IM420 Sistemas Embarcados de Tempo Real

Notas de Aula – Semana 06

Prof. Denis Loubach
dloubach@fem.unicamp.br

Programa de Pós-Graduação em Eng. Mecânica / Área de Mecatrônica
Faculdade de Engenharia Mecânica - FEM
Universidade Estadual de Campinas - UNICAMP

1^o Semestre de 2018

Tópicos

To Code Is Human

In writing code, the smallest mistake can have serious consequences: a single uninitialized variable actually caused a spacecraft to crash in 90's (Mars Polar Lander failed to land). That's why clean, simple code is so important.

As painful as this process [punch card programming] might seem today, it taught me to be very careful with my code.
How are things different now? Thank goodness, the long wait to find out how we're messing things up is gone, now that we have more compute power in our watch than a roomful of hardware in the dark ages. But we're still human, and we still make mistakes. We can just make and discover our mistakes much faster than before.
***Near the top of the list of the most common types of mistakes programmers make is the unintentional use of uninitialized data**.*
Writing reliable code is hard. What's so interesting about software development is that even the smallest mistake can have the largest consequences.
Keep your code clean and simple and heed the warnings from the tools that are designed to protect you. Either do that or risk having a dumb machine show you it can continue to outsmart you.

Fonte: *Gerard J. Holzmann, "To Code Is Human", **IEEE Software**, vol.32, no. 1, pp. 14-17, Jan.-Feb. 2015, doi:10.1109/MS.2015.19*

Definição de RTOS

Real Time Operating Systems - RTOS

Código de software responsável por escalonar execuções de maneira temporal, gerenciar recursos do sistema, e prover uma fundação/camada consistente para o desenvolvimento de códigos de aplicações

Em alguns casos, o RTOS pode ser composto apenas de seu *kernel*, que é o núcleo de software supervisor que provê apenas lógica mínima, escalonador e algoritmos de gerenciamento de recursos

Em outros, pode ser a combinação de vários módulos incluindo o *kernel*, sistema de arquivos, suporte a rede, entre outros

Ilustração das camadas de um sistema embarcado de tempo real

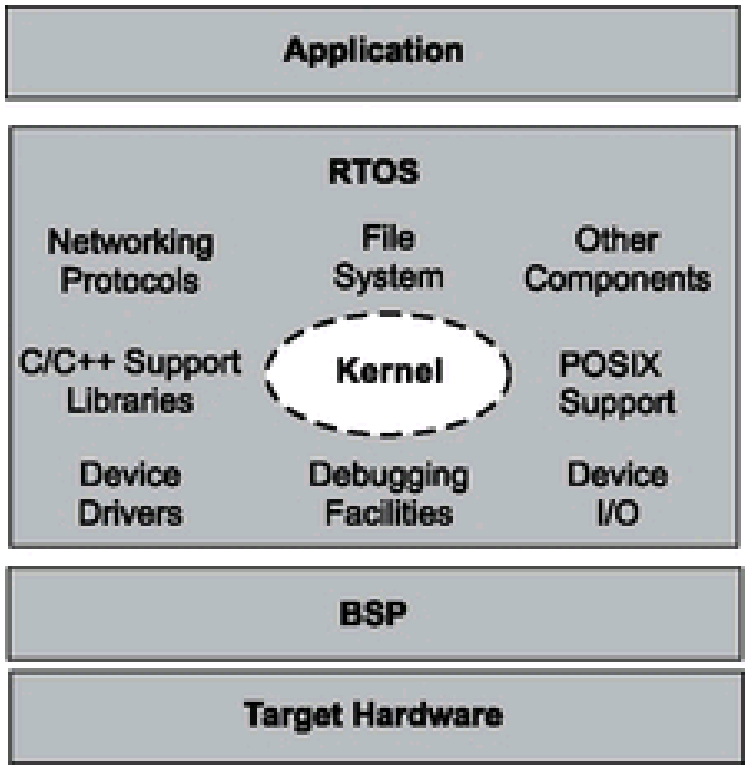
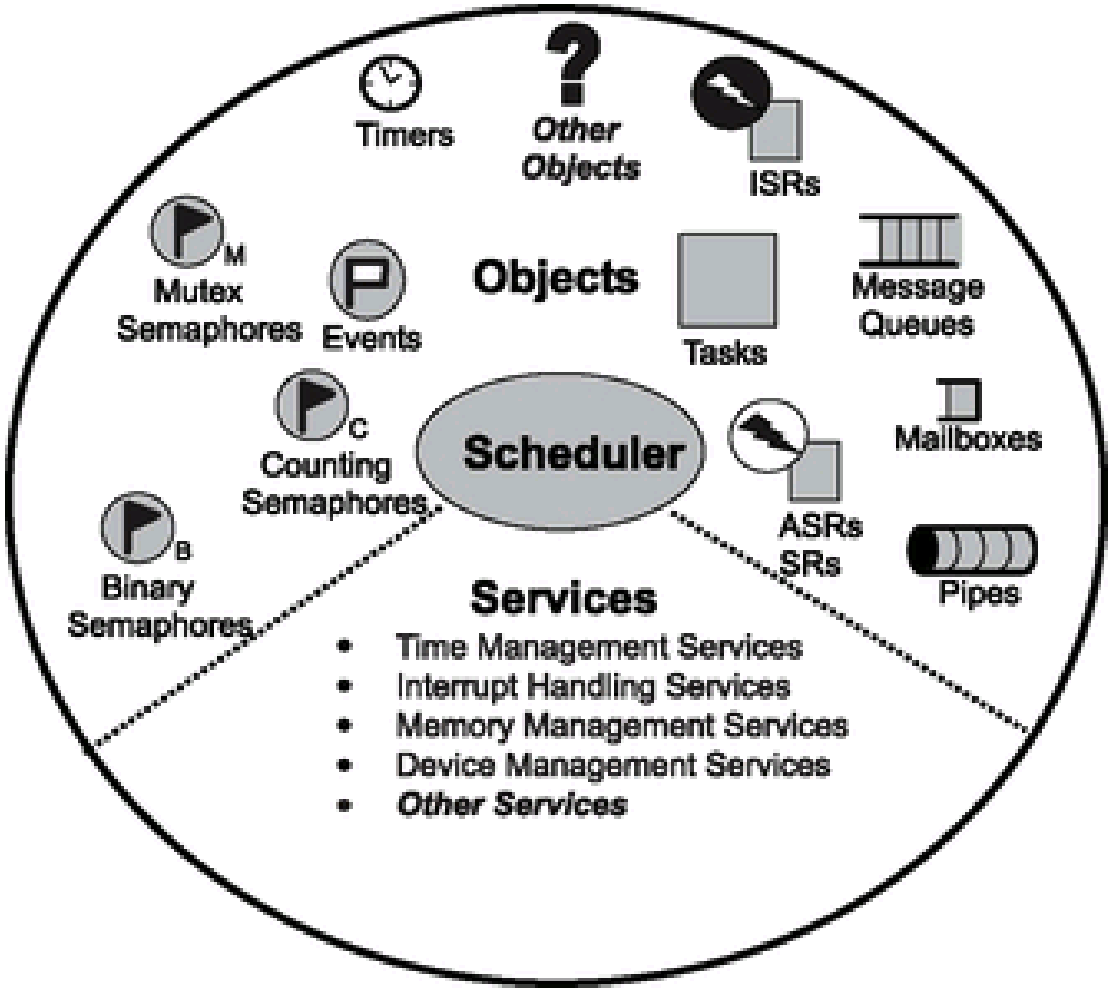


Ilustração do *kernel* de um RTOS



Board Support Package - BSP

A interface entre o RTOS e a plataforma de hardware é conhecida como *board support package* - BSP

BSP pode ser definido como um conjunto de *drivers* de software específicos para os componentes de hardware

BSP possui informações sobre as características do hardware, ou seja, é a implementação de suporte específico para uma dada plataforma de hardware (placa) para um dado sistema operacional de tempo real

Board Support Package - BSP

Uma estrutura de BSP deve incluir (entre outros):

- Configuração da CPU
- Configuração da memória (*i.e., stack, heap*)
- Configuração das interrupções de hardware
- Configuração do *timer* do escalonador do sistema

Implementação de um BSP

A implementação de um BSP exige conhecimento profundo da plataforma de hardware (*e.g.*, uC), algum nível de linguagem *assembly* e de sistemas operacionais

Geralmente, este desenvolvimento não se inicia do zero (*from scratch*)

Ao invés, utiliza-se como base um BSP já desenvolvido, onde adaptações e novos desenvolvimentos são realizados

Este processo denomina-se **RTOS *porting***

Definições - Escalonador

Escalonador ou *Scheduler*

Algoritmo que determina qual entidade/tarefa executar e quando executar

Neste caso, uma entidade escalonável é um objeto do *kernel* que compete por um tempo de execução no sistema, baseado num algoritmo de escalonamento pré-definido

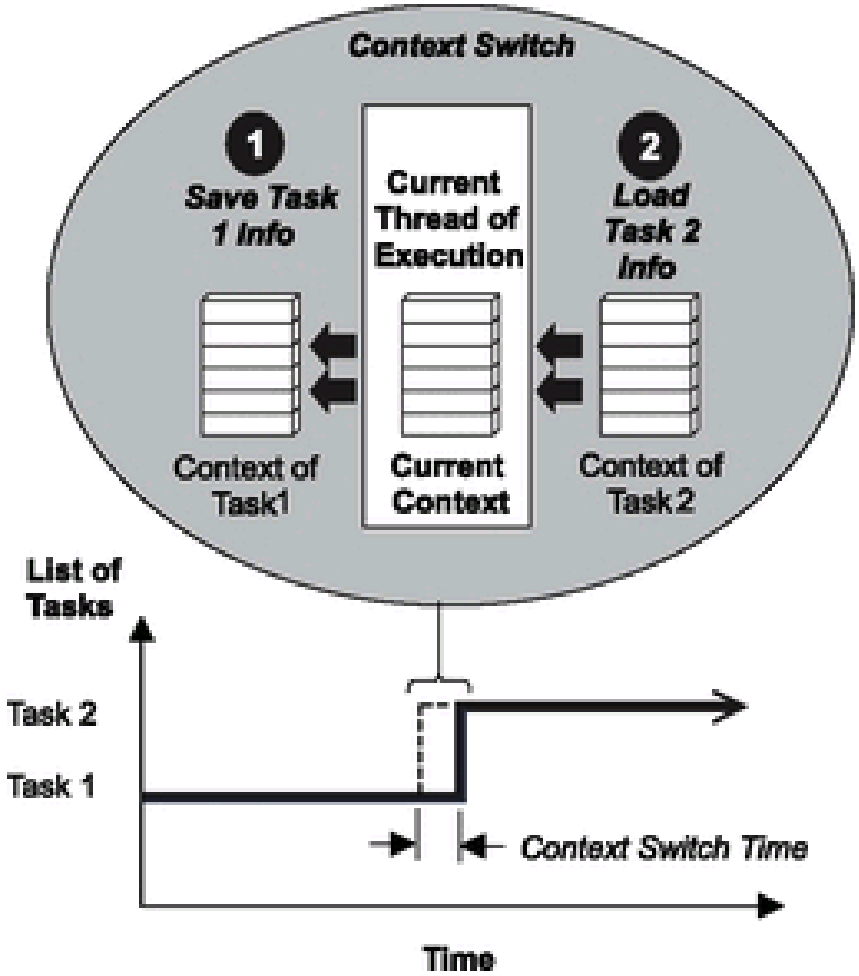
Tarefas ou processos, são exemplos de entidades escalonáveis

Definições - Troca de contexto

Troca de contexto (*context switching*)

- Cada tarefa possui seu próprio contexto (estado dos registradores da CPU requeridos a cada vez que a tarefa é escalonada para ser executada)
- Uma troca de contextos ocorre quando o escalonador realiza a troca de execução no processador da tarefa A para a tarefa B
- Em alguns casos, as tarefas possuem um *task control block* - TCB associado a elas
- TCB é uma estrutura de dados que o *kernel* mantém para armazenar dados específicos das tarefas

Ilustração do TCB



Sequência de passos para troca de contexto

1. O *kernel* salva o contexto da tarefa 1 no seu respectivo TCB
2. O *kernel* carrega as informações de contexto da tarefa 2, que passa a ser a tarefa em execução
3. O contexto da tarefa 1 permanece congelado enquanto a tarefa 2 é executada
4. Entretanto, se o escalonador necessitar rodar a tarefa 1 novamente, ela é executada no ponto onde parou antes da troca de contextos

Tempo de troca de contexto

1. O tempo levado pelo escalonador para trocar de uma tarefa para outra denomina-se **tempo de troca de contextos**

2. Tipicamente, esse tempo é muito menor que o tempo de execução da tarefa, entretanto não deve ser desconsiderado

Definições - Despachador (1/2)

Despachador

Parte do escalonador responsável por executar a troca de contextos e alterar o fluxo de execução

Durante o tempo em que o RTOS está rodando, o fluxo de execução (ou fluxo de controle) passa por uma das seguintes áreas:

- Tarefa de aplicação
- ISR
- *Kernel*

Definições - Despachador (2/2)

Quando uma tarefa ou ISR faz alguma chamada do sistema, o fluxo de controle passa pelo *kernel* para executar a função do sistema provida pelo *kernel*

Quanto a execução da função é terminada, o despachador é responsável por passar o controle para uma das tarefas de aplicação ou ISR

Ou seja, é o despachador que realmente faz o trabalho de troca de contexto e passagem do controle de fluxo

Deadlines

Os requisitos de temporalidade de sistemas de tempo real são definidos em termos de ***deadlines*** das tarefas (*i.e., soft, hard*)

Deadline

Tempo máximo no qual uma tarefa dever terminar sua execução de maneira completa

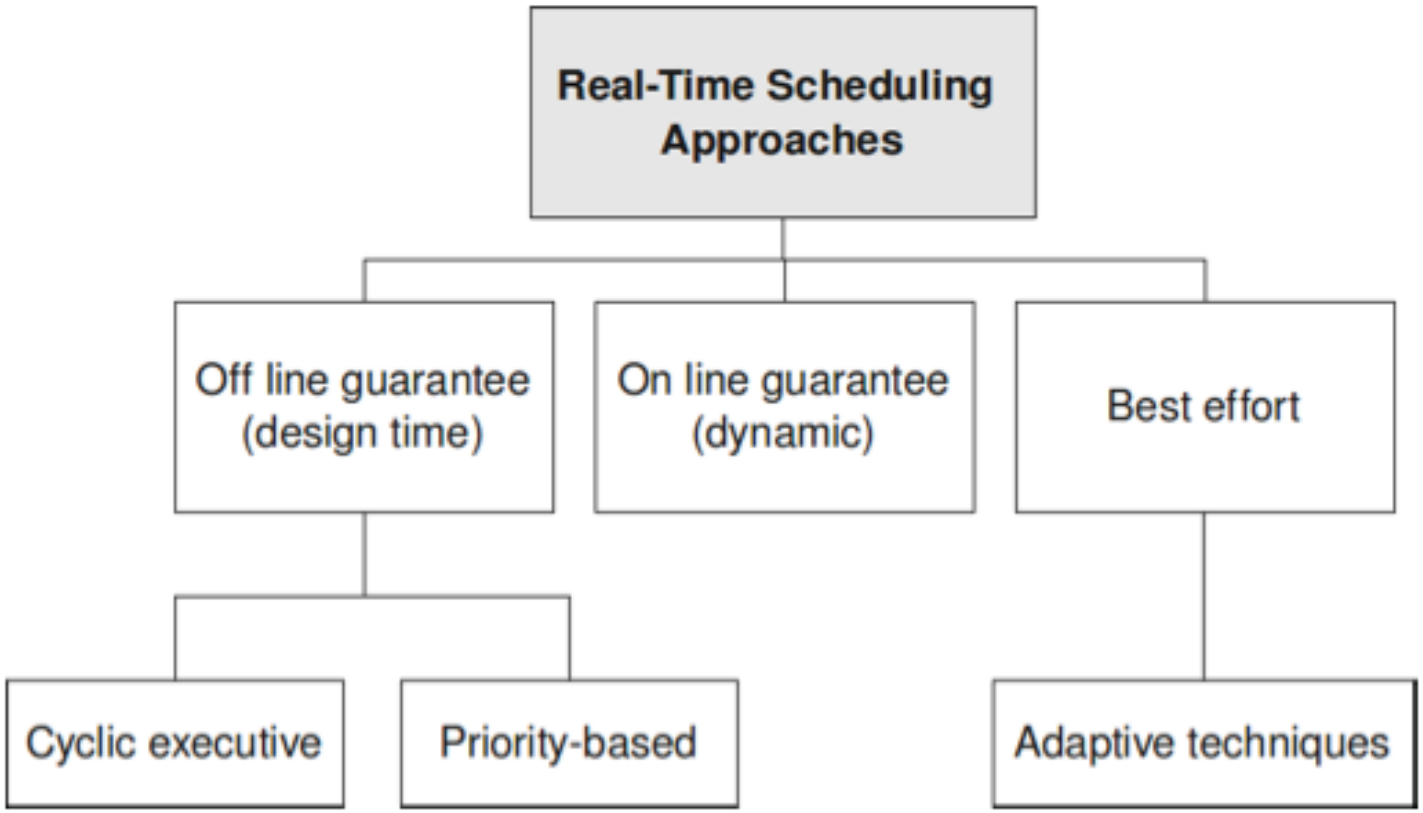
Algoritmos de escalonamento

O escalonador determina qual tarefa irá executar seguindo um algoritmo de escalonamento (ou política de escalonamento)

Os escalonadores mais comuns são:

- Prioridade-fixa (*fixed based-priority*) e preemptivos
  - *Preemptive priority-based*
  - *Rate Monotonic* - RM [1]
- Prioridade-dinâmica (*dynamic based-priority*) e preemptivo
  - *Earliest Deadline First* - EDF [1]
- Tempo de execução e preemptivo
  - *Round-Robin* ou *Time Slicing*
- Colaborativo (não-preemptivo)
  - Executivo Cíclico

Abordagens de escalonador de tempo real [2]

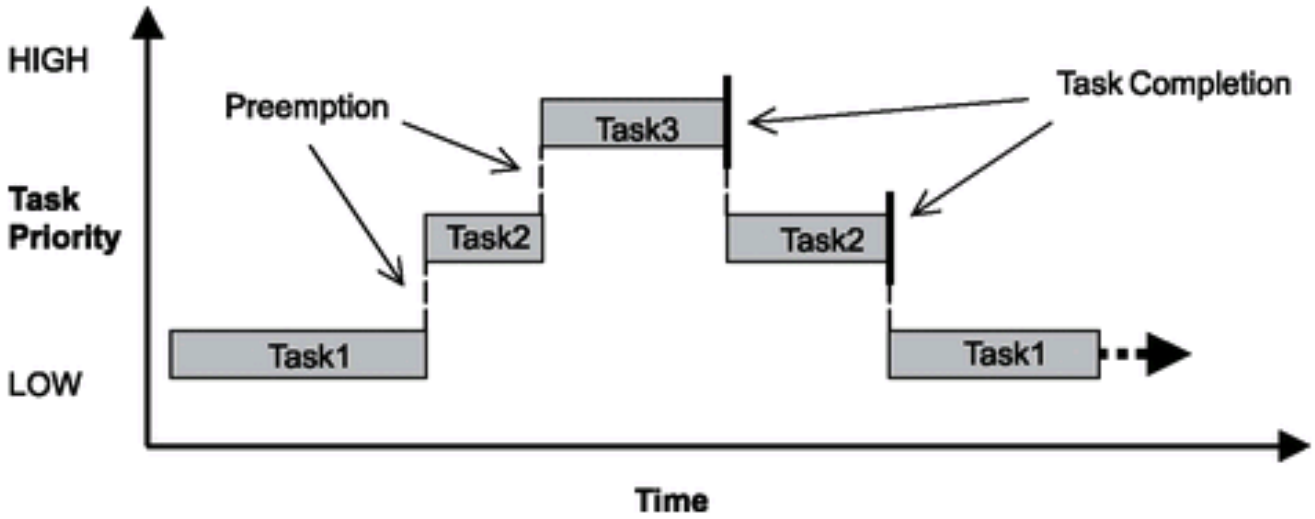


Preemptive priority-based

- A tarefa que ganha o processador para ser executada, considerando qualquer instante de tempo, é a tarefa de maior prioridade entre as tarefas no estado "pronto" do sistema
- Prioridades são fixas (intervalo conhecido, ex [0..255]) e definidas ainda em tempo de projeto (*off line*)
- Se alguma tarefa de maior prioridade (em relação a tarefa corrente) entra no estado "pronto", o *kernel* preempta a de menor prioridade para esta de mais alta prioridade assumir a execução
- Ocorre então a troca de contexto

Preemptive priority-based

Ilustração



Rate Monotonic - RM

- Método de dar prioridade para as tarefas como uma função monotônica da taxa de execução de uma tarefa
- Em outras palavras $\Rightarrow$ menor o período de execução, maior a prioridade atribuída
- Considera algumas suposições:
  - todas as tarefas são periódicas
  - as tarefas são independentes e não possuem inter-comunicação
  - o *deadline* de uma tarefa é o início do seu próximo período ($D = T$)
  - tempo de execução constante para as tarefas
  - todas tarefas possuem o mesmo nível de criticidade
  - tarefas aperiódicas são limitadas a inicialização e algum procedimento de recuperação de falhas

Utilização do processor segundo RM

O fator de utilização do processor (definido pela equação 1), segundo o RM, é dado pela fração do tempo gasto pelo processador para execução de um conjunto de tarefas

$$U = \sum_{i=1}^{m} \frac{C_i}{T_i} \tag{1}$$

onde:

U é o fator de utilização

m é o número total das tarefas do sistema

C é o tempo de computação no pior caso (*worst-case execution time - WCET*), relacionado à uma tarefa

T é o período da tarefa

Restrição de U segundo RM

A equação 2 define o limite teórico de utilização do processador, segundo RM

$$U = m(2^{\frac{1}{m}} - 1) \tag{2}$$

Quando $m \to \infty$, $U \simeq \ln 2 \simeq 0.69$ [1]

Earliest Deadline First - EDF

- Algoritmo da classe de escalonamento dinâmico, isto é, prioridades dinâmicas (mudam ao longo do tempo)
- Prioridades são atribuídas para as tarefas de acordo com os *deadlines* das chamadas correntes
- Ou seja, menor o *deadline*, maior a prioridade da tarefa

Utilização do processor segundo EDF

O fator de utilização do processor, segundo o EDF, é definido pela equação 3

$$U = \sum_{i=1}^{m} \frac{C_i}{T_i} \tag{3}$$

Neste caso do EDF, o limite encontra-se na utilização completa do processador

$$U \leq 1 \tag{4}$$

onde:

U é o fator de utilização

m é o número total das tarefas do sistema

C é o tempo de computação no pior caso (*worst-case execution time - WCET*), relacionado à uma tarefa

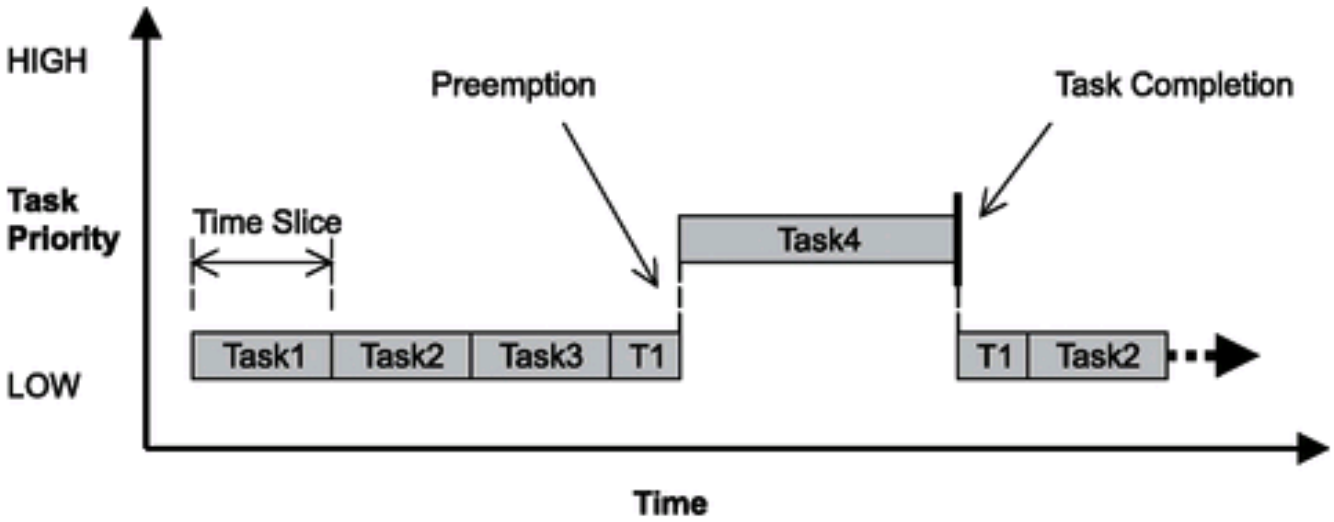
T é o período da tarefa

Round-Robin - RR ou *Time Slicing*

- Como o nome já sugere, aloca uma fatia de tempo igual e pré-determinada (*slice*) para as tarefas executarem
- O *Round-Robin* puro não atende aos requisitos de sistemas de tempo real, uma vez que as tarefas executam papéis em diferentes níveis de importância (*i.e.*, prioridades)
- Pode ser utilizado em conjunto com o *Preemptive priority-based* como **critério de desempate** entre tarefas de prioridades iguais no estado de "pronto"

Round-Robin - RR ou *Time Slicing*

Ilustração



Executivo Cíclico Cooperativo - ECC

- Realizado através de um *loop* de funções/tarefas denominado executivo cíclico
- Não-preemptivo (colaborativo)
- As tarefas (ou funções) são chamadas uma após a outra para serem executadas em sequência, caracterizando um comportamento cooperativo
- Por esta razão, não deve haver *loop* infinito nas tarefas

Executivo Cíclico Cooperativo - ECC

```
1   #define TRUE  1
2   #define FALSE 0
3   unsigned int flagNextPeriod = FALSE;
4   ...
5   void ecc_period_isr_handler(void)
6   {
7     // the interrupt hardware timer counting controls the ECC period
8     flagNextPeriod = TRUE;
9   }
10  ...
11  int main(void)
12  {
13    // setup and initializations go here
14    ...
15
16    // main program loop, runs forever
17    while(TRUE)
18    {
19      funcDoSomething();
20      funcDoSomethingElse();
21      refreshWatchdogTimer();
22
23      // wait until period completion...
24      // flag is set by a ISR of a timer configured
25      // to interrupt in the cyclic excutive predefined period
26      while(!flagNextPeriod);
27      flagNextPeriod = FALSE;
28    }
29    return(0); // program will never reach this point
30  }
```

Análise do aspecto temporal do executivo cíclico

$$\sum_{i=1}^{n} C_i \leq T_{EC} \tag{5}$$

onde:

n é o número das tarefas/funções do sistema

C é o tempo de computação no pior caso (*worst-case execution time - WCET*), relacionado à uma tarefa/função

T_{EC} é o período do executivo cíclico

Definições - Análise de escalonabilidade

Análise de escalonabilidade

Determina se (i) todas as tarefas do sistema podem ser escalonadas para executar e cumprir seus *deadlines* segundo o algoritmo de escalonamento utilizado; e (ii) ainda atingir uma utilização ótima do processador

Nesta análise leva-se em conta somente requisitos temporais, não funcionais

Definições - Tarefa

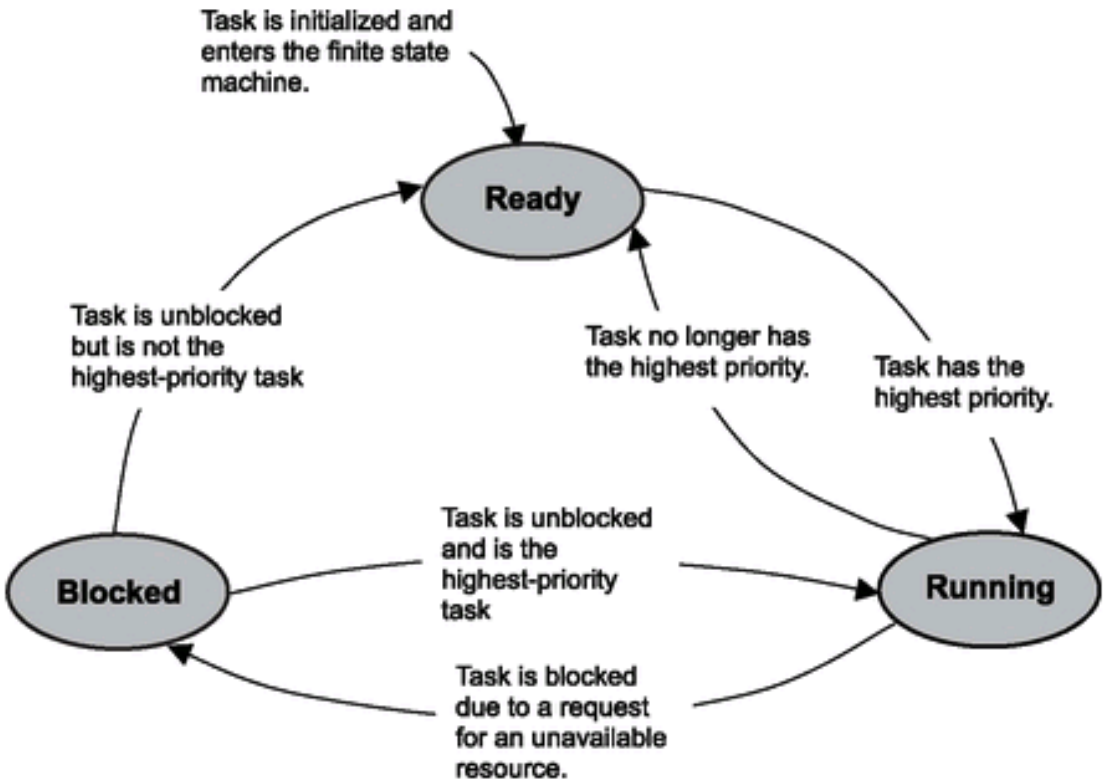
Tarefa

Computação executada por um processador numa ordem sequencial

- Um sistema pode ser decomposto em múltiplas tarefas concorrentes
- Principais estados de uma tarefa: pronta, executando, bloqueada
- Classificação: periódica, aperiódica

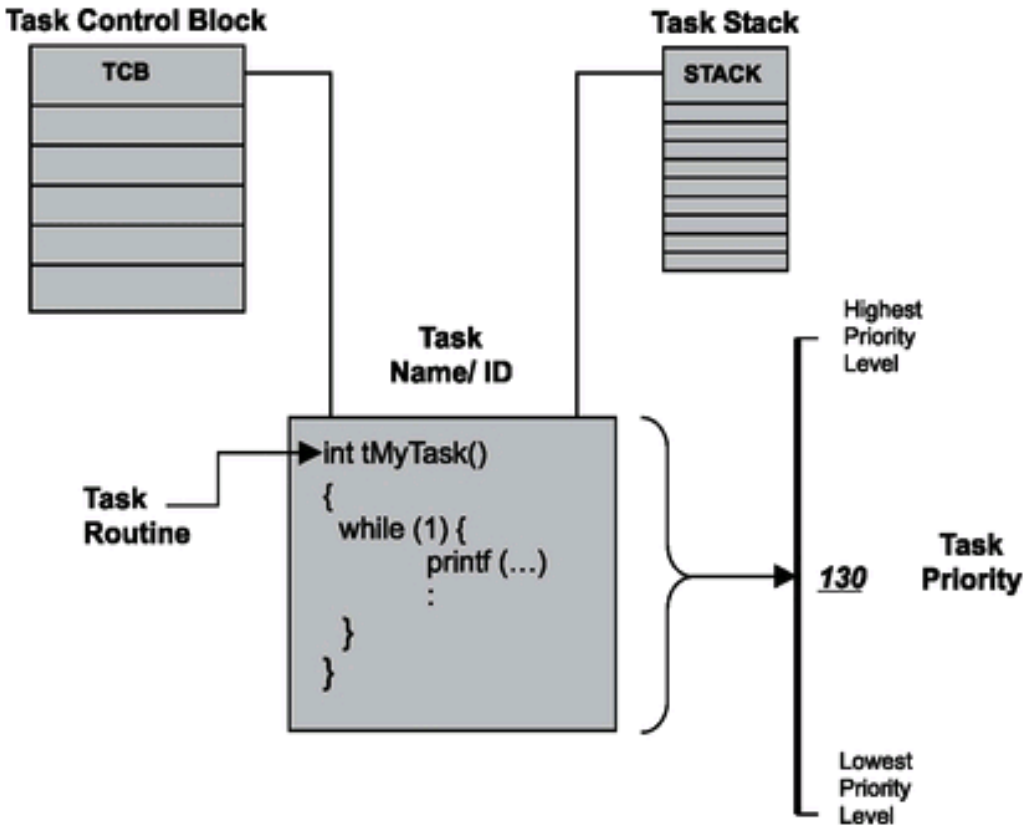
Estados de uma tarefa

Ilustração dos principais estados de uma tarefa



Tarefa, TCB e *stack*

Ilustração da tarefa com seu TCB e sua *stack*



Definições - Sistemas multi-tarefas

Sistema multi-tarefas

Habilidade do sistema operacional (ou moldura de tempo real) de gerenciar várias atividades/tarefas considerando seus *deadlines*

Isto implica em:

- Concorrência pela CPU
- Mais de uma tarefa competindo para ser executada
- Consideração do algoritmo do escalonador e o modelo do tempo real (*hard*, *soft*) utilizados

Informação ao leitor

Notas de aula baseadas nos seguintes textos:


C. L. Liu and J. W. Layland, “Scheduling algorithms for multiprogramming in a hard-real-time environment,” *J. ACM*, vol. 20, pp. 46–61, Jan. 1973.


J.-M. FARINES, J. d. S. FRAGA, and R. S. d. OLIVEIRA, *Sistemas de Tempo Real*. Florianópolis: Universidade Federal de Santa Catarina, 2000.


Q. Li and C. Yao, *Real-Time Concepts for Embedded Systems*. CMP books, 2003.


A. S. Berger, *Embedded Systems Design: An Introduction to Processes, Tools, and Techniques*. CMP Books, 2002.