

# ***Sistemas de Tempo-Real***

## **Aula 4**

# **Conceitos básicos de escalonamento**

**Escalonamento de tarefas, taxonomia básica  
Técnicas de escalonamento preliminares  
Escalonamento estático cíclico**

Adaptado dos slides desenvolvidos pelo Prof. Doutor Luís Almeida  
para a disciplina “Sistemas de Tempo-Real”  
Revisto em Out/2009 por Paulo Pedreiras

# *Aula anterior (3)*

## Executivos de tempo real

- Os **estados de execução** de uma tarefa (diagrama de transição de estados)
- A **arquitetura genérica** de um kernel de tempo-real
- Os **componentes básicos** de um kernel de tempo-real, estruturas de dados e funções
- Exemplos: ReTMiK, OReK, RTKPIC18, SHaRK e RTAI



# Complexidade temporal

- Medida do **crescimento** do **tempo de execução** de um algoritmo quando **aumenta a dimensão do problema** (i.e. dos dados de entrada)
- Costuma expressar-se com o operador  **$O()$**
- Aritmética do operador  $O()$ ,  $n$ =dimensão do problema,  $k$ =cons.
  - $O(k) = O(1)$
  - $O(kn) = O(n)$
  - $O(k_1n^m + k_2n^{m-1} + \dots + k_{m+1}) = O(n^m)$

```
for (k=0;k<N;k++)  
    a[k]=0;  
Compl. =  $O(N)$ 
```

```
for (k=0;k<N-1;k++)  
    for (m=k;m<N;m++)  
        if a[k]<a[m]  
            swap(a[k],a[m]);  
Compl. =  $O(N^2)$ 
```

```
Cálculo das permutações  
de um dado conjunto  
 $A = \{a_i, i=1..N\}$   
Compl. =  $O(N^N)$ 
```

# Complexidade temporal

## Classes P e NP em problemas de decisão

- **P** – problema que se resolve em tempo polinomial,  $O(p(N))$
- **NP** – problema que não se resolve em tempo polinomial mas em que cada solução se pode verificar em tempo polinomial
  - **NP-complete**
    - não é conhecida nenhuma “solução rápida”.
  - **NP-hard**
    - pelo menos tão difícil como NP, mas não necessariamente do tipo NP

**A complexidade temporal é uma importante medida de desempenho de um algoritmo (e.g. de escalonamento)**



# Definição de escalonamento

## Escalonamento de tarefas

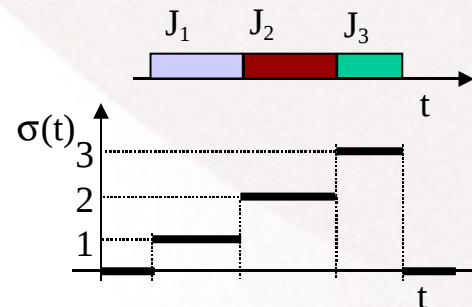
- Sequência de execução de tarefas num ou mais processadores
- Aplicação de  $R^+$  (tempo) em  $N_0^+$  (conjunto de tarefas), fazendo corresponder a cada instante de tempo  $t$  uma tarefa  $i$  que está executando nesse instante.

$$\sigma: R^+ \rightarrow N_0^+$$

$$i = \sigma(t), t \in R^+ \quad (i=0 \Rightarrow \text{processador livre})$$

$\sigma(t)$  é uma função degrau cujo traçado é um *gráfico de Gantt*

$J = \{J_1, J_2, J_3\}$   
(conjunto de tarefas)  $\rightarrow$



# Definição de escalonamento

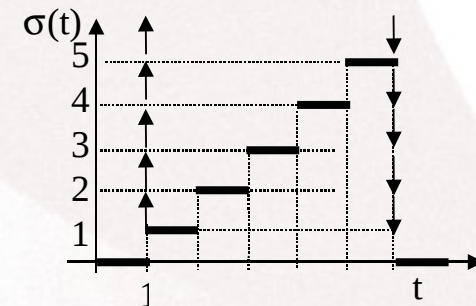
- Um **escalonamento** diz-se **praticável** (*feasible schedule*) se cumpre as restrições associadas ao conjunto de tarefas (**temporais**, não preempção, recursos partilhados, precedências)
- Um **conjunto de tarefas** diz-se **escalonável** (*schedulable task set*) se existe pelo menos um **escalonamento praticável** para esse conjunto.



# Problema de escalonamento

- Dados:
  - um **conjunto de tarefas**
  - **restrições** que lhes estão associadas (ou função de custo)
- Encontrar uma **atribuição de tempo de processador às tarefas** que lhes permita :
  - **executar as tarefas completamente**
  - **cumprir as suas restrições** (ou minimizar a função de custo)

e.g.  $J = \{J_i (C_i=1, a_i=1, D_i=5, i=1..5)\}$  →



# Algoritmos de escalonamento

- Um algoritmo **de escalonamento** é um método de resolução de um problema de escalonamento.
  - **Nota:** não confundir algoritmo *de escalonamento* com *escalonamento*
- Cassificação de algoritmos de escalonamento:
  - **Preemptivo** *versus* **não-preemptivo**
  - **Estático** *versus* **dinâmico**
  - **Off-line** *versus* **on-line**
  - **Ótimo** *versus* **sub-ótimo** (heurístico)
  - Com **garantias** de pior caso *versus* **melhor possível** (*best effort*)



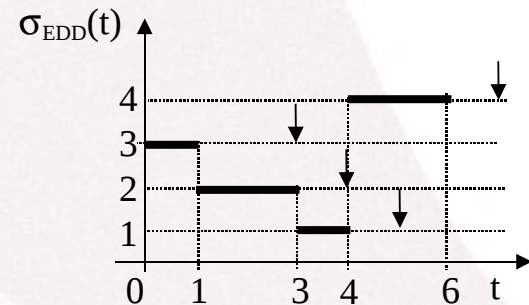
# Algoritmos preliminares

## EDD - Earliest Due Date (Jackson, 1955)

- Tarefas de única instância e disparadas sincronamente:  
 $J = \{ J_i (C_i, a_i=0, D_i) \ i=1..n \}$
- Executar as tarefas por **ordem não decrescente de *deadline* minimiza o atraso máximo**  $L_{\max}(J) = \max_i (f_i - d_i)$
- Complexidade:  **$O(n \cdot \log(n))$**

e.g.  $J = \{ J_1(1,5), J_2(2,4), J_3(1,3), J_4(2,7) \}$

$$L_{\max, \text{EDD}}(J) = -1$$



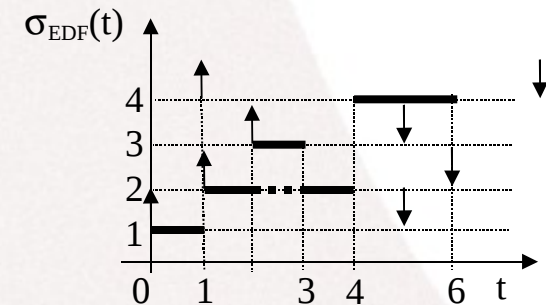
# Algoritmos preliminares

## EDF - Earliest Deadline First (Liu and Layland, 1973; Horn, 1974)

- Tarefas de única instância ou periódicas, assíncronas, preemptivas:  
 $J = \{ J_i (C_i, a_i, D_i) \ i=1..n \}$
- Executar em cada instante a tarefa com **deadline mais próxima**  
**minimiza o atraso máximo**  $L_{\max}(J) = \max_i (f_i - d_i)$
- Complexidade:  **$O(n \cdot \log(n))$** , **Ótimo** entre todos desta classe

e.g.  $J = \{ J_1(1,0,5), J_2(2,1,5), J_3(1,2,3), J_4(2,1,8) \}$

$$L_{\max, EDF}(J) = -2$$



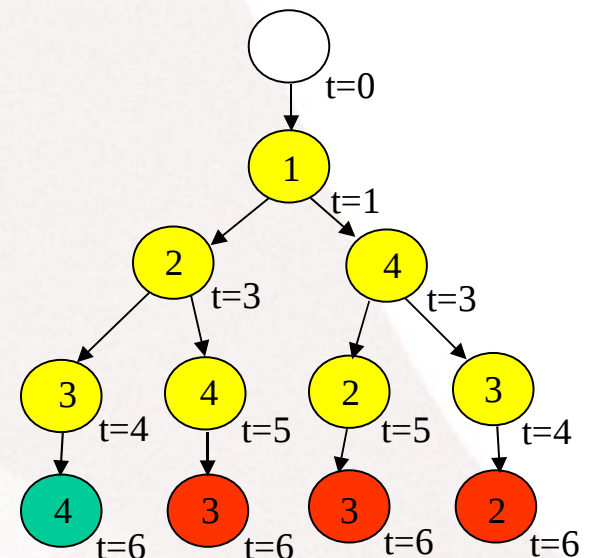


# Algoritmos preliminares

## BB – Branch and Bound (Bratley, 1971)

- Tarefas de única instância ou periódicas, assíncronas, não preemptivas:  
 $J = \{ J_i (C_i, a_i, D_i) \mid i=1..n \}$
- Construção do escalonamento por **busca exaustiva** no espaço de permutações (árvore)
- Complexidade:  **$O(n!)$**

e.g.  $J = \{ J_1(1,0,5), J_2(2,1,3), J_3(1,2,4), J_4(2,1,7) \}$



# Escalonamento de tarefas periódicas

Os instantes de activação são conhecidos a priori

$$\Gamma = \{ \tau_i (C_i, \Phi_i, T_i, D_i, i=1..n) \} \rightarrow a_{i,k} = \Phi_i + (k-1)T_i$$

Assim, o escalonamento pode ser construído quer

- **Com o sistema em execução (on-line)**  
a próxima tarefa é escolhida à medida que o sistema vai funcionando.
- **Antes do sistema entrar em execução (off-line)**  
a ordem de execução é determinada antes do sistema entrar em funcionamento e é guardada numa tabela que é lida em tempo de execução para iniciar as tarefas (**escalonamento estático cíclico**).



# Escalonamento estático cíclico

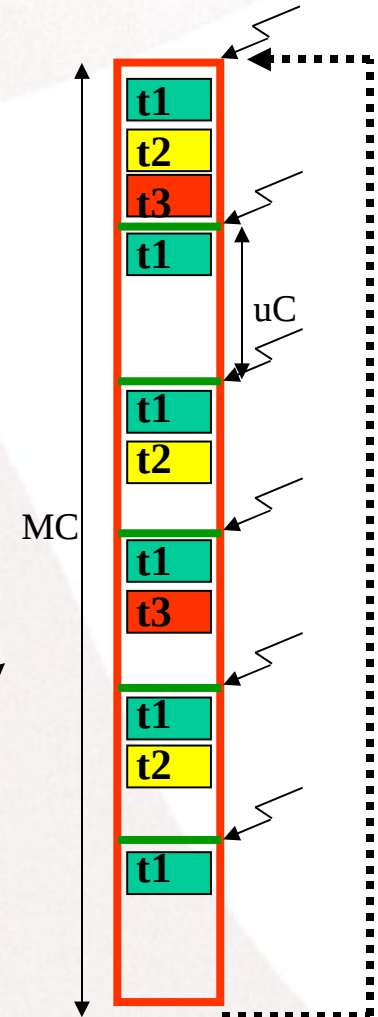
- A tabela é organizada em **micro-ciclos (uC)** de duração fixa para que, quando varrida, se obtenha o carácter periódico das tarefas.
- Os micro-ciclos são disparados por um timer.
- O varrimento contínuo da tabela resulta num padrão cíclico global chamado **macro-ciclo (MC)**

$$\Gamma = \{ \tau_i (C_i, \Phi_i, T_i, D_i, i=1..n) \}$$

$$uC = MDC(T_i) \quad (GCD)$$

$$MC = MMC(T_i) \quad (LCM)$$

$$\begin{aligned} \Phi_i &= 0, C_i = 1\text{ms}, \\ T_1 &= 5\text{ms} \\ T_2 &= 10\text{ms} \\ T_3 &= 15\text{ms} \end{aligned}$$



# Escalonamento estático cíclico

## A favor

- Implementação simples (timer+tabela)
- *Overhead* de execução muito baixo (*dispatcher*)
- Permite optimização do escalonamento (e.g. controlo de *jitter*, relações de precedência)

## Contra

- Pouco escalável (alterações nas tarefas podem causar grandes alterações na tabela, em particular podem levar a tabelas enormes!)
- Pouco robusto a sobrecargas (sensível ao efeito dominó)



# *Escalonamento estático cíclico*

## Construção da tabela

- Calcular o micro-ciclo uC e o macro-ciclo MC
- Expressar os períodos e fases iniciais em micro-ciclos
- Determinar os ciclos onde as tarefas são activadas
- Utilizando um critério de escalonamento adequado, determinar a ordem de execução das tarefas activas
- Verificar se todas as tarefas activas num micro-ciclo podem ser completamente executadas nele. Senão algumas terão que ficar para ciclos seguintes
- Poderá ser necessário partir uma tarefa em várias partes de modo a cada uma poder ser executada dentro de um micro-ciclo.

# ***Resumo da Aula 5***

- O conceito de **complexidade temporal**
- Definição de **escalonamento** e de **algoritmo de escalonamento**
- Algumas **técnicas preliminares** de escalonamento (EDD, EDF, BB)
- Escalonamento **estático cíclico**