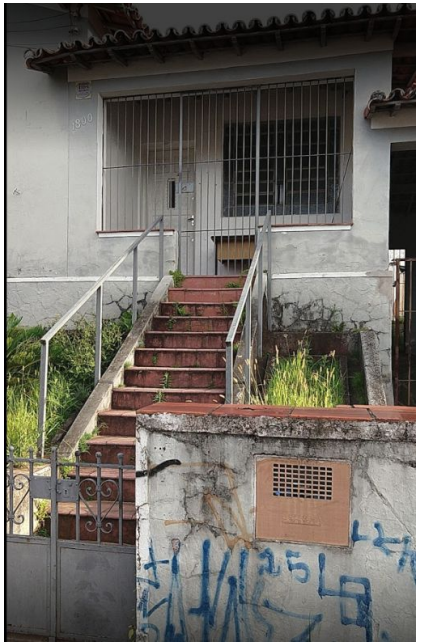
Aprendendo a programar microcontroladores ARM da ST com FreeRTOS

Jorge Guzman

O 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.
  - Biblitecas CMSYS-Math, CMSYS-NN.
  - FreeRTOS, FATFS, LWIP, USB, etc.
- Debuggers:
  - ST-Link
  - J-Link

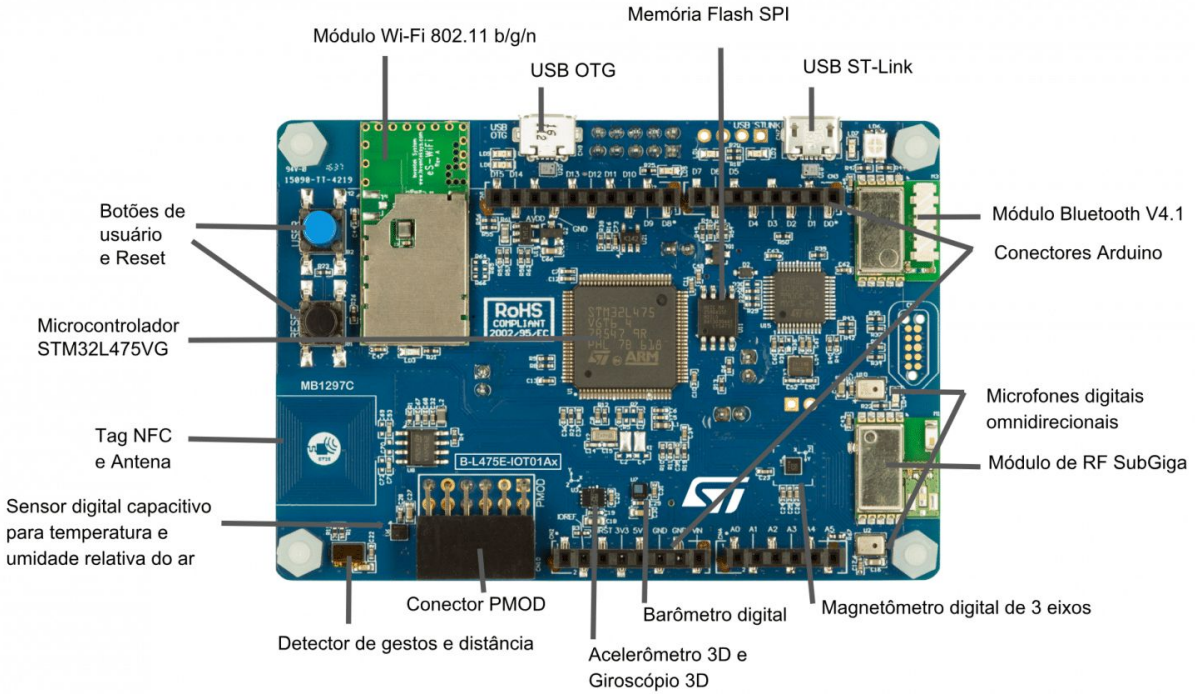
STM32CubeIDE

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

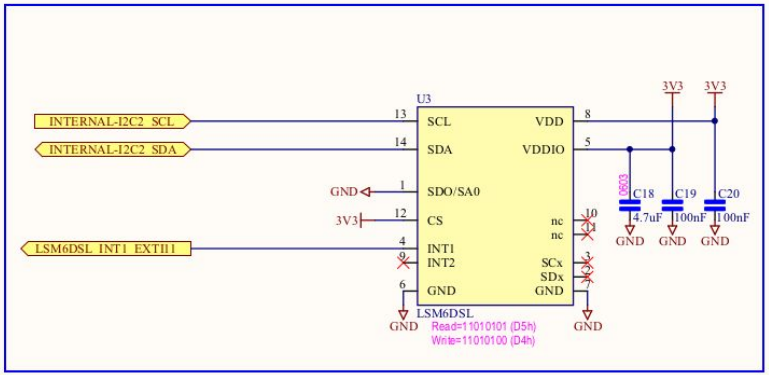
KITS DE DESENVOLVIMENTO

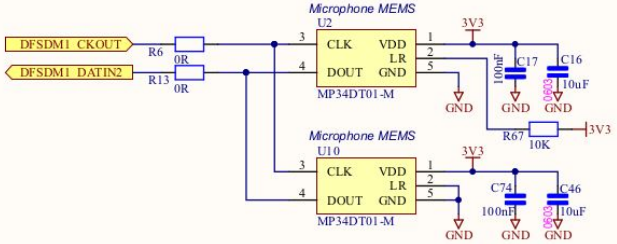
STM32 Nucleo development boards	Discovery kits	Evaluation boards	STM32 Nucleo expansion boards	Third-party boards
Flexible prototyping	Key feature prototyping	Full feature evaluation	Add-on functionalities	From full evaluation to open hardware

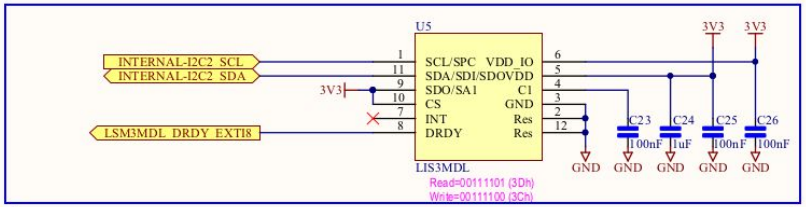
B-L475E-IOT01A

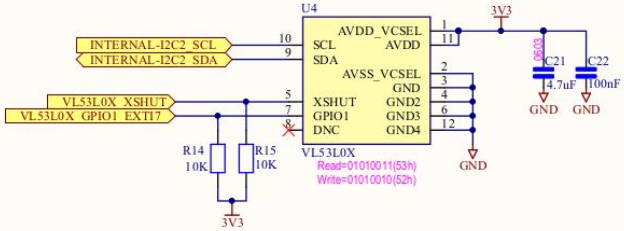


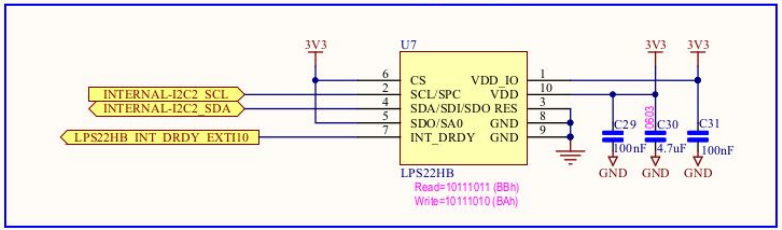
Conexão Elétrica Sensores (I2C)

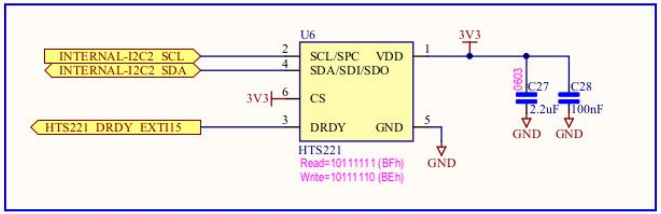




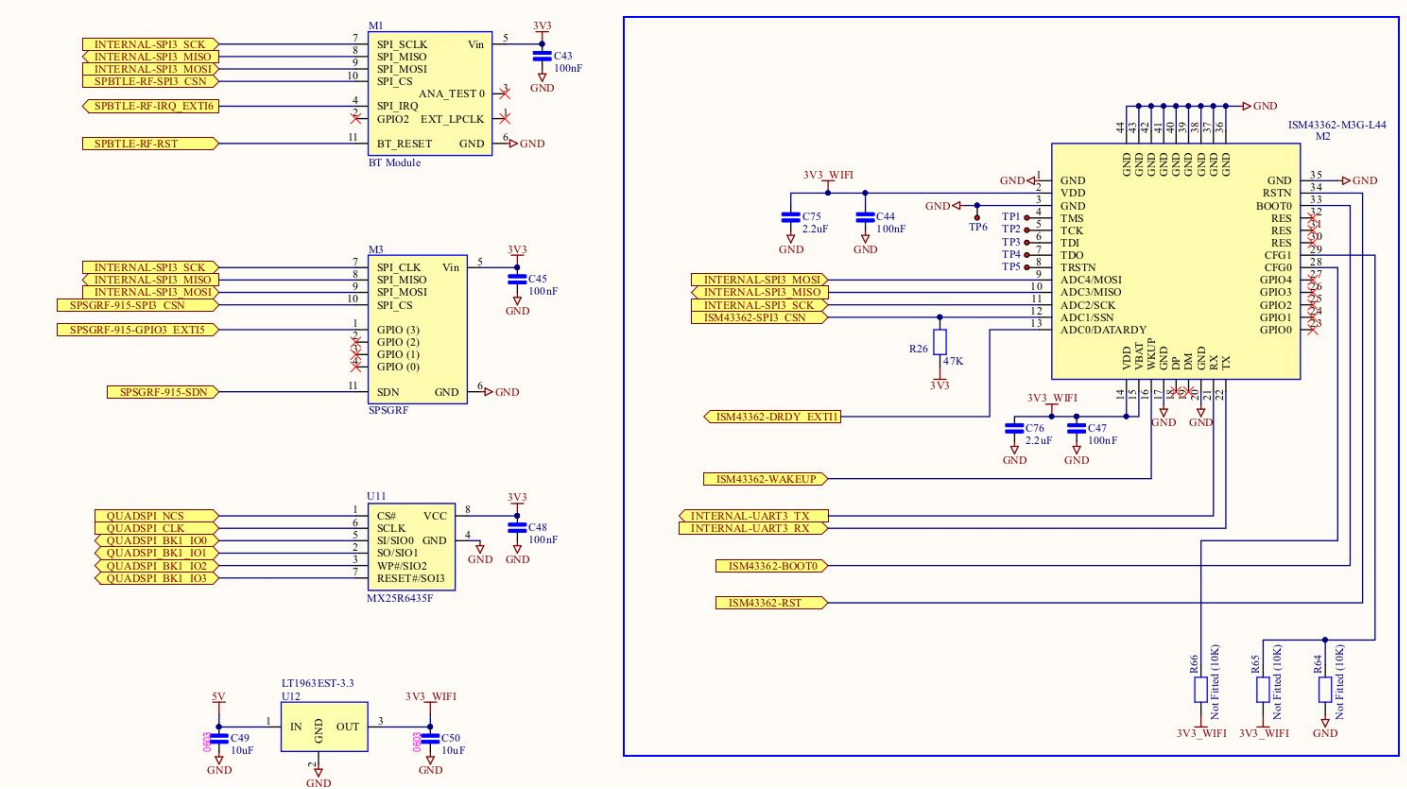








Conexão Elétrica Módulo Wifi (SPI)

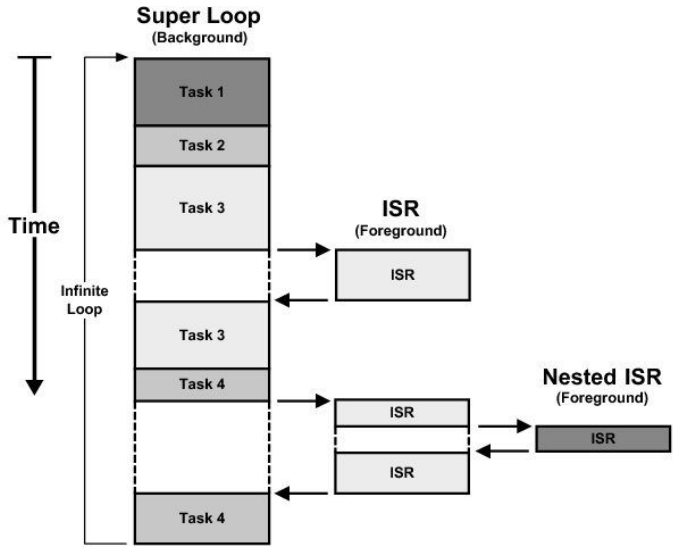


O QUE IMPLEMENTAMOS

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

SISTEMAS SUPER LOOP

- Sistemas Foreground/background

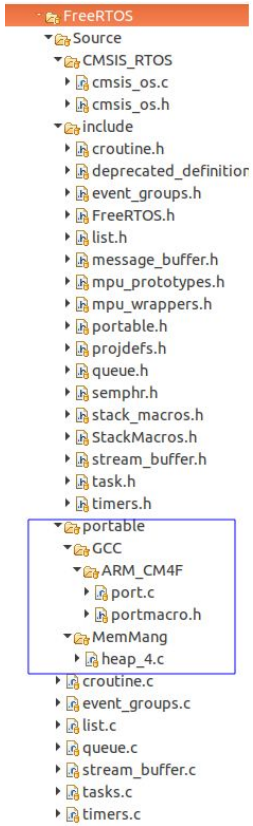


SISTEMAS OPERACIONAIS DE TEMPO REAL

- **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



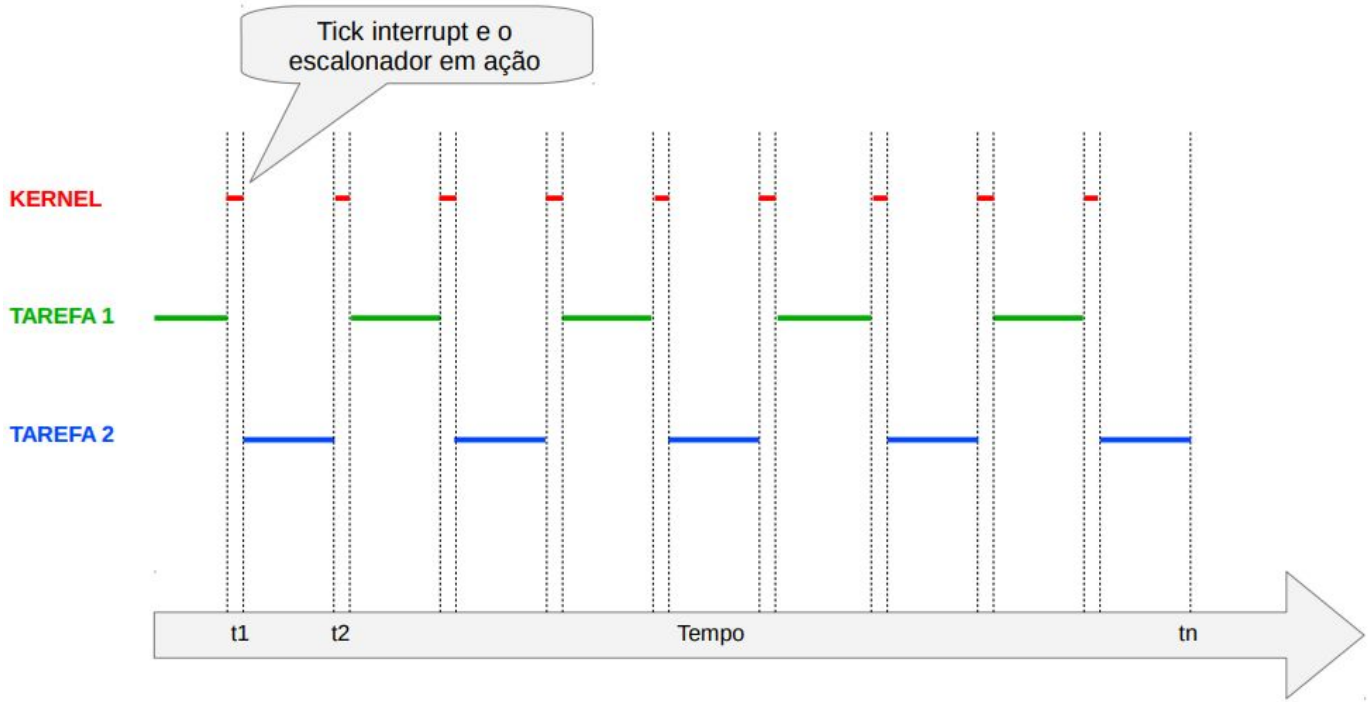
RESPONSABILIDADES DE UM RTOS

- 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

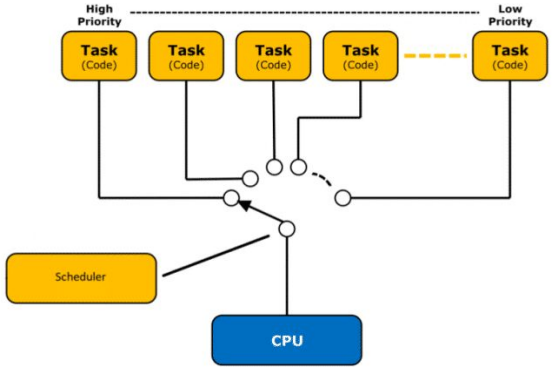


MUDANÇA DE CONTEXTO

- 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.

O 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 de rotinas malloc() e free().

ALOCAÇÃO NO FREERTOS

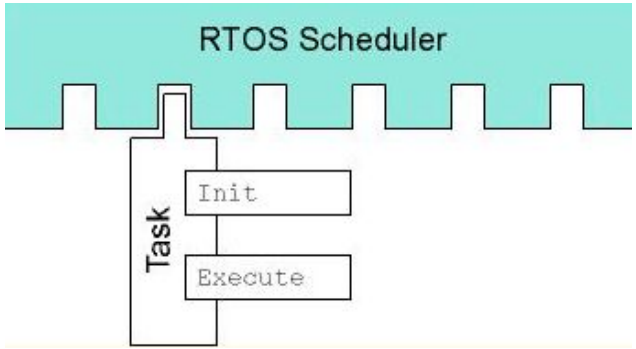
- **heap_1.c:** apenas aloca memória.
- **heap 2.c:** aloca e desaloca memória, mas não trata fragmentação.
- **heap_3.c:** usa a implementação padrão de malloc() e free() da biblioteca C.
- **heap_4.c:** aloca e desaloca memória, trata fragmentação e é mais eficiente que a maioria das implementações 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.

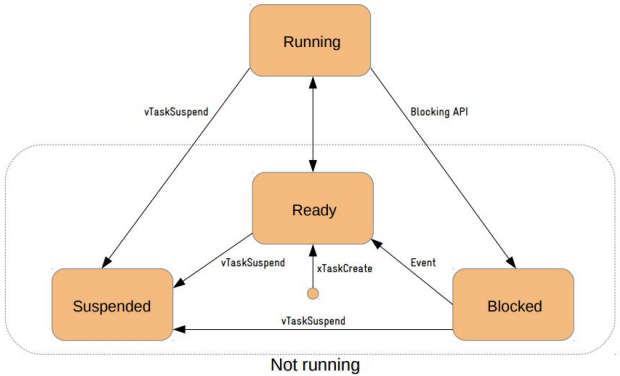
APIs DO FREERTOS

- 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.



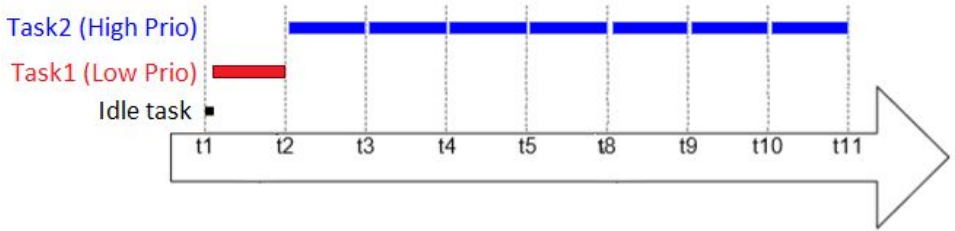


TASK (cont.)

- 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)



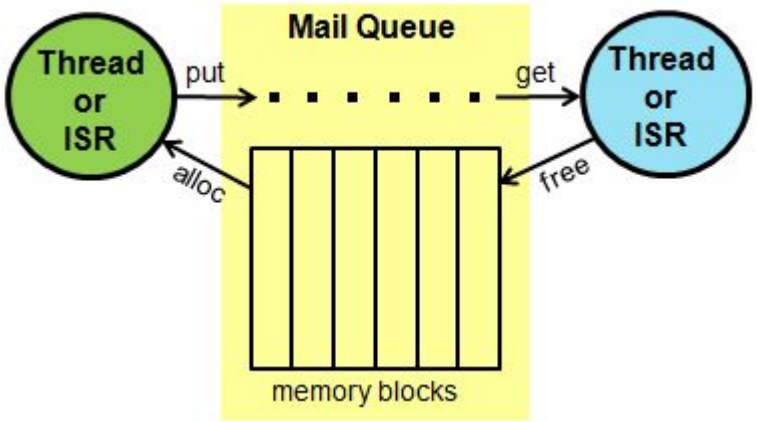
TASK (cont.)

- 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.)


Mail Queue
Thread or ISR
put
get
Thread or ISR
alloc
free
memory blocks

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

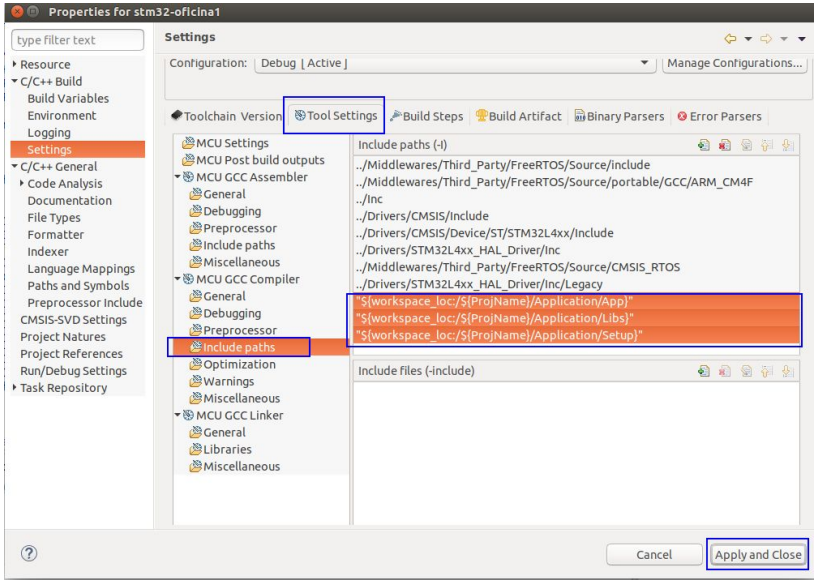
Problemas Frequentes

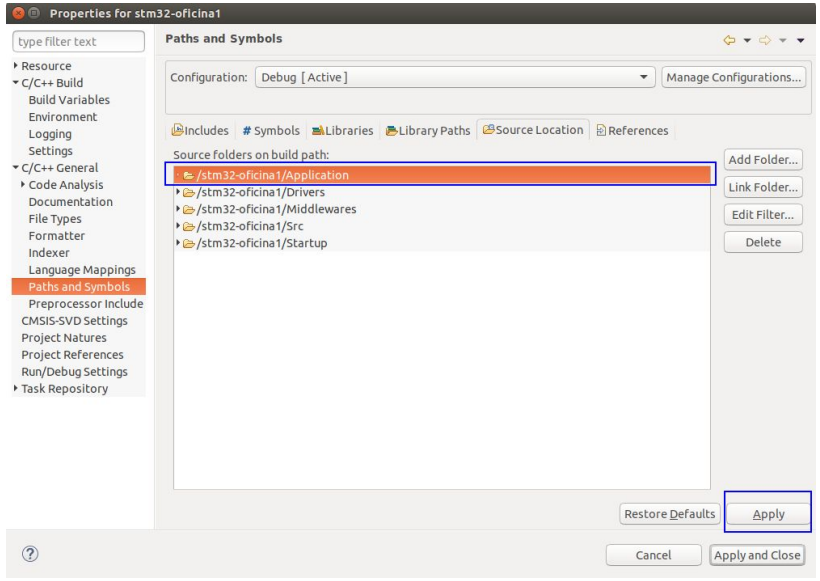
- **Sobrecarga (Overload):** O RTOs não consegue executar as tarefas solicitadas sem perda de prazos.Tarefas de menor prioridade são prejudicadas.
    - Solução: Revisar arquitetura
    - Adquirir um microcontrolador com mais recursos(RAM, Clock)

MÃOS À OBRA

Configurando o Projeto

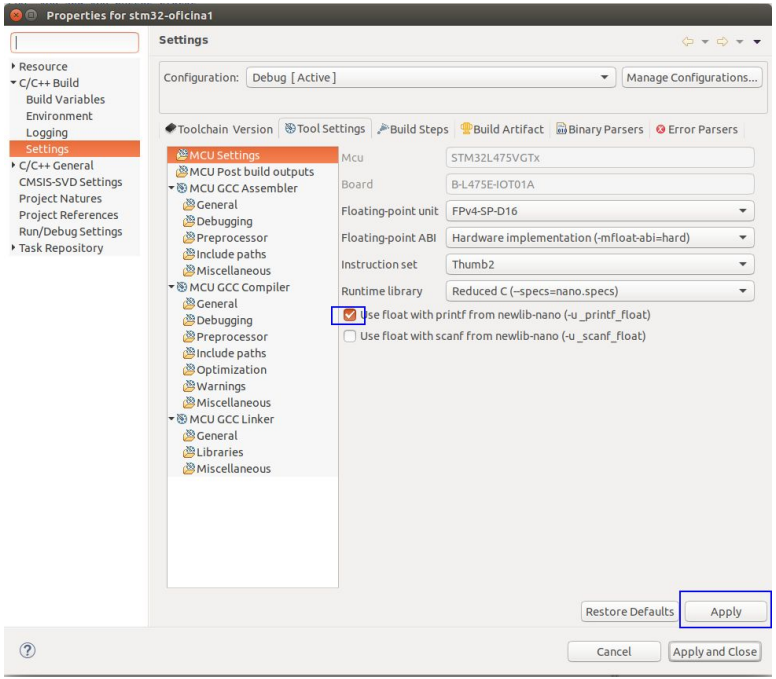
- Adicionando e mapeando libs externas ao projeto.





Configurando o Projeto

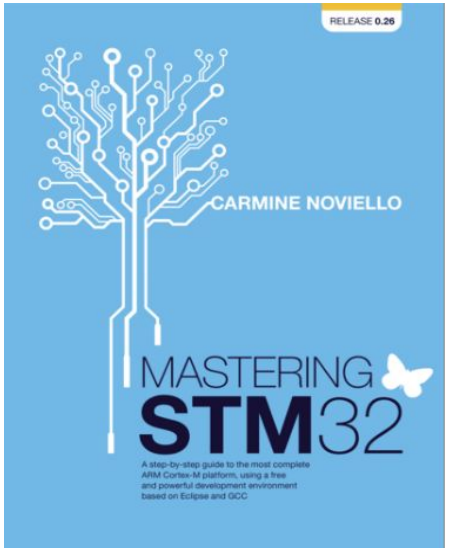
- Habilita o uso de %f para imprimir números float.

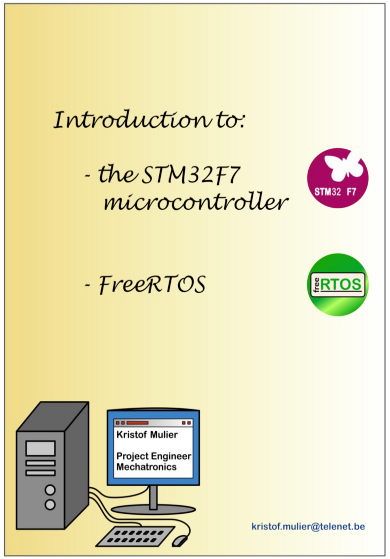


Canais no YouTube

- 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