UNIVERSIDADE DO PORTO
FACULDADE DE ENGENHARIA

Análise e Concepção de Sistemas de Tempo-Real

por
Carlos Jorge Almeida Costa

Dissertação de Mestrado em Engenharia Electrotécnica e de Computadores
Outubro 1998

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Departamento de Engenharia Electrotécnica e de Computadores

Análise e Concepção de Sistemas de Tempo-Real

Carlos Jorge Almeida Costa

Licenciado em Engenharia Informática pela Escola Superior de Tecnologia e Gestão do Instituto Politécnico da Guarda

Dissertação submetida para a satisfação parcial dos requisitos do grau de mestre em Engenharia Electrotécnica e de Computadores (Área de especialização de Informática Industrial)

Dissertação realizada sob a supervisão do Professor Doutor António José Pessoa de Magalhães, da Secção de Automação, Instrumentação e Controlo do Departamento de Engenharia Mecânica e Gestão Industrial da Faculdade de Engenharia da Universidade do Porto

Porto, Outubro de 1998

Aos meus Pais.

Resumo

Os sistemas de tempo-real fazem cada vez mais parte dos vários aspectos da nossa existência, sendo encontrados numa grande variedade de aplicações que utilizamos frequentemente. Assim, estes podem ser encontrados em simples sistemas embebidos, que controlam sistemas físicos, de entre eles: leitores de CD, sistemas de travagem ABS, pequenos controladores de processos fabris dedicados, etc.; ou em sistemas de tempo-real complexos, de entre eles: sistemas de navegação aeronáutica e aeroespacial, controlo de centrais nucleares, controlo de mísseis, etc. Não subestimando esta integração, a evolução faz-se no caminho do aparecimento de sistemas de tempo-real, cada vez mais complexos e adaptativos.

Acontece, no entanto, que até aos nossos dias estes sistemas, essencialmente os críticos e mais complexos, têm sido desenvolvidos por processos *'ad-doc'* e verificados por simulação. Tais processos, por um lado, não oferecem qualquer garantia demonstrável da sua conformidade com as restrições impostas pelo seu meio ambiente e, por outro, o seu desenvolvimento é bastante dispendioso, quer em termos de tempo, quer em termos de custo económico. Foi neste âmbito que surgiu a presente dissertação.

Assim, esta dissertação pretende contribuir para a criação de uma base científica na análise e concepção de sistemas de tempo-real. Concretamente, pretende estudar os problemas resultantes da implementação prática de sistemas de tempo-real e fornecer meios de teste

da conformidade destes com os requisitos temporais impostos pelo meio ambiente. Como ponto de partida apresenta, as principais *'benchmarks'* adequadas à avaliação de sistemas de tempo-real, disponíveis na literatura. Em seguida, utilizando algumas destas métricas efectua a dedução experimental de metodologias que, entrando em linha de conta com as distorções introduzidas pelo suporte prático aos modelos teóricos de escalonamento, permitam garantir a escalonabilidade de aplicações de tempo-real. São assim apresentadas metodologias para o escalonamento estático de tarefas com base no algoritmo *'Round-Robin'*; e para o escalonamento preemptivo por prioridades fixas baseado no algoritmo *'Rate-Monotonic'*. São apresentadas algumas experiências que permitem validar estes novos modelos de escalonamento práticos.

Palavras-chave:

sistemas de tempo-real, análise e concepção de sistemas de tempo-real, suporte prático, algoritmos de escalonamento, sistemas operativos de tempo-real, *hardware*.

Abstract

Real-Time Systems are more and more common in many aspects of our daily life being found in a great variety of situations that we use frequently. Thus, these can be found on simple systems that control physical systems such as CD players, ABS systems, small automated plants; to complex real-time systems such as aeronautic navigation systems, aerospace navigation systems, control of nuclear reactors, air traffic control and missile control.

Up to the present these more complex and critical systems have been developed by *ad-hoc* techniques and evaluated by simulations. On one hand such techniques do not offer any predictability and conforming to temporal constrains imposed by the environment. On the other hand their development is very expensive both in terms of time and economics.

It is against this background that this thesis presented itself. Thus, through this work the objective is to contribute to the founding of a scientific basis for the analysis and conception of real-time systems. More specifically, this thesis want to study problems that result from practical implementation of real-time systems and provide several testing methods of how these conform with temporal requirements dictated by the environment.

To do this it will present the main *benchmarks* adequate to the evaluation of real-time systems available in the literature. Next, using the most appropriate methods of measuring, it will carry out an experimental calculation of methodologies, taking into consideration the distortions

introduced by the practical support to the theoretical scheduling models to guarantee the schedulability of the real-time application.

This methodologies developed specifically for the static scheduling of tasks are based on the *Round-Robin* algorithm and for the preemptive fixed priorities scheduling based on the *Rate Monotonic* algorithm, This thesis presents also some experiments that will confirm these new scheduling methods.

Keywords:

real-time systems, analysis and conception of real-time systems, practical support, scheduling algorithms, real-time operating systems, hardware.

Résumé

Les systhèmes à temps-réel font, de plus en plus, partie de notre existence. On les retrouvent dans une grande variété d' apllications qu' on utilise souvent. Cela permet de dire que ces systhèmes peuvent être trouvés en systhèmes symples, qui controlent systhèmes physiques, comme par exemple: les CDs, les systhèmes de ABS, les petits controleurs de processus d' usines, etc., et peuvent être aussi des systhèmes à temps-réel complexes, comme par exemple : les systhèmes de navigation aéronautique, les systhèmes de navigation spacial, le contrôle de centres nucléaires, le contrôle aérien, le contrôle de missiles, etc.

Sans oublier cette intégration, l' évolution se fait à travers un chemin où les systhèmes à temps-réel sont de plus en plus complexes et adoptables.

Ce qui arrive, toutefois, c' est que jusqu' au présent ces systhèmes, surtout les plus complexes et critiques, se sont développer par des processus *'ad-hoc'* et ils n' ont été verifiés que par des simulations. Ces processus n' offrent, alors, aucune garantie demonstrative de leur conformité avec les restrictions imposées par l' environnement et leur développement est très cher, au niveau du temps et des coûts.

C' est pour cela que cette dissertation a été elaboré. On prétend, ainsi, avec ce travail développer et contribuer à la création d' une base scientifique pour l' analyse et conception des systhèmes à temps-réel et de fournir les moyens de test qui montrent les restrictions temporels imposés par l' environnement.

Pour cela, on cherche des principales *'benckmarks'* adéquates à l' évaluation des systhèmes à temps-réel disponibles dans la littérature.

Ensuite et en utilisant quelques métriques découvertes plus adéquates à ce propôs, on a effectué la déduction expérimentale des méthodes qui tenant en compte les distortions introduites par les support pratique aux modèles théoriques de *'scheduling'* permettaient de garantir la *'schedulabilyty'* d' applications à temps-réel.

Ayant comme support les méthodes développées au niveau de *'static scheduling'* de tâches soutenues dans le algoritme *'Round-Robin'* et pour les *'preemptive scheduling'* par priorités fixes appuyées dans algoritme *'rate-monotonic'*, on a effectué quelques expériences qui nous on permis de valider ces nouveaux modèles de *'scheduling'*.

Mots-clé:

Systhèmes à temps-réel, analyse et conception de systhème à temps-réel, support pratique, algoritme de *'scheduling'*, systhème opératif de temps-réel, *'hardware'*.

Agradecimentos

De entre as muitas pessoas e entidades inteiramente merecedoras dos meus sinceros agradecimentos pelo auxílio, compreensão e consideração que me dispensaram ao longo de todo este curso de Mestrado cuja última etapa foi a elaboração desta dissertação, e sem querer menosprezar o contributo das omitidas, gostaria de destacar as seguintes.

As minhas primeiras palavras vão para o meu orientador, Professor António Pessoa de Magalhães, pela orientação científica, apoio e disponibilidade na resolução dos problemas que sempre demonstrou.

Não posso deixar de agradecer ao Professor Álvaro Bento Leal o incentivo, a facilidade para a frequência do curso e a amizade demonstrados, ainda quando Presidente do Instituto Politécnico da Guarda.

Ao Professor Fernando Pires Valente, anterior Director da Escola Superior de Tecnologia e Gestão, o meu muito obrigado pela sua amizade, pelas facilidades concedidas para a frequência do Curso e pelo material disponibilizado, sempre útil para a elaboração desta dissertação.

Agradeço, também, à Professora Maria Clara Silveira e ao Professor Gabriel Tavares, amigos e colegas de Departamento, pelas leituras feitas a este trabalho.

Os meus mais sinceros agradecimentos à Dra. Elisabete Constante de Brito, à Professora Maria Paula Martins das Neves e ao Professor Samuel Walter Best, pelo auxílio prestado na traduções dos resumos desta dissertação.

Dirijo um sentido agradecimento a toda a minha família pelo entusiasmo e compreensão demonstrados durante o tempo que durou esta minha formação. Um obrigado especial às minhas irmãs, Mónica e Anita, e à minha mãe, pelo auxílio prestado no processamento de texto e correcções da dissertação ao nível da expressão, o que sem elas não teria sido possível.

Aos meus grandes amigos, Joaquim Baía da Fonseca, Paulo Nunes, José Quitério Figueiredo, Carlos Jorge Brigas, Noel Lopes e José Carlos Miranda, agradeço o incentivo e algumas trocas de impressões que em muito me ajudaram a superar esta dura prova.

Por fim, não podia deixar de agradecer aos colegas e amigos do Instituto Politécnico da Guarda que, directa ou indirectamente, me incentivaram e animaram na progressão do trabalho.

A todos, obrigado.

Índice

Índice de Figuras

Índice de Tabelas

Capítulo 1
Introdução

Um sistema de controlo em tempo-real pode ser definido como um sistema que, desempenhando concretamente as suas tarefas no domínio lógico, é capaz de responder a eventos assíncronos externos e internos, dentro de um intervalo de tempo previamente definido. Tal requisito deve-se ao facto deste tipo de sistema interagir com ambientes complexos e dinâmicos, tipicamente, sistemas físicos. O conjunto – sistema de controlo (controlador) e sistema controlado (ambiente) – constitui um Sistema de Tempo-Real (STR).

Num STR, o valor de um cálculo depende, pois, não só do resultado lógico, mas também do tempo a que os resultados são obtidos. Isto porque este tipo de sistemas tem como restrição o ter que produzir um resultado dentro de um prazo específico, designado por *'deadline'*, com o risco de se tal não acontecer, o resultado ser desnecessário ou mesmo catastrófico. Assim, este tempo não corresponde à rapidez, mas sim ao momento em que são obtidas as respostas. O factor importante é a existência de determinismo no tempo de resposta, ou seja, uma garantia firme do cumprimento das *'deadlines'* a que o sistema está sujeito. É preciso não confundir resposta em tempo-real com resposta rápida, que é um dos mal-entendidos avançados por Stankovic [Sta88].

Em sistemas de tempo-real, o encadeamento da execução dos processos (ou tarefas)[1], é normalmente feito por um escalonador, por forma a que todas as *'deadlines'* sejam cumpridas, sempre que tal seja possível. O conjunto de regras que permitem conduzir o desempenho do escalonador é designado por algoritmo de escalonamento. Actualmente, encontra-se na literatura uma grande quantidade de técnicas e algoritmos para efectuar o escalonamento de várias tarefas de tempo-real. Apesar da sua extrema importância, a maioria destas técnicas e algoritmos desprezam determinados problemas práticos – tanto a nível de *'hardware'* como de *'software'* – arriscando-se, assim, a ver as suas validades e eficiências comprometidas em situações reais. Por esta razão, tanto a análise como a síntese de qualquer sistema de tempo-real requerem um equacionamento conjunto das características das tarefas de tempo-real, no mínimo do algoritmo de escalonamento empregue, do sistema operativo adoptado e do *'hardware'* de suporte. Porém, este tipo de abordagem levanta alguns problemas teóricos e exige disciplina.

Assim, há dois aspectos importantes a ter em consideração no âmbito dos sistemas de tempo-real. Em primeiro lugar, os aspectos teóricos: estes correspondem a desenvolvimentos teóricos resultantes da investigação em torno dos sistemas de tempo-real. São objecto de estudo, nomeadamente: os vários algoritmos de escalonamento e as suas condições de aplicabilidade (garantias teóricas), o estudo das condições necessárias para que um sistema se comporte de forma previsível e as consequências do cumprimento ou não das *'deadlines'* – previsibilidade de sistemas de tempo-real. Em segundo lugar, temos os aspectos práticos: estes correspondem à implementação dos desenvolvimentos teóricos a vários

---

[1] Apesar de poderem ser utilizados ambos os termos (processo ou tarefa) para designarem um mesmo conjunto de operações a ser realizado, para atingir um fim específico, utilizaremos preferencialmente o termo tarefa em detrimento de processo, atendendo ao significado de ambos.

níveis, como sendo *'hardware'*, sistema operativo e linguagem de programação.

Directamente relacionados com os dois aspectos apresentados, temos o desempenho de sistemas de tempo-real (STRs) e a sua medida. Com efeito, podem ser colocadas duas questões de fundo quando se fala em desempenho de STRs. Em primeiro lugar, o que poderemos esperar de um sistema de tempo-real desenvolvido segundo determinadas regras, e porquê. Em segundo, como desenvolver um STR que garanta um determinado desempenho.

Existem várias maneiras de verificar se um sistema de tempo-real cumpre ou não os seus requisitos. Uma aproximação natural é testar o STR através da simulação em numerosos cenários, nos quais é esperado que o sistema funcione. Esta não é uma boa solução: primeiro porque só podemos ter confiança no cumprimento dos requisitos, dentro das áreas cobertas pelos cenários de teste; em segundo porque uma alteração mínima nos parâmetros do sistema pode afectar o seu desempenho, aquando da sua instalação no ambiente final, obrigando a repetir determinados testes. Isto é especialmente verdade quando consideramos requisitos temporais, já que a alteração de uma única instrução pode ter um impacto não desprezável a diversos níveis temporais do sistema. Em sistemas grandes e complexos, esta abordagem causou o adiamento de missões e o aumento de custos [Sta88].

Outra aproximação é a medida directa do desempenho do sistema através da introdução de sistemas de análise e medida (*'probes'*[2]) para monitorar o comportamento do sistema. Mais uma vez esta abordagem não é a solução: em primeiro lugar, as alterações no desempenho são inevitáveis, uma vez

---

[2] Poderemos designar este termo em português por "provas", fazendo uma tradução livre. Contudo, pensamos que perde algum significado relativamente ao termo original e, por conseguinte, utilizaremos o termo original.

que há um aumento da carga do sistema pela introdução das *'probes'*; em segundo, encontrar problemas nesta fase de desenvolvimento de um sistema (em fase de exploração) pode implicar um aumento significativo de custos e/ou tempo para voltar a desenvolver o sistema. Tanto as aproximações baseadas em simulação como em experimentação apresentadas conseguem dar uma garantia relativa, em termos estocásticos (probabilísticos), que o sistema cumpre os requisitos. Porém, as modernas aplicações de tempo-real, muitas delas de alto risco, exigem garantias absolutas de desempenho, pelo que uma "garantia estocástica" não é suficiente.

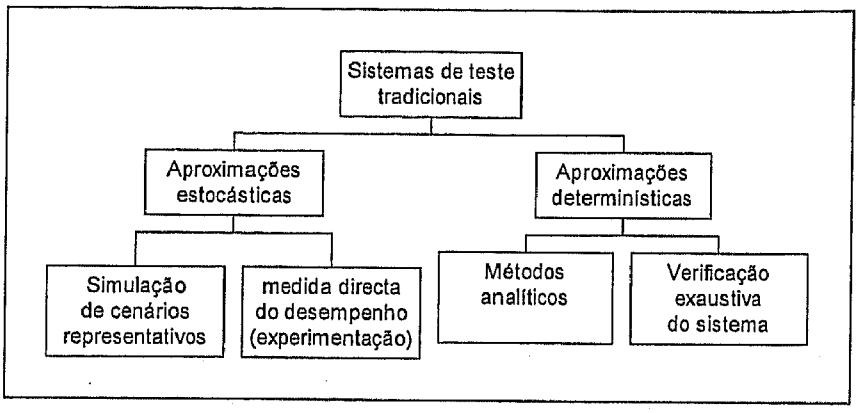

Fig. 1– Teste de sistemas de tempo real em termos de cumprimento dos requisitos temporais.

Contrastando com a abordagem probabilística, aparecem aquelas baseadas em métodos analíticos que equacionam o comportamento do sistema nos cenários mais desfavoráveis. Caso os resultados sejam satisfatórios, sabe-se que o sistema cumpre na perfeição os requisitos a que está sujeito. Estas técnicas entram em linha de conta com os padrões de chegada de estímulos, tempos de execução das tarefas e hipóteses de falhas. Estas abordagens, baseadas, como se disse, nos casos mais desfavoráveis, são essenciais para verificar a confiabilidade das partes críticas de um sistema de tempo-real. É importante notar que os tempos de execução de programas no pior caso dependem do *'hardware'*, do sistema operativo, do compilador e da linguagem de programação. Há pois que as equacionar devidamente e em conjunto. Muitas das características de *'hardware'* introduzidas para aumentar o desempenho no caso médio de programas (*'caches'* de disco e memória, *'buffers'* para dispositivos lentos) levantam problemas, quando é

necessário informação sobre o comportamento no caso mais desfavorável. Apesar de ter um comportamento aceitável na maior parte das vezes, poderá haver situações em que o comportamento é péssimo. Por exemplo, no caso das *'caches'*, estas normalmente guardam a informação que estatisticamente é mais solicitada e neste caso a resposta é rápida, mas se o pretendido é menos solicitado, então o tempo de resposta é mais demorado. De forma similar, as optimizações introduzidas pelos compiladores contribuem para a fraca previsibilidade do tempo de execução do código, isto porque, normalmente, as optimizações são levadas a cabo pela supressão, reorganização e substituição de código, que afecta as características temporais da aplicação. As interferências do sistema de suporte (sistema operativo) para manipulação de interrupções, acesso a memória partilhada e preempção de processos complica ainda mais a questão. Em suma, qualquer aproximação à determinação dos tempos de execução de programas de tempo-real deve ter em conta muitos factores, sendo pois uma actividade bastante complexa.

1.1 Objectivos

Considerando os aspectos focados na secção anterior, constatamos com toda a naturalidade que é necessário, por um lado, encontrar um conjunto de métricas designadas por *'benchmarks'*[3] que permitam avaliar a qualidade dos projectos concebidos seguindo determinadas regras – algoritmos de escalonamento, desempenho de sistemas operativos, características do *'hardware'*, etc; e por outro lado, encontrar ou definir metodologias (eventualmente heurísticas) que permitam conduzir um projecto no sentido

[3] *'Benchmarks'* é o termo original para designar medida de desempenho, o qual iremos utilizar ao longo desta dissertação, por não termos encontrado um equivalente em português.

de vir a alcançar determinado desempenho (o respeito pelas *'deadlines'* é um aspecto fundamental).

Isto é importante porque:

- Os sistemas de tempo-real em geral, e os críticos em particular, tendem a encerrar algum secretismo – o comportamento dos STRs tende a tornar-se imprevisível, quando alguns parâmetros são alterados, ainda que ligeiramente. Acontece, nomeadamente, com os tempos de execução de tarefas, *'deadlines'*, etc., os quais influenciam o resto do sistema, concretamente o seu desempenho.

- Acreditar unicamente nos desenvolvimentos teóricos não é solução – estes são desenvolvidos para cenários bem conhecidos, com restrição de variáveis e algumas condições pré-definidas, e não se sabe como se comportam em todas as situações.

- O desempenho do suporte prático (*'hardware'*, sistema operativo, linguagem de programação, etc.) tem que ser bem equacionado, mas nem sempre são conhecidas as formas de o fazer.

Esta dissertação pretende, pois, contribuir para uma melhor compreensão das vicissitudes dos sistemas de tempo-real, tanto a nível de concepção, como a nível de validação. Concretamente, é objectivo desta dissertação contribuir para a compreensão das vicissitudes introduzidas pelo suporte prático dos sistemas de tempo-real, com especial relevo para a sobrecarga computacional introduzida pelo *'software'* de suporte às aplicações. Esta compreensão é importante, no sentido em que permite quantificar diversas restrições temporais a que os sistemas de tempo-real estão efectivamente sujeitas e, a partir daí, desenvolver metodologias que permitam avaliar de forma efectiva o desempenho de um sistema genérico.

Por outro lado, e para que seja possível entrar em linha de conta com os condicionalismos temporais mencionados, é necessário quantificá-los. Muitos deles (tempo de mudança de contexto, tempo para tratamento de

interrupções, etc.) são de algum modo fornecidos pelos fabricantes. Porém, verifica-se que, por vezes, estes não indicam as condições (se o pior caso ou o caso médio) em que foram obtidos, pelo que é necessário saber como deduzi-los em determinado contexto. O segundo objectivo desta dissertação é precisamente encontrar técnicas que nos permitam obter os condicionalismos temporais, com o nível de confiança necessário às aplicações de tempo-real.

1.2 Organização da dissertação

Esta dissertação está estruturada de forma a apresentar e discutir paulatinamente os vários assuntos, que permitirão atingir os objectivos propostos. Assim, os vários capítulos a desenvolver são os que a seguir se apresentam.

No capítulo dois faz-se a definição do problema. Aí procuramos dar uma definição formal de sistemas de tempo-real, nomeadamente em termos do âmbito de Aplicação, principais características e factores diferenciadores dos Sistemas Computacionais tradicionais. Seguidamente, faremos uma retrospectiva das técnicas de Concepção e Análise de Sistemas de Tempo-Real, apontando os principais problemas inerentes, nomeadamente as técnicas *'ad-hoc'* e por simulação que constituíram, até aos nossos dias, os principais meios de desenvolvimento de sistemas de tempo-real. Em consonância com alguns autores consagrados na área, apontaremos as vantagens da criação de uma base científica para o desenvolvimento de sistemas de tempo-real, de forma a que se torne mais fácil demonstrar a previsibilidade destes. Assim, previsibilidade é a palavra-chave, que só é conseguida através de um conhecimento profundo em torno dos sistemas de tempo-real, concretamente dos desenvolvimentos teóricos e da implementação prática em aplicações concretas. Aqui, salientaremos os principais problemas encontrados nos dois aspectos – teórico e prático.

Nos aspectos teóricos são definidas, concretamente, as subtilezas presentes nas principais características dos sistemas de tempo-real que podem condicionar a sua previsibilidade. Em relação a esta última, serão descritos os níveis de detalhe nos quais a previsibilidade deve ser demonstrada – macroscópica e microscopicamente – para além de algumas considerações sobre a previsão de comportamento, em termos determinísticos e estocásticos.

Nos aspectos práticos serão salientados os principais problemas, especialmente aqueles com o escalonamento de tarefas de tempo-real sobre suportes práticos.

O capítulo três é designado por Projecto de sistemas de tempo-real. Aqui são materializados os sistemas de tempo-real nos seus aspectos teórico e prático. Sem dúvida nenhuma, um dos componentes essenciais dos sistemas de tempo-real são os algoritmos de escalonamento, devido à simultaneidade de operações. Esta questão tem suscitado muita investigação, sendo muito vasta a documentação a respeito. Por isso, é feito um levantamento sobre esta matéria, sendo abordadas, concretamente, as principais classificações dos algoritmos de escalonamento, assim como os principais algoritmos. Em relação a estes últimos serão tidos em atenção as características das tarefas (independentes, dependentes, periódicas, aperiódicas) e a estrutura do próprio sistema (sistemas monoprocessador e multiprocessador).

Em relação aos aspectos práticos, faremos a concretização dos vários níveis de suporte em termos de disponibilidade, funcionalidade e garantias efectivas para os sistemas de tempo-real. Os níveis considerados são: *'hardware'*, sistema operativo e linguagens de programação de tempo-real. Incidir-se-á mais sobre os sistemas operativos de tempo-real, como base principal para os STR. Aqui, começaremos por definir as principais características e os factores importantes para a diferenciação: multitarefa, protecção de memória, modularidade, suporte normativo, comportamento em operação. Em seguida, define-se uma classificação dos sistemas de tempo-real existentes em três classes distintas: núcleos pequenos,

extensões a sistemas operativos clássicos e resultantes de investigação. No que diz respeito às linguagens de programação e do *'hardware'* efectuar-se-á a definição das principais características que estes suportes devem apresentar para serem considerados em conformidade com os requisitos dos STRs. Deste capítulo, ressaltam essencialmente concepções de sistemas de tempo-real.

O capítulo quatro, designado por "Teste de Sistemas de Tempo-Real", é dedicado à apresentação das metodologias e marcas de desempenho (*'benchmarks'*) encontradas na literatura, que permitem testar e validar sistemas de tempo-real. Este capítulo constitui um estudo que "serve de guia" às experiências a apresentar posteriormente nesta dissertação. Os testes aqui apresentados poderão ser utilizados nas várias fases de desenvolvimento ou exploração. Na fase de concepção, destacamos e descrevemos o método RMA (*''Rate Monotonic Analisys'*), que constitui uma ferramenta de análise de escalonabilidade baseada no algoritmo *'Rate Monotonic'*, independente da implementação. Na fase de desenvolvimento, são apresentadas várias técnicas de avaliação de desempenho que podem ser utilizadas nos dois níveis seguintes: linguagens de programação e sistema operativo.

É sugerido neste capítulo uma classificação bastante utilizada na literatura da especialidade e que permite agrupar as várias *'benchmarks'* segundo o grau de detalhe, no qual se avalia o sistema alvo. Assim, temos as *'fine-grained benchmarks'*, que avaliam um sistema de baixo nível, isto é, pela medida da eficiência das primitivas do sistema mais passíveis de serem utilizadas pelos sistemas de tempo-real. O segundo grupo são as *'Application-oriented benchmarks'*, que avaliam o sistema a um nível superior, normalmente ao nível da aplicação em termos de *'deadlines'* mantidas ou perdidas. A terceira abordagem agrupa as avaliações por simulação e são designadas por *'Simulation-based evaluations'*. Cada uma das *'benchmarks'* apresentadas pode pertencer a uma ou mais das classes mencionadas.

No capítulo cinco é feita uma caracterização do meio de desenvolvimento. Pretendemos, assim, dar a conhecer o ambiente experimental em termos de *'hardware'* e sistema de medida utilizados para a realização dos vários testes. Estes testes constituem a parte experimental desta dissertação. Relativamente ao *'hardware'*, optámos por uma plataforma genérica, simples, com muito suporte em termos de *'hardware'* e *'software'* e que constitui a base de muitos sistemas embebidos. Estamos a falar de um computador pessoal baseado no processador Intel 80 386 SX a 25 MHz. O *'software'* engloba as soluções de *'software'* de base (sistema operativo), linguagem de programação e aplicação sintética desenvolvida para testar os modelos de escalonamento. Relativamente ao sistema operativo, optámos por um núcleo de domínio público sem *'royalities'*, que utiliza o MS-DOS como sistema operativo hospedeiro, de entre vários encontrados na Internet. Esta escolha deve-se essencialmente à facilidade de obtenção, ao facto de estar disponível à maioria dos investigadores e de permitir que estes possam confirmar os dados obtidos, de utilizar um sistema operativo bem documentado e com várias ferramentas de desenvolvimento disponíveis, etc. O núcleo escolhido foi o *Ctask*, um Executivo multitarefa para a linguagem de programação *'C'*. Por conseguinte, a linguagem de programação utilizada foi o *'C'*, o que apresenta várias vantagens: ser uma linguagem de âmbito geral, bem documentada, com facilidades de manipulação ao nível do SO (sistema operativo), com ambientes de desenvolvimento bastante poderosos e compiladores fáceis de utilizar para MS-DOS, a partir do próprio ambiente de desenvolvimento. A aplicação sintética corresponde à aplicação pensada para testar os princípios de escalonabilidade e é constituída por um conjunto variável de processos que são colocados em execução concorrente. Em relação ao sistema de medida, optou-se por um contador de 32 bits, baseado no circuito integrado 8253 da Intel, alimentado por um oscilador de 2 MHz, apresentando uma precisão de 500 *ms* e podendo medir até 35,8 minutos. A principal vantagem, em relação a outros sistemas, é ser prático e suficientemente preciso para os testes a desenvolver.

No capítulo seis são concretizadas as metodologias de análise e concepção, deduzidas experimentalmente a partir de dois modelos de escalonamento – o não preemptivo e o preemptivo. Começamos por caracterizar os dois modelos de escalonadores mais implementados em termos de núcleos: orientados por eventos e orientados por tempo. Para ambos os modelos de escalonadores são definidos e identificados os *'overheads'* que podem comprometer a escalonabilidade. Em relação ao caso não preemptivo, consideramos o escalonamento simples, correspondente à execução de uma tarefa na totalidade, uma após outra e o escalonamento *'round-robin'*, fazendo, em primeiro lugar, uma análise de escalonabilidade qualitativa e, depois, quantitativa dos modelos respectivos. Por fim, é feito o estudo do caso preemptivo, protagonizado pelo escalonamento *'Rate Monotonic'*. Desenvolvemos também uma análise de escalonabilidade qualitativa e em seguida quantitativa deste modelo.

No capítulo sete, o qual encerra esta dissertação, são apresentadas as conclusões do trabalho desenvolvido e hipóteses para futuras investigações.

Capítulo 2
Definição do problema

As aplicações de tempo-real diferem consideravelmente das aplicações tradicionais, no sentido em que as primeiras impõem restrições temporais rígidas ao comportamento do sistema de controlo. Tais restrições têm que ser cumpridas sob o risco de porem em causa a integridade da aplicação.

Os sistemas que suportam a execução de aplicações de tempo-real e asseguram, assim, que os requisitos temporais são cumpridos são designados, em termos genéricos, como sistemas de tempo-real. Tem-se, desta forma, a visão tradicional de que a exactidão de um sistema computacional é unicamente função do seu comportamento lógico e funcional, pelo que não se aplica a sistemas de tempo-real. Para estes a exactidão depende, também, das propriedades temporais do comportamento.

Como o desempenho dos computadores digitais continuam a aumentar e o preço, tamanho, peso e consumo energético continua a diminuir, tem-se verificado um aumento considerável na utilização de sistemas de tempo-real baseados em computadores numa grande variedade de campos. Domínios de aplicação como o militar, indústria, medicina e outros, constituem um amplo âmbito de possíveis implementações. Exemplos de sistemas de tempo-real actuais incluem o controlo de centrais nucleares, controlo de produção industrial, monitorização médica, sistemas de auxílio à pilotagem de aviões, sistemas de controlo e automatização de armamento, navegação e controlo espacial, sistemas de reconhecimento, etc. Muitos

destes sistemas complexos requerem um controlo bastante elaborado e facilidades computacionais para permitir o seu correcto funcionamento em contínuo. Estes sistemas utilizam muitas vezes *'hardware'* dedicado como controladores.

Quando um computador é usado num sistema maior para permitir funções de controlo e computação, é normalmente referido como um sistema embebido. Actualmente encontramos sistemas embebidos em praticamente todos os aspectos da nossa vida, com computadores a serem introduzidos em novos sistemas a um ritmo cada vez maior. O sistema embebido tem que gerir e controlar o resto do sistema, obtendo dados através de sensores e exercendo acções de comando sobre actuadores mecânicos, electromecânicos e electrónicos. A principal característica dos sistemas embebidos é que estão dedicados a um determinado número de funções invariantes no tempo e de acção sobre o resto do sistema. Isto é, as necessidades de processamento não mudam e manipulam uma carga computacional (*'workload'*[4]) fixa e bem definida. Por exemplo, o sistema embebido usado nos automóveis para controlar a injecção de combustível e a ignição executa um programa fixo que pode ajustar os parâmetros à medida que os sensores fornecem novas informações sobre as condições do ambiente [LA90].

Como se pode subentender, os sistemas de tempo-real abrangem um âmbito extremamente grande, podendo ser materializados a partir de simples microcontroladores até sistemas complexos, distribuídos e altamente sofisticados. Estes sistemas complexos poder-se-ão encontrar, num futuro próximo, em aplicações tão diversas quanto: estações espaciais, integração de sistemas de visão/robóticos/inteligência artificial, coordenação de

---

[4] *'Workload'* é o termo original para designar a carga computacional. Por facilidade e porque não existe um termo constituído por uma única palavra em portugês, utilizaremos o termo original.

humanos/robots para atingir objectivos comuns (normalmente em ambientes perigosos como fábricas químicas ou exploração subaquática) e várias aplicações de comando e controlo [SR90].

Até aos dias de hoje, a concepção e análise de sistemas de tempo-real têm sido efectuadas, muitas vezes, seguindo técnicas *'ad hoc'* vulgarmente apoiadas em simulações. Este tipo de procedimento levanta alguns problemas:

- As simulações contemplam um determinado número de cenários, tidos como representativos, mas não é possível ter garantia que são contempladas todas as situações práticas possíveis. Além do mais, quanto mais perto do real a simulação está, mais complexa tem de ser e mais custos acarreta.

- O ambiente de interacção é normalmente dinâmico o que implica capacidade de adaptação dos sistemas de tempo-real, que actualmente não possuem.

- Quaisquer alterações efectuadas nos parâmetros do sistema resultam num outro conjunto de testes dispendiosos – é necessário alterar ou mesmo voltar a desenvolver os simuladores.

Todos estes factores levantam problemas em termos da previsibilidade. Assim, é necessário caminhar no sentido de definir uma base cientifica, isto é, de definir métodos, metodologias ou regras que permitam avaliar se um STR cumpre ou não os requisitos temporais a que está sujeito.

De qualquer forma, para se poder chegar ao objectivo definido, é necessário percorrer um longo caminho, saber, concretamente, quais as dificuldades com que os STR e a sua concepção se deparam, quais as envolventes dos projectos de sistemas de tempo-real em termos teóricos e práticos, quais as garantias efectivas de desempenho para um determinado projecto e como poderão ser testadas. Só, então, é possível deduzir modelos qualitativos e quantitativos que permitam avaliar o cumprimento das

restrições temporais a que um sistema de tempo-real está sujeito. Assim, o resto deste capítulo pretende clarificar os pontos focados atrás, não deixando no entanto de começar pela apresentação de alguns conceitos e definições no âmbito dos sistemas de tempo-real.

2.1 Definições e conceitos

Como é sabido, não existe uma definição concreta para sistemas de tempo-real. A maior parte da literatura, afecta aos sistemas TR, apresenta definições que se enquadram melhor nos âmbitos específicos desses textos. Aquela que parece ser mais consensual, devido à característica de controlo de muitos sistemas, tem a ver com a definição de que um **sistema de tempo-real** tem capacidade para prestar um ou mais serviços de tempo-real.

Um **serviço de tempo-real** é um serviço que, de acordo com a sua especificação, deve ser entregue dentro de um determinado intervalo de tempo. Nesta perspectiva, é legítimo argumentar, com toda a generalidade, que o conceito de serviço de tempo-real corresponde à inclusão da dimensão temporal no espaço da sua especificação [Mag95]. Assim, dadas as especificações temporais que, por definição, recaem sobre os serviços de tempo-real, qualquer serviço deste tipo tem o seu tempo de entrega sujeito a uma *'deadline'*. A ***'deadline'*** de um serviço de tempo-real é um marco que denota o tempo limite para a sua entrega. A *'deadline'* de um serviço de controlo é sempre quantificada a partir das características do objecto controlado e das especificações temporais inerentes à função desempenhada sobre o último. Em termos genéricos, entende-se **objecto controlado** como o meio ambiente do STR, com o qual este interactua.

Os serviços prestados por um sistema de tempo-real são devidos à execução de um conjunto de actividades computacionais concorrentes. Tais actividades computacionais são habitualmente designadas por **tarefas**.

Estas quando estão sujeitas a especificações temporais, assumem a designação de **tarefas de tempo-real**. Assim, uma tarefa de tempo-real está sujeito a uma *'deadline'*.

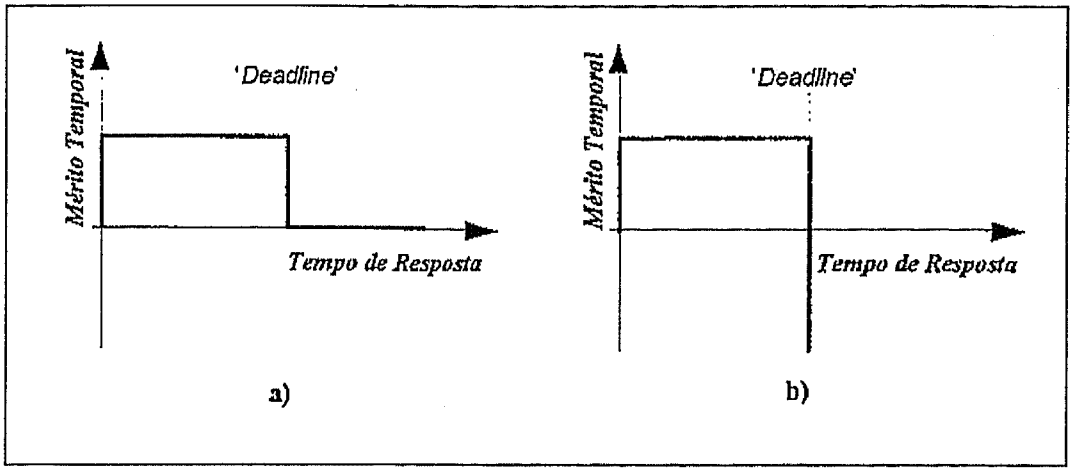

Fig. 2 – tipos de *'deadlines'* a) *'deadlines* não críticas, b) *'deadlines'* críticas [Mag95].

No âmbito dos sistemas de tempo-real, distingue-se sistemas de tempo-real críticos (*'hard real-time systems'*) e sistemas de tempo-real não críticos (*'soft real-time systems'*). Recorrendo à definição universalmente aceite e que resulta da necessidade de distinguir as consequências que advêm do incumprimento das restrições temporais, definimos um **sistema de tempo-real crítico** como sendo um sistema de tempo-real em que o desrespeito por pelo menos uma das suas especificações temporais pode ter consequências extremamente gravosas para o ambiente. Por sua vez, um **sistema de tempo-real não crítico** é um sistema de tempo-real em que qualquer serviço entregue fora de tempo pode ser inútil ou ter efeito diminuído, mas não acarreta consequências catastróficas.

Assim, uma tarefa de tempo-real pode estar sujeita a uma ***'deadline'* não-crítica** (*'soft deadline'*) ou a uma ***'deadline'* crítica** (*'hard deadline'*), conforme o seu cumprimento tem, respectivamente, consequências benignas ou catastróficas. Por sua vez, uma **tarefa de tempo-real** é considerada **crítica** ou **não-crítica**, conforme a *'deadline'* que lhe está associada ser, respectivamente, crítica ou não crítica.

2.2 Dificuldades na concepção de STR

Como o leque de aplicação dos sistemas de tempo-real é muito grande, existe um determinado número de subtilezas relacionadas com o projecto de sistemas de tempo-real que devem ser bem conhecidas e analisadas. Assim, Stankovic e Ramamritham [SR90] fizeram uma caracterização das principais dimensões que podem pôr em causa a previsibilidade dos sistemas de tempo-real. As várias dimensões a caracterizar na próxima secção são: granularidade das *'deadlines'* e lassidade das tarefas; o rigor das *'deadlines'*; a confiabilidade de STR; a complexidade do sistema e o grau de coordenação entre componentes; e as características do ambiente em que o sistema interactua.

Estas características, por seu turno, influenciam claramente quão estático ou dinâmico o sistema poderá vir a ser. Assim, dependendo das considerações tomadas para as várias dimensões mencionadas, muitas opções para o desenvolvimento de sistemas de tempo-real podem ser tomadas. No entanto, o denominador comum para o desenvolvimento de sistemas de tempo-real é a previsibilidade. Isto é, deve ser possível demonstrar que um determinado sistema cumpre os requisitos temporais, para além dos lógicos e funcionais, dentro de um conjunto de pressupostos: um modelo de carga em modelo de falhas. Se, eventualmente, a garantia de previsibilidade não puder ser a 100%, deveremos, pelo menos, saber até que ponto o sistema é previsível. Para isso é necessário, por um lado, conhecer bem as características do sistema e, por outro, conhecer o ambiente de interacção o melhor possível.

2.2.1 Caracterização formal de sistemas de tempo-real

Existem determinados pressupostos e requisitos que os sistemas de tempo-real devem cumprir, mas todos eles estão dependentes de certas subtilezas que podem verificar-se nas cinco dimensões seguintes.

A primeira questão coloca-se relativamente à granularidade das *'deadlines'* face à lassidade das tarefas. A **granularidade das *'deadlines'*** é o tempo entre quando uma tarefa requer execução até ao momento em que a deve completar. Por seu turno, a **lassidade da tarefa** é a *'deadline'*, menos o tempo de execução da tarefa. Se a granularidade da tarefa é pequena, estamos perante uma *'deadline'* "apertada". Isto implica que o tempo de reacção do suporte prático seja curto e, por conseguinte, o algoritmo de escalonamento a executar deva ser rápido e simples. No entanto, pode acontecer que a *'deadline'* seja apertada mesmo quando a granularidade é grande, se o tempo de execução da tarefa necessário for também grande. Isto é, a lassidade é pequena. Ora, isto são restrições predominantes e, consequentemente, na concepção de STR opta-se por desenvolver técnicas simples e rápidas para colmatar as necessidades destes sistemas.

A segunda dimensão diz respeito ao **rigor das *'deadlines'***. Esta refere-se ao valor de executar uma tarefa depois da *'deadline'*. Se estamos perante uma tarefa crítica, o valor da *'deadline'* poderá ser no mínimo nulo e no máximo catastrófico. Se, por outro lado, estivermos perante uma tarefa não-crítica, o valor da *'deadline'* é diminuído, o que constitui uma degradação graciosa. Daí que, eventualmente, mesmo depois de perdida uma *'deadline'* de uma tarefa não-crítica, ser possível ao sistema continuar em execução e cumprir as funções para as quais foi desenvolvido. Esta dimensão chama a atenção para o eventual relaxamento da dicotomia do valor da execução de uma tarefa após perder a *'deadline'*. Isto é, uma tarefa poderá ter algum valor, mesmo quando perde uma *'deadline'*.

A terceira dimensão tem a ver com a **confiabilidade** no desempenho de um sistema, relativamente ao cumprimento dos requisitos impostos pelo meio ambiente. Considerando as tarefas críticas, estas podem ser de dois tipos: aquelas em que a perda de uma *'deadline'* tem consequências catastróficas (*'critical tasks'*) e as segundas, cuja perda de uma *'deadline'* não tem qualquer resultado (*'hard real-time tasks'*). O problema aqui reside no facto de, em muitas concepções, serem tratadas da mesma forma, dando lugar a sistemas sobredimensionados e inflexíveis [SR90]. Ora, só o primeiro tipo necessita de ser tratado com crítica, isto é, é garantido que as *'deadlines'* sejam cumpridas através de uma análise *'off-line'* e esquemas que reservam os recursos necessários, mesmo sujeitas aos pressupostos considerados nos modelos de cargas e falhas.

A quarta dimensão corresponde à **complexidade do sistema** e ao **grau de coordenação**. Os sistemas de tempo-real variam consideravelmente em tamanho e complexidade. Na maioria dos sistemas actuais é comum que os sistemas sejam completamente carregados em memória ou em fases bem definidas antes da execução dessa fase. Por outro lado, acontece que muitos dos subsistemas das aplicações (tarefas, por exemplo) sejam completamente independentes uns dos outros, ou então tenham uma cooperação limitada entre eles. Estes factores significam muitos factores da concepção e análise de sistemas de tempo-real. O problema surge porque cada vez mais os sistemas são maiores e mais complexos, o que compromete significativamente a noção de previsibilidade.

A última dimensão tem a ver com o **meio ambiente** no qual os sistemas de tempo-real têm que operar. Este tem um papel importante e decisivo no desenvolvimento dos sistemas de tempo-real. Muitos dos ambientes podem ser bem definidos – como, por exemplo, uma experiência de laboratório, um motor de um automóvel ou uma linha de montagem – ou, então, serem parcialmente desconhecidos e dinâmicos. Os primeiros são considerados determinísticos e, mesmo que o não sejam intrinsecamente, são forçados a ser. Estes dão, normalmente, origem a sistemas de tempo-real pequenos e

estáticos, onde todas as *'deadlines'* podem ser garantidas à prior. É, assim, possível fazer uma análise quantitativa *'off-line'* das restrições temporais, uma vez que sabemos exactamente o que queremos. Os últimos são considerados não determinísticos e darão origem a sistemas de tempo-real grandes, complexos, distribuídos, adaptativos, com vários tipos de restrições temporais e que evoluirão ao longo do tempo. É difícil forçar ambientes como estes a ser determinísticos e, mesmo que seja possível, os sistemas seriam inflexíveis e incapazes de reagir a eventos inesperados. O grande problema é que as abordagens e garantias conseguidas em sistemas relativamente pequenos e estáticos (a operar em ambientes determinísticos) não são possíveis de generalizar a ambientes maiores, mais complexos e menos controláveis. São necessários avanços significativos para alcançar a previsibilidade destes sistemas em termos científicos.

2.2.2 Previsibilidade em sistemas de tempo-real

Um sistema de tempo-real é previsível é capaz de exibir um comportamento conforme foi pensado no seu desenvolvimento. Em tempo-real, um aspecto muito importante é o cumprimento dos requisitos temporais impostos pelo meio ambiente. Consequentemente, a previsibilidade de um sistema depende da garantia de que os requisitos temporais são cumpridos em todas as circunstâncias, mesmo na presença de falhas e sobrecargas.

A previsibilidade é muito difícil de garantir e provar, sobretudo, em sistemas complexos onde todas as questões mencionadas no ponto anterior estão presentes em simultâneo e existe interacção com meios não determinísticos. É dito em [SR90] que o caminho para garantir a previsibilidade passa pela demonstração que os requisitos são cumpridos a dois níveis de detalhe e para cada classe de tarefas. Entenda-se como classe de tarefas, as tarefas individuais ou as que estão em grupos perfeitamente definidos.

Assim, a previsibilidade deverá ser demonstrada:

- Ao nível macroscópico - todo o sistema -, onde deve ser provado em primeiro lugar que todas as tarefas críticas cumprirão sempre as suas *'deadlines'* e que todas as tarefas não críticas cumprirão a maioria dos requisitos.

- Ao nível microscópico - numa base de tarefa individual ou grupo de tarefas -, onde é necessário garantir a máxima previsibilidade possível.

Hoje em dia, já é viável garantir a previsibilidade de alguns sistemas, considerando determinados pressupostos.

Assim, a previsibilidade é facilmente garantida e demonstrável, num sistema simples ou constituído unicamente por tarefas periódicas, se considerarmos todos os pressupostos para um conhecimento perfeito dos Tempos de Execução no Pior Caso (TEPC) de todas as tarefas. Isto significa que: os tempos de execução das instruções no *'hardware'* devem ser conhecidos, os tempos de cada uma das primitivas do sistema operativo devem ser majorados, qualquer restrição de precedência entre tarefas e conflitos sobre recursos partilhados devem ser considerados e a memória virtual e *'caches'* deve ser eliminada. É necessário salientar que os sistemas resultantes são completamente estáticos e inflexíveis.

Como consequência da previsibilidade, aparece a previsão do desempenho de sistemas de tempo-real. Assim, a previsão do desempenho de tarefas constitui a base de praticamente todas as abordagens utilizadas na concepção e implementação de sistemas de tempo-real. Estas previsões podem ser divididas em duas classes: determinísticas e estocásticas. As previsões determinísticas são aquelas que estão *"sempre"* correctas. Isto é, são previsões baseadas no tempos de execução no pior caso. Assim, estas previsões *"nunca"* são violadas, sujeitas aos pressupostos considerados. Por seu turno, as previsões estocásticas, contrastam com as anteriores, visto que descrevem o comportamento em termos probalísticos. Estas podem ser expressas: como um único tempo médio de execução, como resultado de uma distribuição de tempos, ou ainda como uma combinação

específica com o tempo de execução no pior caso. Desta forma, e geralmente, as previsões determinísticas são a base para o desenvolvimento de sistemas de tempo-real críticos, enquanto que as previsões estocásticas são a base para os sistemas não críticos.

2.3 Problemática da simultaneidade de operações

Existem dois aspectos que necessitam ser conjugados de forma a que o processo de desenvolvimento de sistemas de tempo-real deixe de ser *'ad hoc'* (na base da tentativa e erro) e passe a ser um processo sistemático. Os dois aspectos são o teórico, que engloba os desenvolvimentos conseguidos com a investigação na área, e os aspectos práticos, que engloba a tecnologia de suporte à implementação dos aspectos científicos para aplicações concretas.

Este ponto pretende realçar os principais problemas relacionados com a simultaneidade de operações em ambos os aspectos referidos.

A simultaneidade de operações é a principal característica de qualquer sistema de tempo-real, uma vez que estes, normalmente, na sua interacção com o meio ambiente, recebem dados de diversos sensores e têm, eventualmente, que agir sobre vários actuadores. Acontece que esta situação traduz-se, regra geral, num estado de concorrência entre as várias tarefas do sistema, uma vez que é fácil chegar à situação de escassez de recursos. Por exemplo, a maioria dos sistemas de tempo-real actuais são baseados em sistemas monoprocessadores e, por isso, as várias tarefas que constituem o sistema concorrem pela obtenção de tempo de processador – um recurso escasso.

2.3.1 Aspectos teóricos

A maioria da investigação, em termos de teoria de sistemas de tempo-real, prende-se com a impossibilidade de executar em simultâneo todas as tarefas e utilizar o mínimo de recursos possível ou aproveitar ao máximo os recursos disponíveis.

Assim, o estudo nesta área recai sobre a definição de técnicas de atribuição de tarefas a recursos, também designado por sequenciamento de operações, de forma a utilizar os últimos da maneira mais eficiente. O conjunto de operações para levar a cabo a função descritiva é, normalmente, designado como **algoritmo de escalonamento**.

Como estamos a falar de sistemas de tempo-real, os algoritmos de escalonamento deverão efectuar o sequenciamento de operações, sempre com a garantia do cumprimento de todas as *'deadlines'* do sistema.

A disciplina que estuda esta questão é designada por teoria do escalonamento e, actualmente, conta com bastante investigação já feita.

É necessário ter em atenção que quando falamos de sistemas de tempo-real críticos, a maioria das instâncias do problema de escalonamento são computacionalmente intratáveis (problemas NP – completos). Assim, por exemplo, o escalonamento de processos com restrições de recursos é geralmente um problema *'NP-hard'*[5] e a presença de restrições de precedência e exclusão exacerba o problema ainda mais [RS94]. É preciso referir que existem determinadas soluções pontuais. A solução encontrada pela maioria dos autores é a utilização de heurísticas e algoritmos de aproximação.

---

[5] Termo original para designar problemas que não têm uma solução polinomial, isto é, são intratáveis.

A literatura actual sobre teoria do escalonamento é tão vasta que é impossível abordar todo o trabalho desenvolvido. Duas das comunidades com grande responsabilidade na área dos sistemas de tempo-real são a Investigação Operacional e a Ciência dos Computadores, tendo cada uma delas uma percepção particular do problema do escalonamento e, consequentemente, um conjunto de pressupostos para a condução das decisões de escalonamento diferentes. A primeira valoriza a minimização de recursos (minimizar a soma simples ou ponderada dos tempos de execução, minimizar o comprimento do escalonamento, minimizar o número de processadores necessários ou minimizar o máximo atraso), não dando especial atenção ao cumprimento das *'deadlines'*. A segunda valoriza mais o tempo de execução dos processos, mas em termos médios. Os desenvolvimentos feitos por ambas as comunidades é o que actualmente designamos por teoria clássica do escalonamento [SSN+95]. No entanto, um sistema de tempo-real e, em particular, o escalonamento de tempo-real, têm como principal objectivo cumprir a pontualidade (*'timeliness'*) do sistema, o que introduz um conjunto de métricas diferentes que dependem da área de aplicação.

2.3.2 Aspectos práticos

Os modelos teóricos, discutidos na secção anterior, não permitem, por si só, o desenvolvimento de sistemas de tempo-real. Normalmente, nos desenvolvimentos teóricos são negligenciados certos problemas – quase sempre para simplificação – que, em termos práticos, afectam consideravelmente o desempenho, pelo que as diferenças entre os aspectos teóricos e a sua implementação são bastante acentuadas. Assim sendo, os suportes práticos introduzem, geralmente, várias distorções aos modelos teóricos, mesmo em termos qualitativos.

A previsibilidade de um sistema de tempo-real depende directamente do *'hardware'*, sistema operativo e algoritmo de escalonamento que lhe

servem de base. Assim, ocorre que em termos práticos os sistemas de tempo-real continuem a não cumprir as restrições temporais (*'deadlines'*) a que estão sujeitos, mesmo que as garantias teóricas dos algoritmos de escalonamento sejam respeitadas.

Encarando as questões levantadas numa perspectiva optimista, os sistemas de tempo-real deveriam ser capazes de integrar o escalonamento da CPU e a atribuição de recursos, de forma que conjuntos de tarefas cooperativas pudessem obter os recursos que necessitam em tempo oportuno, para cumprirem as restrições temporais. Para isso, seria necessário existir suporte (*'hardware'* e sistema operativo) e meios de desenvolvimento (linguagem de programação) adequados aos sistemas de tempo-real, isto é, pensados de forma a que as restrições temporais fossem especificadas facilmente e, posteriormente, em fase de exploração, o suporte fornecesse os serviços de modo adequado.

Este, infelizmente, não é o caso, pelo que só nos resta obter um conhecimento profundo dos suportes práticos, de maneira a ser possível conduzir um projecto de uma aplicação de tempo-real para atingir os objectivos traçados inicialmente.

2.4 Enquadramento formal da dissertação

Esta dissertação pretende contribuir para uma melhor compreensão das vissitudes da concepção e análise de sistemas de tempo-real, de forma que sejam desvendados alguns dos mistérios que envolvem o processo respectivo, e que este decorra de modo cada vez mais sistemático.

Com efeito, é necessário, por um lado, encontrar maneiras de testar o desempenho dos sistemas de tempo-real e, por conseguinte, procurar métricas que atribuam figuras de mérito a suportes de sistemas de tempo-real. Estes suportes são constituídos pela conjunção de elementos concretos

de *'hardware'*, sistema operativo. A cada par é, então, atribuída uma figura de mérito que o caracteriza perante outras combinações. Isto permite-nos optar pela melhor solução em termos de suporte para o Sistema de tempo-real a desenvolver. Como se pode constatar, as várias *'benchmarks'* que existem para os testes descritos serão utilizadas na fase de implementação do projecto, não havendo qualquer conhecimento do comportamento do sistema em desenvolvimento, devido ao algoritmo de escalonamento a utilizar. É necessário também aqui, encontrar maneiras de garantir que o escalonamento do sistema é possível, tendo em atenção o respectivo algoritmo. Para a fase de concepção do sistema, existe pelo menos um método baseado no algoritmo *'Rate Monotonic'* que permite testar a escalonabilidade com base nas suas características temporais.

Este método, designado por RMA (*'Rate Monotonic Analysis'*), é independente da implementação. Para a fase de implementação, será necessário, aqui também, encontrar *'benchmarks'* que nos permitam atribuir uma figura de mérito, agora ao trio constituído pelo: *'hardware'*, sistema operativo e algoritmo de escalonamento.

Acontece, no entanto, que em termos teóricos é possível testar a escalonabilidade de um conjunto de tarefas, tendo em atenção a política de escalonamento seguida. É, assim, possível saber se um determinado conjunto de tarefas, com características específicas, cumpre ou não as restrições temporais (*'deadlines'*) a que está sujeito, independentemente do sistema prático de suporte. Trata-se de um conjunto de axiomas (o RMA já mencionado) que ditam a escalonabilidade de um determinado sistema, baseados, essencialmente, na carga do processador. No entanto, como já foi referido oportunamente, o suporte prático introduz determinadas restrições, resultantes dos serviços que este tem que efectuar para dar cumprimento aos algoritmos implementados pelas aplicações. Estas são designadas como o *'overhead'* do suporte prático. Este *'overhead'* é normalmente contabilizado como a soma dos vários tempos necessários à prestação dos

serviços, como por exemplo: tempo de mudança de contexto, tempo de preempção, tempo para o tratamento de interrupções, etc.

Daqui resulta que embora, em termos teóricos o sistema seja escalonável, em termos práticos pode deixar de o ser, devido aos vários *'overheads'* introduzidos pelo sistema operativo e o *'hardware'*.

Neste cenário surge a necessidade do desenvolvimento de metodologias que, entrando em linha de conta com as várias restrições temporais introduzidas pelo suporte prático, permitam construir sistemas de tempo-real previsíveis, tendo em atenção as características temporais do mesmo.

A forma mais indicada para proceder à definição dos modelos de escalonamento em questão, pensamos que passa pela dedução experimental. Isto porque, para além de permitir a definição da metodologia pela análise do algoritmo de escalonamento específico, permite-nos também testar o modelo com base em experiências concretas.

2.5 Síntese

Neste capítulo abordámos os vários problemas com que nos deparámos na Concepção e Análise de sistemas de tempo-real.

Começámos por abordar as dificuldades encontradas na concepção de sistemas de tempo-real. Concretamente fizemos uma caracterização formal de sistemas de tempo-real em termos das cinco principais dimensões que podem condicionar o desenvolvimento de sistemas de tempo-real e, consecutivamente, a previsibilidade dos mesmos. As dimensões focadas foram: granularidade das *'deadlines' versus* a lassidade das tarefas, o rigor das *'deadlines'*, a confiabilidade do sistema, a complexidade do sistema *versus* a correlação entre tarefas e o meio ambiente. Para cada uma delas foram apontados os problemas que podem pôr em questão a previsibilidade.

Depois, fizemos várias considerações sobre a previsibilidade de sistemas de tempo-real, as formas de garantir a previsibilidade e abordámos a problemática da previsão do desempenho.

No ponto seguinte, tratámos da problemática da simultaneidade de operações. Concretamente, definimos os problemas que esta levanta, no que diz respeito aos aspectos teóricos e práticos dos sistemas de tempo-real. Aqui, nos aspectos teóricos, abordámos os problemas do escalonamento de tarefas e os problemas com os algoritmos de escalonamento. Nos aspectos práticos, abordámos os problemas com a implementação prática de sistemas de tempo-real e respectiva implementação dos algoritmos de escalonamentos. Assim, foram identificados os principais problemas introduzidos pelo suporte prático.

Por fim, fizemos o enquadramento formal desta dissertação, de onde ressalta, por um lado, a necessidade de procurar *'benchmarks'* que permitam analisar sistemas de tempo-real criados com base em suportes práticos específicos de *'hardware'*, sistema operativo e algoritmo de escalonamento. Por outro lado, ressalta a necessidade de desenvolvimento de metodologias de análise e concepção de sistemas de tempo-real, que entram em linha de conta com as restrições temporais introduzidas pelo suporte prático.

Todas as questões levantadas neste capítulo serão tratadas nos restantes capítulos desta dissertação.

Neste momento e para a prossecução do trabalho, pretendemos saber a resposta às perguntas que cada um dos aspectos descritos – teórico e prático – levantam.

Concretamente, os aspectos teóricos pretendem responder essencialmente a uma pergunta: como efectuar uma atribuição de recursos óptima, garantindo a previsibilidade do sistema e o respectivo cumprimento das restrições temporais?

Por seu turno, os aspectos práticos pretendem responder à seguinte pergunta: quais os suportes a utilizar para a implementação de sistemas de tempo-real?

Estas perguntas serão respondidas no próximo capítulo.

Capítulo 3
Projecto de sistemas de tempo-real

Neste capítulo são caracterizados os diversos suportes para os projectos de sistemas de tempo-real. Começamos por fazer um levantamento sobre os vários algoritmos de escalonamento de tempo-real.

Em seguida, faremos uma caracterização dos Sistemas Operativos de tempo-real.

Por fim, faremos uma breve caracterização de mais dois suportes de menor interesse para esta dissertação. Estes são as linguagens de programação de tempo-real e o *'hardware'*.

Debruçar-nos-emos, assim, sobretudo sobre os dois primeiros suportes (Algoritmos de Escalonamento e Sistemas Operativos de Tempo-real), por um lado porque são essenciais para o projecto de sistemas de tempo-real e o seu conhecimento é indispensável para a prossecução desta dissertação. Por outro lado, os dois últimos perdem relevância face aos primeiros por duas razões: as linguagens ou são dependentes dos sistemas operativos e/ou algoritmos de escalonamento, ou apresentam soluções próprias (algoritmos de escalonamento proprietários ou executivos, que praticamente substituem o SO). Em relação ao *'hardware'*, este influencia directamente o sistema operativo.

3.1 Algoritmos de escalonamento

Existem inúmeros recursos que, por serem escassos, por razões económicas ou estratégicas, têm obrigatoriamente que ser partilhados por diversas entidades. Isto gera uma condição de concorrência entre as entidades em causa pela obtenção destes recursos escassos. Assim, é necessário encontrar meios que permitam disciplinar de forma eficaz a disputa pelos recursos escassos. Foi para resolver esta situação de concorrência que surgiram os algoritmos de escalonamento.

Um **algoritmo de escalonamento** é um conjunto de regras que, tendo em conta a concorrência verificada a cada instante, define o modo de partilha de um recurso ou conjunto de recursos escassos, pelas diversas entidades que deles pretendem usufruir, por forma a que um determinado intento venha a ser alcançado. Por exemplo, quando um processador está confinado a executar duas ou mais tarefas que, idealmente, se deveriam desenrolar em simultâneo, este processador torna-se um recurso escasso. O nosso interesse, no âmbito desta dissertação, resume-se ao escalonamento de tarefas de tempo-real, pelo que estes algoritmos são designados por **algoritmos de escalonamento de tempo-real**.

Desta feita, para que um determinado algoritmo de escalonamento seja válido, é necessário que ele garanta, sujeito a determinados pressupostos, que todas as tarefas de tempo-real críticas cumprem as suas *'deadlines'*.

De uma forma geral, um algoritmo de escalonamento de tempo-real deve ser absolutamente previsível, eficiente, suportar tarefas periódicas e aperiódicas e eventuais sincronizações das mesmas, e ser utilizável tanto em controladores centralizados como distribuídos.

Desta forma, em sistemas de tempo-real, as tarefas podem ser, por um lado periódicas e aperiódicas; e por outro independentes e dependentes.

Tarefas periódicas são aquelas que pedem execução numa base regular no tempo (execução cíclica ao longo do tempo). Tarefas aperiódicas solicitam execução a tempos impossíveis de prever e normalmente por eventos externos, pelo que não podem ser associadas a *'deadlines'* críticas. Dentro das tarefas aperiódicas aparece uma classe especial de tarefas designadas por tarefas esporádicas, às quais são normalmente associadas *'deadlines'* críticas, desde que seja possível garantir um período mínimo entre duas solicitações de activação consecutivas. Estas são normalmente associadas a sinais de alarme. Por seu turno, tarefas independentes são aquelas que executam sem qualquer tipo de relação entre elas e, por isso, não têm qualquer condicionalismo de sincronização. Tarefas dependentes possuem relações de precedência ou exclusão entre elas, havendo lugar a condicionalismos de sincronização.

Cada um destes tipos de tarefas apresentam características particulares. Assim:

- as tarefas periódicas são caracterizadas pelo tempo de execução, pela *'deadline'* e pelo período;
- as tarefas aperiódicas são caracterizadas pelo tempo de execução e pela *'deadline'*;
- as tarefas esporádicas, como caso particular das tarefas aperiódicas, para além de execução e da *'deadline'*, são caracterizadas pelo tempo mínimo entre dois pedidos de execução consecutivos.

3.1.1 Classificações dos algoritmos de escalonamento

Os algoritmos de escalonamento, em termos gerais, podem ser: estáticos ou dinâmicos.

Nos algoritmos estáticos, o escalonamento das várias tarefas é feito *a priori*, durante a fase de projecto. Para isso, é necessário um conhecimento

prévio do meio com o qual o sistema deve interactuar, nomeadamente, todas as tarefas que podem solicitar execução num dado momento, bem como as suas características: tempos de pedido de activação, tempo de execução, *'deadline'*, período, restrições de precedência e exclusão. Este tipo de escalonamento é também designado como escalonamento *'off-line'* ou *'pre-run-time'*. O resultado desta abordagem é um plano bem definido – *tabela de escalonamento* - que se mantém fixo durante todo o tempo de execução e é seguido na integra pelo *despachante* (*'dispatcher'*) que, em função da informação da tabela de escalonamento e da progressão do tempo, materializa o escalonamento.

Um escalonamento de tempo-real é dinâmico, se é feito aquando da execução do sistema, isto é, em *'run-time'*, pelo que o algoritmo só tem conhecimento das tarefas que estão a solicitar execução a cada momento. Um algoritmo de escalonamento dinâmico atribui uma **prioridade** a cada tarefa e concede o processador àquela que, dentro do conjunto de tarefas a solicitar execução a cada instante, possua a prioridade mais elevada. Desta maneira, estamos perante um escalonamento por prioridades. Os algoritmos de escalonamento dinâmicos dividem-se em duas categorias: os de **prioridades fixas** e os totalmente **dinâmicos** (conferir Fig. 3). Os primeiros definem uma única vez a prioridade de cada tarefa e permanece fixa durante o tempo de execução da aplicação. Os segundos redefinem sistematicamente a prioridade de cada tarefa, durante todo o tempo de execução.

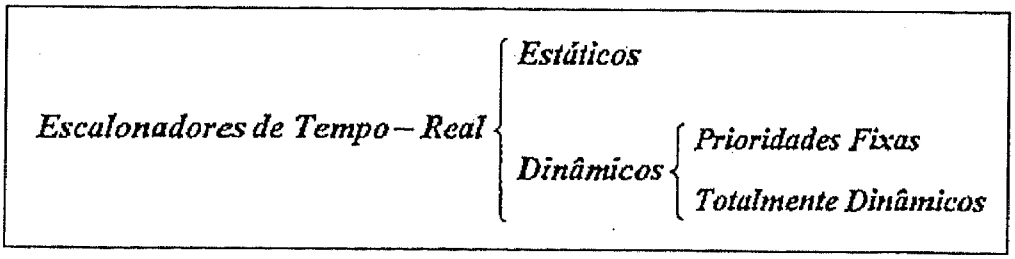

Fig. 3 – Classificação dos Escalonadores de tempo-real [Mag95].

No caso dos escalonadores de prioridades fixas, a prioridade de cada tarefa é sempre decidida durante a fase do projecto do sistema, baseada

normalmente no equacionamento das suas características temporais impostas pelo meio. Por seu turno, quando o escalonamento é totalmente dinâmico, a prioridade exibida por cada tarefa em cada momento é em função das suas características e da concorrência verificada nesse instante. Compete, assim, ao algoritmo totalmente dinâmico equacionar sistematicamente estas variáveis e decidir qual a prioridade a atribuir a cada tarefa, em cada instante.

O escalonamento estático é aquele que apresenta uma menor sobrecarga do sistema aquando da execução, uma vez que o plano de escalonamento já está definido e, por conseguinte o despachante só tem que o pôr em prática. Não há necessidade de algoritmos mais ou menos elaborados para tomar decisões de escalonamento. Desta forma, esta política de escalonamento é de todas aquela que apresenta a melhor previsibilidade, porque todos os testes são elaborados *'off-time'* e o plano de escalonamento é invariante no tempo. Torna-se a principal desvantagem deste algoritmo, uma vez que é estático e não há qualquer hipótese dos sistema responder a estímulos que não tenham sido previstos na fase de projecto. É de facto, um algoritmo inflexível, mas que continua a não deixar de ter adeptos devido à sua previsibilidade [Mag95]. É a melhor solução para sistemas de tempo-real críticos.

Os escalonamentos dinâmicos por propriedades fixas são aqueles que são cada vez, mais utilizados na implementação de projectos de tempo-real. Em relação à sobrecarga do sistema, embora não tão pequena como a do caso estático, esta é perfeitamente aceitável, uma vez que o algoritmo de escalonamento continua a não ter que decidir o escalonamento por técnicas apuradas e complexas. Isto é, este, em cada momento, unicamente permite execução à tarefa com maior prioridade a solicitar activação. Este tipo de política de escalonamento continua a garantir a total previsibilidade do sistema, já que as prioridades são atribuídas estaticamente na fase de projecto e é possível criar o plano de escalonamento, seja em que situação

for. Daí a nossa afirmação do aumento de popularidade deste tipo de algoritmo.

Estes tipos de algoritmos apresentam uma maior flexibilidade, mas continuam a não conseguir responder a estímulos que não tenham sido previstos na fase de projecto.

Curiosamente, os algoritmos totalmente dinâmicos aqueles que não apresentam grande representatividade em termos de aplicações práticas. Em primeiro lugar, a sobrecarga do sistema é a maior dos três tipos descritos. Isto é perfeitamente justificável, já que todas as decisões de escalonamento são tomadas em fase de execução do sistema, o que resulta num algoritmo mais complexo e consecutivamente mais pesado para o sistema, em termos temporais. Apesar do argumento da flexibilidade, que é perfeitamente compreensível, a previsibilidade deste tipo de algoritmos é pouca, pelo que, e em situações de sobrecarga transitória, o comportamento é perfeitamente imprevisível. Sem dúvida nenhuma, o principal argumento favorável destes algoritmos é a flexibilidade, podendo responder a estímulos que eventualmente não tenham sido previstos na fase de projecto do sistema.

3.1.2 Escalonamento de tarefas periódicas independentes

O caso mais fácil de escalonar corresponde a conjuntos de tarefas periódicas e independentes. Foi provado [LL73], [LSD89], [SG90] que os algoritmos óptimos para escalonar este tipo de tarefas são preemptáveis (uma tarefa menos prioritária pode ser interrompida para que outra mais prioritária possa executar) e com prioridades que podem ser atribuídas estática ou dinamicamente.

Os principais algoritmos de escalonamento neste âmbito são: para esquemas de escalonamento com atribuição de prioridades estática, o *'Rate Monotonic'* (RM); e para esquemas com atribuição dinâmica de prioridades,

o *'Earliest-Deadline-First'* (EDF) [LL73] e o *'Least-Laxity-First'* (LLF) [DM89].

No algoritmo *'Rate Monotonic'* [LL73], as prioridades são atribuídas inversamente proporcional ao período, isto é, quanto menor o período maior a prioridade. Estas mantêm-se fixas ao longo de toda a execução do algoritmo e a tarefa actual pode ser interrompida se alguma de prioridade superior solicitar execução. O *'overhead'* do sistema para a materialização deste algoritmo é pequeno e a escalonabilidade pode ser facilmente verificada na fase de projecto (*'off-line'*), a partir dos tempos de execução de cada tarefa no pior caso. Este algoritmo pode escalonar um conjunto de tarefas, com a garantia de que todas cumprem as suas *'deadlines'*, se o total de recursos utilizados (utilização do processador) for menor que um certo limite. Este limite é geralmente $U=n(2^{1/n}-1)$, para n tarefas. e aproxima-se de $ln(2)$ = 0,69 (69%), quando o número de tarefas é suficientemente grande (tende para infinito). Este limite é o caso mais desfavorável; no entanto, verifica-se que, na maioria das vezes, é substancialmente melhor. Quando os períodos são uniformemente distribuídos o limite aumenta para 88% [GMS94], [Bur91] e se estes são harmónicos do período mais pequeno, o limite atinge mesmo 100% [GMS94], [Bur91], [RS94]. A formulação RM assume que todas as tarefas têm a *'deadline'* igual ao período. Caso as *'deadlines'* serem menores que o período, então o algoritmo *'Deadline Monotonic'* (DM) é óptimo.

Existem pelo menos mais dois algoritmos considerados óptimos para o escalonamento de tarefas independentes, mas em que as prioridades são atribuídas dinamicamente. No EDF, a prioridade mais elevada é atribuída, em tempo de execução, à tarefa cuja *'deadline'* está mais próxima. No LLF, a prioridade mais elevada é para a tarefa cuja lassidade (tempo para a *'deadline'* menos o tempo de execução que falta cumprir) é menor. Foi provado [LL73] que a utilização de recursos para o EDF é de 100%. O LLF, como extensão do EDF, também o é. Estes algoritmos permitem uma maior utilização do processador mas aumentam consideravelmente o *'overhead'*

do sistema para a implementação em fase de execução. Como já foi referido e porque estes são algoritmos totalmente dinâmicos, a complexidade do algoritmo é maior e por conseguinte o tempo para a sua execução também é. O principal problema é a falta de previsibilidade, pelo que não há maneira de saber qual a tarefa que perde a *'deadline'* numa situação de sobrecarga transitiva do sistema.

3.1.3 Escalonamento de tarefas aperiódicas independentes

A maioria dos sistemas de tempo-real contemplam o escalonamento de tarefas periódicas e aperiódicas em simultâneo. Foi provado [SLD89] que o algoritmo de escalonamento EDF e as suas variantes, não obstante serem óptimos neste cenário, apresentam alguns problemas práticos, especialmente em termos de previsibilidade.

Tendo por base [PD93], existem algumas alternativas para o escalonamento de tarefas aperiódicas. Concretamente, a primeira política e mais simples consiste no escalonamento de tarefas aperiódicas unicamente quando não houver tarefas periódicas a solicitar execução. Outra alternativa passa pela modificação dos algoritmos de prioridades fixas (o RM, por exemplo), de forma a aceitar tarefas aperiódicas. Assim, podemos criar uma tarefa periódica caracterizada por possuir uma prioridade fixa, que é responsável por servir os pedidos das tarefas aperiódicas. Esta política é designada por sondagem (*'polling'*). Outras alternativas são a definição de políticas *'Priority Exchange'* (PE), *'Deferrable Server'* (DS) e *'sporadic Server'* (SS), as quais são também designadas como *'bandwidth preserving'*, uma vez que fornecem um mecanismo para preservar a largura de banda na atribuição de recursos a tarefas aperiódicas. Ambas as políticas minimizam o tempo de resposta a tarefas aperiódicas através da definição de uma tarefa (servidor) de prioridade alta que manipula os pedidos das tarefas aperiódicas.

3.1.4 Escalonamento de tarefas dependentes

Na maioria dos sistemas de tempo-real realistas, as várias tarefas interagem umas com as outras de forma a satisfazer os requisitos do meio ambiente. Estas interacções podem ser de vários tipos: sincronização de tarefas, restrições de precedência e exclusão mútua para proteger recursos partilháveis. Normalmente, o suporte de aplicações (linguagem de programação concorrente e sistema operativo) fornecem várias primitivas de sincronização, entre elas: semáforos, monitores, *'rendezvous'*, etc.

Assim, para calcular o tempo de execução de tarefas dependentes, é necessário o conhecimento do maior tempo de bloqueio imposto por qualquer uma das primitivas de sincronização a utilizar. É preciso notar que a maioria dos problemas de escalonamento para tarefas com requisitos temporais e qualquer dos tipos de interacções referidos, são normalmente problemas computacionalmente intratáveis (*'NP-Hard'*). Estes tipos de interacções levantam ainda mais problemas, se as tarefas forem preemptáveis.

A principal dificuldade com semáforos, monitores ou sistemas baseados em mensagens, é que tarefas com prioridades mais elevadas podem ser bloqueadas por tarefas com prioridades mais baixas, um número ilimitado de vezes em secções críticas (acesso a recursos partilhados), que são controladas por uma destas primitivas de sincronização. Este fenómeno é conhecido como inversão de prioridades e pode provocar a perda das *'deadlines'* das tarefas mais prioritárias.

Uma das formas mais usuais de evitar este fenómeno é através do protocolo de herança de prioridade [SRL90]. Este efectua a mudança dinâmica da prioridade da tarefa que está a provocar o bloqueio para a prioridade da tarefa que está bloqueada, de forma a que a primeira execute o mais rápido possível a secção crítica sem ser interrompida. No pior caso, cada secção crítica pode ser bloqueada por uma tarefa de baixa prioridade. No entanto,

este limite pode ser alto e inaceitável, e por outro lado, os impasses (*'deadlocks'*) não podem ser evitados.

Todas estas dificuldades são resolvidas pelo *'Priority ceiling protocol'* (PCP) [SRL90].

O benefício grande do PCP é que tarefas com prioridade elevada só podem ser bloqueadas uma vez (por activação) por tarefas de prioridade inferior. Assim, é perfeitamente possível testar a previsibilidade do escalonamento de tarefas dependentes, uma vez que é conhecido o máximo tempo de bloqueio de tarefas de prioridade elevada provocado por tarefas de menor prioridade.

3.1.5 Escalonamento em sistemas multiprocessador

O desenvolvimento de algoritmos de escalonamento apropriados para sistemas multiprocessador é problemático, essencialmente porque os algoritmos desenvolvidos para sistemas uniprocessador não são directamente aplicáveis. Desta forma, os algoritmos que são óptimos para sistemas com um só processador deixam de o ser para um número mais elevado de processadores. Como já foi referido, encontrar um escalonamento óptimo para sistemas multiprocessador é um problema *'NP-hard'*. Assim, é necessário procurar formas de simplificar o problema, através de algoritmos que forneçam resultados adequados, embora possam não ser os óptimos, isto é, através de heurísticas. Como foi demonstrado em [Bur91], em sistemas multiprocessador é melhor efectuar uma distribuição estatística e as tarefas periódicas, para que não aconteça a perda das *'deadlines'* devido a decisões incorrectas feitas por escalonamento dinâmico.

O escalonamento em sistemas multiprocessador pode assumir duas perspectivas, conforme se trate de sistemas distribuídos fortemente ligados

(sistemas de memória partilhada) ou, então, sistemas distribuídos fracamente ligados (sistemas multi-computador).

O escalonamento em sistemas distribuídos fortemente ligados ou de memória partilhada pode ser efectuado por um único escalonador responsável por atribuir tarefas a elementos de processamento. Por outro lado, em sistemas distribuídos fracamente ligados, devido ao elevado custo na migração de tarefas entre processadores, e consequentemente à falta de previsibilidade dos mesmos, as tarefas podem ser estatisticamente atribuídas aos processadores do sistema. Nestes sistemas, o escalonador é normalmente dividido em dois componentes separados: um afectador e um escalonador local. O afectador é responsável por atribuir tarefas aos processadores distribuídos e o escalonador local (um em cada processador) é implementado por um dos algoritmos de escalonamento uniprocessador, mencionados anteriormente. Normalmente, o afectador é implementado com base em heurísticas para reduzir o espaço de procura.

3.2 Sistemas operativos de tempo-real

Um sistema operativo (SO) consiste num conjunto de módulos de *'software'* e *'firmware'*, que funcionam como extensão do *'hardware'*, e permite controlar as operações de um sistema computacional, proporcionando um ambiente para a execução e desenvolvimento de aplicações. Trata-se de uma camada de *'software'* entre o *'hardware'* e os programas de aplicação. Um sistema operativo é suposto fornecer vários serviços. As quatro principais áreas funcionais que estes suportam são:

- gestão e sincronização de processos;
- gestão de memória;
- comunicação entre processos (IPC);

- entrada/saída (E/S).

A forma como estas áreas são suportadas pelos Sistemas Operativos de Tempo-real (SOTR) difere dos sistemas operativos convencionais, uma vez que os sistemas operativos de tempo-real dão maior importância à previsibilidade e incluem características para suportar restrições temporais. O problema é que a previsibilidade e pontualidade em SOTR não é fácil de implementar, quando a maioria do *'hardware'* é construído com base em técnicas de optimização que valorizam o desempenho no caso médio. Por outro lado, é extremamente difícil de implementar as várias técnicas de escalonamento totalmente dinâmico ao nível do SOTR. Consequentemente, os SOTRs, que implementam alguns ritmos de escalonamento de tempo-real, são geralmente baseados em escalonamento preemptivo com prioridades fixas, e por conseguinte não permitem escalonamento totalmente dinâmico.

Acontece, no entanto, que implementações de algoritmos de escalonamento no SO são bastante recentes. Pelo que, a maioria dos SO utilizados como suporte de sistemas de tempo-real oferecem um *'dispatcher'* que coloca os processos em execução, em função da prioridade do mesmo, que normalmente é dinâmica. Na maioria são sistemas operativos preemptivos por prioridades.

3.2.1 Principais características e factores para a diferenciação de SOTR

Existem certos requisitos que os SOTR devem cumprir. Entre eles destacam-se:

- fornecer suporte para a criação, eliminação e escalonamento de várias tarefas de tempo-real - multitarefa;

- garantir que tarefas de prioridade mais elevada, quando prontos a executar, preemptem tarefas de prioridades mais baixas para a execução das primeiras – escalonamento preemptivo;

- ser capaz de rapidamente reconhecer a ocorrência de um evento e tomar acções determinísticas com base nele – garantia de resposta a interrupções;

- ser capaz de suportar comunicação interprocesso (IPC) através de facilidades rápidas e de confiança, tais como semáforos, memória partilhada e passagem de mensagens;

- fornecer meios para uma fácil incorporação de *'drivers'* dedicados e suportar E/S assíncrono, isto é, um processo pode iniciar uma operação de E/S e continuar a execução das restantes tarefas em simultâneo – E/S em concorrência;

- permitir ao utilizador um controlo específico dos recursos do sistema incluindo: CPU, memória e E/S, para um melhor controlo de operações que comprometam a previsibilidade.

Todos os requisitos mencionados tornam um SOTR um suporte previsível para aplicações de tempo-real. No entanto, estes requisitos não estão todos presentes nos sistemas operativos actuais e, por conseguinte, condicionam a previsibilidade dos STRs.

Há, assim, um conjunto de factores importantes que podem conduzir a escolha de um SOTR específico, de acordo com as necessidades que o sistema de tempo-real deve satisfazer. Estes são os que a seguir se indicam.

A *Multitarefa* corresponde à característica que permite que várias operações sejam executadas em concorrência, cada uma com as suas restrições temporais. Esta por si só, quando estamos perante um sistema uniprocessador, é já um problema. Assim, existem vários métodos que permitem a implementação de multitarefa e que podem influenciar a

previsibilidade de um sistema. O primeiro método consiste em que cada um dos processos regularmente ceda o controlo para que outros processos possuam tempo de processador – *sistema 'round-robin'*. Este método provoca tempos crescentes de latência[6] entre processos, uma vez que um processo de prioridade baixa (menos importante) pode atrasar um outro processo de prioridade mais elevada (maior importância), se o primeiro iniciar uma operação longa. Outro método utiliza um *'dispatcher'* (normalmente orientado por prioridades) para decidir quando executar um processo com base na sua prioridade - sistema preemptivo por prioridades. Neste método, os vários processos recebem níveis de prioridade e dependendo da política particular de escalonamento, recebem os recursos apropriados. Acontece, no entanto, que em sistemas operativos modernos, a multitarefa é feita ao nível do *'thread'* – um processo é constituído por vários *'threads'* que partilham memória. Isto origina mais um fenómeno que é a contenção de memória e o respectivo problema de inversão de prioridade associado. Tudo o resto se passa exactamente da mesma forma que para os processos (níveis de prioridade, precedência entre processos, sincronização, etc.). Outra característica importante inerente à multitarefa é a reentrância – uma função não reentrante não pode ser chamada por outro processo enquanto está a ser executada pelo primeiro. A reentrância é uma característica importante a ter em atenção, uma vez que pode permitir o fenómeno de inversão de prioridade.

A *Protecção de memória* é uma característica muito importante para sistemas complexos com muitos processos em execução, simultaneamente. Um sistema com protecção de memória mantém os processos com espaços distintos de memória e a falha (*'crash'*) de um processo só afecta o espaço de memória desse, continuando todos os outros em execução.

---

[6] *Latência* é o tempo que vai desde a ocorrência de um evento até que este seja atendido pelo Sistema Operativo (SO).

Um outro factor importante é a *Modularidade*. Trata-se da característica que permite retirar funcionalidades de uma versão completa de um SOTR (alguns SOTR têm mais de 1000 chamadas ao sistema) até atingir um *'microkernel'* compacto. Através da modularidade, podemos ter controlo sobre as funcionalidades que pretendemos para o sistema. Assim, um sistema de tempo-real modular é um *'microkernel'* com vários *'plugins'*, que pode ser personalizado pelos módulos que pretendemos incluir. Este tipo de sistemas operativos de tempo-real têm grandes *'overheads'*, mas permitem ter um sistema operativo à medida das necessidades. Claramente não serão SOTR a utilizar em aplicações de tempo-real críticas.

O *Respeito por normas* é a característica que estabelece um API[7] normalizado para os sistemas operativos – o POSIX . As principais vantagens são, por um lado, a possibilidade de facilmente mudar de SO, e, por outro, ter mais facilidade em encontrar *'software'* já desenvolvido (*'off-the-shelf software'*), tal como *'drivers'* ou aplicações. Acontece no entanto, que as normas não garantem o desempenho do SOTR. Por exemplo, a norma POSIX.1b especifica as extensões de tempo-real em termos de funcionalidade e não em termos de desempenho. Normalmente, estes SOTR devem ser usados unicamente para suportar sistemas de tempo-real não-críticos. De qualquer forma, dependendo da aplicação, o desempenho e a máxima previsibilidade podem não ser tão importantes como a portabilidade, facilidade de programação e tempo de desenvolvimento. É preciso não esquecer que é útil que um SO inclua ferramentas de desenvolvimento de várias companhias, que podem valorizar significativamente o sistema, já que um sistema operativo com muitas relações de parceria é uma norma informal, tendo muitas das vantagens de uma norma formal e aberta.

---

[7] O *API* é o acrónimo para a *'Application Program Interface'* e corresponde ao conjunto de funções que um sistema oferece para o desenvolvimento de aplicações.

O *Comportamento quantitativo do sistema operativo* deve ser conhecido pelo projectista de uma aplicação através dos vários tempos gastos pelas operações do sistema. Existem várias figuras de mérito que devem ser fornecidas claramente pelo construtor do SOTR e ser correspondentes ao caso. Por exemplo: a latência das interrupções mais desfavorável, o tempo de mudança de contexto, o tempo de preempção, etc.

Em relação à *adequação ao 'hardware'*, é preciso dizer que se temos um processador específico com extensões poderosas, o sistema operativo de tempo-real deve ser optimizado para o processador para tirar dele o melhor partido.

3.2.2 Classificação de Sistemas Operativos de Tempo-Real

Os factores apresentados na secção anterior conduzem ao aparecimento de um grande número de sistemas operativos de tempo-real. Estes podem ser classificados em três categorias [RS94]: núcleos pequenos e proprietários (comerciais e *'homegrown'*[8]), extensões de tempo-real para sistemas operativos clássicos e resultantes de investigação.

Existem dois tipos de núcleos pequenos, rápidos e proprietários: *'homegrown'* e de oferta comercial. Ambos são utilizados para pequenos sistemas embebidos, quando se pretende garantir rapidez e máxima previsibilidade de execução. Os núcleos *'homegrown'* são, normalmente, específicos para uma determinada aplicação, uma vez que são desenvolvidos para suportar um sistema de tempo-real específico. O custo de desenvolvimento e manutenção destes núcleos únicos, assim como a crescente qualidade dos de oferta comercial, está a reduzir significativamente a prática de desenvolvimento deste tipo de sistemas.

---

[8] É a designação para núcleos constituídos por um construtor de sistemas de tempo-real para suportar as suas aplicações.

Estes são normalmente versões reduzidas e optimizadas de sistemas operativos *'time-sharing'*[9], para se conseguir velocidade e previsibilidade. Para reduzir os *'overheads'* de execução introduzidos pelo núcleo e tornar o sistema rápido, estes têm: um tempo de mudança de contexto curto, tamanho reduzido, respondem rapidamente a interrupções externas, minimizam os intervalos de tempo em que as interrupções estão inibidas, não suportam memória virtual e fornecem a possibilidade de bloquear (*'lock'*) código e dados em memória. Para lidarem com os requisitos temporais, estes núcleos fornecem: tempos de execução no caso mais desfavorável para todas as primitivas, mantêm um relógio de tempo-real, fornecem um mecanismo de escalonamento por prioridades, etc. Normalmente, estes núcleos executam em multitarefa, permitem a comunicação e sincronização entre processos através de primitivas bem conhecidas e usuais, tais como mailboxes, semáforos, etc.

Uma aproximação diferente para a criação de sistemas operativos de tempo-real é expandir sistema operativos *'time-sharing'* existentes, para versões tempo-real (UNIX para RT-UNIX ou MACH para RT-MACH). Estas versões tempo-real dos sistemas operativos clássicos são, normalmente, mais lentas e menos previsíveis que os núcleos proprietários, mas apresentam maior funcionalidade e melhores ambientes de desenvolvimento de aplicações. Estes SOTR são baseados num conjunto de *'interfaces'* usuais, eventualmente normas, que facilitam a portabilidade. Vários problemas podem surgir quando se tenta converter um sistema operativo *'standard'* numa versão tempo-real. Os problemas podem aparecer a dois níveis: ao nível da *'interface'* do sistema (API) e ao nível da implementação. Ao nível da *'interface'* do sistema, normalmente existe falta de funções que permitam definições de tempo-real (definição de limites temporais, políticas de escalonamento, etc.). Ao nível da implementação, podem ocorrer vários problemas: *'overheads'* intoleráveis,

---

[9] *'Time-sharing'* é a designação original para sistemas operativos concorrentes.

latência excessiva na resposta a interrupções, não preemptividade do núcleo, fraca granularidade temporal, etc. Assim, este tipo de SOTR tendeu a ser utilizado em aplicações de tempo-real não críticas, onde a perda de uma *'deadline'* não tem consequências graves.

Enquanto muitas aplicações de tempo-real continuam a ser desenvolvidas, baseadas nos dois tipos de sistemas operativos apresentados, continuam a verificar-se a existência de vários problemas. Em particular: os núcleos proprietários apresentam dificuldades quando utilizados em aplicações mais complexas e as extensões a sistemas *'time-sharing'* valorizam a velocidade em vez da previsibilidade. Assim, actualmente, na investigação em sistemas operativos de tempo-real estão a fazer-se tentativas que correspondem ao desenvolvimento de novos modelos de tempo-real que possam garantir previsibilidade e cumprimento das restrições temporais, para suportar os sistemas de tempo-real complexos e distribuídos da próxima geração.

Diz-se em [RS94] que, no contexto dos sistemas operativos, a maioria dos núcleos proprietários comerciais, assim como as extensões de tempo-real a sistemas operativos *'time-sharing'*, não cumprem as necessidades de previsibilidade dos sistemas de tempo-real. Em relação aos núcleos resultantes de investigação, isso é uma área em desenvolvimento.

Para um comparativo entre os vários tipos de SOTR mencionados consultar [San97].

3.3 Linguagens de programação de tempo-real

O nível da linguagem fonte é onde tipicamente os programadores especificam as propriedades de uma aplicação de tempo-real. Esta

descrição deve fornecer informação suficiente ao Executivo[10] responsável pela gestão dos processos na fase de execução, para que a aplicação exiba o comportamento desejado. Esta definição de linguagem de programação de tempo-real contraria um dos mal-entendidos apresentados por Stankovic em [Sta88]: "A programação em tempo-real é escrita de código *assembly*, programação de interrupções e escrita de *'device drivers'*". Isto não é verdade, porque embora a previsibilidade seja, aparentemente, fácil de garantir, uma vez que a programação é feita ao nível da máquina, estas técnicas são muito trabalhosas e por vezes introduzem condicionalismos temporais adicionais os quais podem influenciar a previsibilidade da implementação. Pese ainda que não é fácil codificar restrições temporais ao nível do código máquina. Assim, deverão ser criadas formas para que o desenvolvimento de sistemas de tempo-real seja simplificado. Isto poderá ser conseguido através da criação de linguagens de alto nível, que facilitem o desenvolvimento de sistemas de tempo-real.

Existem várias linguagens que dizem ser próprias para o desenvolvimento de sistemas de tempo-real, segundo vários argumentos. Alguns são baseados na visualização de sistemas de tempo-real como simples extensões dos sistemas convencionais, e por conseguinte, as linguagens são enriquecidas por métodos de controlo da progressão do tempo (representação do tempo actual) e colocação de alarmes. Outros, que mascaram a manipulação de prioridades de processos, adicionam, normalmente, à sintaxe das linguagens instruções para as restrições temporais. No entanto, a maioria das linguagens de tempo-real actuais, usando o modelo de processos para a programação, assumem também o modelo de execução convencional. Este modelo manipula um conjunto de processos que são preemptados em conformidade com as prioridades de

---

[10] Outra designação para núcleo e corresponde à parte do sistema que faz o sequenciamento das operações.

execução, competindo por recursos e boqueando quando os recursos já estão a ser utilizados (programação concorrente).

É preciso notar que o aspecto da previsibilidade surge em todas as fases do desenvolvimento de sistemas de tempo-real, mas começa ao nível da linguagem de programação. Isto porque é aí que o comportamento desejado é especificado e de onde a descrição de execução da aplicação é derivada.

De acordo com [Nie94], Halang e Stoyenco definiram que propriedades uma linguagem de programação deve ter e discutiram como é que certas linguagens existentes cumprem os requisitos temporais. O critério foi dividido em três grupos: elementos da linguagem que facilitam o desenvolvimento de aplicações, os serviços da linguagem na fase de execução necessários para um desempenho previsível e fiável da aplicação de tempo-real e os mecanismos de análise e verificação do *'software'*. Utilizando este critério, Halang e Stoyenco definem um número de linguagens que são essencialmente convencionais com manipulação de prioridades e com propriedades relacionadas com o tempo, que na maior parte das vezes são adicionadas inadequadamente. Em relação a estas, as propriedades e pressupostos da linguagem fonte determinam: o vocabulário disponível para descrever operações, a informação disponível para suportar a análise para a previsão do comportamento e muitos aspectos de como os processos devem ser geridos em tempo de execução.

É, assim, importante que as linguagens de programação de tempo-real forneçam formas de verificação da previsibilidade das aplicações de tempo-real resultantes. Ou seja, que o seu desempenho não esteja dependente do sistema operativo em grande parte. Por outro lado, é necessário que estes mecanismos de verificação sejam adicionados adequadamente, assim como as capacidades de definição das características temporais, de forma a que, e mais uma vez, a previsibilidade das aplicações resultantes seja garantida.

3.4 Considerações sobre o *'hardware'*

Todos os sistemas e aproximações considerados pressupõem *'hardware'* convencional. A motivação económica para este facto é evidente, mas a orientação para a valorização do caso médio da maioria do *'hardware'* impõe limitações no grau e tipo de previsibilidade que os métodos de desenvolvimento e implementação descritos conseguem atingir. Optimizações para o caso médio tornam uma grande parte do desenvolvimento de *'hardware'* perfeitamente inadaptado para os projectos de tempo-real. Por exemplo, o desenvolvimento de um *'pipeline'* de processadores é normalmente optimizado para os casos mais comuns onde existem *'caches'* de instruções e dados para os mais solicitados. Esta é a aproximação correcta quando o desenvolvimento considera os sistemas para os quais o caso médio é o critério mais valorizado. Não é a melhor aproximação, quando a previsibilidade do comportamento é o critério primário de desenvolvimento.

Até ao momento, pouco trabalho foi feito para o desenvolvimento e implementação de *'hardware'* específico para sistemas de tempo-real por várias razões [Nie94]. Em primeiro lugar, as restrições à previsibilidade resultantes das propriedades do *'hardware'* são muitas vezes mais subtis do que as resultantes dos outros níveis do sistema. Estas podem permanecer mascaradas até que os métodos e desenvolvimentos sobre previsibilidade a estes outros níveis (sistema operativo e linguagem de programação) se tornem suficientemente precisos para revelar as limitações do *'hardware'*. Outra razão prende-se com o esforço significativo e custos de desenvolvimento e implementação de novo *'hardware'*, face ao requerido para o *'software'*. Mais, os desenvolvimentos dos outros níveis dos sistemas influenciam claramente o conjunto de propriedades que o *'hardware'* deve ter para fornecer o melhor suporte. Assim, as restrições dos sistemas devem ser suficientemente conhecidas para que se assista à estabilização dos requisitos e só então se deve considerar o

desenvolvimento de *'hardware'* específico. Para algum trabalho na área, nomeadamente uma proposta de *'hardware'* específico, consultar [Nie94].

3.5 Síntese

Foram abordados neste capítulo os aspectos teóricos e os práticos em torno do projecto de sistemas de tempo-real, com a finalidade de os conjugar no âmbito do desenvolvimento de sistemas de tempo-real.

Nos aspectos teóricos, fizemos um levantamento sobre Algoritmos de escalonamento. Aqui apresentamos uma classificação possível dos algoritmos escalonamento como: estáticos e dinâmicos. Dentro dos algoritmos de escalonamento dinâmicos distinguimos os: de prioridades fixas e os totalmente dinâmicos. A seguir, fazendo um percurso do mais simples para o mais complexo, apresentámos paulatinamente as técnicas, algoritmos e restrições para o escalonamento de tarefas periódicas independentes, de tarefas aperiódicas de tarefas dependentes e de sistemas multiprocessador.

Nos aspectos práticos, fizemos a caracterização dos vários suportes, em termos da sua aptidão para sistemas de tempo-real. Começámos pelos sistemas operativos, definindo as características e evidenciando os vários factores que permitem optar por um deles para o desenvolvimento de STRs: multitarefa, protecção de memória, modularidade, respeito por normas, comportamento e adequação ao *'hardware'*. Continuámos com a apresentação de uma classificação e respectivas características para o agrupamento dos vários SO existentes: núcleos pequenos, extensões de SO tradicionais e resultantes de investigação. Foram apresentadas, também, as características que as linguagens de programação e o *'hardware'* devem possuir para serem considerados em conformidade com os sistemas de

tempo-real. No caso do *'hardware'*, foram descritas as razões pelas quais este ainda não é pensado em conformidade com os STR.

Cabe-nos, agora, apresentar as várias hipóteses de teste de sistemas de tempo-real encontrados na literatura, de forma a identificar um ou mais, que nos possibilite avaliar, segundo determinados preceitos, o seu desempenho durante as várias fases de desenvolvimento.

Capítulo 4

Testes de sistemas de tempo-real

Este capítulo pretende apresentar as várias técnicas e metodologias de medição do desempenho de sistemas de tempo-real encontradas na literatura sobre estes. Esta avaliação pode ser efectuada, tendo em linha de conta os vários objectivos e estágios do desenvolvimento de uma aplicação de tempo-real. Assim, por um lado, podemos efectuar a avaliação do sistema na fase de projecto, onde podemos verificar *a priori* o eventual cumprimento dos pressupostos de tempo-real. Isto é o que é tradicionalmente designado por **análise de escalonabilidade**. Por outro lado, o teste poderá ser efectuado na fase de desenvolvimento, concretamente, para optar ou simplesmente avaliar, um suporte prático (*''hardware''*, sistema operativo, algoritmo de escalonamento) para a implementação de uma aplicação de tempo-real. Isto é o que designamos por técnicas de avaliação de desempenho ou simplesmente *'benchmarks'*.

Este é o assunto mais importante deste capítulo, uma vez que nos vai permitir saber como testar um suporte prático concreto.

4.1 Análise de escalonabilidade

O processo de garantir que um sistema de tempo-real cumprirá os requisitos temporais é designado como análise de escalonabilidade e normalmente é efectuada na fase de projecto.

O método mais conhecido e utilizado para analise da escalonabilidade de sistemas, normalmente, na fase de projecto é o *'Rate Monotonic Analysis'* (RMA). Trata-se de um conjunto de métodos com prova matemática, que permite analisar a escalonabilidade de um sistema de tempo-real em desenvolvimento. Estes métodos têm um importante papel no desenvolvimento de sistemas de tempo-real previsíveis e confiáveis. Isto porque, conhecendo as características temporais do modelo de tarefas que constituem um sistema de tempo-real, é possível, ainda na fase de projecto, avaliar se o modelo é escalonável ou não, através do método de análise de escalonabilidade.

O método RMA só permite avaliar modelos que sejam para ser escalonados por prioridades fixas, uma vez que é baseado no algoritmo RM.

Dado o papel de relevo que a análise de escalonabilidade ocupa no processo de desenvolvimento de aplicações de tempo-real, existem actualmente propostas de ferramentas para o efeito. Estas ferramentas são normalmente independentes das implementações, e cobrem as necessidades de pré-análise, durante a primeira fase do desenvolvimento de sistemas de tempo-real. A principal utilidade é prestar auxílio à correcta análise do comportamento temporal ainda na fase de desenvolvimento. Tendo em atenção esta perspectiva, estas ferramentas podem ser, também, de grande ajuda a programadores de sistemas operativos de tempo-real, uma vez que fornecem critérios que permitem determinar a responsividade dos mesmos.

Foi assim proposto uma ferramenta de análise de escalonabilidade independente da implementação para aplicações de tempo-real, designada

por SAT – *'Schedulability Analysis Tool'* [GT95]. A análise suportada pelo SAT é efectuada e aplicável ao desenvolvimento de aplicações de tempo-real críticas. No entanto esta ferramenta pode ser também utilizada durante a implementação, numa perspectiva de refinamento. Isto é conseguido, pela remodelação e refinamento da aplicação, incluindo os efeitos que o suporte prático (SOTR e *'hardware'*) introduzem no comportamento temporal. Os tipos de tarefas suportadas são as periódicas e aperiódicas. Actualmente esta ferramenta suporta dois algoritmos de escalonamento: o *'Rate Monotonic'* e o *'Deadline Monotonic'*. Esta ferramenta dá um conjunto de respostas apropriadas ao programador, nomeadamente: se todo o conjunto de tarefas é escalonável e propõe alternativas para soluções possíveis caso não o seja; quais as possíveis tarefas penalizadas; etc.

4.2 Técnicas de avaliação de desempenho

A avaliação de suportes práticos para sistemas de tempo-real pode ser feita a vários níveis do sistema: ao nível da linguagem de programação – onde no final da última década apareceram várias *'benchmarks'* para a linguagem *Ada* e ao nível do sistema operativo – que é mais eficiente porque é o nível de *'software'* mais baixo sobre o *'hardware'*. Assim sendo, apresentaremos aqui várias metodologias e métricas para, medir, objectivamente, o desempenho de tempo-real, tanto ao nível da linguagem de programação como ao nível do sistema operativo. Estas metodologias ou métricas têm como objectivo, por um lado, permitir a escolha do suporte adequado (linguagem de programação, sistema operativo e/ou *'hardware'*) à aplicação de tempo-real a desenvolver e, por outro, a verificação das características de tempo-real após a implementação.

4.2.1 Classificação das *'benchmarks'*

A tecnologia de avaliação para a medição do desempenho de sistema de tempo-real assume várias formas. Apresentamos aqui uma taxonomia que permite agrupar estas técnicas em três tipos que diferem no grau de detalhe em que avaliam o sistema alvo. Acontece, contudo, que estas metodologias e métricas podem pertencer a uma ou mais das seguintes aproximações [WK92], [GMS94].

Em primeiro lugar temos as *'Fine-grained benchmarks'*, que efectuam a medida de um executivo de tempo-real a um nível baixo, avaliando a eficiência do *'hardware'* e *'software'* de base para as primitivas mais frequentemente disponibilizadas pelo núcleo de tempo-real. Normalmente, estas *'benchmarks'* tentam isolar uma característica de tempo-real específica e medi-la. Face ao desempenho das várias primitivas (combinadas de forma específica), é atribuída uma figura de mérito ao conjunto constituído pelo *'hardware'* e *'software'*. O principal problema associado a este tipo de *'benchmarks'* é que para obter medidas precisas é, muitas vezes, necessário recorrer a dispositivos de medida próprios (*'hardware'* especializado), uma vez que são necessários dispositivos de precisão suficiente para a medida de certas primitivas, como por exemplo a latência de uma interrupção. De outra forma, a experiência pode não ser repetível, tornando o processo de medida inútil. Uma das *'benchmarks'* puras desta categoria, que será descrita posteriormente, é a *'Rhealstone'* [KP89].

As *'benchmarks'* **orientadas para aplicação** (*'Application-oriented benchmarks'*), efectuam a medida do executivo de tempo-real a um nível mais alto, em termos do número de *'deadlines'* mantidas ou perdidas e o ponto de utilização no qual o sistema começa a tornar-se insatisfatório (*'breakdown point'*). Estas são normalmente implementadas como aplicações sintéticas que executam sobre o executivo de tempo-real. Esta designação de sintético é no sentido que os requisitos das *'deadlines'* são

obtidos a partir de uma determinada distribuição, ou então, porque se trata de uma aplicação de âmbito geral que simula as características de tempo-real com certos requisitos: tarefas concorrentes periódicas ou aperiódicas com o seu tempo de execução e *'deadlines'* estabelecidas ou impustas de qualquer forma.

Com base no número de *'deadlines'* mantidas, a combinação de *'hardware'* e *'software'* conseguiu obter, é atribuída uma figura de mérito. A *'benchmark' 'Hartstone'* [WK92] é a melhor *'benchmark'* conhecida nesta categoria, devido às suas características.

Ao nível das linguagens de programação existem outras duas *'benchmarks'* muito referenciadas na literatura são: Whetstone [Wei89] e Dhrystone [Wei84]. Nestas são testadas uns conjuntos de instruções específicos, baseados na frequência de instruções das linguagens, utilizadas em aplicações caracsterísticas para obter as ditas figuras de mérito. Esta categoria de *'benchmarks'* têm como principal vantagem fornecer ao programador figuras de mérito para o comportamento completo do sistema de tempo-real. Isto é, com base nas características tempo-reais definidas para a aplicação sintética e obtida a figura de mérito respectiva, conseguimos saber qual o comportamento de qualquer STR, com as mesmas características.

Nas avaliações baseadas em **simulações** (*'Simulation-based evaluations'*) que modeliza o sistema de tempo-real é modelado a um nível conveniente de detalhe. Estas simulações podem ser usadas para prever o desempenho de um sistema de tempo-real baseado em determinados pressupostos à cerca do seu desenvolvimento, incluindo os requisitos de processamento e os algoritmos de escalonamento. Este é o nível mais abstracto para a avaliação de sistemas de tempo-real e tem como principais desvantagens, por um lado, não ser possível validar o modelo de simulação desenvolvido, e por outro, o facto de raramente ter em atenção todos os componentes de *'hardware'* e *'software'* (suporte prático do STR). Uma simulação pode levar a conclusões deturpadas se o modelo não reflectir a realidade

adequada em termos de detalhe e precisão. Por outro lado, simulações detalhadas são normalmente proibitivas em termos de custos e requerem um grande esforço de codificação. A maior vantagem desta aproximação é que o próprio *'hardware'* pode ser modelado em conjunto com o *'software'*. Isto permite aos projectistas conhecer as necessidades de desempenho dos núcleos de tempo-real no *'hardware'*, que podem ainda não estar completamente desenvolvido [GMS94].

Poderão aparecer *'benchmarks'* constituídas por combinações destas técnicas, as quais poderão ser mais adequadas às aplicações de tempo-real a desenvolver.

4.2.2 Avaliação ao nível da linguagem de programação

Existem várias *'benchmarks'* para a avaliação das linguagens de programação utilizadas no desenvolvimento de sistemas de tempo-real. No entanto, a linguagem *Ada* foi aquela que despertou maior interesse nos criadores das *'benchmarks'*, uma vez que foi a linguagem criada e adoptada para o desenvolvimento de sistemas de tempo-real pelo Departamento de Defesa Americano (DOD) e por conseguinte foram desenvolvidas várias métricas para avaliar as características desta linguagem. Por outro lado, esta linguagem funciona "como" um sistema operativo, daí a importância de ser abordada nesta dissertação

Patrick Donohoe fez uma pesquisa sobre as *'benchmarks'* para medir o desempenho de tempo-real da linguagem *Ada* [Don87]. Esta pesquisa fornece uma descrição sumária das principais *'benchmarks'* disponíveis para esta linguagem. As *'benchmarks'* descritas são: as da Universidade do Michigan, as do Grupo de Trabalho do ACM para os aspectos de desempenho (PIWG – *'ACM Performance Issues Working Group'*) e o protótipo de avaliação de capacidade do compilador *Ada'* (ACEC – *'Ada Compiler Evaluation Capability'*).

As *'benchmarks'* da Universidade do Michigan para a linguagem *Ada*, concentram-se em técnicas para a medida de desempenho das características individuais desta linguagem. Por exemplo: chamada a subprogramas, afectação dinâmica de espaço de armazenamento, manipulação de excepções, operações sobre tarefas (criação, activação e terminação), rendezvous de tarefas, etc.

As *'benchmarks'* do PIWG são agrupadas em três categorias gerais: *'benchmarks'* compostas (inclui versões *Ada* das *'benchmarks'* *'Whetstone'* [Wei89] e *'Dhrystone'* [Wei84]), testes individuais das características temporais (tais como os das *'benchmarks'* da Universidade do Michigan) e testes de compilação (medida da velocidade de compilação). As *'benchmarks'* sintéticas ao serem programas que reflectem as frequências das estruturas da linguagem utilizadas nos programas *Ada* medem, consequentemente, na forma compilada, não só a arquitectura do sistema mas também a habilidade do compilador gerar código eficiente.

O propósito do protótipo ACEC é fornecer: um conjunto organizado de testes de desempenho a seguir, *'software'* de suporte para a execução dos testes e coligir estatísticas de desempenho.

4.2.3 Avaliação ao nível do sistema operativo

Tradicionalmente, os sistemas operativos para sistemas de tempo-real têm sido *'homegrown'* e especializados para as aplicações criadas por cada construtor. Este tipo de sistemas operativos são, normalmente, adaptados aos requisitos e pressupostos específicos das aplicações a desenvolver e, por conseguinte, são dispendiosos. Actualmente, existem vários sistemas operativos de tempo-real de âmbito geral (núcleos pequenos e proprietários e extensões de tempo-real para sistemas operativos tradicionais), desenvolvidos por vários fabricantes ou, eventualmente, propostos por equipas de investigação.

O grande problema é saber se estes tipos de sistemas operativos suportam os requisitos temporais das aplicações de tempo-real. Mais ainda, cada fabricante declara que o seu sistema é o mais modular, rápido, fácil de utilizar e bem documentado. A maioria das especificações que acompanham os sistemas operativos de tempo-real fornecem tempos básicos para operações, tais como, mudança de contexto e tempo de resposta de interrupções. No entanto, a metodologia para a obtenção destes tempos não é normalmente apresentada. Muitas vezes, cada característica é medida de forma que faça com que o sistema operativo particular pareça melhor que os da concorrência: alguns tempos são médios, outros são típicos e apenas outros são no caso mais desfavorável – o valor que realmente interessa. Assim, a maioria dos fabricantes e vendedores de sistemas operativos de tempo-real falham no fornecimento de informação sobre as medidas adequadas para a avaliação dos seus sistemas.

Existem, assim, várias metodologias e métricas que podem auxiliar o programador a fazer a escolha acertada em termos de sistema operativo de tempo-real.

Começamos pela métrica Rhealstone, que pertence às *'fine-grained benchmarks'* e que foi proposta por Kar e Porter [KP89]. Esta consiste na medição quantitativa de seis componentes que podem influenciar o desempenho de sistemas de tempo-real. Este seis componentes são: tempo de mudança de contexto, tempo de preempção, tempo de latência das interrupções, tempo de obtenção de semáforo após ser libertado por outra tarefa (*'semaphore shuffling time'*), tempo de resolução de *'deadlock'* e tempo de *'datagram throughput'* (número de kilobytes por segundo enviados de uma tarefa a outra). Todos estes tempos são conseguidos pela média das medidas de várias acções sequenciais, dado que os tempos obtidos são de uma ordem de grandeza bastante pequena (micro-segundos ou mili-segundos). Kar e Porter propuseram um procedimento para o cálculo da figura de mérito designada por **número Rhealstone**. Este consiste na combinação das várias medidas num só valor através da

conversão para uma mesma unidade (segundos) e fazendo a inversão aritmética de cada uma (frequência da medida). O número *'Rhealstone'* é então calculado pela soma simples ou ponderada de todas as frequências traduzindo-se em, quanto maior o valor, melhor o desempenho do sistema. Se for considerado que todos os componentes influenciam de forma similar o desempenho, então, a soma simples é considerada: Se não for o caso, é considerada a soma ponderada, cujos pesos reflectem a importância de cada componente para o desempenho do sistema. A figura de mérito da *'benchmark' 'Rhealstone'* caracteriza uma combinação de núcleo e *'hardware'* que é praticamente independente da aplicação, a não ser que seja usado o método da soma ponderada para o cálculo do número *'Rhealstone'*.

A *'benchmark' 'Rhealstone'* têm sérios problemas [GMS94]: Em primeiro lugar, as *'deadlines'*, que são de primeira importância em sistemas de tempo-real, não são consideradas – a *'benchmark'* avalia a velocidade de operações em vez de ter em atenção os pormenores de tempo-real. Em segundo lugar, um único valor para o tempo de preempção não é fácil de obter, uma vez que é um problema complicado (devido ao fenómeno da inversão de prioridade) num sistema de tempo-real multitarefa. Em terceiro lugar, o critério de bloqueio por *'deadlock'* perde importância face aos protocolos de herança de prioridade usados pela maioria dos escalonadores. Finalmente, as seis categorias de medidas consideradas são algo *'ad hoc'*.

Foi feita uma implementação da *'benchmark'* Rhealstone por Kar [Kar90], um dos seus autores. Este artigo apresenta a definição refinada da *'benchmark'*, através de programas em linguagem C para o núcleo de tempo-real da Intel IRMX. As principais diferenças para a *'benchmark'* proposta inicialmente são: a substituição do critério *'datagram throughput'* por latência da mensagem entre tarefas e o cálculo do número *'Rhealstone'*. Neste caso, após a conversão em segundos, é calculada a média aritmética dos componentes e só depois é que o resultado é invertido aritmeticamente para obter a frequência (*'Rhealstones'*/segundo).

Uma outra metodologia de avaliação foi proposta por Eric McRae [McR96] e foi designada por "Medição de sistemas operativos de tempo-real, utilizando uma *'benchmark' 'Dhrystone'* modificada para representar a carga da aplicação". Embora apareça a designação de *'Dhrystone'*, esta *'benchmark'* pode ser considerada com uma *'fine-grained benchmark'*, uma vez que efectua a medida de várias características do sistema operativo de tempo-real. O *'Dhrystone'* é unicamente utilizado para representar a carga para o sistema operativo a ser avaliado. Esta *'benchmark'* não foi desenvolvida para ser combinada numa única figura de mérito e, assim, não permite a comparação global com outras ofertas de outros vendedores. Esta *'benchmark'* possibilita o reconhecimento intuitivo do desempenho de um SOTR. Isto é, permite ter uma primeira impressão sobre um SOTR, de acordo com os requisitos temporais do sistema, enquanto que a maioria dos testes não permitem, intuitivamente, saber como se comporta um SO num *'hardware'* específico. Os testes propostos por esta *'benchmark'* pretendem caracterizar os aspectos de um SOTR que são mais importantes para os programadores de sistemas embebidos. Estamos a falar de: desempenho na mudança de tarefa, desempenho na gestão de prioridades de tarefas, desempenho na manipulação de semáforos, desempenho na afectação de memória, desempenho na passagem de mensagens, latência de interrupções. Ou seja, qual o desempenho global do sistema). A chave desta abordagem é que os resultados da *'benchmark'* são apresentados como percentagem dos recursos de CPU/*'hardware'* são consumidos pelo SOTR, em termos temporais. A relação é determinada correndo o sistema com o SOTR inibido para obter uma base de comparação (*'baseline'*) e, posteriormente, correr os testes nos quais o SOTR deve executar as tarefas especificas. Esta *'benchmark'* é uma boa aproximação na avaliação de SOTR, porque cobre a maioria das características que necessitamos conhecer a fundo para a avaliação de sistemas embebidos. A única desvantagem é que as medidas são relativas e é difícil fazer uma avaliação precisa baseada em medidas relativas em vez das absolutas consideradas noutros casos.

A *'benchmark' 'Hartstone'* [WK92], pertence à classe das *'application-oriented benchmarks'* e permite avaliar sistemas de tempo-real críticos em termos das *'deadlines'* mantidas. Originalmente desenvolvida para sistemas de tempo-real uniprocessador, a *'benchmark'* Hartstone é constituída por: tarefas periódicas, tarefas aperiódicas e outras tarefas, para representar pontos de sincronização na aplicação. As tarefas periódicas têm *'deadlines'* críticas enquanto que as tarefas aperiódicas podem ter *'deadlines'* críticas e não críticas. A implementação da *'benchmark'* uniprocessador *'Hartstone'* é completamente independente do SO e *'hardware'* de suporte. Inicialmente, é definido um conjunto de base (*'baseline task set'*), o qual é caracterizado pelas tarefas que o compõem, as suas *'deadlines'*, períodos e tempos de pedido de activação. As experiências na *'benchmark'* seguem por etapas, sendo um parâmetro modificado em cada etapa, enquanto os outros se mantêm constantes. As várias séries de experiências utilizam uma mistura de tarefas sendo constituídas por: *Séries PH* – tarefas periódicas com frequências harmónicas (frequências múltiplas da mais pequena); *Séries PN* – tarefas periódicas com frequências não harmónicas; *Séries AH* – tarefas periódicas e tarefas aperiódicas; *Séries SH* – tarefas periódicas com sincronização e *Séries AS* – tarefas periódicas, aperiódicas com sincronização. Os tempos de execução, *'deadlines'*, tempos de bloqueio e número de tarefas são variados, apropriadamente, em cada etapa e em cada experiência, até que as *'deadlines'* comecem a ser perdidas, ou até outras condições de paragem se tornem verdadeiras (como por exemplo, tempos de resposta muito grandes). O ponto em que isto se verifica, é a figura de mérito para a combinação *'hardware'* e sistema operativo de tempo-real em avaliação. Isto é medido em total de *'Kilo-Whetstone Instructions Per Second'*, ou simplesmente, KWIPS, porque nas primeiras experiências [Wei89], a *'benchmark' 'Whetstone'* foi utilizada para representar a carga do sistema). Para além da figura de mérito, uma análise cuidada do ponto de quebra pode resultar em melhorias de desempenho para o executivo de tempo-real.

Esta *'benchmark'* foi expandida para ter em linha de conta sistemas de tempo-real distribuídos [KW91].

A *'benchmark' 'Hardstore'* pode ser utilizada para avaliar sistemas relativamente complexos, uma vez que é orientada para as *'deadlines'*. A sua grande vantagem, em relação às *'fine-grained benchmarks'*, é fazer uma avaliação do sistema alvo com base nas *'deadlines'* cumpridas. A principal desvantagem é que não são conhecidos os tempos de execução das primitivas individuais do SOTR, pelo que é difícil saber onde reside o problema, no caso do sistema alvo não ser escalonável.

Foi proposto em [MCV96] uma metodologia abrangente (*'comprehensive methodologie'*) e as respectivas métricas para medirem quantitativamente o desempenho de tempo-real e o determinismo dos sistema operativos de tempo-real actuais. De acordo com os autores do artigo, as duas *'benchmarks'* (*'Rhealstone'* e *'Hartstone'*) tradicionalmente consideradas como as melhores para a avaliação de SOTR, não são adequadas para aplicações de tempo-real que exijam um conhecimento profundo da previsibilidade do SOTR de suporte. A *'benchmark' 'Rhealstone'* não está completa, uma vez que existem situações que não são consideradas nas suas métricas (por exemplo, tempo para a inversão de prioridade na gestão de um determinado recurso) que pode ter especial relevância para várias aplicações de tempo-real actuais. Por seu turno a *'benchmark' 'Hartstone'* é excessivamente genérica. Esta apresenta faltas de detalhe tais como resposta a eventos externos, transferência de dados entre tarefas, etc. As métricas propostas no artigo pretendem ser abrangentes, úteis e fáceis de implementar. Estas métricas podem ser consideradas na classe das *'fine-grained benchmarks'*, pelo que efectuam as medidas das seguintes características essenciais dos SOTR actuais: resposta a eventos externos - através da manipulação de interrupções de *'hardware'*, que é considerado como uma das características principais para sistemas de tempo-real; sincronização entre tarefas e partilha de recursos - que são os serviços do SOTR utilizados para controlar o acesso a recursos partilhados, sinalização

de eventos síncronos e transferência de dados entre tarefas. Neste conjunto de métricas, consideramos que há falta de medidas importantes para a avaliação de SOTR, como por exemplo, tempo de mudança de contexto, tempo de preempção, etc. Por outro lado consideramos, que a resposta a eventos externos e avaliação de sincronização entre tarefas são medidas bastante importantes, uma vez que os sistemas de tempo-real interagem com o meio ambiente, respondem atempadamente aos eventos deste e as tarefas normalmente não são independentes.

Em [Sac95] é sugerido um conjunto de testes simples que podem ser utilizados para efectuar a medida da eficiência do sistema operativo. Os testes baseiam-se unicamente em ferramentas de *'software'* e não necessitam de qualquer *'hardware'* especial. As medidas consideradas neste artigo são baseadas numa lista das capacidades mais populares para comunicação entre tarefas, sincronização de tarefas e sinalização de eventos. Assim, o âmbito do artigo é limitado à área dos sistemas operativos de tempo-real. Os vários testes considerados são: testes para a medição da duração da transferência de uma mensagem e da comunicação através de um *'pipe'*; testes para a medição da velocidade de sincronização de tarefas através de *'proxies'* e sinais; um teste para a medição da duração da mudança de tarefa e um teste para a medição da latência da interrupção do *'timer'*. Todos os testes apresentados foram implementados e aplicados ao sistema operativo distribuído de tempo-real QNX.

4.3 Avaliação por simulação

As simulações são utilizadas em efectivo, para a avaliação do desempenho de sistemas implementados e não implementados. O processo de simulação de sistemas consiste numa formulação de um modelo, implementação desse modelo e avaliação do desempenho baseado nas estatísticas obtidas.

Existem dois tipos de simulação de sistemas: simulações de eventos discretos e simulações continuas (*'timestepped'*). As simulações de eventos discretos (DES) executam, normalmente, melhor do que as simulações continuas, uma vez que as estas modelam as mudanças verificadas no estado do sistema, enquanto que as simulações contínuas modelam o sistema em intervalos específicos. Estas simulações tendem a ser extremamente dispendiosas em termos de tempo e custo. Em primeiro lugar porque, os modelos iniciais normalmente têm que ser refinados e executados várias vezes. Por outro lado, porque os simuladores são mais lentos que os sistemas alvo a serem modelados.

Uma forma de aumentar a velocidade destas simulações é através da execução em paralelo. No entanto, as restrições de sincronização e devido ao acesso ordenado à informação do estado em fase de execução, têm um papel crucial nestas simulações. Por outro lado, é praticamente impossível prever as dependências entre eventos. Assim, se uma determinada simulação toma uma aproximação conservadora (caso mais desfavorável) para a paralelização o desempenho pode não ser muito melhor que a execução em série. Uma aproximação optimista foi proposta para a paralelização de simulações [GPF+93].

4.4 Síntese

Neste capítulo identificámos várias metodologias e métricas, que possibilitam a avaliação de sistemas de tempo-real. Esta avaliação pode ser feita a vários níveis e estádios de desenvolvimento de uma aplicação. Isto é, pode ser feito ao nível da linguagem de programação ou ao nível do sistema operativo. Por outro lado, a avaliação pode ser feita na fase de análise, na fase de desenvolvimento ou mesmo na fase de implementação e teste.

Fizemos uma descrição mais aprofundada das técnicas de avaliação de desempenho ao nível do sistema operativo de tempo-real, mas não deixámos de discutir as outras fases e níveis para a avaliação. Assim, foram descritas várias *'benchmarks'* nas várias aproximações consideradas: *'fine-grained benchmarks'*, e *'application-oriented benchmarks'*.

Assim, consideramos que as várias metodologias e métricas para a avaliação do desempenho, podem ser aplicadas de acordo com a complexidade dos sistema e objectivos a cumprir. As *'benchmarks'* puramente *'fine-grained'* podem ser aplicadas com sucesso principalmente para determinar tempos de execução das várias primitivas dos SOTR. Por outro lado, podem ser aplicadas a sistemas embebidos, baseados em núcleos pequenos e rápidos, de forma a obter uma figura mérito global. Nesta perspectiva de avaliação global de uma combinação de SO e *'hardware'*, é importante acentuar que uma eventual combinação dos vários testes das várias metodologias podem conduzir a uma *'benchmark'* óptima. Por outro lado, as *'application-oriented benchmarks'* devem ser aplicadas para avaliar um sistema de suporte de aplicações de tempo-real em termos abstractos, numa perspectiva de cumprimento ou não das restrições temporais.

Chamemos à atenção que estas *'benchmarks'* podem definir o comportamento da aplicação de tempo-real a desenvolver, se os parâmetros para a *'benchemark'* corresponderem aos parâmetros temporais do nosso futuro sistema de tempo-real.

Com a informação apresentada até aqui, pensamos ter todo o suporte para passar à Dedução Experimental dos modelos de Escalonamento. Antes disso é necessário fazer uma caracterização do meio de desenvolvimento das experiências a desenvolver, o que constitui o assunto do próximo capítulo.

Capítulo 5
Caracterização do meio de desenvolvimento

Neste capítulo pretendemos definir o meio de desenvolvimento das experiências efectuadas no âmbito desta dissertação. Este meio é constituído por vários componentes. Em primeiro lugar, um *'hardware'* de suporte *'standard'*, relativamente modesto e relevante do ponto de vista dos STR. Depois, o *'software'* de suporte (núcleo de tempo-real) que: tenha um bom desempenho, seja de domínio público, adequado ao *'hardware'* escolhido, tenha ferramentas de desenvolvimento poderosas e de preferência conhecidas e por fim seja bem documentado. A seguir, é necessário caracterizar a aplicação sintética para implementar os programas para as várias experiências, assim como a linguagem de programação a utilizar. Por fim, um sistema de medida autónomo e com uma precisão suficiente, para efectuar as medidas de tempos do conjunto *'hardware'* e SO escolhidos.

5.1 Caracterização do *'hardware'*

A escolha do *'hardware'* recaiu num PC modesto em relação aos padrões actuais. Isto é, um Computador Pessoal equipado com um processador 80386Sx a da Intel 25 MHz, 2 MB de memória RAM, 125 MB de disco rígido e uma unidade de disquetes 3 ½".

A razão da escolha prende-se com vários factores:

- Por se tratar de um sistema modesto, podemos beneficiar da simplicidade do *'hardware'*, já que não inclui nenhuns mecanismos desenvolvidos para melhorar a performance em termos de tempos médios de resposta, por exemplo, *'caches'*. Assim, é de prever o comportamento.
- É genérico (e *'standard'*).
- Está bem documentada.
- Tem muito suporte: *'hardware'* e *'software'*.
- Não se trata de um sistema de controlo dedicado pelo que possui características suficientes de computação genérica para ser utilizado no controlo de sistemas físicos, executando programas razoavelmente sofisticados e desenvolvidos em linguagens de programação de alto nível.
- Muitos dos sistemas embebidos são baseados neste tipo de sistemas: centrais telefónicas, monitorização de processos fabris, etc.
- É barato!

5.2 Caracterização do *'software'*

Embora muitas vezes confundidos com sistemas operativos, os núcleos de tempo-real fornecem um conjunto mais reduzido de serviços, normalmente: capacidades de multitarefa, sincronização e comunicação entre processos, preempção baseada em prioridades. Este tipo de sistemas também são designado por Executivos.

Optámos pela utilização de um núcleo, de domínio público (logo, gratuito e sem royalities), para efectuar as experiências que nos propusemos. A primeira vantagem é o facto de ser gratuito e por conseguinte disponível.

Depois, optou-se por realizar experiências sobre um conjunto de recursos que estão disponíveis à maioria dos investigadores de STR, por forma a que estas possam confirmar os resultados.

Para efectuar uma melhor avaliação dos vários núcleos de tempo-real encontrados apresentamos, alguns dos principais requisitos que estes devem cumprir tendo em atenção o âmbito do trabalho. Assim o núcleo a utilizar deve:

- permitir a criação, eliminação e execução de múltiplas tarefas de tempo-real;
- garantir que uma tarefa de prioridade mais alta, quando solicite execução, seja interrompida a tarefa de prioridade mais baixa (escalonamento preemptivo);
- fornecer ferramentas de desenvolvimento fáceis de utilizar, nomeadamente compiladores para linguagens de alto nível, como por exemplo o *C*;
- fornecer um bom suporte documental, incluindo exemplos.

Como é compreensível, qualquer núcleo (ou executivo) não constitui por si só um sistema operativo, pelo que deve ser integrado ou eventualmente ter como suporte um sistema operativo, que designamos como sistema hospedeiro ou simplesmente *'host'*.

Existem várias hipóteses, entre elas: UNIX, LINUX, MS-DOS ou qualquer outro sistema operativo tradicional para os quais são criadas extensões de tempo-real.

Dado a natureza do *'hardware'* escolhido é lógico que procuremos núcleos que tenham como suporte o MS-DOS. As vantagens estão à vista e são essencialmente as seguintes: trata-se de um sistema amplamente conhecido e utilizado pelo que existem muitas ferramentas de desenvolvimento disponíveis (compiladores, editores, *'debbugers'*, etc.), o seu desempenho já é razoavelmente conhecido. Trata-se de um sistema aberto.

No entanto, este também enferma determinadas desvantagens do ponto de vista dos SOTR, que têm que ser colmatadas pelo núcleo. Entre elas salienta-se: ser um sistema monotarefa e mono-utilizador, que normalmente é transformado num sistema multitarefa recorrendo ao mecanismo das interrupções; as primitivas não terem tempo de execução conhecidos no caso mais desfavorável (característica herdada do *'hardware'*). Assim, são necessários testes exaustivos.

Far-se-á no resto desta secção uma descrição dos vários núcleos e SOTR encontrados, por forma a que se possa optar por um.

5.2.1 Pesquisa de núcleos de tempo-real

Começamos pelo núcleo designado por **RTOS**. Este núcleo foi desenvolvido por Mike Koller, um engenheiro de *'software'* para sistemas embebidos.

Trata-se de um núcleo multitarefa preemptivo baseado em prioridades que suporta as funções essenciais de manipulação de tarefas, semáforos, filas e gestão de memória. Ao longo de publicação desta dissertação, este núcleo (RTOS) pode ser obtido a partir do *URL http://www.goofee.com/mike.htm*.

O núcleo foi escrito em *C*, mas existem algumas chamadas ao sistema operativo dependentes do sistema alvo para a inicialização e comutação de tarefas que foram codificadas em *'Assembly'*.

O núcleo é fornecido com as fontes e utiliza o MS-DOS como sistema operativo hospedeiro. As ferramentas de desenvolvimento são quaisquer

para DOS, utilizando qualquer compilador de *C* para a criação do programa executável.

A API deste núcleo engloba várias funções que a seguir se apresentam:

- Aproveitamento dos *'ticks'* do *'timer'* através da rotina correspondente do sistema operativo;
- Manipulação de interrupções e excepções: através do registo de entrada e saída numa interrupção;
- Gestão de tarefas: criação, modificação de prioridades, definição de um tempo de espera, suspensão e eliminação;
- Manipulação de filas: criação, colocação de tarefas na fila, teste do estado da fila, obtenção das tarefas da fila e eliminação;
- Gestão de memória: inicialização; afectação e libertação de blocos de memória.

Como principais considerações temos que, a documentação encontrada é bastante escassa, resumindo-se à constante página web colocada pelo autor no *URL*, já referido.

Não existe qualquer referência relativamente ás garantias de tempo-real oferecidas pelo núcleo pelo que seria necessário efectuar alguns testes para verificar o cumprimento dos requisitos temporais.

Foi incluído um pequeno exemplo PCM - que controla e apresenta o estado de uma impressora ligada à porta paralela.

Não é referido nada quanto às possibilidades de escalonamento de tarefas com prioridades idênticas. Por testes efectuados verificamos que só permite o escalonamento de prioridades, pelo que não é possível, pelo menos directamente, escalonar tarefas com prioridades idênticas.

O segundo núcleo encontrado foi o **RTX** (***'Real-Time Operating System'***). Este núcleo pode ser obtido a partir do *URL* *http://world.std.com/~mikep/rtx-page.html*. Mediante a documentação disponível, trata-se de um executivo para sistemas de tempo-real críticos com um tempo de mudança de contexto entre tarefas bastante rápido e uma resposta efectiva das tarefas dentro de um tempo específico. No entanto, estes tempos não são quantificados. Esta versão é específica para processadores da *Intel* a correr em modo real. A maior parte do código é em *C*, no entanto, alguns serviços de interrupções e rotinas de manipulação da pilha foram codificados em *Assembly*.

Tal como no caso anterior também este núcleo é fornecido no formato de fontes pelo que deverá ser compilado e ligado com a aplicação de tempo-real em desenvolvimento, a qual é executada sobre o MS-DOS. O Código fornecido foi testado com o Compilador *'Microsoft C'* ver. 5.1.

O *RTX* tem as seguintes características:

- Suporte de várias tarefas em concorrência;
- Escalonamento preemptivo por prioridades, que podem ser redifinidas dinamicamente;
- Partilha de tempo entre tarefas (escalonamento *'round-robin'*), para tarefas com a mesma prioridade;
- Temporizadores de guarda a eventos (*'Watch dog timers'*);
- Tarefas activadas por tempo (*'Time-triggered'*);
- Tempo de comutação de contextos que requerem poucos ciclos de relógio (não é especificado quanto);
- Comunicação entre tarefas é feito através do pedido de colocação em fila de objectos ou eventos, uma vez que cada tarefa tem uma fila associada;

- As tarefas podem suspender e correr periodicamente.

Este executivo tem um mecanismo interno que permite coexistir pacificamente com o MS-DOS, de forma a que este último não afecte a funcionalidade do núcleo. Todas as chamadas ao sistema são interceptadas pelo executivo, o qual só as passa ao DOS caso seja um serviço crítico, deixando em espera a actividade do núcleo até que se complete o serviço.

Neste núcleo não foram incluídas as várias rotinas de serviço a interrupções a não ser uma mínima para o teclado. Isto não constitui um problema porque no âmbito do trabalho a desenvolver não é necessário a utilização de dispositivos de *'hardware'*. No entanto poder-se-á criar as rotinas que se julguem necessárias.

Por se tratar de um núcleo com aplicação em sistemas de tempo-real críticos e pela documentação que o acompanha, poderá ser uma boa possibilidade de escolha. Um outro ponto a favor é o facto de utilizar um compilador de C para MS-DOS, o que facilita muito o desenvolvimento de aplicações de tempo-real.

Um outro núcleo de domínio público, bastante utilizado é **µC/OS**. Este é um núcleo de tempo-real, multitarefa e preemptivo para microprocessadores, com possibilidade de ser colocado em ROM. A maior parte do núcleo foi codificada em linguagem *C*, com uma parte dependente do microprocessador alvo, codificada em linguagem *Assembly*. Neste momento encontram-se disponíveis versões para vários microprocessadores inclusive os microprocessadores da Intel da família *80x86*. A versão testada no âmbito deste trabalho é para o microprocessador da *Intel i386*.

Por seu turno, o compilador de *C* deve ser compatível com o ANSI e permitir *'in line assembly'* ou conter extensões à linguagem que permitam inibir e desinibir interrupções.

A partir da pouca informação encontrada e das *'sources'* fornecidas é possível enumerar as seguintes características:

- o código encontra-se bastante optimizado;
- as interrupções estão inibidas muito pouco tempo (menos de 400 ciclos de relógio);
- uma tarefa pode eliminar outra tarefa;
- uma tarefa pode mudar a prioridade de outra;
- o núcleo disponibiliza um relógio de tempo-real;
- o núcleo tem implementado os seguintes mecanismos de sincronização: semáforos, mailboxes e filas.

Acompanham, esta distribuição, dois exemplos que mostram como utilizar as funções do núcleo. Os exemplos foram testados no compilador da *Borland para C++* versão 3.0.

Seria necessário uma análise mais aprofundada das *sources* fornecidas para conseguirmos determinar as verdadeiras potencialidades deste núcleo.

No entanto, segundo testes efectuados, o escalonador não permite de forma directa o escalonamento de tarefas com a mesma prioridade. Se esta for tentada a aplicação bloqueia por não saber que tarefa escalonar.

Um outro núcleo encontrado foi o **CTask** ***'A Multitasking Kernel for C'***. CTask é um conjunto de rotinas que permitem a aplicações em *C* executar tarefas em concorrência. O CTask manipula a partilha do tempo do processador com um escalonador preemptivo baseado em prioridades. Este fornece um conjunto completo de rotinas para a comunicação entre tarefas, sinalização de eventos. Este núcleo inclui, também, um número de *'drivers'* para MS-DOS cujo resultado é um conjunto básico de funções que permitem implementar aplicações de E/S através das portas série e paralelas. A distribuição do CTask é constituída por código fonte (*'sources'*) e código pré-compilado (*'binaries'*). A versão pré-compilada foi compilada com o modelo de memória *large*, mas uma vez que, as rotinas

do núcleo não chamam qualquer função da biblioteca de *Cμ*, pode-se utilizar esta versão em qualquer outro modelo de memória excepto nos modelos *Tiny* e *Huge* do *Turbo C*.

No CTask o escalonador é invocado em cada *'tick'* do *'timer'* do sistema. Primeiro guarda o contexto da tarefa actual e em seguida, retira o primeiro elemento da fila de tarefas prontas a executar e repõe o contexto dessa tarefa, recomeçando-a a partir do ponto onde tinha sido interrompida anteriormente.

No CTask são implementados vários mecanismos (primitivas) para comunicação/sincronização entre tarefas. Assim, podemos encontrar:

- *'mailboxes'*, em que uma tarefa aguarda a chegada de uma mensagem através deste mecanismo e por conseguinte não é escalonável.
- *'flags'*, que permitem assinalar um determinado acontecimento.
- *'pipes'*, que são similares às *'mailboxes'*, no entanto e, contrariamente às *'mailboxes'*, os *'pipes'* utilizam os seus próprios *'buffers'*.

Para a protecção de regiões críticas, o CTask fornece um mecanismo designado por *'resource'*, que deverá ser requerido pela tarefa que vai entrar na zona critica e libertado quando sair. Se o recurso estiver tomado, a tarefa que pretende entrar na zona critica tem que aguardar até que este seja libertado.

Embora o MS-DOS não seja reentrante e como o CTask intercepta todas as chamadas ao sistema, este efectua as necessárias reservas de *'resources'*, por forma que duas tarefas concorrentes não chamem funções que ponham em causa a reentrância do MS-DOS.

O CTask permite 65.535 níveis diferentes de prioridades, e as tarefas com prioridades mais altas são escalonadas antes das que têm prioridades mais baixas. Também as tarefas com prioridades mais altas têm acesso ao *'mail'*,

'pipes' e *'resources'* antes das outras tarefas. Em aplicações de tempo-real criticas, pode-se inibir a preempção de tarefas, para que o *'tick'* gerado pelo *timer* não provoque a comutação de tarefas, caso as tarefas tenham prioridades idênticas.

Para compilar aplicações baseadas no CTask, podem ser utilizados os compiladores: *Microsoft C 5.1* ou posterior ou então o *Turbo C 2.0* ou posterior. Para as partes em *Assembler*, podem ser usados o *Microsoft MASM 5.1* ou posterior, ou o *TASM 1.01* ou posterior, respectivamente.

No sistema CTask existe pelo menos uma fila essencial: a fila das tarefas elegíveis (*'eligible queue'*) que contém todas as tarefas que estão à espera de oportunidade para ser executadas. Cada vez que o escalonador é invocado, este guarda em primeiro lugar todos os registos do processador no bloco de controlo da tarefa corrente e adiciona a tarefa corrente na fila apropriada. Só depois: é retirado o primeiro elemento da *'eligible queue'*, a *'stack'* é comutada com a *'stack'* da tarefa retirada e os registos do processador são restaurados do bloco de controlo da tarefa. Esta tarefa recomeça, então, a partir do ponto onde foi interrompida.

Este núcleo apesar de não ser vocacionado para aplicações de tempo-real críticas, por não ser possível conhecer os limites temporais de certas operações, pode ser uma possibilidade a considerar para o trabalho. Como principais pontos a favor temos:

- É extremamente bem documentado, fazendo-se acompanhar de um manual de programação bastante completo, o que facilita a sua utilização.

- Por outro lado trata-se de um sistema bastante completo em termos de funcionalidades.

O principal ponto contra, para além dos já enumerados, é que sempre que é gerado um *'tick'* do *'timer'* existe uma mudança de contexto, quando, eventualmente, a tarefa a escalonar possa ser a mesma. Isto é, um

comportamento bem definido, pelo que para além do aumento do tempo da tarefa, não acarreta outras consequências.

O último núcleo de que falaremos é o **RTEMS** (***'Real-Time Executive for Multiprocessor/Military Systems'***), que é um executivo (núcleo) de grande desempenho, orientado por objectos, multiprocessamento, que fornece um vasto conjunto de possibilidades para o desenvolvimento de aplicações de tempo-real actuais. O RTEMS serve como base para o desenvolvimento de aplicações de tempo-real portáveis, reutilizáveis e eficientes, devido à sua estrutura modular e à orientação por objectos.

O RTEMS foi desenvolvido pelo OLAR (*'On-line Application Research'*) para o programa de comando de mísseis do exército dos Estados Unidos. Este fornece todas as potencialidades e flexibilidade dos executivos de tempo-real disponíveis no mercado comercial, mas sem os custos e *'royalties'* associados.

O RTEMS está disponível na Internet. Este encontra-se actualmente implementado nas linguagens de programação *C* e *Ada*.

A solução RTEMS fornece assim um suporte para o desenvolvimento de sistemas embebidos. Este apresenta as seguintes características:

- Capacidade de multitarefa;
- Sistema multiprocessador homogéneo e heterogéneo;
- Escalonador preemptivo, baseado em prioridades e orientando acontecimentos;
- Escalonamento *'rate monotonic'* (opcional);
- Comunicação e sincronização entre tarefas;
- Herança de prioridades;
- Gestão de *'interrupts'* responsivo;

- Afectação dinâmica de memória;
- Configuração pelo utilizador a alto nível;
- Facilidades na reutilização de *'software'*.

O RTEMS apresenta um desempenho elevado e é determinístico o que permite suportar as características de um sistema de tempo-real (tempos de resposta efectivos com limites apertados mesmo com carga computacional máxima).

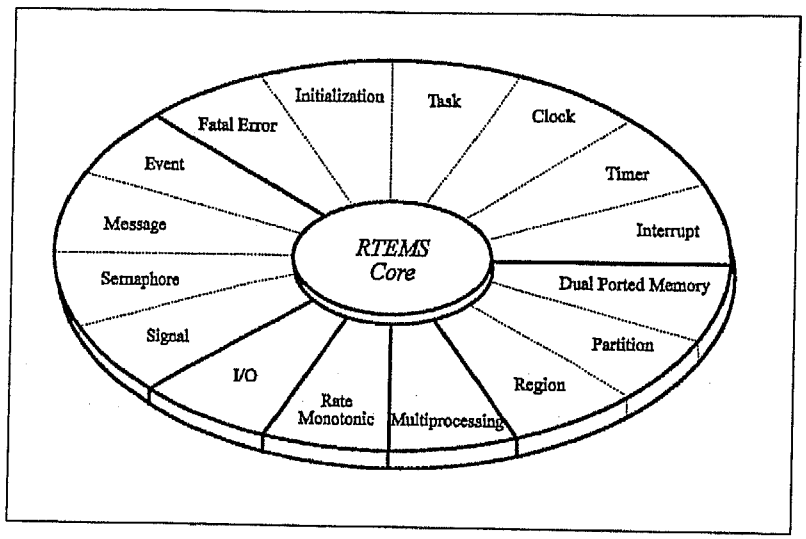

Fig. 4 – Arquitectura interna do RTEMS

A interface do executivo apresentado à aplicação é formado por agrupamento de directivas em conjuntos lógicos chamados resource managers (gestores de recursos). As funções utilizadas pelos vários gestores tais como escalonamento despacho e gestão de objectos são fornecidas pelo núcleo. Em conjunto estes componentes fornecem um ambiente de execução poderoso que garante o desenvolvimento de aplicações de tempo-real eficientes.

Os vários gestores fornecidos conforme se pode constatar pela Fig. 4 são: inicialização, tarefa, interrupção, *'clock'*, *'timer'*, semáforo, mensagem, evento, sinal, partição, região, *'dual ported memory'*, E/S, *'Rate Monotonic'*, erro fatal e multiprocessamento.

As várias ferramentas de desenvolvimento baseiam-se nas ferramentas de desenvolvimento da GNU, bibliotecas ANSIC: biblioteca de *C* da GNU e NEWLIB de Cygnus (biblioteca de funções de tempo-real para sistema embebidos).

A julgar pela informação, sem sombra de dúvidas, estamos perante o melhor núcleo de tempo-real de domínio público com desempenhos anunciados, equivalentes aos comerciais. No entanto, por ser concebido para vários sistemas alvo e várias plataformas de desenvolvimento, torna-se algo complexa a sua utilização para desenvolvimento de aplicações para o *'hardware'* escolhido. Por outro lado, o facto de fornecer garantias do cumprimento escropoloso das restrições temporais e bons desempenhos, poderá não ser a melhor escolha para o trabalho que pretendemos desenvolver, uma vez que é necessário explorar um pouco, qual a confiabilidade de aplicações desenvolvidas sobre suportes que não foram pensados para tempo-real.

5.2.2 Escolha do núcleo

Em relação à aplicabilidade para suporte de aplicações de tempo-real, todos os núcleos podem ser utilizados desde que se trate de aplicações não críticas. Segundo a documentação existente, aqueles que se assumem como suporte para sistemas de tempo-real críticos são o RTX e o RTEMS. Por outro lado, qualquer um apresenta funcionalidades suficientes para o trabalho a desenvolver.

Dois dos núcleos são preferíveis para o trabalho, devido essencialmente à documentação fornecida e facilidade de desenvolvimento. Estes são o *RTX* e o *CTask*.

Como escolha, devido aos vários factores apresentados, optamos pelo CTask para os desenvolvimentos a efectuar.

5.3 Aplicações sintéticas

Tendo como principal objectivo a compreensão do comportamento de aplicações de tempo-real em termos práticos, foram pensadas e desenvolvidas determinadas experiências práticas. Em primeiro lugar importa saber, em concreto, como se comporta uma aplicação de tempo-real, constituída por várias tarefas concorrentes, perante um escalonamento estático. Aqui as prioridades são a mesma para todas as tarefas. Depois, é preciso conhecer o comportamento de outro conjunto de tarefas concorrentes, agora perante o escalonamento preemptivo por prioridades.

Optámos, assim, por implementar um modelo de tempo-real constituído por várias tarefas. Todas executam o mesmo código genérico, sem funcionalidade prática alguma, constituído por um ciclo de duração pré-definida. O tamanho variável do ciclo permite-nos considerar vários tempos de execução para as tarefas.

No caso do escalonamento estático materializado pelo escalonamento *'round-robin'*, as tarefas têm todas a mesma prioridade, sendo executadas numa ordem arbitrária do princípio ao fim e umas a seguir às outras. Os tempos de execução das várias tarefas, assim como o seu número, são arbitrários, sendo introduzidos como parâmetros no programa de teste. Estes tempos de execução são expressos em número de ciclos a executar.

No caso do escalonamento preemptivo por prioridades, é materializado pelo algoritmo de prioridades fixas *'Rate Monotonic'*. Desta forma, as prioridades são atribuídas nos moldes do estabelecido pelo algoritmo, isto é, as tarefas recebem a prioridade em função inversamente proporcional ao seu período. Assim, neste caso para além do número de tarefas a executar e do tempo de execução expresso em número de iterações do ciclo, é fornecido como parâmetros os tempos do período para cada tarefa em mili-segundos.

5.4 Linguagem de programação

Os programas descritos no ponto anterior foram implementados na linguagem de Programação *C*.

A nossa escolha foi baseada, em primeiro lugar, no facto do núcleo escolhido, apresentar as bibliotecas com as várias funções, nesta linguagem de programação. Em segundo lugar é uma linguagem bastante versátil, permitindo programar tanto a alto nível como ao nível do sistema operativo. Por fim, os compiladores e ambientes de desenvolvimento desta linguagem para MS-DOS são fáceis de utilizar. Neste campo optámos por utilizar o Compilador de C++ da Borland, o *'Borland C++ ver.3.1'*.

5.5 Caracterização do sistema de medida

Com o intuito de efectuar as medidas de tempos do sistema operativo de tempo-real e do *'hardware'* de suporte, optámos por utilizar uma placa de E/S – PCL-711BPC-Multilab Card.

Após análise às características da placa, verificamos que esta se encontra equipada com o circuito integrado da *Intel 8253* – contador programável decrescente – responsável pela geração do *'pacer'* para as conversões A/D. Cada *CI Intel 8253* fornece três canais contadores independentes. Como se pode verificar pela Fig. 5, a placa em questão coloca o contador 1 e o contador 2 em cascata, resultando num contador de 32-bits.

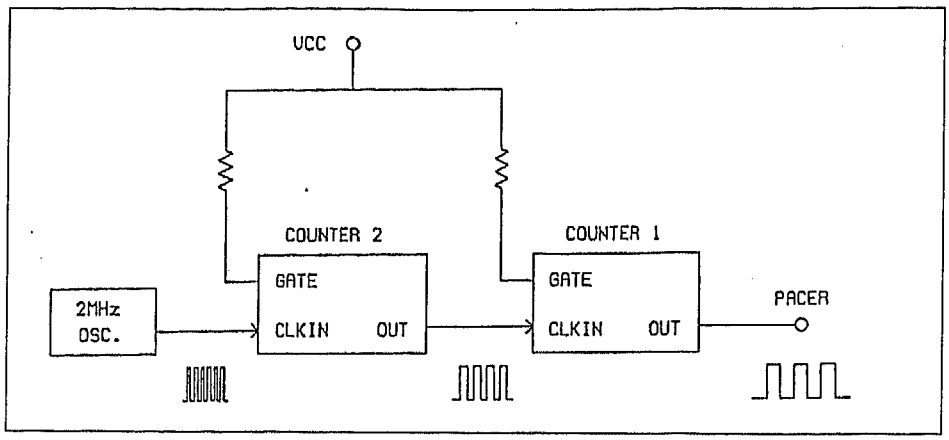

Fig. 5 – Esquema de ligação dos contadores do circuito integrado Intel 8253 na placa PCL-711B

Foi colocado na entrada do contador 2 um oscilador de 2MHz, sendo assim capaz de ter uma precisão de 500ns. É possível medir tempos até ao máximo 35,8 minutos, o que nos parece um cenário bastante bom para o trabalho a desenvolver.

5.6 Síntese

Neste capítulo fizemos a caracterização do meio de desenvolvimento para as experiências a desenvolver. Assim, caracterizamos: o *'hardware'*, o núcleo de tempo-real, a aplicação sintética, a linguagem de programação e o sistema de medida.

Em relação ao *'hardware'*, o escolhido foi um Computador Pessoal equipado com o processador da *Intel 80 386Sx* a 25 MHz, com 2 MB de memória RAM, 125 MB de disco fixo e um drive de disquetes de 3 ½".

Em relação ao núcleo de tempo-real fizemos uma pesquisa dos vários de domínio público disponíveis na Internet e optámos pelo Ctask, por ser aquele que reunia a maioria das condições que enumerámos. Essencialmente, para além das funcionalidades que apresentam e que superam claramente as da maioria, têm um suporte documental muito bom.

A aplicação sintética constituiu o conjunto de programas a desenvolver, para efectuar as expriências pretendidas. Aqui caracterizámos dois modelos. O primeiro contempla o Escalonamento estático a executar com o algoritmo *'round-robin'*, uma vez que as tarefas têm todas a mesma prioridade. O segundo contempla o Escalonamento preemtivo por prioridades fixas materializado pelo algoritmo RM.

Relativamente à linguagem de programação a utilizar, optámos pelo compilador de *C++* da *Borland vers.3.1*. Para além de outras vantagens referidas, apresenta um ambiente de desenvolvimento útil e funcional.

Por fim fizemos a caracterização do sistema de medida que tínhamos assumido à priori que seria autónomo e separado do sistema de tempo-real. A nossa escolha recaiu sobre a placa PCL-711B, que contém um integrado com três contadores decrescentes programáveis (Intel 8253). Dois destes contadores foram colocados em cascata e alimentados por um oscilador de 2 MHz, o que nos permite contar com um contador de tempo com a resolução de 500 *ns* e um alcance máximo de 35,8 minutos.

Com o ambiente de desenvolvimento perfeitamente definido, o passo seguinte é a implementação dos programas que constituem a Aplicação Sintética e efectuar Experiências. No próximo capítulo faremos a dedução experimental dos modelos qualitativos descritos, entrando em linha de conta com as restrições introduzidas pelo suporte prático. Estas restrições serão devidamente quantificadas, assim como os próprios modelos, de forma a confirmarmos os Desenvolvimentos efectuados.

Capítulo 6
Dedução experimental de modelos de escalonamento

Este capítulo será inteiramente dedicado à avaliação em concreto de um *Sistema de tempo-real*, cujo suporte foi definido no capítulo anterior.

Pretende-se, assim, tendo em atenção os parâmetros desprezados pelo algoritmo de escalonamento e as restrições temporais introduzidas pelo sistema de suporte (*'hardware'* e sistema operativo), fazer uma previsão efectiva do comportamento do sistema. Para isso, serão propostas metodologias e experiências específicas. As metodologias propostas são desenvolvidas como resultado de uma plena compreensão das várias restrições introduzidas pelo suporte prático.

Um outro problema põe-se na determinação ou obtenção das sobrecargas computacionais referidas. Uma das possibilidades é trabalhar com as que são apresentadas pelo vendedor do SOTR que podem não ser as mais indicadas pelas razões já mencionadas. Além do mais, se estivermos a trabalhar com núcleos de domínio público, estas medidas nem sequer existem. Assim, seria bom definir formas de obtenção desses valores. Aqui recorremos às métricas descritas em algumas *'benchmarks fine-glained'*, nomeadamente, a Rhealstone, já que é necessário obter os tempos de execução de certas primitivas dos núcleos. Para além destas, são definidas outras que se julguem necessárias para obter os tempos em questão.

São aqui tomados em consideração os custos introduzidos pelos mecanismos de escalonamento do núcleo, que são função do *'hardware'* de suporte. Estes custos podem ser de dois tipos: sobrecargas, também designadas como *'overheads'*[11], e bloqueios.

Os *'overheads'* são definidos como o tempo dispendido no núcleo a efectuar serviço em benefício de uma tarefa específica. Os bloqueios, ou também designados por inversão de prioridade, são, no âmbito deste trabalho, o tempo passado no núcleo quando uma tarefa é impedida de executar por estar em curso uma ou mais operações do núcleo que não podem ser evitadas.

6.1 Modelos de escalonadores

O escalonador é o segmento do núcleo que implementa um algoritmo de escalonamento específico para atribuir, em cada momento, uma tarefa à CPU. Normalmente, a política de escalonamento implementada, em termos genéricos, é por prioridades dinâmicas, sendo garantido que nos pontos de decisão de escalonamento é colocada em execução a tarefa com prioridade mais elevada. Entenda-se pontos de decisão de escalonamento aqueles momentos em que se verifica a interrupção da tarefa actual e é executado o escalonador do sistema para verificar qual a tarefa a escalonar de seguida.

São aqui desenvolvidas as duas implementações mais encontradas em termos de núcleos, assim como os componentes dos *'overheads'* e bloqueios presentes em cada uma delas. Estas são o escalonamento orientado por acontecimentos e o escalonamento orientado por tempo.

---

[11] Apesar do termo correspondente em Português ser "sobrecarga computacional", utilizaremos o termo em Inglês *'overhead'*, por ser sobejamente conhecido e estar de acordo com o vocabulário encontrado na literatura.

Nas implementações orientadas por acontecimentos, que os escalonadores dependem de dispositivos externos de *'hardware'* para a geração de interrupções que permitem o escalonamento de outras tarefas. Assume-se, todavia, que as interrupções coincidem com o período da tarefa, normalmente referido como chegada de tarefa (*'task arrivals'*).

Nas implementações orientadas por tempo, utiliza-se a interrupção gerada por um *'timer'*[12] periódico para que o escalonador execute e desempenhe a sua função.

Existem outras implementações, que permitem aumentar o desempenho, pela redução de alguns dos *'overheads'* e bloqueios, normalmente passando para *'hardware'* algumas funcionalidades ou implementando certas astúcias [KAS93].

6.1.1 Estruturas de dados

Existem várias estruturas de dados genéricas que devem estar presentes no núcleo para guardar a informação de suporte ao escalonamento.

A primeira a ser definida, pela sua importância, é a estrutura de suporte às tarefas a serem executadas sobre um determinado executivo. Esta estrutura designada por bloco de controlo de tarefa (TCB[13]) permite guardar a informação necessária à execução da tarefa, nomeadamente: o contexto da tarefa em todo o momento e o seu estado de execução. O contexto corresponde ao espaço de endereçamento que ocupa, endereço da pilha, prioridade fixa e/ou dinâmica, etc. Em relação ao estado de execução num

---

[12] *'Timer'* é a designação para um Circuito Integrado que efectua a contagem do tempo e gera interrupções em momentos pré-definidos.

[13] TCB é o acrónimo para *'Task Control Block'* que corresponde à designação em Inglês para a estrutura.

determinado núcleo, as tarefas passam pelos seguintes estados principais: *em execução* (*'running'*), *bloqueada* (*'blocked'*) e *pronta a executar* (*'ready'*).

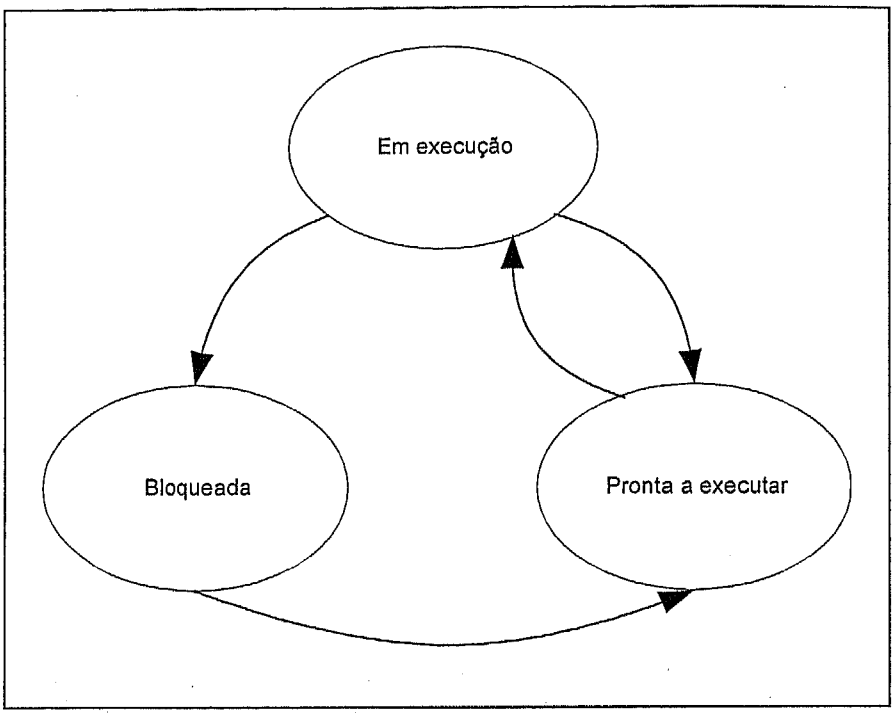

Fig. 6 – Estados possíveis de uma tarefa.

Uma tarefa está no estado *em execução*, se possuir o processador no momento e, por conseguinte, encontra-se a efectuar algum trabalho. Esta está no estado *bloqueada* se tiver sido parada explicitamente (*'stopped'*), tiver sido adiada (*'delayed'*), ou se estiver à espera de um acontecimento ou recurso (*'waiting'*). Por fim, uma tarefa está *pronta a executar* se não estiver a executar, por não lhe ter sido atribuído o processador pelo despachante do sistema, também designado *'dispatcher'*[14], estando parada para que outra tarefa execute. As mudanças de estado são feitas de acordo

[14] Parte do sistema operativo responsável pela afectação de tarefas ao processador a todo o momento, com base num Escalonamento específico. Em portugês pode ser desihnado por *despachante*.

com a política de escalonamento utilizada e respeitando os sentidos e direcções definidos na Fig. 6.

As outras estruturas utilizadas pela maior parte dos núcleos são normalmente constituídas por agrupamento de estruturas simples, como por exemplo TCB, de determinada forma. Normalmente, estas estruturas são preconizadas por filas, cuja característica principal é receberem elementos por uma extremidade designada "por fim" e retirarem elementos pela outra extremidade designada "por topo". Na maioria das vezes, estas filas são organizadas, seguindo uma determinada condição, pelo que normalmente a adição de novos elementos não é feita pelo fim, como foi dito anteriormente. De qualquer forma, o retirar de elementos é sempre pelo topo.

As restantes estruturas essenciais são:

- fila de execução (*'Run Queue'*) – fila de tarefas prontas a executar ordenadas por prioridades;
- fila de espera (*'Start Queue'*) – fila de tarefas bloqueadas, eventualmente à espera de novo período ou evento e ordenadas pelo tempo de activação;
- tarefa activa (*'Active Task'*) – tarefa que se encontra em execução efectiva, e que detém o processador a todo o momento.

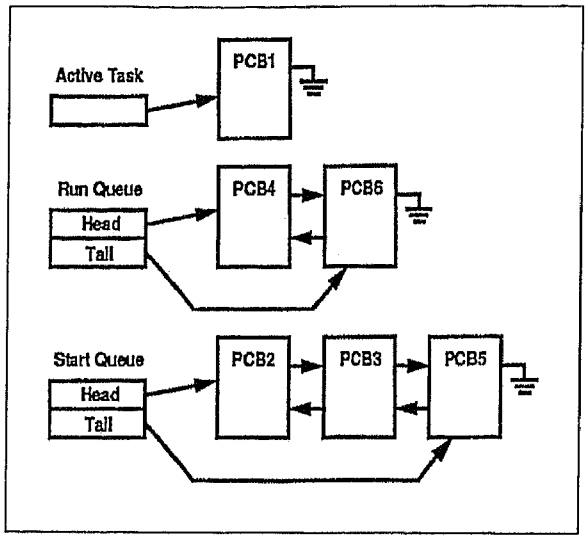

Fig. 7 - Filas para o escalonamento de prioridades fixas [KAS93].

Como se pode verificar pela Fig. 7, cada uma das estruturas descritas é constituída por um ou mais blocos de controlo de tarefas.

6.1.2 Pressupostos

Existem vários pressupostos a que o núcleo deve obedecer, de forma a que ele se enquadre no âmbito deste estudo:

- O manipulador de interrupções deve guardar um conjunto mínimo de registos do contexto para proceder ao processamento de uma interrupção. Se o escalonador decidir não preemptar a tarefa actual, o conjunto mínimo de registos do contexto deverá ser restabelecido e a tarefa activa deve continuar a execução.

- A comutação de contexto requer guardar o contexto da tarefa actual e repor o contexto da próxima tarefa a executar, que corresponde à tarefa cujo bloco de controlo está no topo da fila de execução. Assim, o núcleo deverá proceder também à retirada deste bloco de controlo da fila e à sua referenciação como tarefa activa, isto é, deverá ser colocado na estrutura tarefa activa.

- O fim de uma tarefa requer uma sinalização (*'trap'*) para o núcleo, de forma a que uma eventual rotina de tratamento do *'trap'* ou outro mecanismo seja responsável por repor o TCB da tarefa terminada na fila de espera e seja seleccionada a tarefa do topo da fila de execução para passar a ser a tarefa activa.

- O núcleo deverá assegurar que a tarefa mais prioritária na fila de execução (a que se encontra no cimo) é aquela que será a activada em todos os pontos de análise de escalonabilidade.

- O núcleo deverá ter um mecanismo que permita executar tarefas com a mesma prioridade (implementação do algoritmo *'round-robin'*, por exemplo).

6.1.3 Definições

As várias definições que se seguem são aplicáveis no âmbito deste estudo: escalonamento *'round-robin'* e escalonamento de prioridades fixas. Estas são os tempos atómicos ou também designados parâmetros das medidas que constituem os *'overheads'* do núcleo. É preciso chamar à atenção que os tempos que passaremos a definir podem ser diferentes de implementação para implementação. Assim, os vários tempos são:

- C_{int} – tempo para o tratamento de interrupções, guardar registos mínimos para processar a interrupção e chamada do escalonador. Este tempo é função do suporte prático.

- C_{sched} – tempo para executar o código do escalonador, de forma a determinar a próxima tarefa a executar – eventualmente mover os TCB's da fila de espera para a fila de execução e comparar a tarefa activa com a tarefa no cimo da fila de execução em termos de prioridades. Este tempo varia com a implementação.

- C_{resume} – tempo para voltar à tarefa em execução quando não existe mudança de contexto – repor o contexto de registos e voltar à execução normal da tarefa.

- C_{store} – tempo para guardar o estado da tarefa actualmente em execução no bloco de controlo (TCB) e inserção ordenada do TCB na fila de execução.

- C_{load} – tempo para carregar o estado da próxima tarefa a executar da fila de execução. Este tempo C_{load} é menor ou igual a C_{store} porque, mesmo que o tempo de guardar o estado da tarefa seja idêntico ao tempo de reposição do estado, o segundo tempo inclui a inserção ordenada que pressupõe pesquisa.

- C_{trap} – tempo para tratar o acontecimento gerado pelo terminar normal de uma tarefa – eventualmente, inserir o TCB da tarefa na fila de tarefas em espera até ao início do próximo período (no caso do escalonamento de tarefas periódicas), seguido da obtenção da próxima tarefa a executar da fila de execução (a que se encontra no topo da fila). Isto é, quando uma tarefa termina, o núcleo deverá ser notificado do facto para proceder ao despacho de uma nova tarefa. Este parâmetro corresponde ao tempo que estas operações levam a ser completadas.

6.2 Estudo do caso não preemptivo

Começamos pelo estudo do caso não preemptivo, no sentido de considerar um conjunto de tarefas independentes, todas com a mesma prioridade e escaloná-las umas a seguir às outras, sem haver interrupção de tarefas menos prioritárias. Trata-se do caso mais simples de escalonamento, uma vez que não é necessário aplicar uma política de escalonamento elaborada, a não ser colocar uma outra tarefa a executar quando a anterior terminou, até que todas as tarefas sejam executadas na totalidade. Idealmente, o

tempo de execução do conjunto é igual à soma dos tempos de execução de todas as tarefas. Isto é, se não houver qualquer *'overhead'* introduzido pelo suporte, o fim da execução do conjunto deveria coincidir com a soma dos tempos das tarefas individuais, o que não é o caso, como mostraremos. Existe uma carga computacional introduzida pelo *'dispatcher'* do núcleo resultante da detecção do fim de uma tarefa e do despacho da seguinte. Como também provaremos, quanto mais elaborada for a política de escalonamento, mais carga computacional será introduzida. Pretendemos, assim, identificar os *'overheads'* e definir um modelo que, entrando em atenção com eles, permita caracterizar o comportamento do sistema.

6.2.1 Definição do modelo

O modelo é constituído por n tarefas independentes com prioridades idênticas, sendo cada uma das tarefas τ_i caracterizada pelo tempo de execução (C_i) que poderá ser um qualquer (desde que mensurável com o mínimo de exactidão).

A ordem de execução das tarefas é indiferente, eventualmente definida na fase de escalonamento, podendo ser na forma mais simples, executar por completo uma tarefa após o fim de outra. As tarefas estão prontas a executar todas em simultâneo, sendo o *'dispatcher'* responsável unicamente por seguir o escalonamento previamente estabelecido. Isto significa que se decidiu *a priori* uma ordem qualquer para a execução das tarefas e o sistema operativo cumpre-a.

6.2.1.1 Análise qualitativa

Como se pode verificar pela Fig. 8, existe um overhead introduzido pelo núcleo sempre que uma tarefa termina, relativo à detecção do fim de uma tarefa e lançamento da seguinte que se encontra na fila de execução. Este tempo é constante e resulta da associação dos seguintes parâmetros:

$$C_{exit} = C_{trap} + C_{load} \qquad (6.1)$$

Interessa salientar que o tempo C_{trap} considerado para este caso é menor ou igual ao considerado aquando da definição do parâmetro, uma vez que não há o tempo de inserir o TCB da tarefa que termina na fila de tarefas suspensas, de forma ordenada.

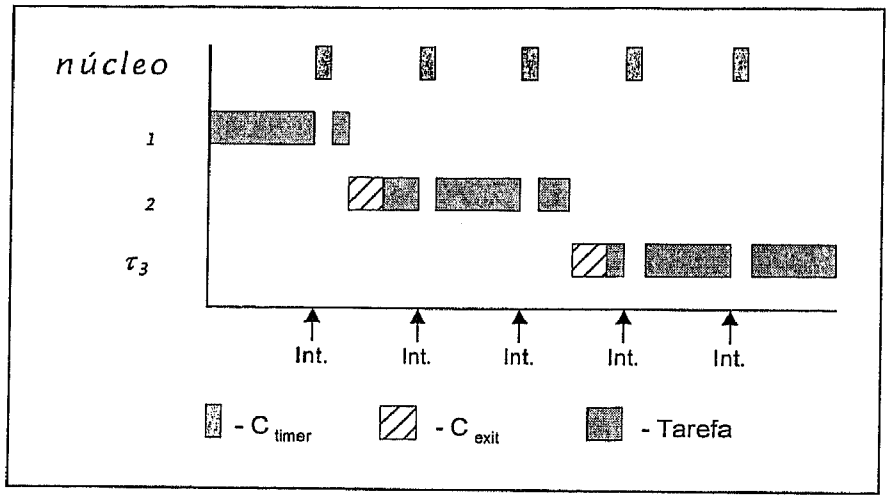

Fig. 8 – Escalonamento estático não preemptivo.

Existe um outro *'overhead'* correspondente à necessidade do núcleo aproveitar as interrupções do *'timer'* para, eventualmente, proceder à mudança para a próxima tarefa. No caso em análise, a mudança não se verifica, pelo que a associação de parâmetros que resulta nesta medida é a seguinte:

$$C_{timer} = C_{\text{int}} + C_{sched} + C_{resume} \qquad (6.2)$$

Se associarmos a cada tarefa τ_i o *'overhead'* correspondente a esta, que depende do número de interrupções que ocorreram durante o tempo que executou, temos que o referido *'overhead'* é o resultado da seguinte fórmula:

$$\left\lceil \frac{C_i}{T_{tic}} \right\rceil \times C_{timer} \qquad (6.3)^{15},$$

onde o T_{tic} corresponde ao tempo entre dois *'ticks'* sucessivos, ou seja, a resolução do *'timer'*. Este tempo constitui o pior caso, uma vez que a tarefa pode não ter sido iniciada numa interrupção.

Em relação ao *'overhead'* C_{exit}, e uma vez que ele não faz perder a *'deadline'* da própria tarefa τ_i, este deverá ser associado à tarefa τ_{i+1}, o que resulta no somatório do C_{exit} das $(i\text{-}1)$ tarefas.

Se o *'overhead'* fosse nulo, o tempo para a execução das n tarefas seria:

$$C_{total} = \sum_{i=1}^{n} C_i \qquad (6.4)$$

Considerando o *'overhead'* introduzido por cada uma das tarefas, temos:

$$C'_{total} = \sum_{i=1}^{n} \left(C_i + \left\lceil \frac{C_i}{T_{tic}} \right\rceil \times C_{timer} \right) + (n-1) \times C_{exit} \qquad (6.5)$$

O que implica que:

$$C_{total} \leq C'_{total} \qquad (6.6)$$

6.2.1.2 Análise quantitativa

Começamos a análise quantitativa pela determinação do parâmetro C_{exit}. Este tempo compreende a detecção do fim da tarefa actual, originada pelo *'trap'* feito ao núcleo, a retirada do topo da fila de execução a próxima

---

[15] ⌈ ⌉ - representa o menor inteiro maior ou igual ao valor. ⌊ ⌋ - representa o maior inteiro menor ou igual ao valor.

tarefa, a colocação da referência desta na estrutura tarefa activa, até à execução da primeira instrução desta nova tarefa.

Se considerarmos n tarefas vazias (não fazem absolutamente nada, unicamente terminam), o tempo que decorre desde que são postas em execução até que termine a última, a dividir pelo número de tarefas menos uma, fornece a medida que pretendemos.

Após alguns testes com 100, 200 e 220 tarefas (foi o número de tarefas próximo do máximo permitido pelo executivo), verificamos que o tempo que apareceram muitas das execuções efectuadas foram entre os 150 e os 160 μs, conforme se pode verificar pela Tabela 1. Assim, fazendo um arredondamento do tempo mais comum para a dezena de micro-segundo superior, obtemos o tempo de ≈ 160 μs.

Tabela 1 – Tempos do parâmetro *Cexit* fazendo várias execuções (tempos em μs)

100 tarefas	200 tarefas	220 tarefas
359,1	257,9	155,2
392,8	257,9	155,2
416,8	176,6	155,2
159,4	244,6	155,2
159,4	178,6	155,2
159,4	177,1	155,2
388,5	271,7	155,2
159,3	264,3	155,2

Uma outra forma de obtenção desta medida, se o sistema de medida tiver resolução suficiente, consiste em medir directamente o tempo, tirando uma primeira leitura do tempo na última instrução de uma tarefa e tirar uma segunda leitura na primeira instrução da outra tarefa. O tempo é obtido pela diferença entre a segunda e a primeira leituras.

Consideremos para o efeito duas tarefas. A primeira tarefa executa por um determinado tempo e lê o contador antes de terminar. A segunda tarefa, por

seu turno, lê o contador na entrada. É necessário garantir que a primeira tarefa execute na totalidade, antes da segunda tarefa iniciar.

O tempo que obtivemos com este processo foi de $\approx$ 163 μs, o que vem confirmar a primeira medida obtida. A medida do C_{exit} a utilizar nas experiências é, então, 163 μs.

A outra medida a obter é o C_{timer}. Esta pretende medir o tempo de tratamento da interrupção do *'timer'*, execução do escalonador e retorno à tarefa actual, uma vez que a interrupção, eventualmente, não provoca a mudança de contexto. Para isso, consideramos uma tarefa com o tempo de execução muito maior que o tempo entre duas interrupções consecutivas. O facto do tempo de execução da tarefa ser muito maior do que a resolução do *'timer'*, permite que a experiência não seja afectada pelo tratamento de uma interrupção no final de uma das tarefas, o que provoca um aumento significativo do tempo e, por conseguinte, deturpa os resultados.

Começamos por executar a tarefa sem o efeito do executivo, obtendo o tempo respectivo. Depois colocamos a tarefa em execução sobre o núcleo, havendo forçosamente a cada interrupção do *'timer'* uma interrupção do programa, a execução do código do escalonador - o qual normalmente actua de forma semelhante, independentemente do número de tarefas em execução - e o retorno à tarefa em execução, uma vez que só é uma. Este tempo também é medido. A diferença entre a segunda e a primeira medidas a dividir pelo número de interrupções ocorridas dá-nos o tempo pretendido:

$$C_{timer} = \frac{Tempo\ com\ Executivo\ -Tempo\ sem\ Executivo}{NumInt} \qquad (6.7)$$

O número de interrupções é dado pela fórmula seguinte:

$$NumInt = \left\lceil \frac{Tempo\ sem\ Executivo}{T_{tic}} \right\rceil \qquad (6.8)$$

O tempo C_{timer} obtido e que será utilizado a partir deste momento é $\approx$ 100 μs.

Tendo obtido os tempos dos *'overheads'* presentes nesta experiência, verificamos como é que esta se comporta.

Assim, se considerarmos oito tarefas, cujos tempos de execução são aproximadamente harmónicos do menor tempo de execução. Isto porque a carga de cada tarefa é expressa em número de iterações e, em termos de tempo, não existe correspondência exacta, como se pode constatar pela Tabela 2.

Tabela 2 – Tempos de execução de 8 tarefas harmónicas da menor

Nº Iterações	Tempo Execução (em ms)
32.000	199,0
64.000	397,9
128.000	753,7
256.000	1.465,5
512.000	2.888,8
1.024.000	5.735,7
2.048.000	11.429,3
4.096.000	22.858,5
C_{total} =	45.728,4

Se aplicarmos a equação (6.3) a cada uma das tarefas, o *'overhead'* total introduzido para o tratamento das interrupções é 83,7 *ms*. Por seu turno, o *'overhead'* introduzido pela mudança de tarefas é no total 1,141 *ms*.

O tempo total para a execução é, então, de:

$$C'_{total} = 45.728,4 + 83,7 + 1,141 = 45.813,2\ ms$$

O resultado obtido por simulação foi de 45.813,8 *ms*. Pelo que a

$$Diferença = 45.813,2 - 45.813,8 = -600\ \mu s.$$

Em termos relativos, a

$$Diferença_{rel} = Diferença \,/\, Tempo\ Simulação \times 100 = -0{,}001\ \%$$

O *'overhead'* efectivo introduzido pelo núcleo é assim de 85,4 *ms*, o que em termos relativos dá 0,19%. Este valor vem confirmar o que foi referido em relação à significância do *'overhead'* em sistemas de tempo-real, cujo escalonamento é efectuado *'off-line'* (*'pre-run-time scheduling'*), ou seja, é muito reduzido.

Apresentamos de seguida um quadro resumo para 8 tarefas, com várias cargas para o sistema:

Tabela 3 – Tempos de várias experiências com carga computacional diferente

Tempo Simulação (*ms*)			Tempo Calculado (*ms*)	Diferença Relativa
Sem Executivo	Com Executivo	Overhead Executivo		
1.554,9	1.573,9	1,22%	1.559,2	-0,93%
3.325,0	3.345,4	0,62%	3.332,7	-0,38%
6.030,0	6.051,8	0,36%	6.042,4	-0,16%
11.723,7	11.765,8	0,36%	11.746,4	-0,16%
45.728,4	45.813,8	0,19%	45.813,2	0,00%
91.204,7	91.372,1	0,18%	91.372,4	0,00%
157.379,4	157.678,1	0,19%	157.667,5	-0,01%
180.049,1	180.397,2	0,19%	180.378,5	-0,01%
182.161,1	182.491,0	0,18%	182.494,4	0,00%
364.114,5	364.789,7	0,19%	364.779,1	0,00%

Na tabela acima, verificamos que a previsibilidade do comportamento de sistemas de tempo-real depende da relação de grandeza das tarefas face ao *'overhead'* do Executivo. Isto é, como os *'overheads'* são fixos, a sua influência nos resultados das experiências é notória (o *'overhead'* do executivo pode ir de 1,22% para uma carga 1555 *ms* a 0,19% para 364.115 *ms*). A seguir, apresentamos um gráfico que permite verificar esta relação e como é que evolui.

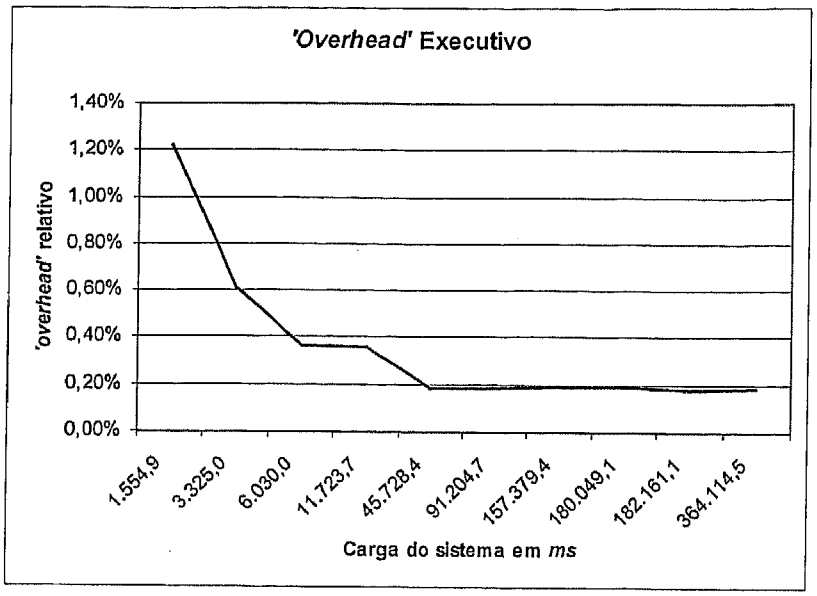

Fig. 9 – Evolução dos *'overheads'* do Executivo face à carga do sistema.

Nota-se aqui uma esbilização do *'overhead'* quando a influência deste no tempo de execução do conjunto é insignificante, isto acontece a partir dos 45 segundos de tempo de execução.

6.2.2 Política de escalonamento *'round-robin'*

Uma das políticas de escalonamento mais antigas, mais simples e mais utilizadas em sistemas tradicionais é o escalonamento *'round-robin'*. A cada tarefa é atribuído um determinado pedaço de tempo (*'time-slice'*[16]), normalmente idêntico para todas as tarefas, que é designado por *'quantum'* e durante o qual a tarefa pode executar (utilizar o processador). Se no fim do intervalo a tarefa ainda não terminou, esta é suspensa e outra tarefa é executada por igual pedaço de tempo. Se a tarefa terminar antes do fim do *'quantum'*, acontece uma mudança para a próxima tarefa. Isto é, as tarefas são executadas até ao fim, havendo mudança de tarefa sempre que se esgota

[16] Será utilizada a designação em Inglês de *'time-slice'* em vez da correspondente em Português de "fatia de tempo", por questões de conveniência da linguagem.

o tempo que lhe é atribuído ou, então, quando termina a execução, eventualmente, antes do fim do *'quantum'*.

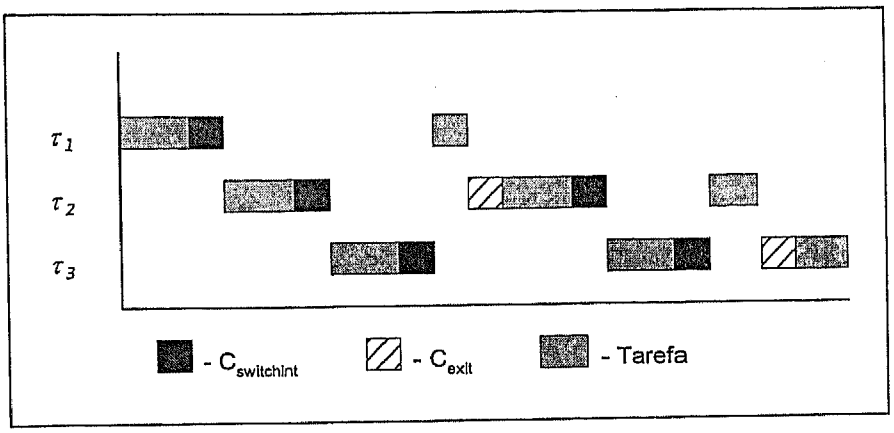

Fig. 10 – Escalonamento *'round-robin'* de três tarefas com os respectivos *'overheads'*

Como se pode verificar pela Fig. 10, quando o *'quantum'* atribuído a uma tarefa para executar esgota e ainda não terminou a tarefa, esta passa para o fim da lista, para retomar a execução quando chegar novamente a sua vez.

Esta política introduz dois tipos de *'overheads'*, como se pode constatar pela mesma Fig. 10. Um é relativo à mudança de tarefa quando é esgotado o *'quantum'* de uma tarefa, e outro relativo à detecção do fim de uma tarefa e respectiva mudança para a tarefa seguinte, o qual já foi devidamente quantificado no ponto anterior.

6.2.3 Análise qualitativa de escalonabilidade

Em todas as implementações, verifica-se uma sincronização temporal com o sistema de interrupções regulares (*'timer'*), o que faz com que o período (*'quantum'*) seja traduzido em termos de *'ticks'* (interrupções do *'timer'*). Isto é devido ao facto de, em todas as implementações que conhecemos as interrupções do *'timer'*, serem aproveitadas pelo núcleo para proceder às mudanças de tarefa.

O *'quantum'* nunca pode ser inferior à resolução do *'timer'*, fazendo com que se apresentem duas situações distintas: o *'quantum'* ser igual ao tempo entre dois *'ticks'* consecutivos (resolução do *'timer'*) ou, então, ser maior que o tempo entre dois *'ticks'* consecutivos. A tradução destas medidas, em termos do sistema convencional de medida de tempo, consegue-se com base na resolução em *ticks/seg.*, característica de qualquer *'timer'*.

6.2.3.1 Quando o *'quantum'* é igual à resolução do *'timer'*

Comecemos por efectuar a concretização dos *'overheads'*, os quais são resultantes da associação de um ou mais termos de medida.

O primeiro *'overhead'* a caracterizar é o tempo de mudança de contexto, o qual é dado pela seguinte expressão:

$$C_{switch} = C_{store} + C_{load} \qquad (6.9)$$

Como as mudanças de contexto são efectuadas em todos os *'ticks'* do *'timer'*, associada a esta medida temos o *'overhead'* introduzido pelo *'timer'* que é dado pela equação (6.2).

Assim, o *'overhead'* introduzido pela mudança de contexto é dado pela expressão seguinte:

$$C_{switchint} = C_{switch} + C_{timer} \qquad (6.10)$$

É preciso ver que esta medida é pessimista, uma vez que o elemento C_{resume} está a mais.

O segundo *'overhead'* é devido ao tempo de detecção de fim de uma tarefa e carregamento do contexto da tarefa seguinte que é dado pela fórmula (6.1).

Consideremos a resolução do timer como sendo T_{tic}. Se associarmos a cada tarefa interrompida o tempo de mudança de contexto ($C_{switchint}$) correspondente ao tempo de execução dessa tarefa, esta inclui os

'overheads' resultantes das mudanças de contexto em todos os *'ticks'* ocorridos no tempo de execução da tarefa. O *'overhead'* resultante é,

$$\left\lceil \frac{C_i}{T_{tic}} \right\rceil \times C_{switchint} \quad , \qquad (6.11)$$

o qual constitui o pior caso, uma vez que a tarefa pode não ter sido iniciada numa interrupção.

Assim, no pior caso, cada tarefa carece do seguinte tempo de execução:

$$\forall_{i,\ 1\le i\le n}, \quad C'_i = C_i + \left\lceil \frac{C_i}{T_{tic}} \right\rceil \times C_{switchint} \qquad (6.12)$$

Mais uma vez no total, o *'overhead'* devido ao C_{exit} é $(n\text{-}1)\times C_{exit}$.

O tempo total de execução do conjunto das tarefas é, assim:

$$C'_{total} = \sum_{i=1}^{n} \left(C_i + \left\lceil \frac{C_i}{T_{tic}} \right\rceil \times C_{switchint} \right) + (n-1) \times C_{exit} \qquad (6.13)$$

6.2.3.2 Quando o *'quantum'* é maior que a resolução do *'timer'*

Claramente existem nesta implementação três tipos de *'overheads'*, um correspondente à mudança de contexto quando o *'time-slice'* é esgotado, o segundo relativo ao tratamento da interrupção do *'timer'* quando não há lugar à mudança de tarefa e o último relativo ao despacho da tarefa seguinte quando termina a actual. Acontece, no entanto, que todos estes *'overheads'* já se encontram caracterizadas pelas fórmulas (6.9), (6.2) e (6.1), respectivamente.

Chamamos à atenção, no entanto, que nas associações referidas em todos os *'ticks'* do *'timer'* se verifica o *'overhead'* C_{timer}, mas só nos correspondentes ao *'time-slice'* se verifica a mudança de tarefa – C_{switch}.

Aqui consideraremos T_{tic} como sendo a resolução do *'timer'*, o que é equivalente à notação utilizada anteriormente, e T_{slice} o *'time-slice'* designado por *'quantum'*.

Seguindo o mesmo raciocínio do ponto anterior, são associados os *'overheads'* às tarefas respectivas, pelo que neste caso dá-se a mudança de contexto unicamente nos múltiplos do *'quantum'*, cujo *'overhead'* é:

$$\left\lceil \frac{C_i}{T_{slice}} \right\rceil \times C_{switch}, \qquad (6.14)$$

Existe, no entanto, outro *'overhead'* que corresponde ao tratamento da interrupção do *'timer'* e que é dado pela fórmula seguinte:

$$\left\lceil \frac{C_i}{T_{tic}} \right\rceil \times C_{timer} \qquad (6.15)$$

O tempo de execução para cada uma das tarefas é, no pior caso:

$$\forall_{i,1\le i\le n}, \quad C'_i = C_i + \left\lceil \frac{C_i}{T_{slice}} \right\rceil \times C_{switch} + \left\lceil \frac{C_i}{T_{tic}} \right\rceil \times C_{timer} \qquad (6.16)$$

E, como no caso anterior, o total do *'overhead' Cexit* é dado pelas (n-1) tarefas, pelo que a fórmula resultante é:

$$C'_{total} = \sum_{i=1}^{n} \left(C_i + \left\lceil \frac{C_i}{T_{slice}} \right\rceil \times C_{switch} + \left\lceil \frac{C_i}{T_{tic}} \right\rceil \times C_{timer} \right) + (n-1) \times C_{exit} \qquad (6.17)$$

6.2.4 Análise quantitativa

Neste caso, faremos a análise quantitativa do caso em que o *'quantum'* é igual à resolução do *'timer'*, já que o executivo CTask não implementa um algoritmo de escalonamento *'round-robin'* que permita definir um *'quantum'* maior do que o tempo de *'tick'*.

6.2.4.1 Resolução do *'timer'* constante

Começaremos esta análise pela obtenção da medida C_{switch}, que corresponde aos tempos de mudança de tarefa - guardar o contexto da tarefa actual e repor o contexto da tarefa seguinte). Isto é conseguido, provocando a mudança de contexto entre duas tarefas um determinado número de vezes.

Normalmente, qualquer núcleo possui uma instrução que permite interromper a tarefa actual e provocar a mudança de contexto. No entanto, é praticamente inevitável, mesmo que as tarefas tenham prioridades idênticas, que o escalonador seja chamado. Acontece, assim, que o tempo para além da mudança de contexto tenha o tempo de execução do escalonador. O tempo pretendido é, assim, menor ou igual do que o tempo realmente obtido, o que em termos de previsão do comportamento é correcto, uma vez que, se o conjunto é escalonável com o tempo considerado, com o efectivo é de certeza.

Consideremos, para o efeito, duas tarefas idênticas, constituídas por um ciclo que efectua a chamada à instrução referida um determinado número de vezes. Comecemos por medir o tempo que leva a executar os dois ciclos, sem a influência do executivo, o que constitui a primeira medida. Depois efectuamos outra medida, colocando as tarefas em execução sobre o executivo. A diferença entre os dois tempos a dividir pelo dobro do número de interacções do ciclo fornece a medida pretendida.

A medida que obtivemos para o parâmetro C_{switch} foi de $\approx 280\ \mu s$.

Considerando a equação (6.10), a medida $C_{switchint} \approx 280+100=380\ \mu s$.

Se considerarmos as mesmas oito tarefas, cujos tempos de execução foram apresentados na Tabela 2, e com a aplicação da equação (6.11) a cada uma delas, o *'overhead'* total devido às várias mudanças de contexto é de ≈ 318 *ms*.

O *'overhead'* introduzido pela mudança de tarefa é igual ao do exemplo anterior, isto é, 1,141 *ms*.

O tempo total para a execução é, então:

$$C'_{total} = 45.728,4 + 318 + 1,141 = 46.047,5\ ms$$

O resultado obtido por simulação foi 46.053,4 *ms*.

A *diferença* = 46.047,5 – 46.053,4 = -5,859 *ms*.

Por seu turno a

Diferençarel = -0,013%

O *'overhead'* efectivo introduzido pelo núcleo é neste caso de 325 *ms*, o que em termos relativos dá 0,71%. Como se pode verificar em relação ao caso anterior, existe um acréscimo no *'overhead'* do Executivo, o que é perfeitamente compreensível.

6.2.4.2 Variação da resolução do *'timer'*

Se considerarmos a resolução do *'timer'* de 27,5 *ms*, a previsão do comportamento do sistema é o total do *'overhead'* introduzido devido às mudanças de contexto, que é de ≈ 633,84.

Assim, o tempo total para a execução é de:

$$C'_{total} = 45.728,4 + 633,84 + 1,141 = 46.363,4\ ms$$

O resultado da simulação é 46.344,7 *ms*.

Diferença = 46.363,4 – 46.344,7 = 18,7 ms

Diferençarel = 0,04%

O *'overhead'* efectivo introduzido pelo Executivo, neste caso, foi de 616,3 *ms*, que em termos percentuais dá 1,35%.

As causas do aumento do *'overhead'*, que fazem aumentar o tempo total das experiências, poderão verificar-se devido à resolução do *'timer'* que pode fazer a diferença. Digamos que numa tarefa com tempo de execução fixo o *'overhead'* pode ser de menos de 1% para a resolução normal do *'timer'* (≈ 55 *ms*) e atingir mais de 10% para resolução mínima de 3,4 *ms*.

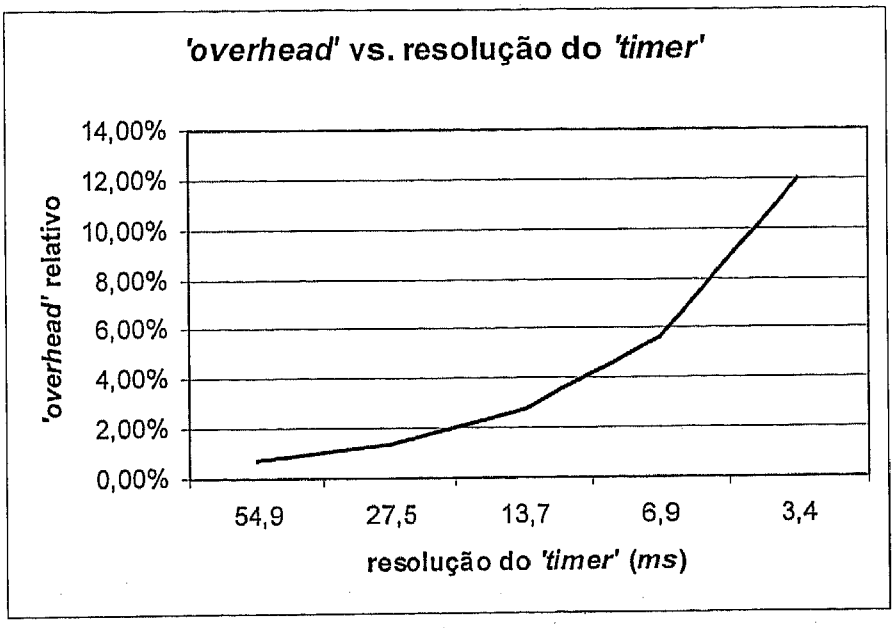

Fig. 11 – Evolução do *'overhead'* em função da resolução do *'timer'*.

Este *'overhead'* é resultante unicamente do tratamento das interrupções, uma vez que todos os outros foram considerados anteriormente.

Tabela 4 – Resultados com resolução do *'timer'* de 55 *ms*

Tempo Simulação (*ms*)			Tempo Calculado (*ms*)	Diferença Relativa
Sem o Executivo	Com o Executivo	Overhead Executivo		
1.554,9	1.585,3	1,96%	1.567,8	-1,11%
3.324,9	3.354,5	0,89%	3.351,0	-0,10%
6.029,7	6.082,1	0,87%	6.073,0	-0,15%
11.723,1	11.822,7	0,85%	11.805,7	-0,14%
45.727,7	46.053,4	0,71%	46.044,4	-0,02%
91.203,8	91.827,8	0,68%	91.832,6	0,01%
157.376,9	158.460,8	0,69%	158.460,0	0,00%
180.048,2	181.277,4	0,68%	181.287,0	0,01%
182.153,9	183.380,5	0,67%	183.407,1	0,01%
364.091,4	366.551,5	0,68%	366.593,9	0,01%

Apresentamos de seguida as várias tabelas resumo dos tempos de execução (carga do sistema), *'overhead'* do executivo, tempo calculado com o modelo deduzido e diferença relativa, para que as várias resoluções do *'timer'* do sistema.

Tabela 5 – Resultados com resolução do *'timer'* de 27,5 *ms*

Tempo Simulação (*ms*)			Tempo Calculado (*ms*)	Diferença Relativa
Sem o Executivo	Com o Executivo	Overhead Executivo		
1.554,9	1.577,6	1,46%	1.578,7	0,07%
3.325,1	3.391,6	2,00%	3.373,3	-0,54%
6.029,7	6.131,2	1,68%	6.115,2	-0,26%
11.723,0	11.903,3	1,54%	11.887,0	-0,14%
45.726,5	46.344,7	1,35%	46.356,1	0,02%
91.200,2	92.452,2	1,37%	92.453,0	0,00%
157.372,5	159.527,2	1,37%	159.532,7	0,00%
180.047,1	182.515,2	1,37%	182.517,6	0,00%
182.148,1	184.624,8	1,36%	184.647,6	0,01%
364.085,1	369.011,4	1,35%	369.078,8	0,02%

Tabela 6 – Resultados com resolução do *'timer'* de 13,7 *ms*

Tempo Simulação (*ms*)			Tempo Calculado (*ms*)	Diferença Relativa
Sem o Executivo	Com o Executivo	Overhead Executivo		
1.554,9	1.620,8	4,23%	1.600,2	-1,27%
3.325,0	3.436,0	3,34%	3.418,9	-0,50%
6.029,7	6.201,4	2,85%	6.196,7	-0,08%
11.723,0	12.075,4	3,01%	12.046,9	-0,24%
45.725,8	46.994,7	2,77%	46.986,2	-0,02%
91.199,3	93.704,6	2,75%	93.710,9	0,01%
157.370,8	161.741,8	2,78%	161.702,9	-0,02%
180.045,3	185.049,8	2,78%	185.002,1	-0,03%
182.145,5	187.140,6	2,74%	187.160,0	0,01%
364.080,8	374.058,3	2,74%	374.101,8	0,01%

Tabela 7 – Resultados com resolução do *'timer'* de 6,9 *ms*

Tempo Simulação (*ms*)			Tempo Calculado (*ms*)	Diferença Relativa
Sem o Executivo	Com o Executivo	Overhead Executivo		
1.554,9	1.645,8	5,84%	1.642,4	-0,21%
3.325,0	3.528,0	6,11%	3.509,0	-0,54%
6.029,7	6.398,0	6,11%	6.362,6	-0,55%
11.723,1	12.428,5	6,02%	12.366,6	-0,50%
45.726,4	48.300,5	5,63%	48.227,4	-0,15%
91.200,4	96.354,7	5,65%	96.186,2	-0,17%
157.372,1	166.338,3	5,70%	165.973,0	-0,22%
180.046,1	190.328,6	5,71%	189.886,2	-0,23%
182.148,7	192.413,1	5,64%	192.103,8	-0,16%
364.087,4	384.576,8	5,63%	383.983,2	-0,15%

Tabela 8 – Resultados com resolução do *'timer'* de 3,4 *ms*

Tempo Simulação (*ms*)			Tempo Calculado (*ms*)	Diferença Relativa
Sem o Executivo	Com o Executivo	Overhead Executivo		
1.554,9	1.752,5	12,71%	1.729,9	-1,29%
3.325,1	3.749,1	12,75%	3.696,8	-1,40%
6.030,0	6.791,6	12,63%	6.700,7	-1,34%
11.723,5	13.173,8	12,37%	13.024,5	-1,13%
45.727,0	51.199,3	11,97%	50.799,2	-0,78%
91.201,7	102.083,7	11,93%	101.316,3	-0,75%
157.374,5	176.388,1	12,08%	174.826,6	-0,89%
180.048,1	201.835,2	12,10%	200.014,5	-0,90%
182.152,2	203.885,4	11,93%	202.351,5	-0,75%
364.095,9	407.507,2	11,92%	404.469,9	-0,75%

Fazendo uma análise global as tabelas 4, 5, 6, 7 e 8 apresentadas, verificamos que os resultados obtidos pela aplicação do modelo de escalonamento desenvolvido satisfazem, uma vez que a diferença relativa nunca ultrapasse o ±1%, 5%. Os piores resultados são encontrados na tabela 8, mas são perfeitamente explicáveis face ao *'overhead'* do executivo. Isto é, o executivo passa muito tempo nas operações de mudança de contexto (≈12%).

6.3 Estudo do caso preemptivo

Uma política de escalonamento é preemptiva, se uma tarefa de prioridade mais baixa é interrompida para que uma tarefa de prioridade superior execute. Assim, no caso preemptivo existe forçosamente o conceito de prioridade de tarefa, fazendo com que qualquer política de escalonamento assegure que as tarefas mais prioritárias executem primeiro do que as menos prioritárias. A variação entre algoritmos de escalonamento está na forma de atribuição e tipo de prioridades (fixas ou dinâmicas).

Devido ao aumento de complexidade dos algoritmos de escalonamento, existe também um aumento no número de restrições contabilizáveis, a ter em atenção no escalonamento de tarefas. Estas prendem-se, essencialmente, com a preemptividade.

A característica comum ao escalonamento preemptivo baseia-se na execução da tarefa mais prioritária das prontas a executar em primeiro lugar.

Todo o estudo a efectuar neste trabalho é sobre o escalonamento preemptivo com atribuições de prioridades fixas, isto porque a maioria dos sistemas de tempo-real que são desenvolvidos hoje em dia são baseados em prioridades fixas, precisamente por serem mais previsíveis. Daí que este estudo seja dirigido aos Projectistas destes sistemas de tempo-real, que continuam a ver os seus projectos falharem devido a questões práticas.

6.3.1 Definição do modelo

O modelo é constituído por n tarefas periódicas independentes, com prioridades diferentes. Cada tarefa τ_i é caracterizada pelo tempo de execução (C_i), prioridade (P_i), *'deadline'* (D_i) e o período (T_i).

Existem alguns pressupostos sobre o ambiente para que seja possível obter resultados analíticos do comportamento de um sistema num ambiente de tempo-real, eventualmente crítico [LL73]:

- Os pedidos de execução para todas as tarefas com deadlines críticas devem ser periódicos, com intervalos constantes entre pedidos;
- Cada tarefa deve terminar a execução antes do próximo pedido de execução;
- Não existe qualquer relação de precedência ou de exclusão mútua entre as tarefas, isto é, a execução de uma determinada tarefa não depende de outra tarefa;
- O tempo de execução para cada tarefa é constante e não varia no tempo.

Existem alguns conceitos importantes no âmbito do escalonamento com atribuição de prioridades fixas. Assim:

- Para um determinado conjunto de tarefas, um algoritmo de escalonamento é válido, se nenhuma tarefa perder a sua *'deadline'* (em tempo algum).
- O tempo de resposta de um pedido de execução de uma tarefa é o tempo decorrido entre o pedido de execução e o término da mesma.
- O instante crítico para uma tarefa, τ_i é o instante no qual um pedido de execução da tarefa tem o maior tempo de resposta. Isto é, corresponde ao instante em que a tarefa τ_i pede execução em simultâneo com todas as tarefas de maior prioridade $(\tau_{i-1}, \tau_{i-2}, \ldots, \tau_1)$. Assim, se cada tarefa cumpre a sua primeira *'deadline'* quando pede execução no mesmo momento de todas as outras, então as *'deadlines'* continuarão a ser cumpridas [SG90].

- A zona crítica para uma tarefa é o intervalo de tempo entre o instante crítico e o limite do *fim da resposta* para o pedido de execução da tarefa, que torna o escalonamento válido.

6.3.2 Escalonamento 'Rate Monotonic'

O algoritmo *'rate monotonic'* efectua a atribuição de prioridades às tarefas de acordo com o número de pedidos de execução e independentemente dos tempos de execução. Assim, tarefas com um número maior de pedidos de execução deverá ter maior prioridade e, consequentemente, o período mais pequeno.

Assim, para determinar se uma tarefa pode cumprir a sua *'deadline'*, quando iniciada no momento crítico, é necessário considerarmos os pedidos de execução feitos pelo conjunto de tarefas em função do tempo [LSD89]. A expressão

$$W_i(t) = \sum_{j=1}^{i} C_j \cdot \left\lceil \frac{t}{T_j} \right\rceil, \qquad (6.18)$$

fornece os pedidos comulativos ao processador feitos pelas *i* tarefas durante o período *[0, t]*. Considerando estes pedidos em função do tempo, temos

$$L_i(t) = W_i(t)/t. \qquad (6.19)$$

A tarefa τ_i só é escalonável, utilizando o algoritmo RM se,

$$\forall_{i,1\le i\le n},\ min_{\{0\le t\le T_i\}}\ L_i(t) \le 1. \qquad (6.20)$$

A função $L_i(t)$ é monotonamente decrescente por períodos: $\lceil t/T_i \rceil / t$ é estritamente decrescente excepto num conjunto finito de valores chamados

pontos de escalonamento. Quando t é múltiplo de um dos T_j, com $1 \leq j \leq i$, $L_i(t)$ encontra-se num mínimo local.

Considerando S_i como sendo o conjunto dos pontos de escalonamento para a tarefa τ_i, esta traduz-se como

$$S_i = \left\{ K \cdot T_j \Big|_{j=1,\ldots,i;K=1,\ldots,\left\lfloor T_i / T_j \right\rfloor} \right\} \qquad (6.21)$$

Resumindo, numa única expressão tudo o que foi apresentado, temos que a tarefa τ_i só é escalonável, utilizando o algoritmo RM se

$$\forall_{i,\ 1 \leq i \leq n} \quad \min_{\{t \in S_i\}} \sum_{j=1}^{i} \frac{C_j}{t} \cdot \left\lceil \frac{t}{T_j} \right\rceil \leq 1, \qquad (6.22)$$

Na equação em cima são analisadas cada tarefa τ_i pelo seu período, até à *'deadline'* ($D_i=T_i$). A soma da carga é avaliada em cada ponto de escalonamento. Se o valor mínimo da carga normalizado pelo tempo é menor ou igual que a unidade, então a tarefa é escalonável, constituindo a condição necessária e suficiente de escalonabilidade.

6.3.3 Análise qualitativa da escalonabilidade

Como já foi visto na secção anterior, a verificação da escalonabilidade de um conjunto de tarefas, com base na condição necessária e suficiente, é feita partindo do princípio que a preempção e consequente mudança de tarefas são instantâneas e que a tarefa está pronta a executar no início do período.

Em termos práticos não é o que se verifica, uma vez que existem várias restrições introduzidas pelo núcleo – *'overheads'* - quando este se encontra a efectuar as operações referidas.

Os dois tipos de implementações dos escalonadores dos núcleos – orientada por eventos e por tempo - consideram as restrições de forma particular. É assim normal que a associação de termos, que constituem os *'overheads'* e os modelos matemáticos que permitem a verificação da escalonabilidade (previsão do comportamento), sejam diferentes. Assim, iremos aqui analisar os dois tipos de implementações.

6.3.3.1 Escalonamento orientado por eventos

Neste tipo de implementação, todas as tarefas são iniciadas por eventos internos ou externos. No início do período da tarefa, quando esta pede execução, é gerada uma interrupção que será detectada pelo processador.

A tarefa actualmente em execução é interrompida, de forma a ser verificado se a tarefa que solicitou execução é mais prioritária ou não. Se for mais prioritária, então dá-se a mudança para a nova tarefa; se não, o controlo é devolvido à tarefa em execução. Isto pode ser verificado pela Fig. 12.

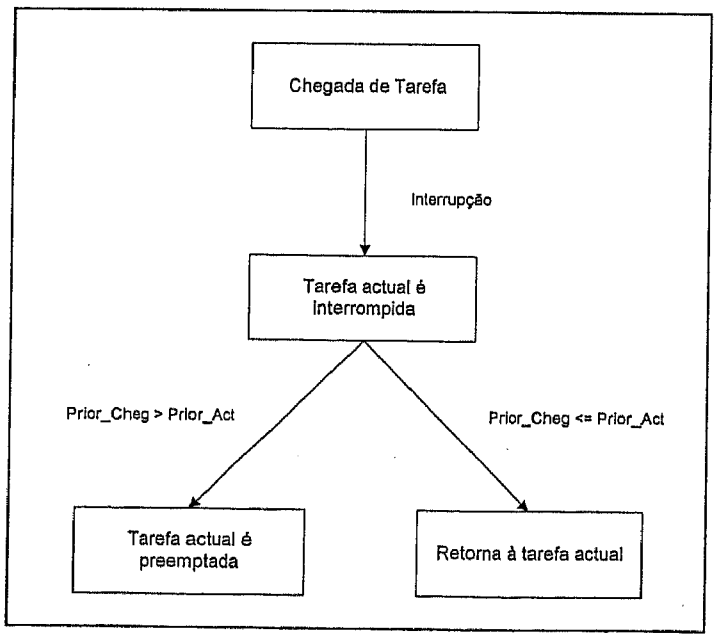

Fig. 12 – Diagrama de estados da implementação orientada por eventos.

Assim, os núcleos com este tipo de implementação do escalonador têm vários *'overheads'* resultantes de determinadas associações de termos.

Em primeiro lugar, temos o *'overhead'* introduzido pela necessidade de preempção de uma tarefa de prioridade mais baixa, quando uma tarefa de prioridade mais alta pede execução:

$$C_{preempt} = C_{int} + C_{sched} + C_{store} + C_{load} \qquad (6.23)$$

O segundo *'overhead'* é introduzido pelo tempo resultante do tratamento do pedido de execução de uma tarefa de prioridade mais baixa:

$$C_{nonpreempt} = C_{int} + C_{sched} + C_{resume} \qquad (6.24)$$

Se considerarmos que $C_{store}+C_{load}>C_{resume}$, então $C_{preempt}>C_{nonpreempt}$.

O último *'overhead'* prende-se com a necessidade de mudança de tarefa quando a actual termina. Este é dado pela fórmula (6.1):

$$C_{exit} = C_{trap} + C_{load} \qquad (6.25)$$

Salientamos que neste caso, o termo C_{trap} já engloba o tempo de inserção ordenada do TCB da tarefa que terminou na fila de espera.

O primeiro *'overhead'* associado com esta implementação é constante e está ligado à tarefa que provoca a preempção. No pior caso, o *'overhead'* para cada tarefa periódica com prioridade fixa é $C_{preempt}$.

Isto prende-se com o facto de que, se uma tarefa, digamos τ_i, solicitar execução e outra tarefa, τ_j, menos prioritária estiver em execução, o escalonador do núcleo deverá zelar para que a segunda seja preemptada e para que a primeira execute. A associação do *'overhead'* à primeira tarefa, tem a ver com o facto da necessidade do escalonamento das tarefas mais

prioritárias em detrimento das menos prioritárias. Isto é, no pior caso, cada vez que a tarefa τ_i pede execução, eventualmente, preempta uma tarefa de prioridade mais baixa.

Acontece, no entanto, que o pedido de execução de uma tarefa menos prioritária provoca a interrupção da tarefa actual, o qual resulta no segundo *'overhead'* considerado.

Assim, para um algoritmo de prioridades fixas e para uma tarefa τ_i, num intervalo t, devido à chegada de tarefas de prioridade idêntica ou inferior num conjunto de n tarefas, o tempo total do *'overhead'* é dado, no pior caso, pela seguinte fórmula:

$$\sum_{j=i+1}^{n} \left\lceil \frac{t}{T_j} \right\rceil \times C_{nonpreempt} \qquad (6.26)$$

Isto é, cada vez que uma tarefa de prioridade igual ou inferior à actual pede execução, introduz um *'overhead'* fixo de tempo $C_{nonpreempt}$. Se considerarmos um intervalo t, cada tarefa menos prioritária τ_j pode pedir execução, no pior caso, $\lceil t/T_j \rceil$ vezes. No caso do escalonamento *'rate monotonic'*, o período de cada tarefa menos prioritária tem que ser maior que T_i ($T_i \leq T_{i+1} \leq \ldots \leq T_n$), o período de τ_i. Assim, num intervalo $t \leq T_i$, o número de interrupções reduz para

$$(n-i) \times C_{nonpreempt}. \qquad (6.27)$$

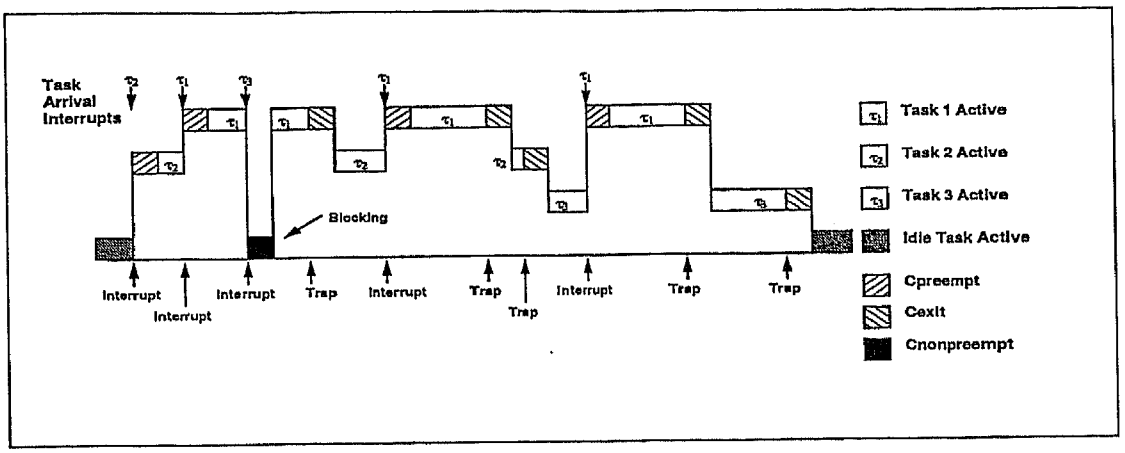

Fig. 13 – Escalonamento orientado por eventos [KAS93].

Como já foi dito para o caso não preemptivo, o *'overhead'* C_{exist} não provoca a perda de *'deadline'* para a tarefa que terminou, pelo que deverá ser tido em conta para a tarefa τ_i as vezes que as tarefas $\tau_1, \ldots, \tau_{i-1}$ terminaram. Assim, num período t e para a tarefa τ_i, *'overhead'* introduzido pelo parâmetro C_{exit} é, no total,

$$\sum_{j=1}^{i-1} \left\lfloor \frac{t}{T_j} \right\rfloor \times C_{exit} \qquad (6.28)$$

Tendo por base a condição necessária e suficiente apresentada pela expressão (6.20) e tendo em atenção que os factores deduzidos influenciam claramente a escalonabilidade do conjunto, esta expressão passa a ter o seguinte aspecto:

$$\forall_{i,1\le i\le n},\ min_{\{t\in S_i\}} \left(\sum_{j=1}^{i} \frac{C_j + C_{preempt}}{t} \times \left\lceil \frac{t}{T_j} \right\rceil \right) + \frac{\sum_{j=i+1}^{n} \left\lceil \frac{t}{T_j} \right\rceil \times C_{nonpreempt}}{t} + \frac{\sum_{j=1}^{i-1} \left\lfloor \frac{t}{T_j} \right\rfloor \times C_{exit}}{t} \le 1 \qquad (6.29)$$

Esta expressão permite testar a escalonabilidade de um conjunto de tarefas $\tau_1, \ldots, \tau_n$, sujeito às restrições impostas pelo suporte prático. Esta expressão apresenta três componentes perfeitamente distintos. O primeiro é o overhead introduzido pelo aparecimento de tarefas mais prioritárias e que na fórmula aparece associado a cada uma das tarefas. O segundo tem a ver com o aparecimento de tarefas menos prioritárias, pelo que para o

escalonamento das tarefas $\tau_1,\ldots,\tau_i$ entra em linha de conta com o *'overhead'* das tarefas $\tau_{i+1},\ldots,\tau_n$. O terceiro tem a ver com o tratamento do fim das várias tarefas. Neste caso para a tarefa τ_i, todas as tarefas $\tau_1,\ldots,\tau_{i-1}$, terminaram um determinado número de vezes.

É possível saber a carga do sistema para o conjunto das tarefas $\tau_1,\ldots,\tau_i$, com $1 \leq i \leq n$. Isto é conseguido através da expressão que fornece os pedidos comulativos no período *[0,t]*, entrando em linha de conta com os vários *'overheads'* descritos:

$$W_i(t) = \sum_{j=1}^{i}\left(C_j + C_{preempt}\right)\times\left\lceil\frac{t}{T_j}\right\rceil + \sum_{j=i+1}^{n}\left\lceil\frac{t}{T_j}\right\rceil\times C_{non\ preempt} + \sum_{j=1}^{i-1}\left\lfloor\frac{t}{T_j}\right\rfloor\times C_{exit} \qquad (6.30)$$

6.3.3.2 Escalonamento orientado por tempo

Este tipo de implementação é a mais usual e utiliza as interrupções geradas por um *'timer'* periódico para interromper a execução actual e invocar o escalonador. É assim assumido que o *'timer'* expira todos os T_{tic} segundos, forçando um ponto de decisão de escalonamento. Como qualquer outra interrupção, também esta é manipulada por uma rotina de serviço a interrupções que entre outras operações chama o escalonador.

Relativamente ao escalonador, podemos definir os seguintes pressupostos:

- o escalonador move, eventualmente, todas as tarefas cujas *'deadlines'*, sejam maiores ou iguais que o tempo actual, da fila de tarefas suspensas (fila de espera) para a fila de prontas a executar (fila de execução); então, decide se deve ou não preemptar a tarefa activa, baseado na prioridade da tarefa no topo da fila de execução – tarefas em novo período de execução;

- quando uma tarefa termina, o escalonador é chamado imediatamente, o qual coloca em execução a próxima tarefa na fila de execução.

Assim, é compreensível que exista um *'overhead'* associado à manipulação das interrupções do *'timer'* e execução do código do escalonador que é designado como o *'overhead'* do *'timer'*. É assumido que este é constante para todas as interrupções, independentemente de haver (ou não) lugar a preempção. Neste caso, o primeiro *'overhead'* a caracterizar é o resultante do tratamento da interrupção do *'timer'*, execução do código do escalonador e retorno à tarefa actual, que já foi descrito no ponto 6.2.1.1. e cuja associação de termos é:

$$C_{timer} = C_{\text{int}} + C_{sched} + C_{resume} \qquad (6.31)$$

Desta forma, o *'overhead'* introduzido pela necessidade de preempção da tarefa actual, reduz-se a guardar o contexto da tarefa actual e repor o contexto tarefa seguinte:

$$C_{preempt} = C_{store} + C_{load} \qquad (6.32)$$

Interessa salientar que este tempo é idêntico ao C_{switch} caracterizado anteriormente.

O último *'overhead'* corresponde ao tempo C_{exit} que também já foi descrito no ponto anterior e cuja associação de termos é dada pela fórmula (6.1).

Pelas mesmas razões apresentadas no ponto anterior, o overhead associado a cada tarefa de prioridade fixa é, também, no mecanismo de escalonamento orientado por tempo, $C_{preempt}$.

Em relação ao *'overhead'* do *'timer'*, se considerarmos um intervalo t e resolução do *'timer'* T_{tic}, este é dado pela expressão seguinte:

$$\left\lceil \frac{t}{T_{tic}} \right\rceil \times C_{timer}. \qquad (6.33)$$

Acontece que neste caso aparece o fenómeno de inversão de prioridade, uma vez que uma tarefa mais prioritária pode ser bloqueada por uma outra de prioridade inferior até à próxima interrupção do timer. Assim, aparece um factor de bloqueio que, no pior caso, é T_{tic}. Isto é, se o pedido de execução da tarefa mais prioritária for feito próximo do início do período, o tempo de bloqueio é maximizado para T_{tic}, pelo que o factor de bloqueio é T_{tic} no pior caso.

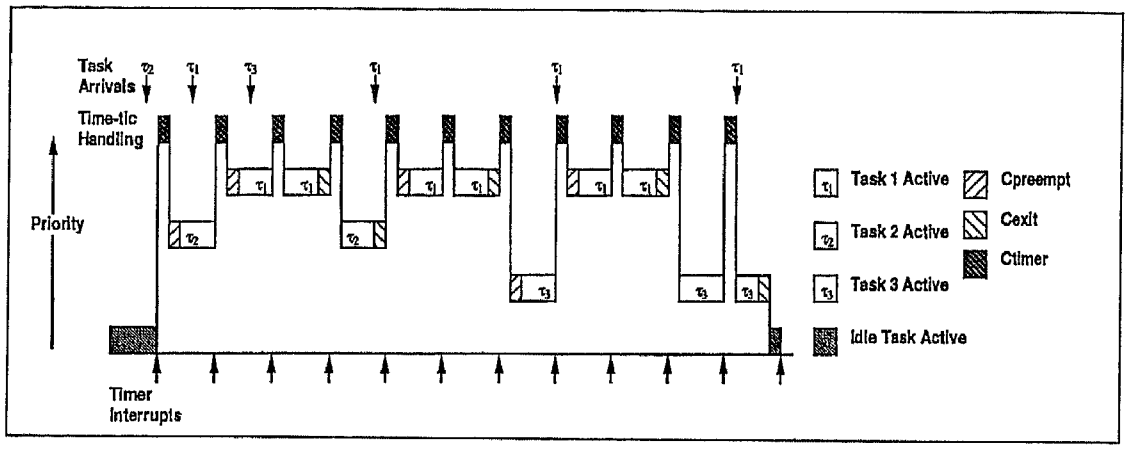

Fig. 14 – Escalonamento orientado por tempo [KAS93].

Em relação ao *'overhead'* introduzido pela medida C_{exit}, este é idêntico ao apresentado no caso orientado por eventos.

Tendo em atenção todas as restrições descritas, a condição necessária e suficiente para testar a escalonabilidade desta aproximação tem a seguinte forma:

$$\forall_{i,\ 1\le i\le n},\ min_{\{t\in S_i\}} \sum_{j=1}^{i}\left(\frac{C_j + C_{preempt}}{t}\times\left\lceil\frac{t}{T_j}\right\rceil\right)+\left\lceil\frac{t}{T_{tic}}\right\rceil\times\frac{C_{timer}}{t}+\frac{T_{tic}}{t}+\frac{\sum_{j=1}^{i-1}\left\lfloor\frac{t}{T_j}\right\rfloor\times C_{exit}}{t}\le 1 \qquad (6.34)$$

Da mesma forma do caso precedente, é possível saber a carga computacional acumulada do sistema para toda a tarefa τ_i, com $1 \leq i \leq n$ no momento t, modificando a expressão dos tempos acumulados (6.18), que passa a ter o aspecto seguinte:

$$W_i(t) = \sum_{j=1}^{i} \left(C_j + C_{preempt}\right) \times \left\lceil \frac{t}{T_j} \right\rceil + \left\lceil \frac{t}{T_{tic}} \right\rceil \times C_{timer} + T_{tic} + \sum_{j=1}^{i-1} \left\lfloor \frac{t}{T_j} \right\rfloor \times C_{exit} \qquad (6.35)$$

6.3.4 Análise quantitativa

Iremos aqui quantificar o caso orientado por tempo. Em relação ao caso orientado por eventos, a sua quantificação exige a utilização de eventos externos, ou eventualmente, a simulação destes, o que pensamos neste momento não ser essencial, uma vez que o caso orientado por tempo corresponde directamente ao ambiente experimental considerado, permitindo assim testar a validade do modelo matemático deduzido. Por outro lado, este caso é mais complexo do que o orientado por eventos.

Comecemos por obter o tempo $C_{preempt}$. Para o efeito, consideremos duas tarefas de prioridades diferentes, as quais iniciaram cada uma delas um ciclo suficientemente longo. A tarefa de prioridade mais baixa para além do ciclo referido encadeia outro que tem a duração de aproximadamente um *'tick'* do *'timer'*, e que permite aguardar pela preempção da outra tarefa. A tarefa mais prioritária, por seu turno, suspende por um *'tick'*, para que o núcleo coloque em execução a outra tarefa. Quando o período de suspensão expira, esta tarefa mais prioritária preempta a outra.

Como acontece com outras medidas, começamos por executar os ciclos que constituem as tarefas, em série e sem influência do executivo, de forma a obter o tempo de base. Em seguida, colocamos as duas tarefas sobre o executivo e obtemos o segundo tempo.

A medida em questão é obtida pela diferença entre os dois tempos, dividida pelo número de iterações.

Feitas várias experiências, o tempo obtido para o $C_{preempt}$ foi de $\approx 280\ \mu s$.

Em relação à medida C_{timer}, como o executivo é o mesmo das experiências não preemptivas, poderemos considerar o mesmo valor. Assim, C_{timer} é $\approx 100\ \mu s$.

A medida C_{exit}, como já foi referido, deverá ser maior ou igual à considerada para o caso não preemptivo, isto porque neste caso inclui o tempo de inserção do TCB da tarefa terminada na fila de tarefas em espera, mas de forma ordenada. O tempo superior é devido à inserção ordenada.

A forma mais simples de obter esta medida, condicionada pela precisão do sistema de medida, é fazendo a medida do valor directamente. Para o efeito consideremos duas tarefas com prioridades diferentes. A mais prioritária lê o primeiro tempo e suspende. A segunda e menos prioritária lê o tempo na primeira instrução. A medida em questão é obtida pela diferença entre o primeiro e o segundo tempos.

O tempo obtido foi 300 μs.

6.3.4.1 Experiências com a resolução formal do *'timer'* (55*ms*)

Consideremos três tarefas periódicas independentes, cujas características são dadas pela Tabela 9:

Tabela 9 – Características do conjunto de tarefas

Tarefa	T. Execução (*ms*)	Período (*ms*)	U_i
1	209,4	549,5	0,381
2	234,6	879,1	0,267
3	548,9	2.087,9	0,263
		U=	0,911

O limite segundo Liu e Layland [LL73] é

$$U = 3(2^{1/3}-1) = 0,78$$

Considerando a condição suficiente o conjunto não é escalonável uma vez que 0,911>0,78.

Se considerarmos a condição necessária e suficiente o conjunto já é escalonável.

Quando i=3 e t=1648 *ms*, o conjunto é escalonável pela equação (6.18).

$W_3(1648) = 1646,2 < 1648$ *ms*.

Não sendo escalonável por nenhuma outra condição.

Contudo, se considerarmos os *'overheads'*:

$C_{preempt} = 0,28$ *ms*

$C_{timer} = 0,123$ *ms*

$C_{exit} = 0,600$ *ms*

$T_{tic} = 54,9$ *ms*

O conjunto deixa de ser escalonável ao aplicarmos a equação (6.35) para i=3 e t=1648 *ms*.

$W_3(1648) = 1653,9>1648$ *ms*.

Com este exemplo verifica-se que os *'overheads'* introduzidos pelo suporte prático podem fazer a diferença.

O resultado da simulação é o que se apresenta na Tabela 10.

Tabela 10 – Simulação de 3 tarefas independentes com resolução do *'timer'* ≈ 55 *ms*

Tarefa	N Iter	Teste Base	Escalonamento Rate Monotonic			U_i
		T Exec(us)	N Exec	T Resp(us)	T_i Ef (us)	
1	40.500	209.376,0	4	211.038,0	549.450,5	0,381
2	45.714	234.550,5	3	448.631,0	879.120,8	0,267
3	115.432	548.936,5	1	2.113.655,0	2.087.912,0	0,263
Perdeu a *'deadline'* na tarefa 3					U=	0,911

Pela tabela verifica-se que a última tarefa perdeu a *'deadline'*, uma vez que o tempo de resposta foi de 2113,7 *ms* o que é francamente superior ao período da última tarefa (2087,9 *ms*).

A Tabela 11 que se segue apresenta os resultados dos tempos de resposta obtidos para este exemplo, considerando e não considerando o factor de bloqueio T_{tic}.

Tabela 11 – Tempos de resposta

Simulação (*ms*)	Calculo s/ bloqueio		Calculo c/ bloqueio	
	Tempo (*ms*)	Diferença relativa	Tempo (*ms*)	Diferença relativa
211,0	210,9	0,0%	265,8	26,0%
448,6	447,5	-0,2%	502,4	12,0%
2.113,7	2.097,5	-0,8%	2.152,4	1,8%

Os resultados obtidos são claramente satisfatórios, principalmente se considerarmos o factor de bloqueio T_{tic}.

Capítulo 7
Conclusão

No início desta dissertação propunhamo-nos contribuir para uma melhor compreensão das vicissitudes dos sistemas de tempo-real, tanto a nível de concepção, como a nível de validação. Isto passava, concretamente, pela compreensão das distorções introduzidas pelo suporte prático de sistemas de tempo-real, com especial relevo para a sobrecarga computacional introduzida. Esta compreensão revelava-se importante, no sentido em que permitiria quantificar diversas restrições temporais a que os sistemas de tempo-real estão efectivamente sujeitas e, a partir daí, desenvolver metodologias que permitissem avaliar de forma efectiva o desempenho de um sistema qualquer.

Por outro lado, e para que fosse possível entrar em linha de conta com os condicionalismos temporais identificados, seria necessário quantificá-los. Daí que traçássemos como um outro objectivo desta dissertação encontrar formas de os deduzir num determinado contexto. Para isso, seria necessário encontrar técnicas que nos permitissem obter estes condicionalismos temporais, com o nível de confiança necessário às aplicações de tempo-real.

Neste contexto, começámos por estudar os aspectos teóricos e práticos em torno dos projectos de sistemas de tempo-real, com a finalidade de os conjugar no âmbito do desenvolvimento de sistemas de tempo-real.

Nos aspectos teóricos, fizemos um levantamento sobre os algoritmos de escalonamento. Aqui distinguimos algoritmos de escalonamento estáticos e dinâmicos e, dentro dos dinâmicos, os de prioridades fixas e os totalmente dinâmicos. Em seguida, apresentámos as várias soluções em termos de algoritmos para o escalonamento de tarefas periódicas independentes, aperiódicas, dependentes e em sistemas multiprocessador.

Nos aspectos práticos, caracterizámos os vários suportes, em termos da sua aptidão para sistemas de tempo-real. Começámos pelos sistemas operativos, onde definimos as características e evidenciámos os vários factores que permitem optar por um deles, para o desenvolvimento de STRs. Continuámos com a apresentação de uma classificação e respectivas características para o agrupamento dos vários SO existentes: núcleos pequenos, extensões de SO tradicionais e sistemas resultantes de investigação.

Foram apresentadas, também, as características que as linguagens de programação e o *'hardware'* devem possuir para serem considerados em conformidade com os sistemas de tempo-real.

Identificadas as possibilidades em termos de algoritmos de escalonamento e sistemas operativos de tempo-real, era necessário identificar formas de proceder ao teste de possíveis soluções para a implementação de sistemas de tempo-real. Procedemos, assim, a uma pesquisa pela literatura da especialidade, com o intuito de identificar *'benchmarks'* que permitissem fazer a avaliação de suportes de sistemas de tempo-real. Identificámos vários testes possíveis de serem utilizados nos níveis e estádios de desenvolvimento de uma aplicação. Deste modo, identificámos um método de análise de escalonabilidade a ser utilizado, preferencialmente, na fase de projecto, e uma ferramenta de análise a ser utilizada na mesma fase de projecto ou, eventualmente, durante a fase de desenvolvimento numa perspectiva de refinamento. Identificámos algumas *'benchmarks'* a serem utilizadas ao nível da linguagem de programação. Estas são, sobretudo, para a linguagem *Ada*. Ao nível do sistema operativo, apresentámos uma

quantidade maior de métricas e metodologias de teste, nas duas aproximações seguintes: *'fine-grained benchmarks'* e *'Application-oriented benchmarks'*. Desta pesquisa, concluímos que as *'fine-grained bechmarks'* puras podem ser aplicadas com sucesso para a determinação dos tempos de execução das várias primitivas dos SOTRs. Por outro lado, podem também ser aplicadas a suportes práticos de tempo-real, constituídos pela combinação de SO e *'hardware'*, para obtenção de uma figura de mérito única que caracteriza esta combinação. Verificámos, igualmente, que numa eventual combinação dos vários testes, das várias metodologias, elas podem conduzir a uma *'benchmark'* óptima.

As *'application-oriented benchmarks'* devem ser aplicadas para avaliar um sistema de suporte de aplicações de tempo-real em termos abstractos, numa perspectiva do mérito de cumprimento das restrições temporais. Concluímos aqui que estas *'benchmarks'* podem definir o comportamento da aplicação de tempo-real a desenvolver, se os outros parâmetros para a *'benchmark'* corresponderem aos parâmetros temporais do sistema de tempo-real a desenvolver. Foi possível, nesta altura, passar à dedução dos modelos de escalonamento práticos.

Estes modelos são mais fáceis de deduzir, se forem feitos com base em experiências concretas, de forma a ser possível perceber bem o comportamento do sistema. Para isso implementámos dois modelos: um para o escalonamento estático a executar com todo o algoritmo *'round-robin'*, onde foram consideradas todas as tarefas com a mesma prioridade; o segundo implementa o escalonamento preemptivo por prioridades fixas baseado no algoritmo RM.

Passámos, então, à avaliação, em concreto, de um sistema de tempo-real baseado na aplicação sintética definida.

Como pretendemos desenvolver metodologias independentes de suportes específicos, começámos por definir os modelos de escalonadores. Assim, qualquer núcleo, independentemente da sua concepção, que tiver um

escalonador em conformidade com os modelos abstractos definidos, as metodologias desenvolvidas poderão ser utilizadas para validar a sua escalonabilidade. Os modelos de escalonadores caracterizados foram os orientados por acontecimentos e os orientados por tempo. Na caracterização dos modelos de escalonadores foram definidas as primitivas dos *'overheads'* presentes em ambos os modelos, mas que têm uma composição específica para cada um dos modelos. Isto é, os dois modelos considerados não têm os mesmos *'overheads'*.

Com isto, chegámos ao estudo do escalonamento estático. Neste campo, foram feitas as análises qualitativas e quantitativas de dois modelos. O primeiro corresponde ao escalonamento de um conjunto de tarefas, do início ao fim e numa ordem arbitrária. O segundo corresponde ao escalonamento do mesmo conjunto de tarefas mas, desta feita, através da política de escalonamento *'round-robin'*. Tudo o resto se passa de igual forma. Em ambos, foram deduzidos modelos qualitativos e validados por análise quantitativa.

Os resultados obtidos foram bons. No primeiro modelo, procedemos também à análise do efeito do *'overhead'* do executivo em função da carga computacional do sistema. Aqui verificou-se que o *'overhead'* relativo varia no sentido inverso da carga computacional. No segundo modelo, procedemos também à análise do efeito do *'overhead'* do executivo em função da resolução do *'timer'* e verificámos que esta aumenta exponencialmente à medida que também aumenta a resolução.

Em relação ao caso preemptivo, foi feita a análise qualitativa para dois modelos. Neste caso: em primeiro lugar para o escalonamento orientado por eventos e, depois, para o escalonamento orientado por tempo. A análise quantitativa só foi feita para o modelo orientado por tempo. Aqui os resultados também foram bons, tendo sido possível verificar, igualmente, a perda de escalonabilidade de um conjunto específico, que em termos teóricos (através da condição necessária e suficiente) era escalonável e na

implementação prática deixou de ser. Mais uma vez, a metodologia desenvolvida confirmou a situação.

Por isto tudo que foi dito, pensamos ter atingido os objectivos traçados. Assim, com base nas metodologias deduzidas e nas *'benchmarks'* identificadas, é possível analisar a escalonabilidade de sistemas de tempo-real em termos práticos, no âmbito dos modelos de escalonadores definidos e dos algoritmos de escalonamento estáticos e preemptivos por prioridades fixas, concretamente escalonamento *'round-robin'* e *'Rate Monotonic'*, respectivamente.

Embora este estudo não tenha contemplado um outro algoritmo de prioridades fixas, o *'deadline monotonic'*, pensamos que as metodologias do caso preemptivo são facilmente adaptáveis, uma vez que o único pressuposto que muda é o facto da *'deadline'* ser menor do que o tempo de período.

Em termos de perspectivas futuras, pensamos que se poderia estender este estudo para o escalonamento de tarefas aperiódicas independentes e dependentes. A grande novidade passa pelo aparecimento de novas restrições, devido aos diversos tipos de bloqueios.

Referências

[Bur91] A. Burns, *"Scheduling hard real-time systems: a review"*, Software Engineering Journal, pp. 116-128, May 1991.

[CDV+86] Russell M. Clapp, Louis Duchesneau, Richard A. Volz, Trevor N. Mudge, and Timothy Schultze, *"Toward Real-Time Performance Benchmarks for Ada"*, Communications of the ACM, Volume 29, No. 8, August 1986.

[Cra97] Nicholas Cravotta, *"Real-Time Operating Systems"*, Embedded Systems Programming, pp. 97-103, March 1997.

[DM89] M. Dertouzos, and A. Mok, *"Multiprocessor On-line Scheduling of Hard-Real-Time Tasks"*, IEEE Transactions on Software Engineering, Vol. 15, No. 12, pp. 1497-1506, December 1989.

[Don87] Patrick Donohoe, *"A Survey of Real-Time Performance Benchmarks for the Ada Programming Language"*, Technical Report, CMU/SEI-87-TR-28, ESD-TR-87-191, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, December 1987.

[FGG+91] Borko Furht, Dan Grostick, David Gluch, Guy Rabbat, John Parker, Meg McRoberts, *"REAL-TIME UNIX SYSTEMS Design and Application Guide"*, Kluwer Academic Publishers, 1991.

[GMS94] Kaushik Ghosh, Bodhisattwa Mukherjee, Karsten Schwan, *"A Survey of Real-Time Operating Systems – Draft"*, Technical report, GIT-CC-93/18, College of Computing, Georgia Institute of Technology, Atlanta, Georgia, February 15, 1994.

[GPF+93] Kaushik Ghosh, Kiran Panesar, Richard M. Fujimoto and Karsten Schwan, *"PORTS: a Parallel, Optimistic, Real-Time Simulator"*, Technical report, GIT-CC-93/71, College of Computing, Georgia Institute of Technology, Atlanta, Georgia, December 7, 1993.

[GT95] Vasilis C. Gerogiannis, Manthos [a] Tsoukarellas, *"SAT – A Schedulability Analysis Tool for Real-Time Applications"*, Proceedings of the 2th EUROMICRO Workshop on Real-Time Systems, IEEE, 1995.

[HS91] Wolfgang [a] Halang, Alexander D. Stoyenko, *"Constructing Predictable Real Time Systems"*, Kluwer Academic Publishers, 1991.

[Kar90] Rabindra P. Kar, *"Implementing the Rhealstone Real-Time Benchmark"*, Dr. Dobb's Journal, pp.46-55, April 1990.

[KAS93] Daniel I. Katcher, Hiroshi Arakawa, Jay K. Strosnider, *"Engineering and Analysis of Fixed Priority Schedulers"*, IEEE Transactions on Software Engineering, Vol. 19, No. 9, pp. 920-934, September 1993.

[KP89] Rabindra P. Kar and Kent Porter, *"RHEALSTONE A Real-Time Benchmarking Proposal"*, Dr. Dobb's Journal, pp.14-24, February 1989.

[KW91] Nick Kamenoff, Nelson Weiderman, *"Hartstone Distributed Benchmark: Requirements and Definitions"*, In Proceedings of the Real-Time Systems Symposium, pages 199-208, IEEE Computer Society Press, December 1991.

[Lan96] Ronald Landman, *"Selecting a Real-Time Operating System"*, Embedded Systems Programming, pp. 79-94, April 1996.

[LL73] C. L. Liu, and James W. Layland, *"Scheduling Algorithms for Multiprogramming in Hard-Real-Time Environments"*, Journal of ACM, Vol. 20, No. 1, pp. 46-61, January 1973.

[LSD89] John Lehoczky, Lui Sha and Ye Ding, *"The Rate Monotonic Scheduling Algorithm: Exact Characterization And Average Case Behavior"*, Proc. IEEE Real-Time Symp., CS Press, Los Alamitos, Calif., Order No. 2004, pp. 166-171, 1989.

[Mag95] António J. Pessoa de Magalhães, *"Estabilização dos Controladores de Tempo-Real através da complacência temporal dos Objectos Controlados"*, Tese de Doutoramento, FEUP, Março 1995.

[McR96] Eric McRae, *"Benchmarking Real-Time Operating Systems"*, Dr. Dobb's Journal, pp.48-58, May 1996.

[MCV96] Alberto García-Martínez, Jesús F. Conde, and Angel Viña, *"A Comprehensive Approach in Performance Evaluation for Modern Real-Time Operating Systems"*, in Proceedings of EUROMICRO-22, pp. 61-68, IEEE, 1996.

[Nie94] Douglas Niehaus, *"Program Representation and Execution in Real-Time Multiprocessor Systems"*, Dissertation for the degree of Doctor of Philosophy, University of Massachusetts, Amherst, February 1994.

[PD93] F. Panzieri, R. Davoli, *"Real Time Systems: A Tutorial"*, Technical Report, UBLCS-93-22, Laboratory for Computer science, University of Bologna, Piazza di Porta S. Donato, 5, Bologna, Italy, October 1993.

[PDP93] F. Panzieri, L. Donatiello, L. Poretti, *"Scheduling Real Time Tasks: A performance Study"*, Technical Report, UBLCS-93-10, Laboratory for Computer science, University of Bologna, Piazza di Porta S. Donato, 5, Bologna, Italy, May 1993.

[PSK90] David L. Parnas, A. John van Schouwen, and Shu Po Kwan, *"Evaluation of Safety-Critical Software"*, CACM, vol. 33, No. 6, June 1990, pp. 636-648.

[RS94] Krithi Ramamritham and John A. Stankovic, *"Scheduling Algorithms and Operating Systems support for Real-Time Systems"*, Proceedings of the IEEE, vol. 82, No. 1, January 1994.

[Sac95] Krzysztof M. Sacha, *"Measuring the Real-Time Operating System Performance"*, in Proceedings of the 2th EUROMICRO Workshop on Real-Time Systems, IEEE, 1995.

[San97] Daniela Santos, *"Sistemas Operativos de Tempo-Real"*, FEUP, 1997.

[SG90] Lui Sha, and John B. Goodenough, *"Real-Time Scheduling Theory and Ada"*, IEEE Computer, Vol. 23, No. 4, pp. 53-62, April 1990.

[SKG91] Lui Sha, Mark H. Klein, John B. Goodenough, *"Rate Monotonic Analisys for Real-Time Systems"*, Technical Report, CMU/SEI-91-TR-6, ESD-91-TR-6, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, March 1991.

[SR90] John Stankovic and Krithi Ramamritham, *"What is Predictability for Real-Time Systems?"*, J. Real-Time Systems, Vol. 2, December 1990, pp. 247-254.

[SRL90] L. Sha, R. Rajkumar, and J. Lehoczky, *"Priority Inheritance Protocols: Na Approach to Real-Time Synchronization"*, IEEE Transactions on Computers, Vol. 39, No. 9, pp. 1175-1185, September 1990.

[SSN+95] John A. Stankovic, Marco Spuri, Marco Di Natale, Giorgio C. Buttazzo, *"Implications of Classical Scheduling Results for Real-Time Systems"*, IEEE Computer, 1995.

[Sta88] John Stankovic, *"Misconceptions About Real-Time Computing"*, IEEE Computer, Vol. 21, No. 10, October 1988, pp.10-19.

[TW97] Andrew S. Tanenbaum, Albert S. Woodhull, *"Operating Systems: Design and Implementation, Second Edition"*, Prentice-Hall, Inc., 1997.

[Wei84] Reinhold P. Weicker, *"DHRYSTONE: A Synthetic Systems Programming Benchmark"*, Communications of the ACM, Volume 27, No. 10, October 1984.

[Wei89] Nelson Weiderman, *"Hartstone: Synthetic Benchmark Requirements for Hard Real-Time Applications"*, Technical Report, CMU/SEI-89-TR-23, ESD-89-TR-31, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, June 1989.

[Wel93] George Wells, *"A Comparison of Four Microcomputer Operating Systems"*, Real-Time Systems, 5, pp. 345-368, 1993.

[WK92] Nelson H. Weiderman and Nick I. Kameoff. *"Hartstone uniprocessor benchmark: Definitions and experiments for real-time systems"*, In Real-Time Systems Journal, volume 4 (4), pages 353-383, Kluwer Academic Publishers, December 1992.

[Zal93] Janusz Zalewski, *"Real-Time Systems Glossary"*, Dept. of Computer Science, The University of Texas of the Permian Basin, December 1993.