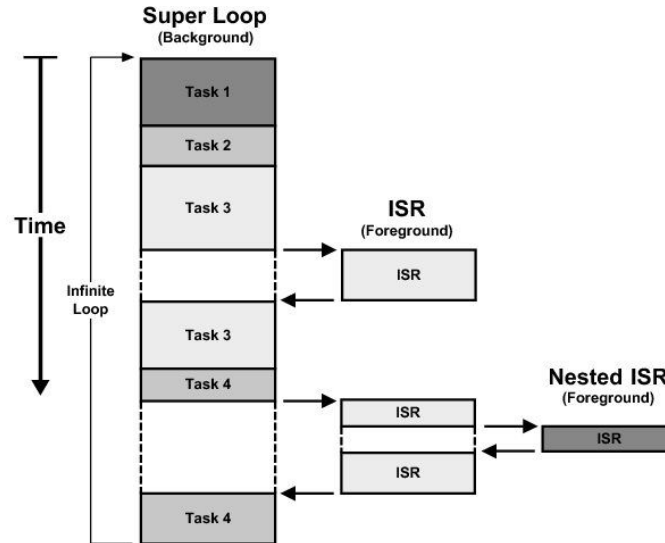
O QUE IMPLEMENTAMOS

- Criar o código de inicialização usando o CubeMX
- Configurar os periféricos UART, SPI, I2C e FreeRTOS
- Implementar exemplos usando os principais recursos do FreeRTOS.
- Adicionar e configurar a lib RTT
- Criar um projeto para trace do FreeRTOS usando System Viewer



SISTEMAS SUPER LOOP

- Sistemas Foreground/background



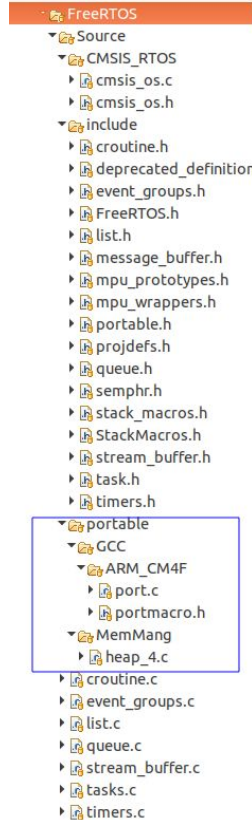


SISTEMAS OPERACIONAIS DE TEMPO REAL

- **Núcleo não preemptivo / Núcleo Cooperativo:** Para a troca de contexto à tarefa deve desistir explicitamente do controle do processador. Liberar a CPU através da macro `taskYIELD()`.
- **Núcleo preemptivo:** A cada interrupção o chamada do sistema o núcleo reavalia qual tarefa deve ser executada. O controle da CPU sempre será dado à tarefa de maior prioridade.



ESTRUTURA FREERTOS





RESPONSABILIDADES DE UM RTOS

- Gerenciar a comunicação entre as tarefas.
- Gerenciar a comunicação entre interrupções e tarefas.
- Gerenciar o acesso aos recursos da aplicação (hardware, estruturas de dados, etc).
- Gerenciar o uso de memória.
- Prover outras funcionalidades como timers, tracing, etc.

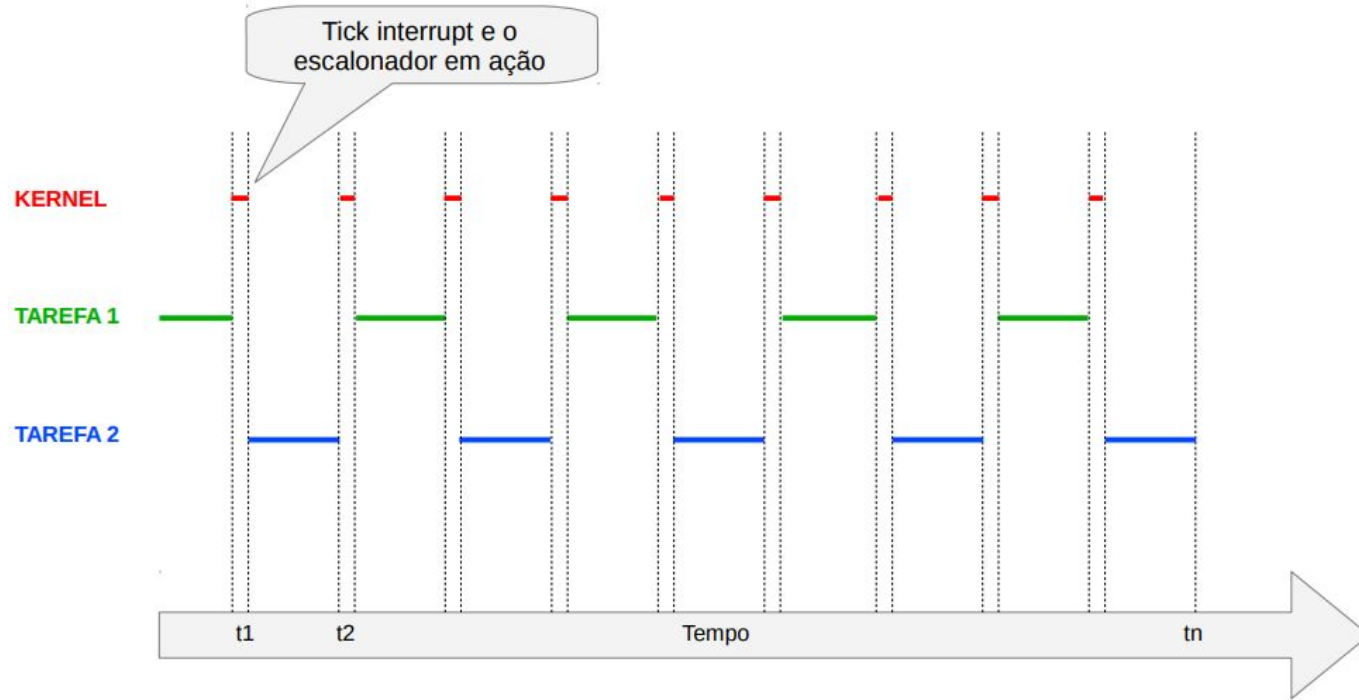


TIMER TICK

- Tempo de passo usualmente configurado entre 1 é 100ms
- Para interromper a tarefa em execução e trocar de contexto para uma nova tarefa, o kernel usa uma interrupção do sistema.



TICK INTERRUPT E PREEMPÇÃO





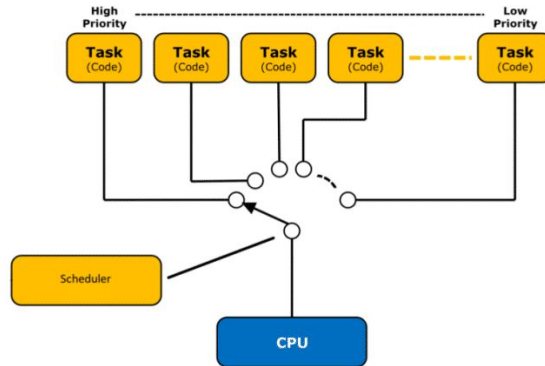
MUDANÇA DE CONTEXTO

- Enquanto uma tarefa está em execução, ela possui determinado contexto (stack, registradores da CPU, etc).
- Ao mudar a tarefa em execução, o kernel salva o contexto da tarefa a ser suspensa e recupera o contexto da próxima tarefa a ser executada.
- O controle do contexto de cada uma das tarefas é realizado através de uma estrutura interna do RTOS.



0 ESCALONADOR

- O escalonador de tarefas entra em ação durante as mudanças de contexto.
- Ele é a parte do kernel responsável por decidir qual é a próxima tarefa a ser executada em determinado momento.
- O algoritmo responsável por decidir qual é a próxima tarefa a ser executada é chamado de política de escalonamento.





GERENCIAMENTO DE MEMÓRIA

- O FreeRTOS irá alocar memória dinamicamente toda vez que precisar criar um objeto do sistema (tarefa, queue, semáforo, etc).
- O FreeRTOS usa rotinas de alocação e desalocação de memória dinâmica, portanto ela implementa suas rotinas malloc() e free().



ALOCAÇÃO NO FREERTOS

- **heap_1.c:** apenas aloca memória.
- **heap 2.c:** aloca e desaloca memória, mas não trata fragmentação.
- **heap_3.c:** usa a implementação padrão de malloc() e free() da biblioteca C.
- **heap_4.c:** aloca e desaloca memória, trata fragmentação e é mais eficiente que a maioria das implementações da biblioteca C padrão.
- **heap 5.c:** utiliza o mesmo algoritmo que a heap_4.c, porém permite utilizar como heap regiões não contínuas de memória.



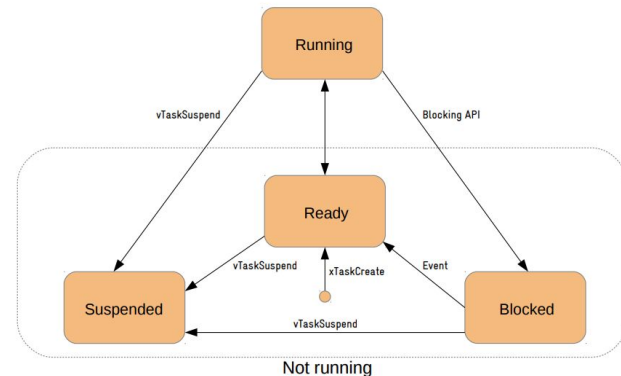
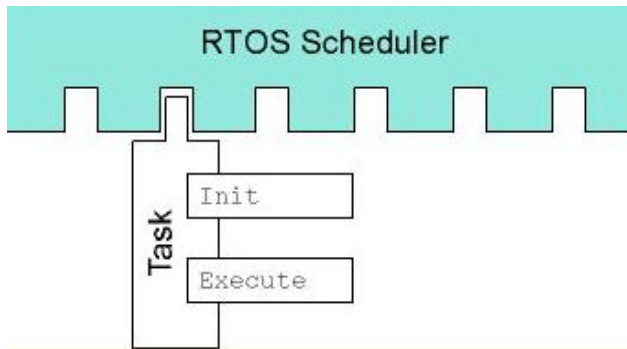
APIs DO FREERTOS

- Principais mecanismos de FreeRTOS:
 - Task Notifications.
 - Queues.
 - Semáforos (binários e contadores).
 - Mutex
 - Event Groups.
 - Queues Sets.
- APIs de sincronização podem ser usados tanto para comunicação entre tarefas quanto para comunicação entre interrupções e tarefas.



TASK

- Cada tarefa se comporta como um programa isolado:
 - Tem um ponto de entrada.
 - É implementada normalmente com um loop infinito.
 - Normalmente não retorna. Se uma tarefa finalizar, é responsabilidade do desenvolvedor removê-la da lista de tarefas do kernel.





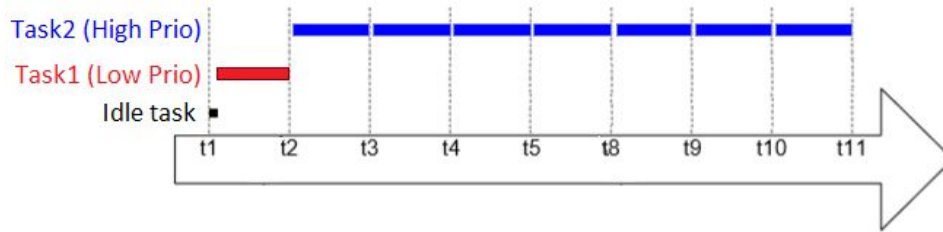
TASK (cont.)

- Cada tarefa pode estar em um determinado estado (Running, Ready, Blocked, Suspended).
- Apenas uma tarefa pode estar no estado Running em determinado momento.
- O escalonador sempre seleciona a tarefa de maior prioridade e no estado Ready para ser executada.



TASK (cont.)

- Toda tarefa tem uma prioridade.
- A menor prioridade do FreeRTOS é 0
- Ex:
 - Idle task: 0 (Menor prioridade)
 - Task1: 1
 - Task2: 5. (Maior prioridade)





TASK (cont.)

- Estado BLOCKED: Uma tarefa esperando um evento está no estado Blocked ou bloqueada.
- Estado SUSPENDED: Tarefas no estado Suspended não são escalonadas (executadas) pelo kernel.
- Estado READY: Estas tarefas estão aguardando na fila, prontas para serem selecionadas e executadas pelo escalonador.
- Estado RUNNING: Uma tarefa em execução é a tarefa atualmente alocada na CPU, é ela que está em processamento.

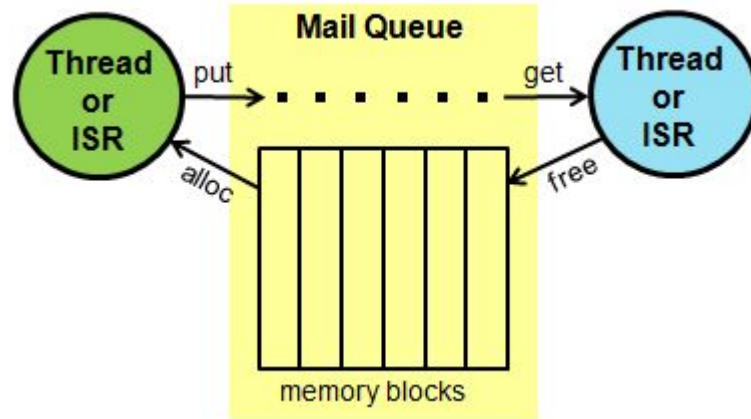


Queue

- Queue é um mecanismo de comunicação (troca de mensagens) entre tarefas ou entre uma tarefa e uma interrupção.
- Um queue não pertence à nenhuma tarefa em específico.
- Diversas tarefas e interrupções podem compartilhar o mesmo queue, tanto para ler, quanto para escrever.
- Diversas tarefas e interrupções podem compartilhar o mesmo queue, tanto para ler, quanto para escrever.
- Cada item do queue pode ter um tamanho fixo de bytes



QUEUE (cont.)





Problemas Frequentes

- **Inversão de Prioridade:** Uma tarefa de menor prioridade causa o bloqueio de uma tarefa de maior prioridade ao consumir um recurso compartilhado.
 - Solução: Usar gatekeeper(tarefa com acesso exclusivo à um recurso compartilhado).
- **Deadlock:** dois ou mais recursos ficam impedidos de continuar suas execuções devido à um compartilhamento mútuo de recursos dependentes um do outro.
 - Solução: Começar a aquisição de recursos pelo recurso de maior prioridade.
 - Usar gatekeeper



Problemas Frequentes

- **Sobrecarga (Overload):** O RTOS não consegue executar as tarefas solicitadas sem perda de prazos. Tarefas de menor prioridade são prejudicadas.
 - Solução: Revisar arquitetura
 - Adquirir um microcontrolador com mais recursos (RAM, Clock)

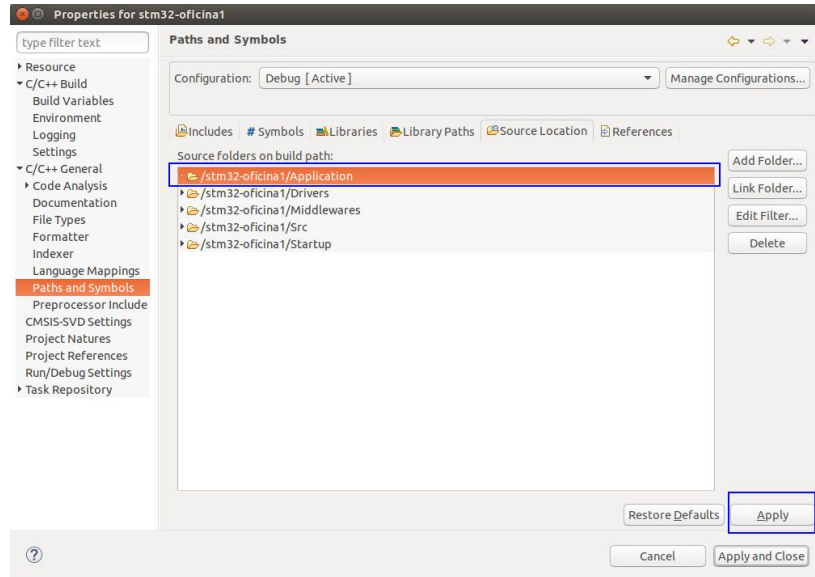
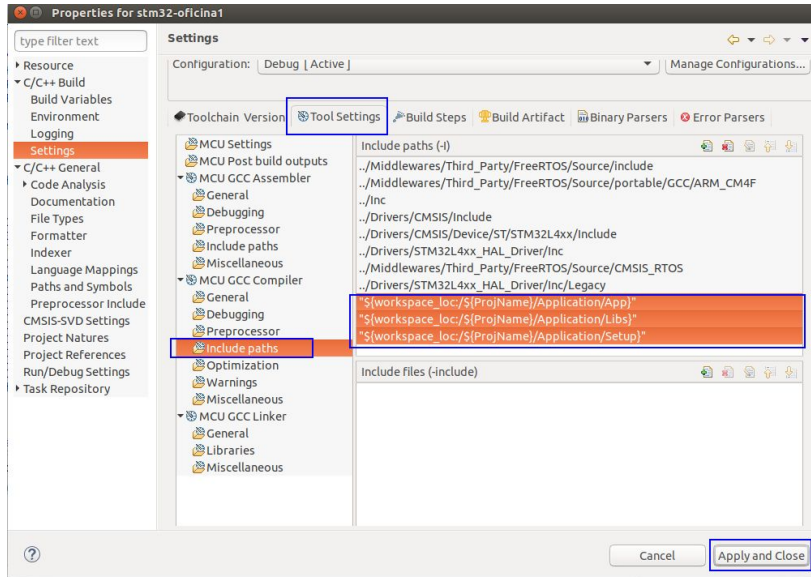


MÃOS À OBRA



Configurando o Projeto

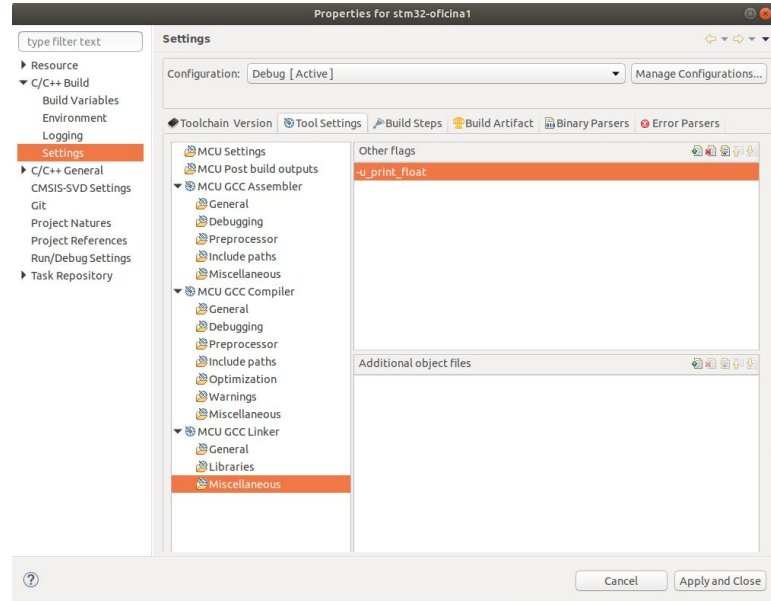
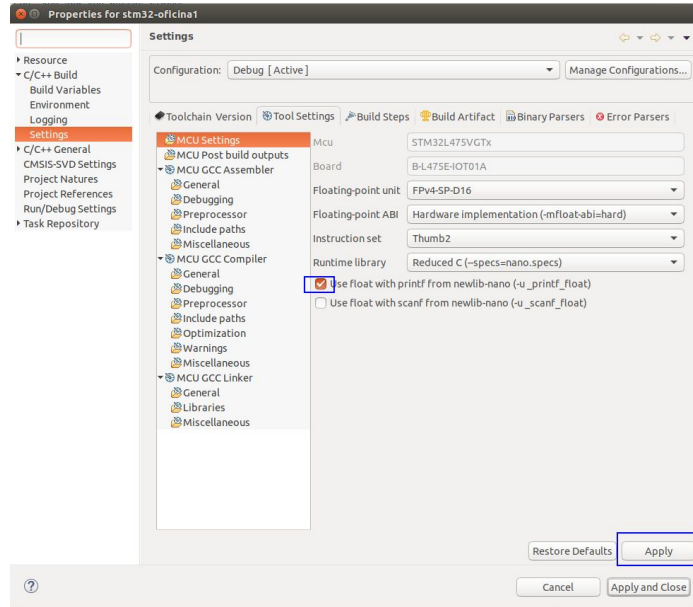
- Adicionando e mapeando libs externas ao projeto.





Configurando o Projeto

- Habilita o uso de %f para imprimir números float.
- Adicione a flag "-u_print_float"





SEMÁFORO BINARIO

- Um Semáforo Binário (Binary Semaphore) é um mecanismo de sincronização disponibilizado pelo FreeRTOS.
- Ele pode ser usado para acordar (desbloquear) uma tarefa quando determinada interrupção acontecer, sincronizando a interrupção com a tarefa.
- Desta forma, apenas o essencial é executado na interrupção, o restante do trabalho é deferido para a tarefa correspondente ao tratamento da interrupção.

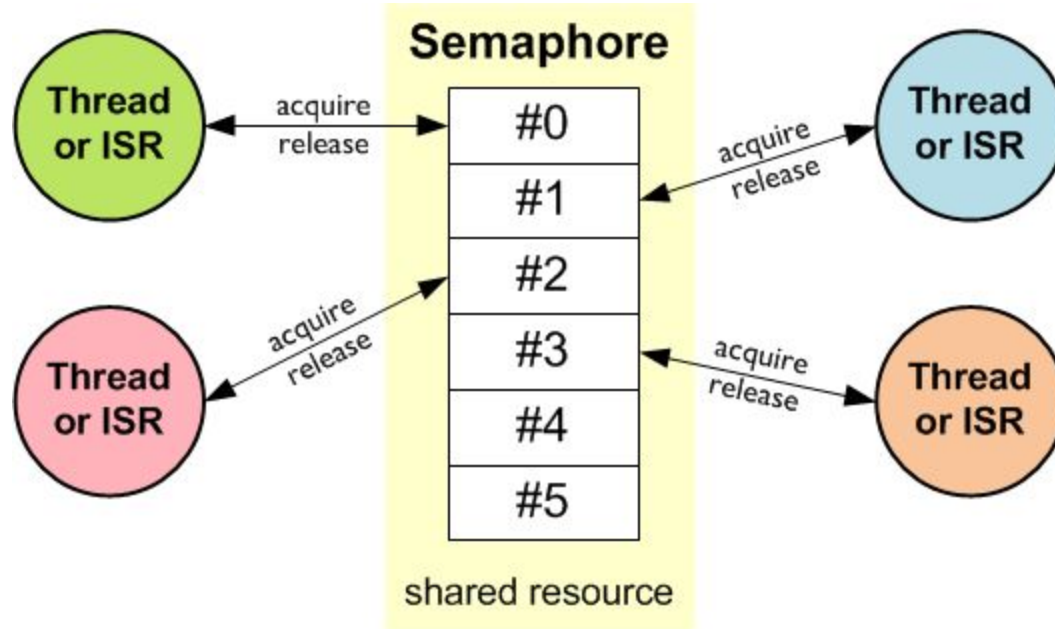


SEMÁFORO BINARIO

- Um Semáforo Binário (Binary Semaphore) é um mecanismo de sincronização disponibilizado pelo FreeRTOS.
- Ele pode ser usado para acordar (desbloquear) uma tarefa quando determinada interrupção acontecer, sincronizando a interrupção com a tarefa.
- Desta forma, apenas o essencial é executado na interrupção, o restante do trabalho é deferido para a tarefa correspondente ao tratamento da interrupção.



SEMÁFORO BINARIO





TASK NOTIFICATIONS API

- Este recurso é chamado de Task Notification e pode substituir o uso de semáforos quando apenas uma tarefa é responsável por tratar determinado evento do sistema.
- Para substituir o semáforo por uma Task Notification, a tarefa deve esperar por notificações através da função `ulTaskNotifyTake()`.
- Da mesma forma, as notificações podem ser enviadas com a função `xTaskNotifyGive()`.

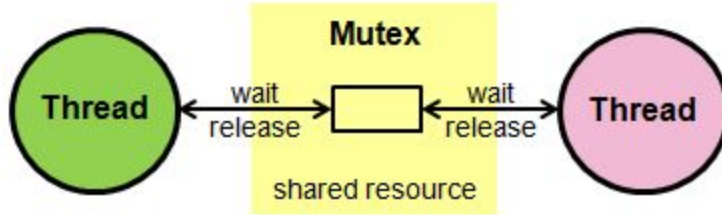


MUTEX

- O mutex é um tipo especial de semáforo binário usado para proteger o acesso a um recurso compartilhado por mais de uma tarefa.
- Para uma tarefa acessar o recurso, ela precisa antes pegar o token
- Se o recurso estiver ocupado, ela deverá esperar a liberação do recurso para poder usá-lo.
- Ao terminar de usar o recurso, ela deverá liberá-lo (give)
- O recurso não pode ser liberado em Interrupções.



MUTEX





TASK E QUEUE

```
QueueHandle_t QueueLeds = xQueueCreate(4, sizeof(MsgLed_t));  
DBG_ASSERT_PARAM(QueueLeds);  
  
xReturned = xTaskCreate(Leds_Task, "LedsTimer", configMINIMAL_STACK_SIZE, QueueLeds, 3, NULL);  
configASSERT(xReturned);
```

```
typedef struct  
{  
    LedsIndex_e index;  
    LedsAction_e action;  
}MsgLed_t;  
  
static void Leds_Task(void *param)  
{  
    QueueHandle_t *pQueue;  
    pQueue = (QueueHandle_t*)param;  
    MsgLed_t MessageLed;  
  
    for(;;)  
    {  
        /* Verifica se recebeu algum comando */  
        if(xQueueReceive(pQueue, &MessageLed, 10) == pdTRUE)  
        {  
            ...  
        }  
    }  
}
```

```
void Leds_Set(LedsIndex_e in_leds, LedsAction_e action)  
{  
    MsgLed_t msgLed;  
    msgLed.index = in_leds;  
    msgLed.action = action;  
  
    ⚡ xQueueSend(QueueLeds, &msgLed, 1000);  
}
```



TASK E SEMÁFORO

```
SemaphoreHandle_t xSemaphoreShell = xSemaphoreCreateBinary();  
configASSERT(xSemaphoreShell);
```

```
xReturned = xTaskCreate(vTaskShell, "tkSheel", configMINIMAL_STACK_SIZE * multi_stack_size, NULL, 3, NULL);  
configASSERT(xReturned);
```

```
static void vTaskShell(void *param)  
{  
    for(;;)  
    {  
        /* Verifica se recebeu algum comando */  
        xSemaphoreTake(xSemaphoreShell, portMAX_DELAY);  
        ...  
    }  
}
```

```
void Shell_ISR_Getc(uint8_t c, BaseType_t pHigherPriorityTaskWoken)  
{  
    if(xSemaphoreShell != NULL)  
    {  
        ...  
        ⚡ xSemaphoreGiveFromISR(xSemaphoreShell, pHigherPriorityTaskWoken);  
        ...  
    }  
}
```




MUTEX

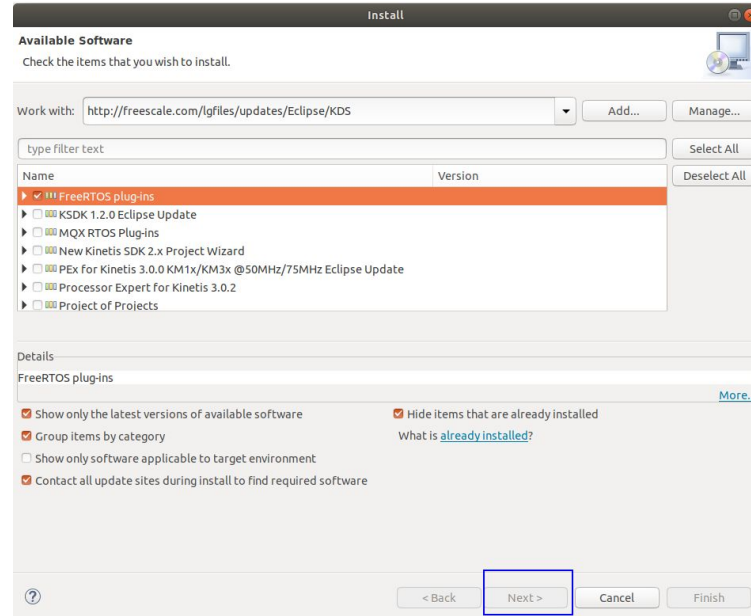
```
SemaphoreHandle_t mutex_debug = xSemaphoreCreateMutex();  
configASSERT(mutex_debug);
```

```
HAL_StatusTypeDef Debug_Printf(char * format, ...)  
{  
    va_list args;  
    uint16_t len;  
    HAL_StatusTypeDef resp = HAL_ERROR;  
  
    ⚡ if( xSemaphoreTake(mutex_debug, 1000) == pdTRUE )  
    {  
        va_start(args, format);  
        len = vsnprintf((char *) bufferSerial, SERIAL_BUFFER_SIZE, format, args);  
        va_end(args);  
  
        resp = HAL_UART_Transmit(pUartDebug, bufferSerial, len, 1000);  
  
        xSemaphoreGive(mutex_debug);  
    }  
    return resp;  
}
```



INSTALANDO PLUGIN DO FREERTOS

- Help > Install new Software...
- Link: <http://freescale.com/lgfiles/updates/Eclipse/KDS>



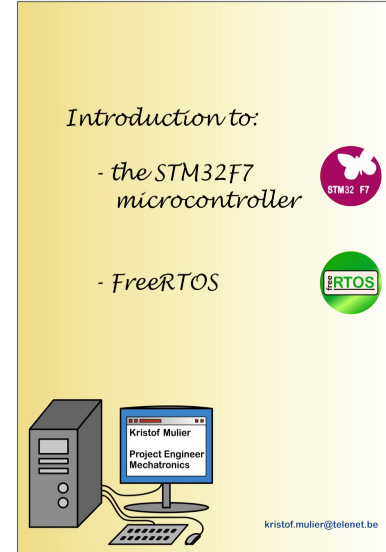
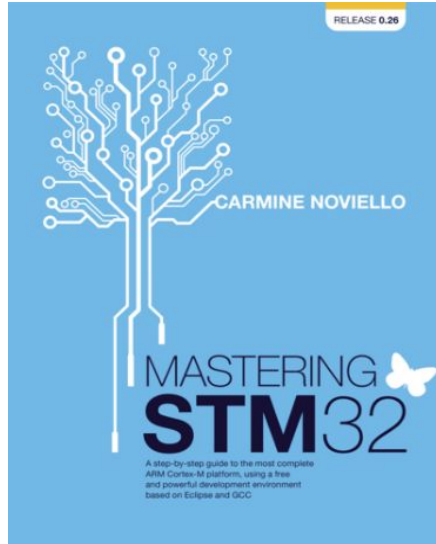


Canais no YouTube

- millsinghion
- [MYaqoobEmbedded](#)
- [Snømann Ingeniør](#)
- [narod stream](#)



Livros



Referências:

https://docs.aws.amazon.com/pt_br/freertos-kernel

<https://e-labworks.com/treinamentos/freertos/slides>



Mídias Sociais

Telegram

https://t.me/lhc_campinas

Facebook:

<https://www.facebook.com/LabHackerCampinas/>

Web:

<https://lhc.net.br/wiki/Categoria:Eventos>