Ensaio sobre os Sistemas de Tempo Real

Arnaldo S. R. Oliveira, Luís Almeida

***Abstract* – This paper introduces the fundamentals of real-time operating systems. The topics covered include the motivation for using this class of systems, the application domains, the tasks used for their implementation and the scheduling policies normally employed.**

***Resumo* – Este artigo faz uma introdução aos sistemas operativos de tempo real. São abordados alguns dos aspectos mais importantes deste tipo de sistemas, tais como a motivação para a sua utilização, as áreas de aplicação, a utilização de tarefas na sua implementação e os algoritmos de escalonamento empregues na execução dessas tarefas.**

***Keywords* – Real-time systems, operating systems, task scheduling**

***Palavras chave* – Sistemas de tempo real, sistemas operativos, escalonamento de tarefas**

I. Introdução

Numa aplicação de controlo podem em geral ser identificados três componentes básicos (figura 1):

- O **controlador**, que envia sinais para o sistema controlado, de acordo com uma estratégia ou algoritmo de controlo predefinido;
- O **sistema controlado**, que interage com o controlador através de sensores e actuadores.

Do ponto de vista de interação entre sistema e o ambiente, os sistemas de controlo podem ser de três tipos distintos:

- **Sistemas de monitoração** - não modificam o ambiente;
- **Sistemas de controlo em malha aberta** - modificam o ambiente de uma forma cega ou pouco precisa;
- **Sistemas de controlo em malha fechada** - modificam o ambiente de forma precisa através de uma interação estreita entre percepção e acção.

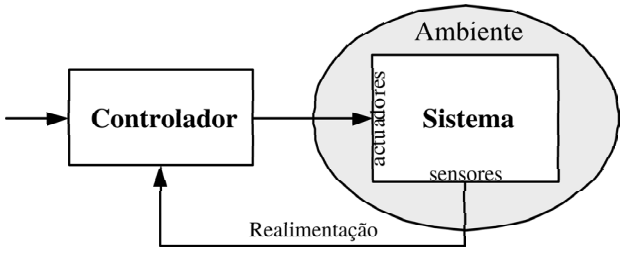

Figura 1 - Diagrama de blocos de um sistema de controlo.

Nesta discussão estamos interessados apenas no último tipo, pois é aquele em que existe uma grande ligação entre as partes sensorial e de controlo do sistema. A interacção estreita com o ambiente requer que o sistema responda aos eventos ocorridos no ambiente dentro de restrições temporais precisas, impostas pela sua dinâmica. Por outras palavras, precisamos de um **sistema que responda no instante certo ou em tempo real**. Daqui resulta que um sistema de tempo real é um sistema reactivo, isto é, um sistema que responde continuamente a eventos externos, neste caso do ambiente. Do ponto de vista computacional, a sua execução é despoletada pela ocorrência de eventos que podem ser de vários tipos, desde acontecimentos assíncronos (como a activação de um sensor) a sinais periódicos produzidos por um temporizador. Em qualquer dos casos, o sistema reaje ao evento executando uma tarefa ou programa onde é determinada e desencadeada a resposta ao evento.

II. Generalidades

A. Definição

Mas afinal o que é um sistema de tempo real? Um sistema de tempo real é:

- Um sistema computacional capaz de responder a eventos dentro de restrições temporais precisas;
- Um sistema cujo funcionamento correcto não depende apenas do valor das saídas mas também do instante em que são produzidas;
- Um sistema cuja evolução temporal deve estar sincronizada com a do ambiente em que opera.

Em sistemas de controlo de tempo real não triviais, para facilitar a análise, projecto e manutenção, a sua implementação é decomposta num conjunto de tarefas ou processos a executar sobre um executivo ou sistema operativo. É responsabilidade do respectivo núcleo efectuar o escalonamento e a execução das tarefas de forma a respeitar as restrições temporais do sistema.

B. Domínios de Aplicação

Os sistemas de tempo real encontram-se nas mais variadas aplicações, tais como:

- Militares e de defesa;
- Controlo de instalações químicas e nucleares;
- Robótica;
- Transportes e controlo aéreo;
- Telecomunicações, etc.

Os sistemas de tempo real podem também ser utilizados para modelar e simular processos, sempre que a precisão temporal também seja importante e não apenas a sequência de operações.

C. Projecto

Apesar do vasto domínio de aplicação, grande parte dos sistemas de tempo real são tradicionalmente projectados usando técnicas empíricas, tais como:

- Programação a baixo nível (utilização de linguagem *assembly* e construção de *device drivers*);
- Uso extensivo de temporizadores;
- Manipulação de prioridades das tarefas usando sistemas operativos convencionais.

Esta abordagem:

- Torna o desenvolvimento da aplicação enfadonho e altamente dependente da habilidade do programador;
- Complica a compreensão do código e a sua manutenção;
- Dificulta a verificação do cumprimento das restrições temporais.

Em conjunto, estas desvantagens conduzem inevitavelmente a uma fraca fiabilidade do sistema. Por este motivo tem-se assistido nos últimos anos a uma utilização crescente de executivos e sistemas operativos especializados, à migração para linguagens de alto-nível e à adopção de metodologias e técnicas formais de análise e projecto.

D. Velocidade, Desempenho e Previsibilidade

É importante notar que um sistema de tempo real não significa um sistema rápido porque a velocidade é sempre relativa ao ambiente em que o sistema opera. Uma execução rápida é uma vantagem mas não é suficiente para garantir um comportamento temporal correcto.

O objectivo de um sistema de tempo real é garantir a correcção temporal de cada tarefa individualmente, enquanto o objectivo de um sistema rápido é minimizar o tempo médio de resposta de um conjunto de tarefas. Dito de outra forma, um bom desempenho médio das tarefas não é suficiente para garantir o desempenho, a previsibilidade e a correcção temporal de cada tarefa individualmente.

As fontes de não determinismo do desempenho podem ser de várias naturezas:

- Arquitectura - *cache*, *pipelining*, interrupções, DMA (*Direct Memory Access*);
- Sistema operativo - escalonamento, sincronização e comunicação entre processos;
- Linguagem - falta de suporte para especificação explícita de parâmetros e restrições temporais;
- Metodologias de projecto - inexistência ou escassez de técnicas de análise e verificação.

A experiência tem demonstrado que a simulação e o teste do sistema, apesar de necessários, permitem apenas avaliar parcialmente o seu comportamento, isto é, o teste de um sistema não faz uma verificação exaustiva do seu comportamento. Além disso, um sistema de tempo real para aplicações críticas deve ser:

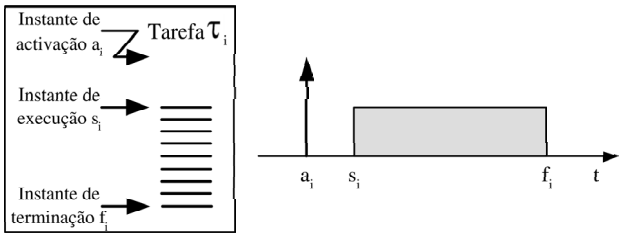

Figura 2 - Tarefas ou processos.

- Projectado assumindo as condições mais desfavoráveis;
- Previsível em situações de sobrecarga;
- Tolerante a falhas.

Tal como já foi referido, em sistemas de tempo real baseados em sistemas operativos é importante que seja o respectivo núcleo a assegurar o cumprimento das restrições temporais, a resposta em tempo real aos eventos e a tolerância a falhas.

III. TAREFAS

Tal como já foi dito acima, os sistemas de tempo real são, em geral, constituídos por um conjunto de tarefas ou processos que executam concorrentemente. Uma tarefa é uma sequência de instruções que, na ausência de outras actividades, é executada ininterruptamente pelo processador até ser completada (figura 2). De forma simplificada, uma tarefa pode-se encontrar num dos seguintes estados (figura 3):

- **A executar** (*running*);
- **Pronta a executar** (*ready*);
- **Bloqueada** (*blocked*).

Uma tarefa diz-se:

- **A executar -** se estiver a ser executada pelo processador;
- **Pronta a executar** - se estiver à espera que lhe seja atribuído tempo de processador.
- **Bloqueada -** se estiver à espera de um evento.

Uma tarefa a executar ou pronta a executar, isto é, se puder ser executada pelo processador num dado instante, designa-se por activa.

A figura 3 ilustra também os eventos responsáveis pela transição entre os diversos estados em que uma tarefa se pode encontrar.

As tarefas activas são armazenadas numa fila de espera chamada *ready queue* (figura 4). À estratégia usada para escolher uma tarefa entre as que se encontram activas e atribuir-lhe tempo de processador (*Central Processing Unit - CPU*) chama-se algoritmo ou política de escalonamento.

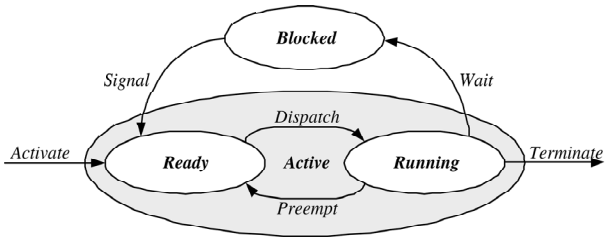

Figura 3 - Diagrama simplificado de transição de estados de uma tarefa.

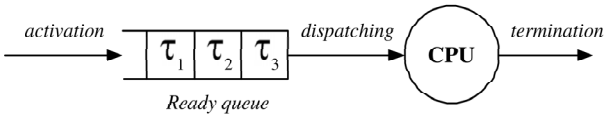

Figura 4 - Lista de tarefas prontas a executar (*ready queue*).

A. *Níveis de Criticalidade*

Em termos de níveis de criticalidade, as tarefas podem ser caracterizadas da seguinte forma:

- **Ordinárias** - se não possuírem quaisquer restrições temporais;
- **Soft real-time** - se o não cumprimento de uma restrição temporal causar apenas uma degradação do desempenho do sistema;
- **Hard real-time** - se o não cumprimento de uma restrição temporal causar efeitos catastróficos no sistema controlado.

Exemplos de tarefas *hard real-time* são: a aquisição de valores de sensores, o controlo de baixo nível e a gestão de alarmes. A leitura de dados do teclado, a interpretação de comandos do utilizador e a gestão do interface gráfico são exemplos de tarefas *soft real-time*. Um sistema operativo capaz de gerir tarefas de tempo real designa-se por sistema operativo de tempo real. Se além disso, também suportar tarefas *hard real-time* é chamado sistema operativo *hard real-time*.

B. *Parâmetros*

Uma tarefa τ_i é caracterizada por vários parâmetros (figura 5 e 6):

- a_i - instante de activação;
- s_i - instante de execução;
- f_i - instante de terminação;
- d_i - *deadline* absoluta;
- D_i - *deadline* relativa;
- C_i - tempo máximo de execução (*Worst Case Execution Time - WCET*);
- $c_i(t_k)$ - tempo máximo de execução residual;
- $x_i(t_k)$ - tempo máximo de relaxamento (*slack*).

O tempo máximo de relaxamento é definido como sendo:

$$x_i(t_k) = d_i - t_k - c_i(t_k)$$

Se $x_i(t_k) < 0$ a tarefa pode não ser capaz de terminar antes de d_i, ou seja, há fortes possibilidades de ocorrer uma violação temporal (*deadline miss*). Note-se que $c_i(t_k)$ é um majorante do tempo de execução remanescente e como tal $x_i(t_k)$ é um minorante do relaxamento da tarefa num dado instante.

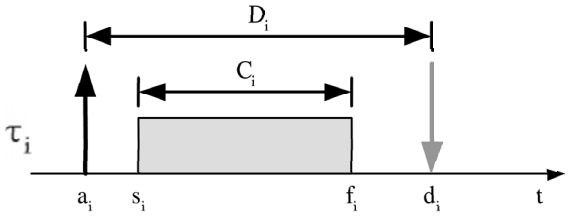

Figura 5 - Parâmetros temporais de uma tarefa.

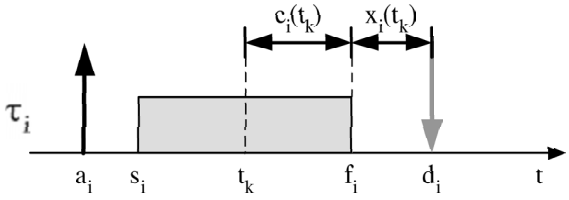

Figura 6 - Parâmetros temporais de uma tarefa (cont.).

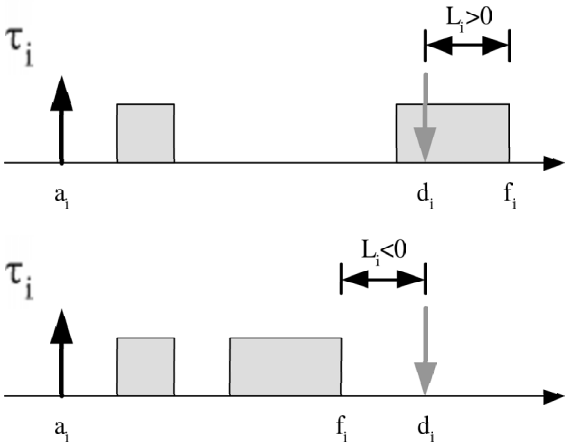

Figura 7 - Definição do atraso (*lateness*) de uma tarefa.

Pode-se ainda definir o atraso L_i de uma tarefa como sendo (figura 7):

$$L_i = f_i - d_i$$

Se $L_i > 0$ significa que a tarefa já ultrapassou o instante de terminação tendo ocorrido uma *deadline miss*.

C. *Modos de Activação*

Em termos de modo de activação, as tarefas podem ser de dois tipos distintos:

- **Periódicas** (*time driven*) (figura 8) - a tarefa é automaticamente activada pelo núcleo em intervalos de tempo regulares;
- **Aperiódicas** (*event driven*) (figura 9) - a tarefa é activada assincronamente quando ocorrer um evento ou através da invocação explícita de uma primitiva de activação.

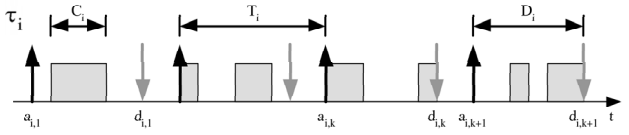

Figura 8 - Modelo periódico de activação de tarefas.

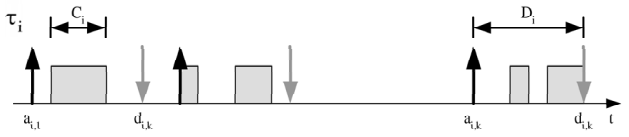

Figura 9 - Modelo aperiódico de activação de tarefas.

Uma tarefa periódica, além dos parâmetros acima, tem também um período T_i associado.

Numa tarefa periódica caracterizada pelo terno $\tau_i(C_i, T_i, D_i)$ verificam-se as seguintes relações:

$$a_{i,k} = a_{i,k-1} + T_i$$

$$d_{i,k} = a_{i,k} + D_i$$

$$C_i \leq T_i$$

em que k é a k-ésima activação da tarefa.

O período e a *deadline* relativa são frequentemente iguais, ou seja:

$$D_i = T_i$$

neste caso, a tarefa deve completar antes do instante da activação seguinte.

Numa tarefa aperiódica:

$$a_{i,k+1} > a_{i,k}$$

Se para uma tarefa aperiódica for possível definir um tempo mínimo entre activações consecutivas, esta designa-se por esporádica:

$$a_{i,k+1} \geq a_{i,k} + MIT_i$$

D. Restrições

As restrições impostas às tarefas podem ser de vários tipos:

- **Temporais** - definem os diversos parâmetros e restrições temporais da tarefas, tais como período, instante inicial de activação, *jitter*, *deadline*, etc.;
- **Precedência** - estabelecem a ordem de execução das tarefas;
- **Recursos** - forçam a que o acesso a recursos partilhados seja mutuamente exclusivo.

IV. Escalonamento de Tarefas

Um escalonamento é uma atribuição particular de tarefas ao processador. De uma forma geral, o problema

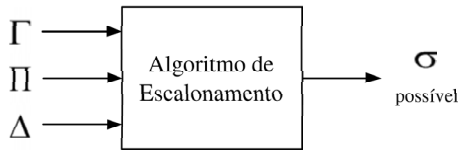

Figura 10 - Problema do escalonamento de tarefas.

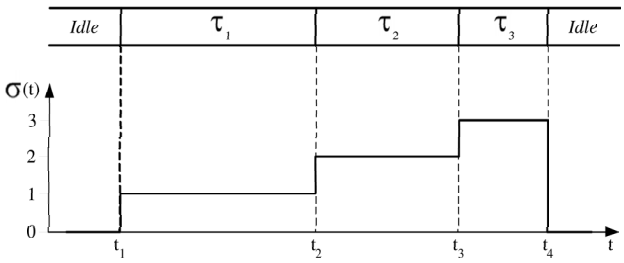

Figura 11 - Exemplo de um escalonamento de 3 tarefas.

do escalonamento de tarefas pode ser formulado da seguinte maneira (figura 10): dados,

- um conjunto Γ de l tarefas,
- um conjunto Π de m processadores,
- um conjunto Δ de n recursos,

determinar uma atribuição de Γ a Π e Δ que produza um escalonamento σ possível.

Um escalonamento diz-se possível se todas as tarefas puderem ser executadas e terminadas cumprindo as restrições impostas. Um conjunto de tarefas diz-se escalonável se existir um escalonamento possível.

Em 1975, Garey e Johnson mostraram que o problema do escalonamento na sua forma geral é *NP-hard*. No entanto, dentro de condições particulares podem ser utilizados algoritmos de complexidade polinomial. Para este efeito é comum assumir algumas simplificações, tais como:

- Processador único;
- Conjunto de tarefas homogéneo;
- Tarefas completamente preemptivas;
- Activação simultânea das tarefas;
- Ausência de restrições de precedência;
- Ausência de restrições de recursos.

A figura 11 ilustra um possível escalonamento de 3 tarefas. Nos instantes t_1, t_2, t_3 e t_4 é realizada a comutação do contexto. Cada intervalo $[t_i, t_{i+1}[$ é chamado *time slice*.

Formalmente, um escalonamento é definido da seguinte forma: dado um conjunto de tarefas Γ

$$\Gamma = \{\tau_1, \ ..., \ \tau_l\}$$

um escalonamento σ é um mapeamento:

$$\sigma : \mathbb{R}^+ \longrightarrow \mathbb{N}$$

tal que,

$$\forall t \in \mathbb{R}^+, \ \exists t_1, t_2 : \ t, t' \in [t_1, t_2[: \sigma(t) = \sigma(t')$$

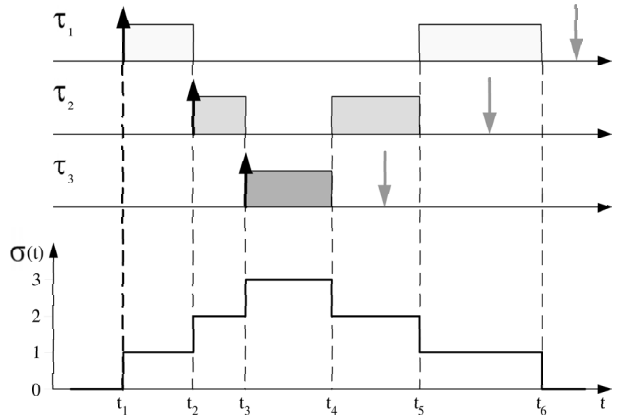

Figura 12 - Exemplo de um escalonamento preemptivo de 3 tarefas.

$$\sigma(t) = \begin{cases} i > 0 & \text{, se } \tau_i \text{ está a executar} \\ 0 & \text{, se o processador estiver livre} \end{cases}$$

A. Taxonomia

Os algoritmos de escalonamento podem em geral ser classificados de acordo com a seguinte taxonomia:

- Preemptivo *vs.* Não preemptivo;
- Estático *vs.* Dinâmico;
- *On-line vs. Off-line*;
- Óptimo *vs.* Sub-óptimo.

Diz-se que um algoritmo de escalonamento é preemptivo se a tarefa a executar puder ser temporariamente suspensa na *ready queue* de forma a executar uma tarefa mais importante. Caso contrário, isto é, se a tarefa a executar não puder ser suspensa até terminar diz-se que o algoritmo é não preemptivo. Na figura 12 é ilustrado um exemplo de um escalonamento preemptivo.

Se as decisões de escalonamento forem tomadas com base em parâmetros fixos, atribuídos estaticamente às tarefas antes das sua activação, diz-se que o algoritmo de escalonamento é estático. Se em vez disso o escalonamento for baseado em parâmetros cujo valor pode variar com o tempo, o algoritmo diz-se dinâmico.

Por outro lado, se todas as decisões forem tomadas antes da activação da tarefa, isto é, se o escalonamento estiver armazenado numa tabela, o escalonamento diz-se *Off-line*. Se o escalonamento for feito durante a execução para o conjunto das tarefas activas diz-se *On-line*.

O algoritmo diz-se óptimo se no caso de existir um escalonamento possível, ele for capaz de o determinar, caso contrário designa-se por sub-óptimo.

B. Políticas Clássicas

Nos sistemas operativos convencionais são usadas diversas políticas de escalonamento clássicas, tais como:

- *First Come First Served*;
- *Shortest Job First*;
- *Priority Scheduling*;
- *Round Robin*.

Todas estas políticas têm vantagens e desvantagens e visam distribuir de uma forma "justa" o tempo de processador por todas as tarefas do sistema. A escolha de uma política em particular depende do fim em causa. No entanto, estas políticas não devem ser utilizadas em sistemas de tempo real porque não garantem o cumprimento das restrições temporais impostas às tarefas, não garantindo uma resposta em tempo real e previsível do sistema.

C. Políticas para Sistemas de Tempo Real

Os sistemas de tempo real têm restrições temporais que devem ser tomadas em conta pelo algoritmo de escalonamento. Mais concretamente, possuem um tempo limite de execução (*deadline*) que deve ser respeitado sob a pena dos valores produzidos não serem válidos. Os algoritmos para escalonamento de tarefas de tempo real baseiam-se essencialmente nos parâmetros período, *deadline* relativa e *deadline* absoluta. Nos primeiros dois casos o escalonamento é baseado num parâmetro estático enquanto no segundo caso baseia-se num parâmetro dinâmico.

C.1 Escalonamento Cíclico

O primero algoritmo ou política usada no escalonamento de tarefas de tempo real periódicas foi o escalonamento cíclico (*timeline schedule*). O escalonamento usando esta política é feito *off-line* e consiste no seguinte método:

- O eixo do tempo é dividido em intervalos de igual duração (*time slots*);
- Cada tarefa é alocada estaticamente nas *slots* necessárias de forma a cumprir a periodicidade desejada;
- A execução em cada *slot* é activada por um temporizador.

Esta política de escalonamento é:

- Simples de implementar, não necessitando de um sistema operativo de tempo real;
- Possui uma sobrecarga computacional reduzida durante a execução, uma vez que todo o escalonamento é feito *off-line*;
- Permite um melhor controlo sobre flutuações (*jitter*) na execução das tarefas.

No entanto, tem as seguintes desvantagens:

- A adição de novas tarefas obriga à reconstrução de todo o escalonamento;
- A integração de tarefas aperiódicas é bastante difícil;
- Não é robusto em situações de sobrecarga.

Em situações de sobrecarga, isto é, quando não existirem os recursos computacionais necessários para executar todas as tarefas cumprindo as restrições impostas, pode-se:

- Permitir que a tarefa que esteja a violar as restrições temporais continue a executar;

- Abortar a sua execução.

No primeiro caso pode ocorrer o efeito de dominó nas restantes tarefas do sistema uma vez que o atraso provocado por uma tarefa pode propagar-se às restantes. No segundo caso o sistema pode ficar num estado inconsistente.

Para flexibilizar e tornar mais robusto o escalonamento de tarefas de tempo real em situações de sobrecarga foram propostas outras políticas baseadas em prioridades.

C.2 Escalonamento Baseado em Prioridades

No escalonamento baseado em prioridades, a cada tarefa é atribuída uma prioridade calculada a partir das suas restrições temporais. A execução das tarefas é feita por um executivo baseado em prioridades. Este método permite também efectuar uma análise de escalonabilidade usando técnicas analíticas. As prioridades são em geral atribuídas com base nos parâmetros período e *deadline*. Certas políticas de escalonamento têm também em conta o tempo máximo de execução quer para efeitos de escalonamento quer para análise de escalonabilidade. Assim, em vez do escalonamento cíclico, em sistemas de tempo real são normalmente empregues as seguintes políticas de escalonamento:

- *Rate Monotonic (RM)*;
- *Deadline Monotonic (DM)*;
- *Earliest Deadline First (EDF)*;
- *Least Slack First (LSF)*.

No algoritmo **RM**, a cada tarefa é atribuída uma prioridade fixa directamente proporcional à sua frequência de activação. Para o algortimo RM considera-se que $T_i = D_i$, isto é, uma tarefa deve terminar antes da activação seguinte.

O algoritmo RM é considerado óptimo entre os algoritmos de prioridade fixa. Se existir uma atribuição de tarefas ao processador baseada em prioridades fixas que leve a um escalonamento possível então a atribuição RM é possível para Γ. Dito de outra forma, se Γ não for escalonável pelo algoritmo RM, então também não pode ser escalonado por qualquer outro algoritmo de prioridade fixa.

O algoritmo **DM** é uma generalização do RM e seleciona a tarefa com a *deadline* relativa mais pequena. Neste caso $D_i <= T_i$. O algoritmo DM possui as seguintes características:

- Todas as tarefas são activas simultaneamente;
- Prioridade fixa inversamente proporcional a D_i;
- Preemptivo;
- Minimiza o atraso máximo da tarefa com menor D_i.

No algoritmo **EDF** seleciona-se para execução a tarefa com a *deadline* absoluta mais próxima, isto é, cada tarefa recebe uma prioridade proporcional à sua *deadline* absoluta. Possui as seguintes características:

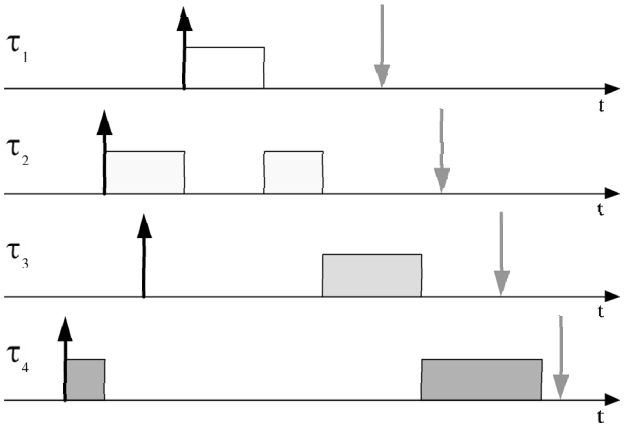

Figura 13 - Exemplo de um escalonamento de 4 tarefas usando o algoritmo EDF.

- Prioridade dinâmica (d_i depende do instante de activação a_i);
- Preemptivo;
- Minimiza o número de mudanças de contexto relativamente às políticas RM e DM.

A figura 13 mostra um exemplo de 4 tarefas usando o algoritmo EDF. O algorimto EDF é considerado óptimo entre todos os algoritmos de escalonamento. Em cada instante o processador é atribuído à tarefa com a *deadline* absoluta mais próxima. Ao contrário do RM permite alcançar uma taxa de ocupação do processador de 100% garantido a escalonabilidade.

Finalmente, o algoritmo **LSF** tem em conta o tempo máximo de execução de cada tarefa. Neste algoritmo, em cada instante é selecionada para execução a tarefa com o menor tempo de relaxamento. Desta forma, à custa de uma sobrecarga computacional mais elevada, é possível fazer um melhor planeamento e prevêr dinamicamente a ocorrência de violações temporais antes de efectivamente acontecerem.

V. CONCLUSÃO

Neste artigo foi feita uma introdução aos sistemas de tempo real. Como conclusão é importante reter que o sistema operativo é a entidade mais responsável por um comportamento previsível. A execução concorrente das tarefas deve ser feita através de algoritmos de escalonamento adequados. Fora desta discussão ficaram os protocolos de sincronização e os mecanismos de comunicação apropriados para sistemas de tempo real, assim como a apresentação das técnicas analíticas para análise de escalonabilidade.

VI. BIBLIOGRAFIA

1. Buttazo, G. (1997). Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications. Kluwer Academic Publishers.
2. Kopetz, H. (1997). Real-Time Systems Design Principles for Distributed Embedded Applications. Kluwer Academic Publishers.
3. Liu, J.W.S. (2000). Real-Time Systems. Prentice Hall.
4. Krishna, C.M. and K. Shin (1997). Real-Time Systems. McGraw-Hill.