Universidade Tecnológica Federal do Paraná (UTFPR)
Departamento Acadêmico de Eletrônica (DAELN)

SISTEMAS EMBARCADOS

Programação Concorrente e CMSIS RTOS

Prof. André Schneider de Oliveira

andreoliveira@utfpr.edu.br

Concorrência

- Um programa concorrente descreve diversas atividades que ocorrem **simultaneamente**, de modo diferente de programas comuns, que descrevem apenas uma atividade (ex: função main em linguagem C)

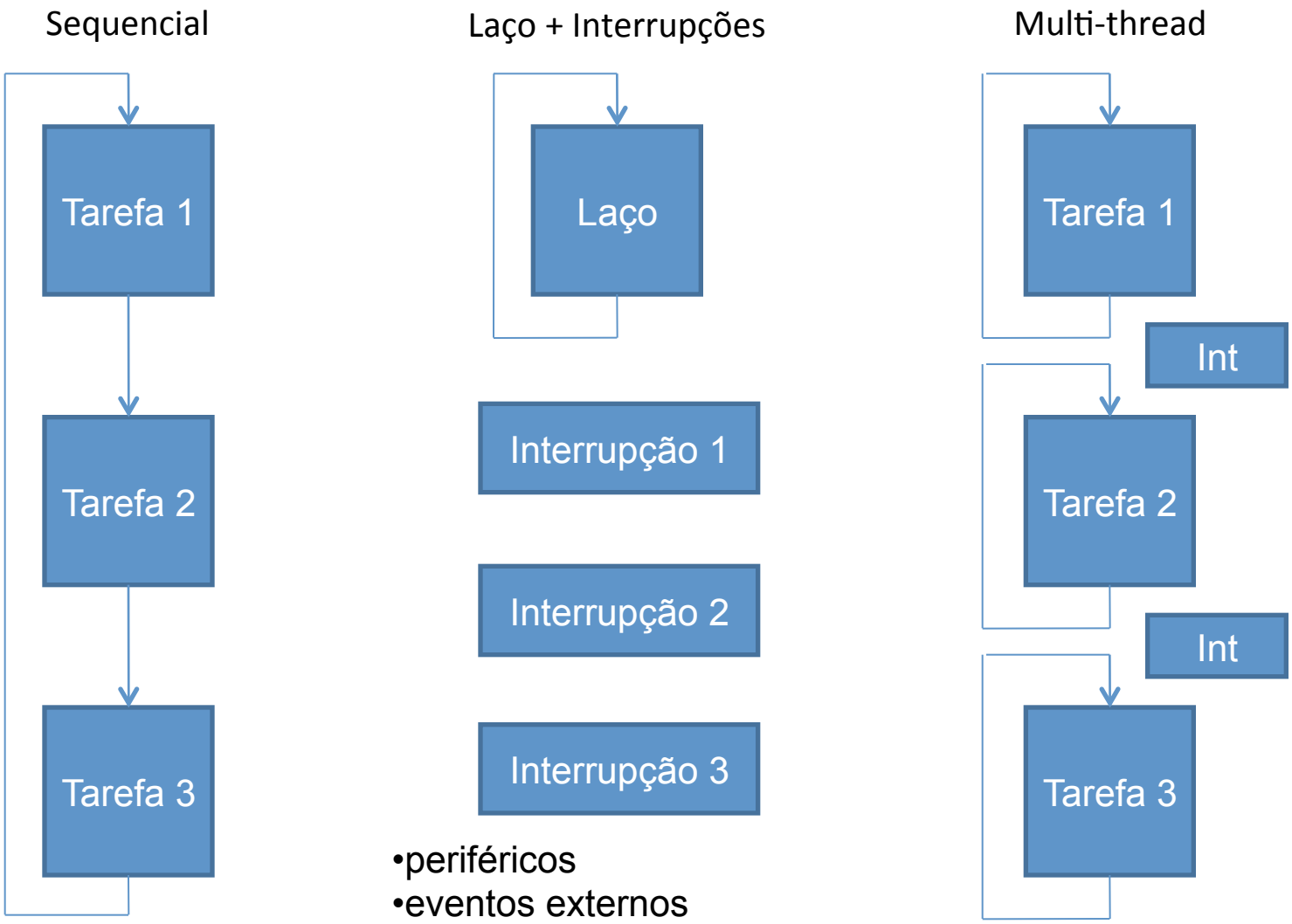
Prog. Sequencial x Prog. Concorrente

- Analogia indivíduo x equipe
  - Indivíduo:
    - Responsável por todas as tarefas
  - Equipe:
    - Ocorre divisão de tarefas entre vários indivíduos

Vantagens do Trabalho em Equipe

- Em uma equipe é menos complexo definir o trabalho de cada indivíduo
- Não há necessidade de um indivíduo mudar de atividade ao longo do tempo
- Há separação entre atividade e controle de atividade (priorização)

Implementação de Concorrência

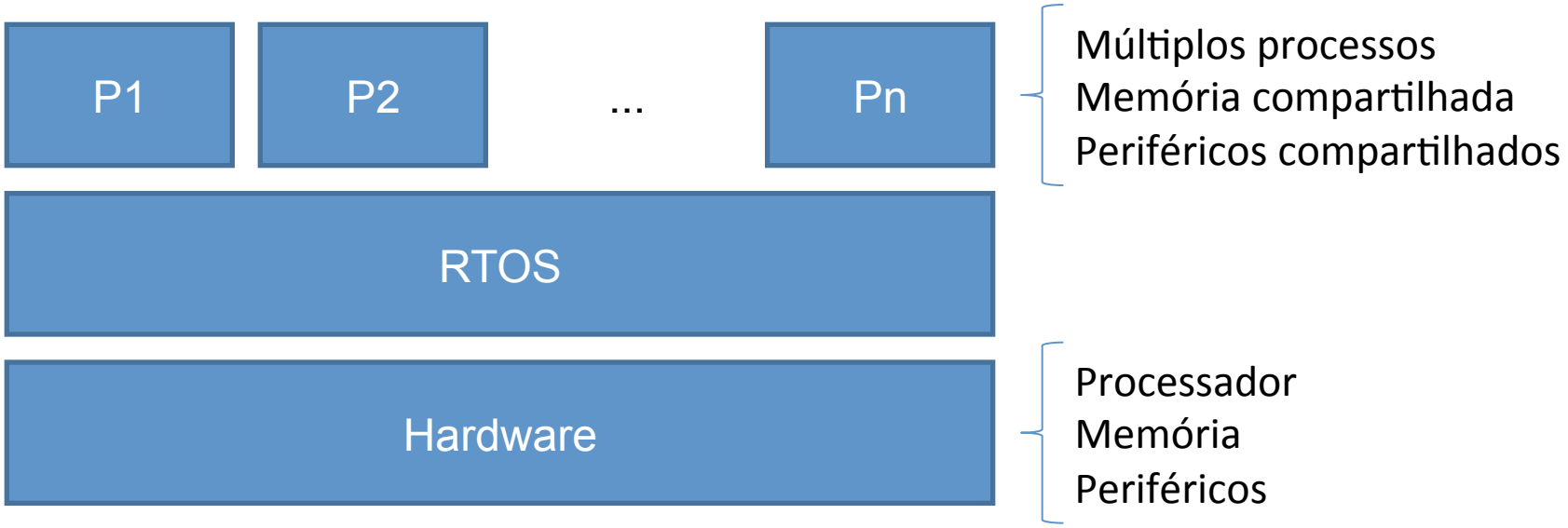


Desafios

- Multi-core
  - Sincronização de tarefas - 1 processo dividido em 2 ou mais threads (ociosidade, paralelismo)
- Mono-core
  - Preempção de tarefas - capacidade de interromper o processo e trocar por outro (troca de contexto, pilha, prioridade)
- Compartilhamento de recursos
  - Processador (chaveamento de contexto)
  - Regiões de memória (variáveis compartilhadas) - conflito
  - Periféricos - múltiplos acessos, acessos simultâneos, conflito de interrupção

Multi-threading

- Múltiplas "linhas" de execução

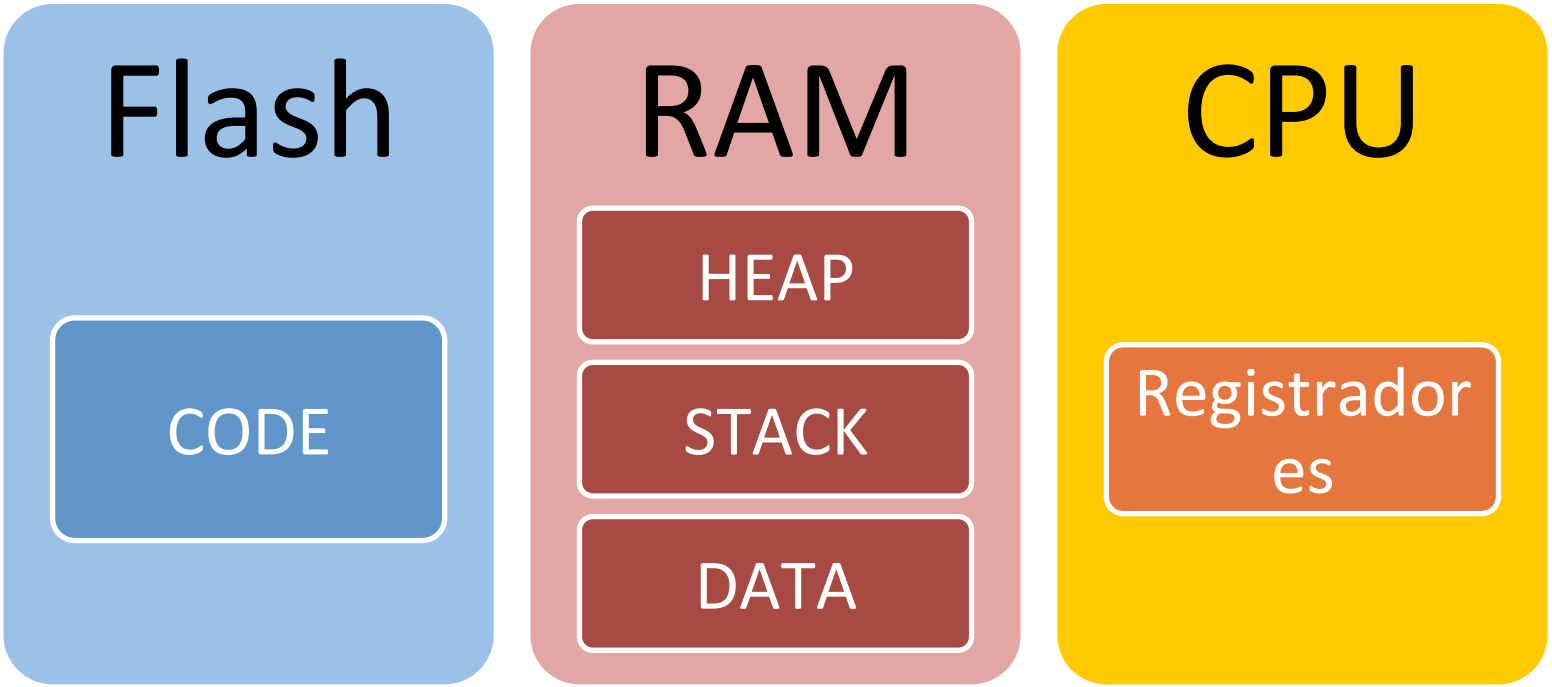


Multi-threading

- criação de ***processadores virtuais*** **idênticos**
  - Registradores e pilha individuais
- Sistemas multi-core
  - Muito mais processadores virtuais do que núcleos
- A soma da capacidade de processamento dos processadores virtuais é igual à capacidade de processamento original

Regiões de Memória

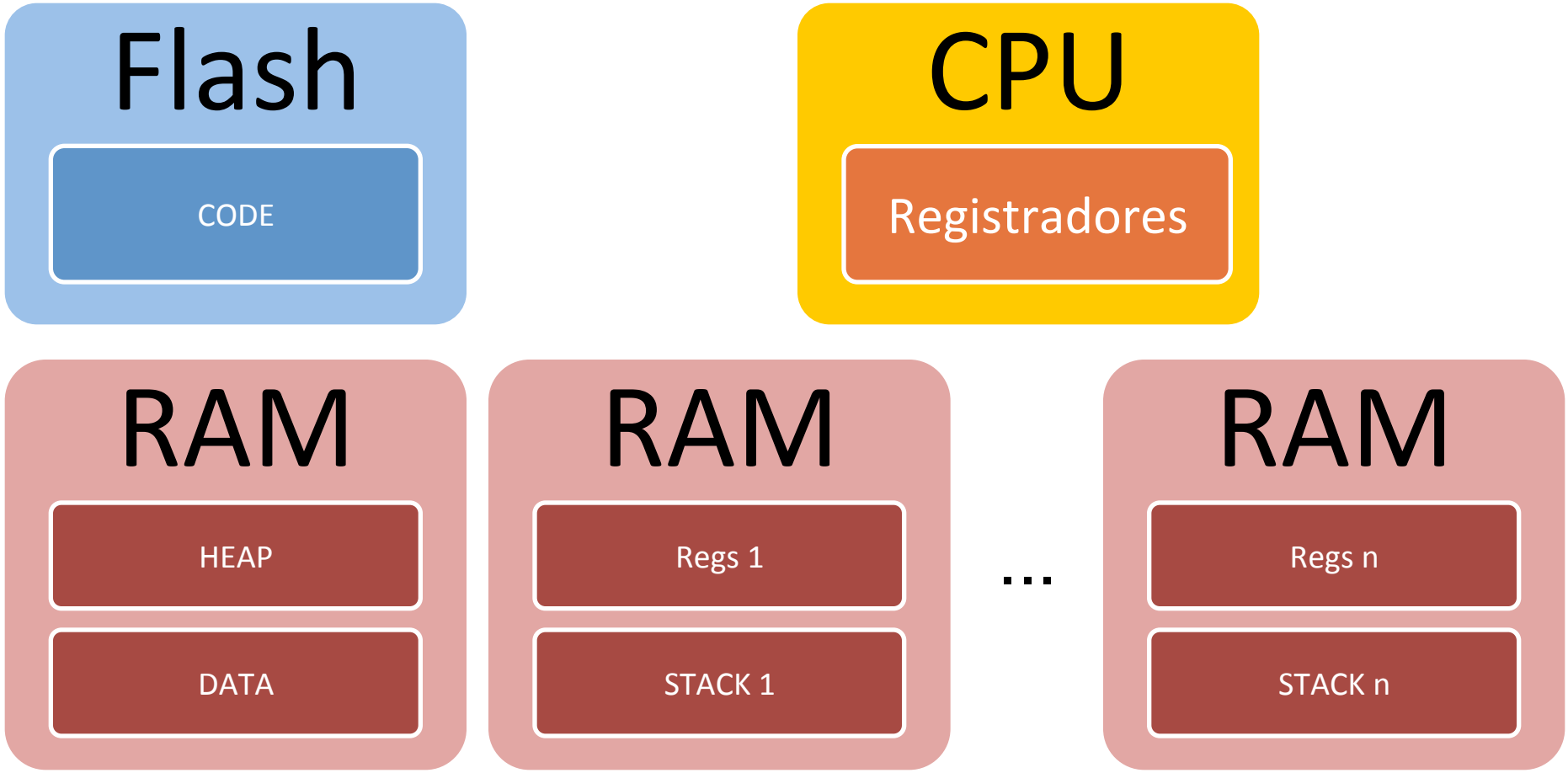
Programação sequencial



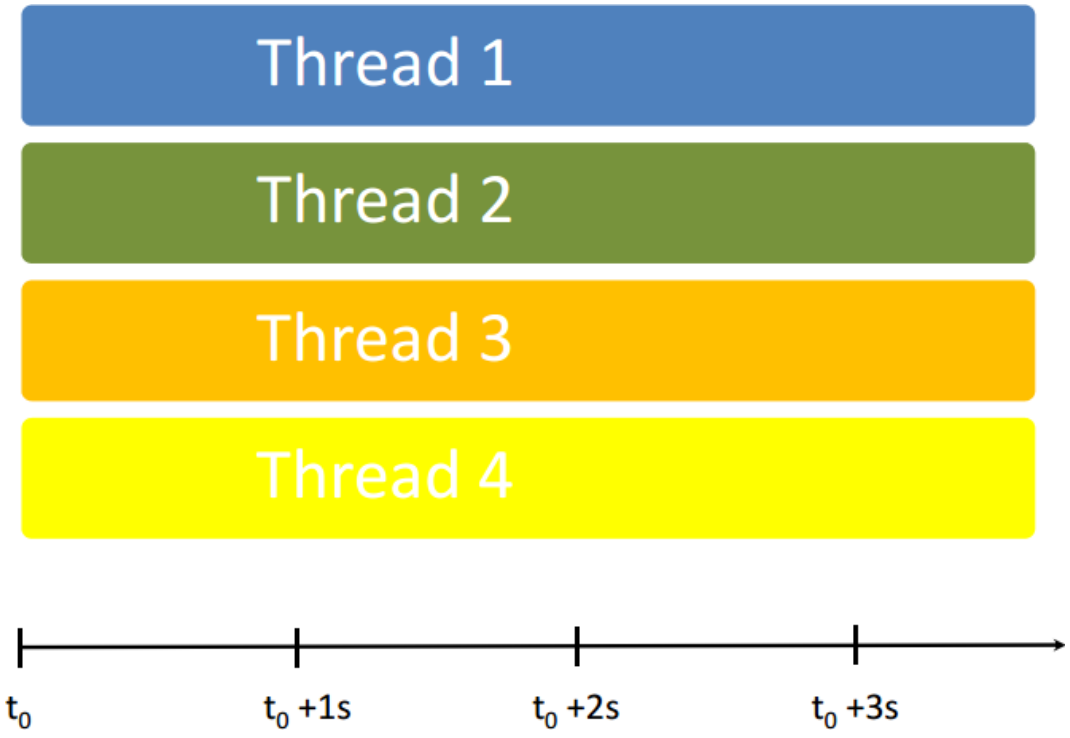
Heap: **alocado dinamicamente ou randomicamente, flexível**
Stack: execução, chamada de função
Data: dados, variáveis

Regiões de Memória

Programação concorrente com Multi-threading

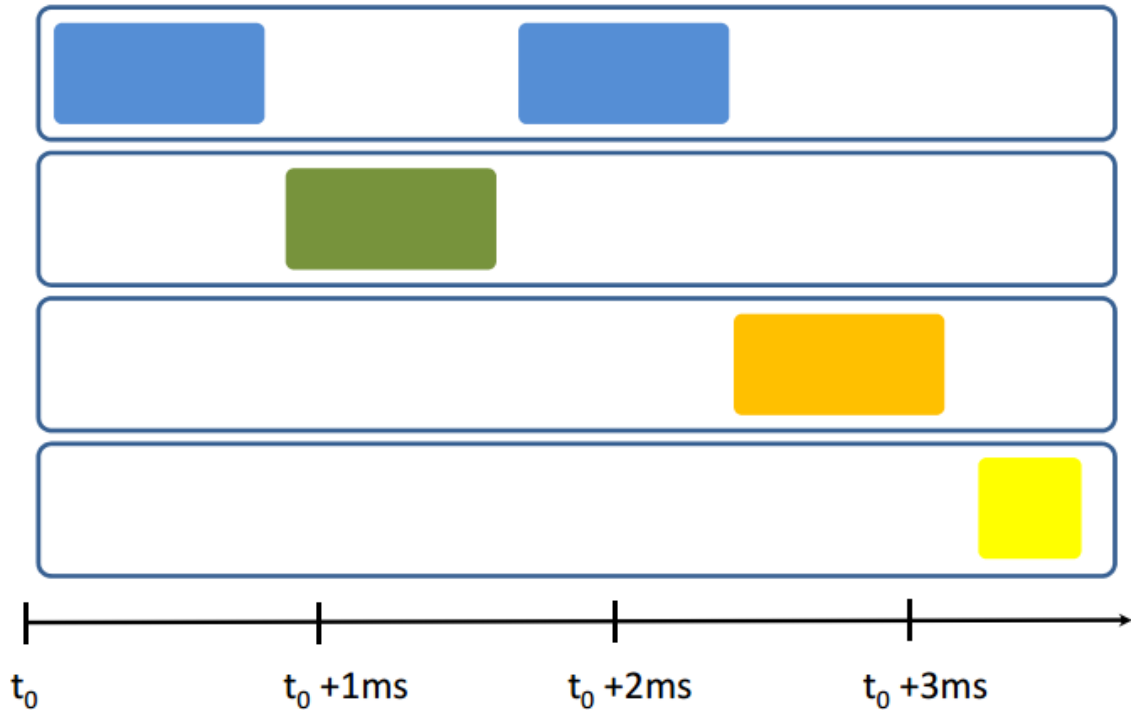


Efeito ao Longo do Tempo (granularidade grossa)



uma thread é iniciada até uma ociosidade (desperdício), nesse ponto inicia outra thread

Efeito ao Longo do Tempo (granularidade fina)



rodízio aproveitando todos os ciclos de clock, sem ociosidade

Efeito ao Longo do Tempo (granularidade)

Thread A

A1 A2 X X A3

Thread B

B1 X X B2 X

Thread C

C1 C2 C3 C4 X

Granularidade fina

A1 B1 C1 A2 B2 C2 A3 C3 C4

Granularidade grossa

A1 A2 X B1 X C1 C2 C3 C4 X A3 B2

Acesso a Recursos Compartilhados

- Acessos múltiplos a recursos compartilhados, *sem modificações de estado*, podem ocorrer em paralelo sem conflitos (leitura x atualização)

- Problemas surgem quando acessos aos recursos compartilhados modificam o seu estado
  - Acesso concorrente a memória compartilhada
  - Acesso concorrente a periférico compartilhado

Conceito de Seção Crítica

- Thread pode ser dividida em **seção crítica** e não-crítica

- A **seção crítica** está relacionada com a área de código que acessa o **recurso compartilhado**

Conceito de Seção Crítica

- Solução: apenas uma thread pode estar acessando a sua seção crítica

- Um único processo pode estar nesta seção em **determinado instante de tempo**

- Certificar-se de que no máximo um processo pode entrar na sua seção crítica (**exclusão mútua**)

Maiores Preocupações da Programação Concorrente

- Exclusão mútua
  - Apenas um processo na seção crítica em determinado instante de tempo (**concorrentemente)**
- Ausência de impasse (*deadlock*)
  - Se dois ou mais tentarem entrar, ao menos um terá sucesso **(disputa pelo acesso à seção crítica)**

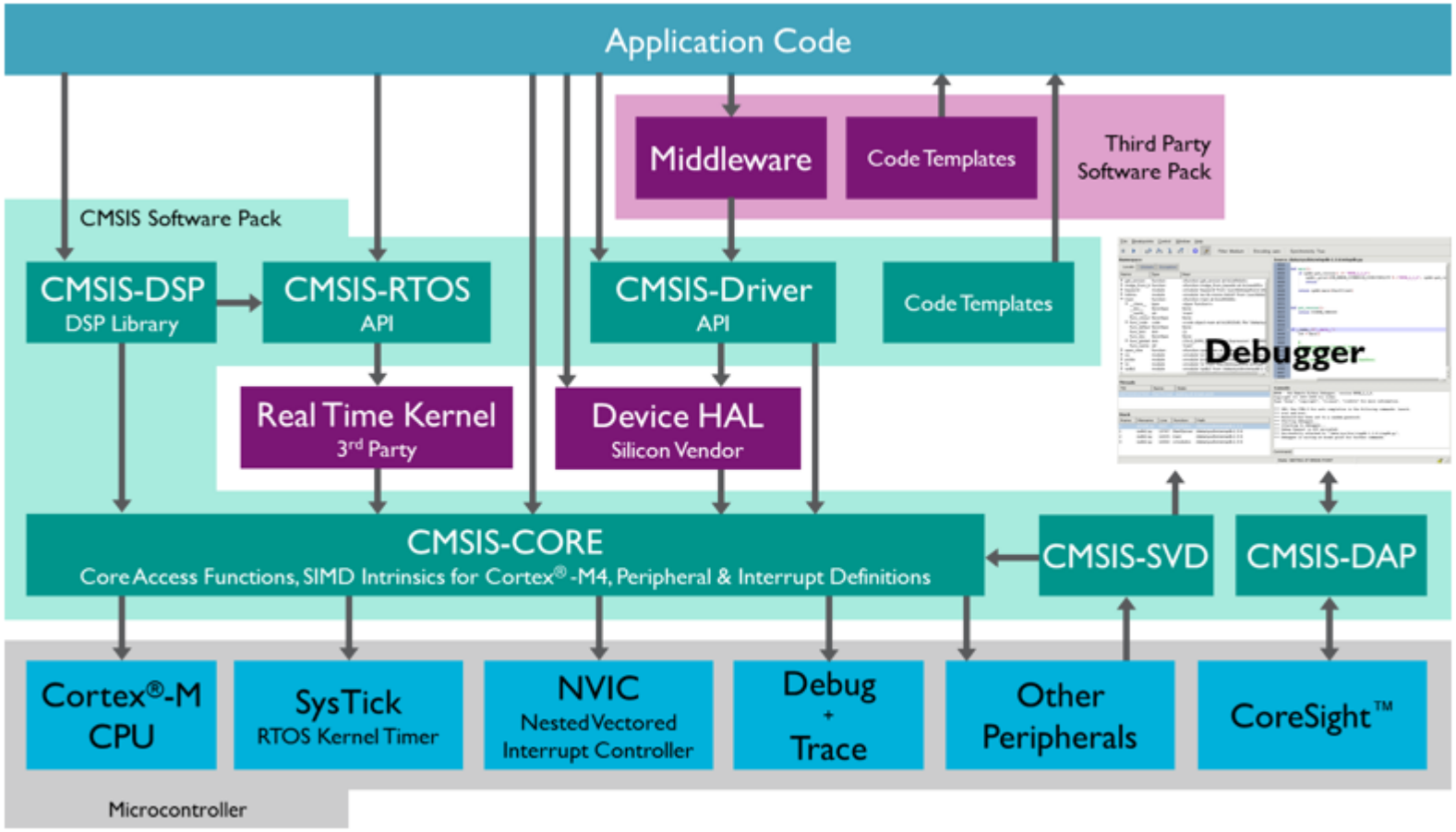
Maiores Preocupações da Programação Concorrente

- Ausência de atrasos desnecessários
  - Se não houver outros processos na seção crítica, um processo tentando entrar **não deve sofrer atrasos**

- Garantia de entrada
  - todas as thread devem ter a oportunidade de acessar a sua seção crítica

CMSIS

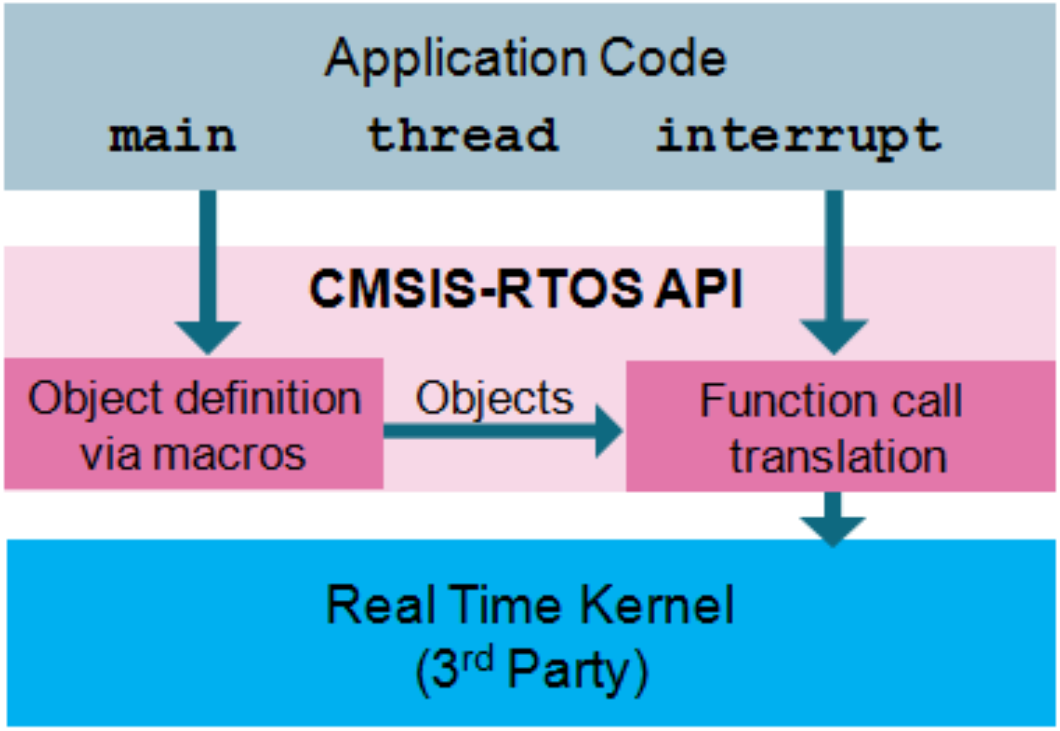
Cortex Microcontroller Software Interface Standard



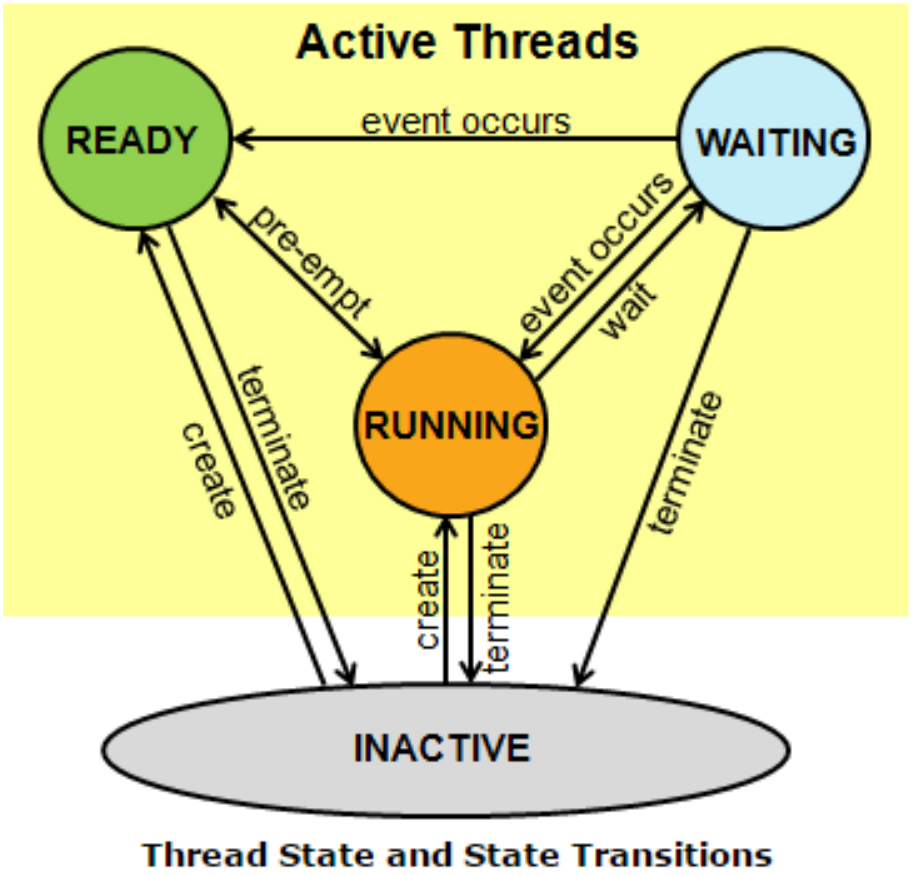
CMSIS Structure

CMSIS RTOS

- Real-Time Operating System (RTOS)
- Segue o padrão CMSIS voltado para a linha Cortex

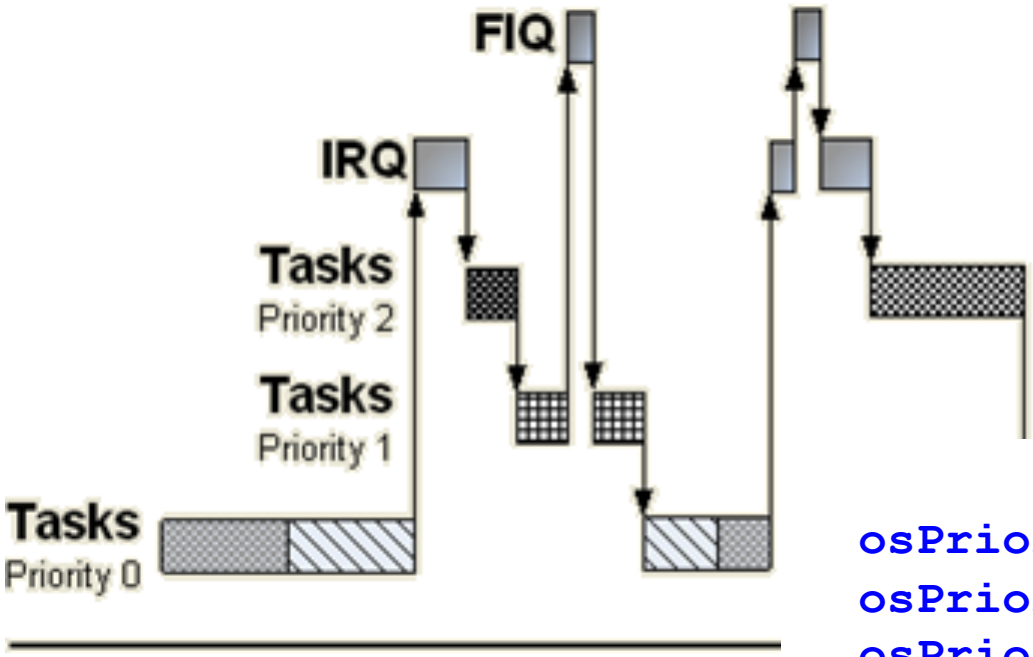


Estado das tarefas no CMSIS RTOS



Thread State and State Transitions

Prioridades




```
osPriorityIdle = -3,
osPriorityLow = -2,
osPriorityBelowNormal = -1,
osPriorityNormal = 0,
osPriorityAboveNormal = +1,
osPriorityHigh = +2,
osPriorityRealtime = +3,
osPriorityError = 0x84
```

Ciclo de Vida dos Elementos do Kernel

- Os elementos do Kernel seguem um ciclo de vida:

1. **Definição**
2. **Inicialização**
3. **Operação**
4. **Término**

Informação e Controle do Kernel

- **`osKernelInitialize`** – Inicializa o kernel
- **`osKernelStart`** – Ativa o kernel
- **`osKernelRunning`** – Consulta se o kernel está ativo
- **`osKernelSysTick`** – Obtém o valor do temporizador do kernel
- **`osDelay`** (função de espera genérica) - Espera por um tempo específico

Definição e Acesso das Threads

`osThreadDef(job1, osPriorityAboveNormal, 1, 0)`

- Macro que cria uma instância de uma estrutura do tipo `osThreadDef_t`, que contém:
  - nome da tarefa
  - prioridade
  - número máximo de instâncias
  - tamanho da pilha em bytes
- O nome desta instância é: `os_thread_def_job1`

`osThread(job1)`

- Macro que é expandida para: `&os_thread_def_job1`, ou seja, um ponteiro para a estrutura criada com `osThreadDef`

Gerenciamento de Tarefas

- **`osThreadCreate`** – Ativa a execução de uma tarefa
- **`osThreadTerminate`** – Desativa a execução de uma tarefa
- **`osThreadYield`** – Passa a execução à próxima tarefa que pronta
- **`osThreadGetId`** – Obtém o identificador que referencia a tarefa
- **`osThreadSetPriority`** – Altera a prioridade de uma tarefa
- **`osThreadGetPriority`** – Obtém a prioridade atual de uma tarefa

Projeto CMSIS RTOS

- **Adicionar o arquivo de configuração RTOS no projeto:**
  RTX_Conf_CM.c

- **Adicionar a biblioteca Cortex-M3 no projeto:**
  RTX_LIB_CM.a

- **Incluir o header no programa principal:**
  #include "cmsis_os.h"

Estrutura Básica

```
#include "cmsis_os.h"

// definições de tarefas, temporizadores, etc.
// declarações das funções das tarefas e de callback

void main(){
  osKernelInitialize();

  // inicializações de hardware
  // ativação de tarefas, temporizadores, etc.

  osKernelStart();

  // laço de repetição ou encerramento da tarefa main()
} // main
```

Exemplo de Uso (Tarefa)

```
#include "LPC13xx.h"                    // CMSIS-Core
#include "gpio.h"                       // Lib_EABaseBoard
#include "cmsis_os.h"                   // CMSIS-RTOS


void thread1(void const *argument){
  while(1){
    GPIOSetValue(0, 7, 0); // apaga LED2
    osDelay(500);
    GPIOSetValue(0, 7, 1); // acende LED2
    osDelay(500);
  }
}
osThreadDef(thread1, osPriorityNormal, 1, 0);
```

Exemplo de Uso (Tarefa)

```
void main(){
  osKernelInitialize();

  SystemInit();
  GPIOInit();
  GPIOSetDir(0, 7, 1); // LED2 como saída

  osThreadCreate(osThread(thread1), NULL);

  osKernelStart();
  osDelay(osWaitForever);
}
```

Exemplo Thread no RTOS LPC1343

```
#include "libdemo.h"
/*============================================================================
 *                  Exemplos de utilização do RTOS CMSIS
 *           LPCXpresso 1343 + Embedded Artists Development Board
 *----------------------------------------------------------------------------*
 *                   Gerenciamento de múltiplas Threads
 *----------------------------------------------------------------------------*
 *                    Prof. André Schneider de Oliveira
 *            Universidade Tecnológica Federal do Paraná (UTFPR)
 *============================================================================*/

void led2_thread(void const *args) {
    while (1) {
        pca_toggle(2);
        osDelay(1000);
    }
}
osThreadDef(led2_thread, osPriorityNormal, 1, 0);

int main() {

    osKernelInitialize();

    I2CInit( (uint32_t)I2CMASTER, 0 );

    osThreadCreate(osThread(led2_thread), NULL);

    osKernelStart();

    while (1) {
        pca_toggle(1);
        osDelay(500);
    }
}
```

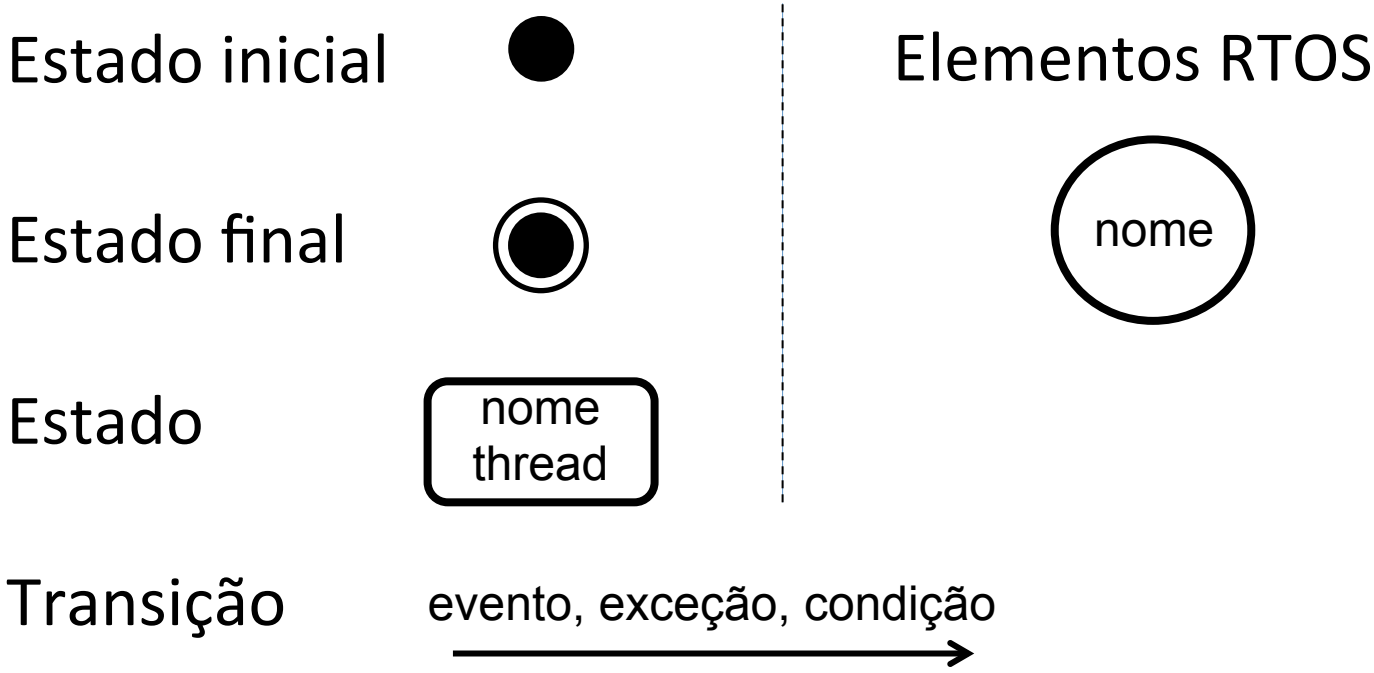
Projeto e Análise de sistemas concorrentes

- **Projeto teórico**
  - Diagrama de estados e transições
- **Análise dos resultados**
  - Diagrama de Gantt

Diagrama de Estados e Transições

- Realizar a representação gráfica das principais ações e eventos da solução proposta
- Organizar o estudo do problema e esboço da solução
- Importante ferramenta para agilizar o desenvolvimento

Entidades DET



- ***evento de estouro de timer :*** *especificar o tempo/frequência*
- ***exceção (interrupção ou falta) :*** *especificar o tipo/pino/evento*
- ***condição booleana****: especificar o estado*

Exemplo DET

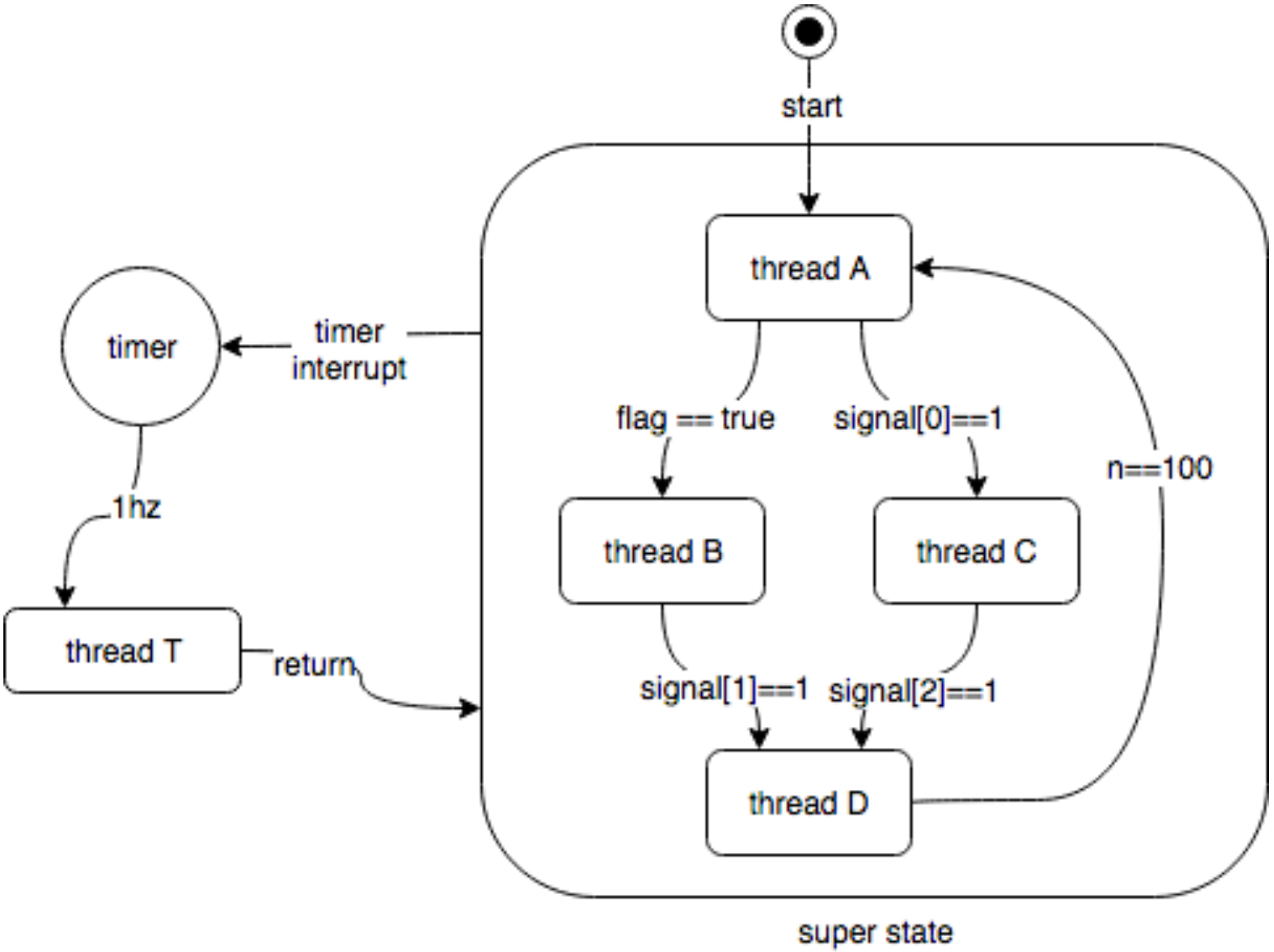

start
thread A
timer
timer
interrupt
flag == true
signal[0]==1
n==100
1hz
thread B
thread C
thread T
return
signal[1]==1
signal[2]==1
thread D
super state

Diagrama de Estados e Transições

- Draw.io (ferramenta de desenho na nuvem)
  - https://www.draw.io/

- Yakindu Statechart Tools (desenho + simulação)
  - http://statecharts.org/

Diagrama de Gantt

- Objetiva descrever a descrição das etapas de um projeto em relação ao tempo

- Também pode ser aplicado para ilustrar a execução de um sistema multithread,
  - cada etapa representa o estado de uma thread

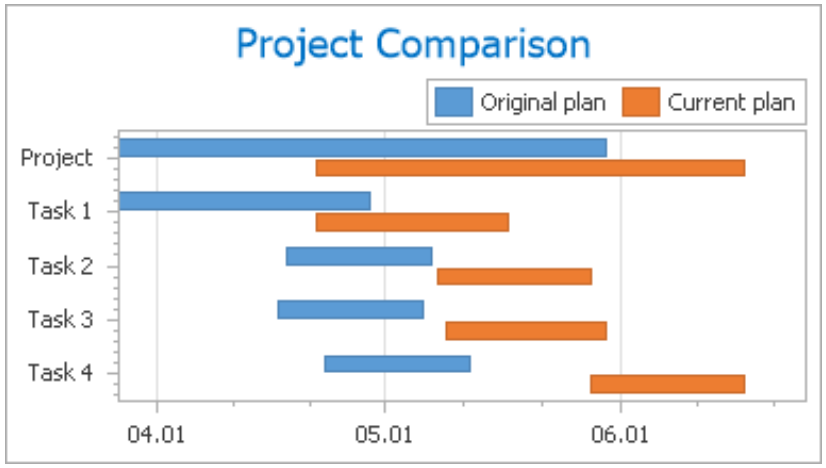
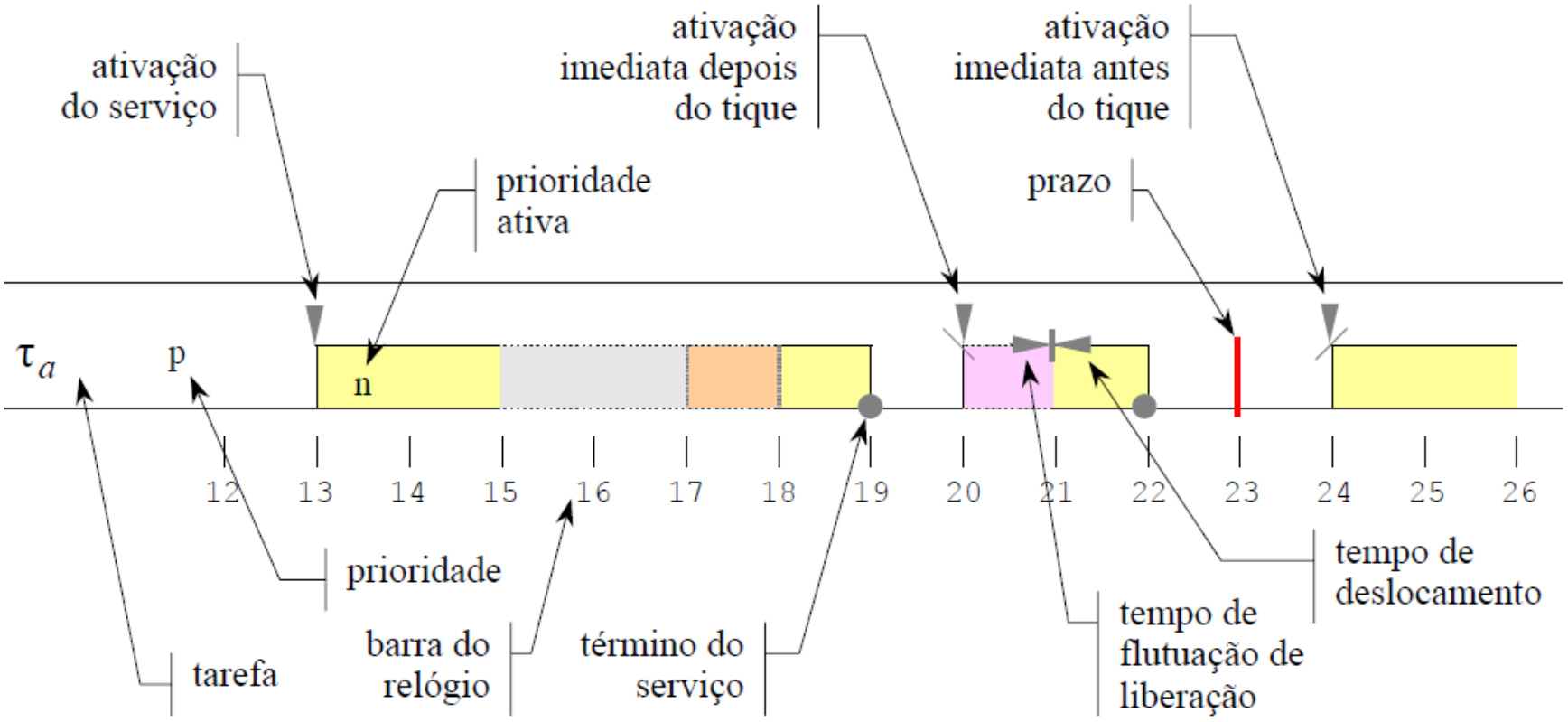


Diagrama de Gantt



Tipos de Tarefas

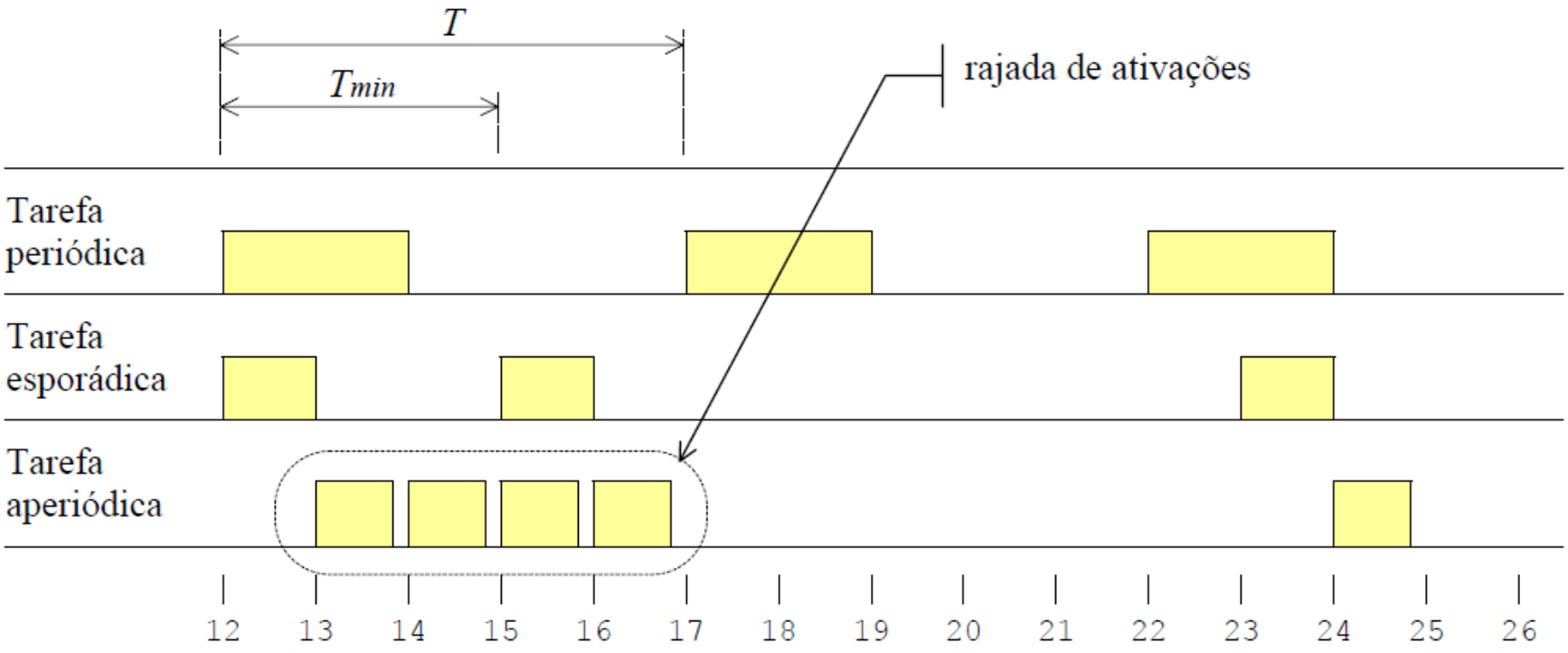


Diagrama de Gantt

- Mermaid Live Editor

https://knsv.github.io/mermaid/live_editor/

- Documentação Diagrama de Gantt

https://mermaidjs.github.io/gantt.html

Exemplo RTOS Gantt 1/2

```
#include "libdemo.h"
/*============================================================================
 *                  Exemplos de utilização do RTOS CMSIS
 *           LPCXpresso 1343 + Embedded Artists Development Board
 *----------------------------------------------------------------------------*
 *                            Diagrama de Gantt
 *----------------------------------------------------------------------------*
 *                https://knsv.github.io/mermaid/live_editor/
 *              Documentação : https://knsv.github.io/mermaid/
 *----------------------------------------------------------------------------*
 *                   Prof. André Schneider de Oliveira
 *            Universidade Tecnológica Federal do Paraná (UTFPR)
 *============================================================================*/
FILE *file;
int ticks_factor = 10000;

void led1_thread(void const *args) {
    uint32_t time;
    while (1) {
        time = osKernelSysTick()/ticks_factor;
        pca_toggle(1);
        osDelay(500);
        fprintf(file," Led1 : %i, %i\n", (int)time, (int)osKernelSysTick()/ticks_factor);

    }
}
osThreadDef(led1_thread, osPriorityNormal, 1, 0);

void led2_thread(void const *args) {
    uint32_t time;
    while (1) {
        time = osKernelSysTick()/ticks_factor;
        pca_toggle(2);
        osDelay(2000);
        fprintf(file," Led2 : done, %i, %i\n", (int)time, (int)osKernelSysTick()/ticks_factor);

    }
}
osThreadDef(led2_thread, osPriorityNormal, 1, 0);
```

Exemplo RTOS Gantt 2/2

```
void led3_thread(void const *args) {
    uint32_t time;
    while (1) {
      time = osKernelSysTick()/ticks_factor;
      pca_toggle(3);
      osDelay(300);
      fprintf(file," Led3 : crit, %i, %i\n", (int)time, (int)osKernelSysTick()/ticks_factor);
    }

    printf("%i\n", (int)osKernelSysTick());

}
osThreadDef(led3_thread, osPriorityHigh, 1, 0);

int main() {

    osKernelInitialize();
    I2CInit( (uint32_t)I2CMASTER, 0 );

    osThreadCreate(osThread(led1_thread), NULL);
    osThreadCreate(osThread(led2_thread), NULL);
    osThreadCreate(osThread(led3_thread), NULL);

    osKernelStart();

    file = fopen("gantt.txt","w");

    fprintf(file,"gantt\n");
    fprintf(file,"    title A Gantt Diagram\n");
    fprintf(file,"    dateFormat x\n");

    osDelay(osWaitForever);
}
```