UNIVERSIDADE TÉCNICA DE LISBOA
INSTITUTO SUPERIOR TÉCNICO

Integração Modular de Dispositivos de Entrada/Saída em Plataformas de Controlo Distribuído

Manuel Pedro Macara Coutinho

(Licenciado)

Dissertação para obtenção do Grau de Mestre em

Engenharia Electrotécnica e de Computadores

DOCUMENTO PROVISÓRIO

Dezembro 2007

Modular Input/Output Integration in Distributed Control Systems

Manuel Pedro Macara Coutinho

Tese submetida para provas

de mestrado em

Engenharia Electrotécnica e de Computadores

Departamento de Engenharia Electrotécnica e de Computadores

Instituto Superior Técnico

Lisboa

Dezembro 2007

Este trabalho foi parcialmente financiado por:

FCT - Fundação para a Ciência e a Tecnologia
(através do Projecto POSC/EIA/56041/2004 - DARIO)

ESA - European Space Agency
(ao abrigo da Iniciativa Triângular de Inovação AIR - ARINC 653 In RTEMS)

Tese realizada sob a orientação do

Prof. Doutor Carlos Manuel Ribeiro Almeida

Professor Auxiliar do Departamento de Engenharia Electrotecnia e de Computadores do Instituto Superior Técnico da Universidade Técnica de Lisboa

e co-orientação do

Prof. Doutor José Manuel de Sousa de Matos Rufino

Professor Auxiliar do Departamento de Informática da Faculdade de Ciências da Universidade de Lisboa

Resumo

Esta dissertação aborda a temática dos Sistemas de Tempo Real Distribuídos. No meio industrial, estes sistemas são componentes fundamentais pois distribuem as tarefas em várias células e operam de modo a garantir que, em condições de funcionamento normais, nem equipamento nem vidas humanas são postos em perigo.

A primeira intervenção apresentada na dissertação centra-se no desenvolvimento de condições para que as tarefas do sistema cumpram a meta temporal. Se o sistema não obedecer às restrições matemáticas impostas, o sistema é não escalonável. Em particular, o modelo de sistema adoptado centra-se em tarefas periódicas assíncronas juntamente com tarefas esporádicas, usando um escalonador preemptivo de prioridades fixas.

A segunda intervenção aborda a especificação ARINC 653 oriunda da indútria aeronáutica e a sua implementação recorrendo a COTS RTOS. Em particular, é usado o RTEMS como um caso de estudo. São apresentadas duas soluções arquitecturais, sendo escolhida aquela que melhor se adequa às restrições dos sistemas alvo.

Por último, são abordados os dispositivos de E/S de forma a torná-los tolerantes a faltas recorrendo exclusivamente a mecanismos presentes no CPU nativo (relógio de sistema). Se o número de eventos despoletados pelo hardware externo for demasiado alto, o sistema pode entrar em sobrecarga e o restante processamento, por ventura mais crítico, pode não obedecer às restrições temporais impostas. É proposto um mecanismo que a partir de uma métrica estabelecida (e.g ritmo dos eventos, tempo mínimo entre dois eventos consecutivos, etc) inibe o processamento imediato dos eventos.

Abstract

This thesis handles the Distributed Real-Time Operating Systems thematic. In the industrial world, these systems are fundamental components since they distribute the tasks through a number of cells and operate in such a manner that, in normal working conditions, neither equipement nor human lives are placed in jeapordy.

The first intervention of this thesis focus on the developement of analytical conditions to ensure that the tasks fulfill their deadlines. If the system does not obey these mathematical conditions then it is unschedulable. In particular, the targeted system model focus on assynchronous periodic tasks together with sporadic tasks, using a fixed-priority preemptive scheduler.

The second intervention approaches the ARINC 653 specification, that comes from the aeronautical industry, and its implementation using COTS RTOS. In particular, the RTEMS OS is used as a case study. Two architectural solutions are presented where the fittest, regarding the target platform restrictions, is chosen.

Lastly, the I/O devices are handled to make them fault tolerant using exclusively hardware mechanisms already present in the native CPU (system clock). If the number of events triggered by the external hardware is too high, the system can enter an overload state and the remaining tasks, which can be more critical, can miss their deadlines. This thesis proposes a mechism that temporarily inhibits event processing when it detects an overlad scenario (e.g. event rate too high, minimum inter-arrival time violated, etc).

Palavras Chave

Sistemas de Tempo-Real

Tolerância a faltas

Arquitectura Event-triggered

Arquitectura Time-triggered

Escalonabilidade

Fiabilidade

Keywords

Real-time Systems

Fault-tolerance

Event-triggered architecture

Time-triggered architecture

Schedulability

Dependibility

Agradecimentos

Aos meus orientadores, Prof. Carlos Almeida e Prof. José Rufino, a quem desejo expressar o meu reconhecimento pelo empenho que colocaram na orientação deste trabalho. As suas críticas, sugestões, incentivo e apoio constantes contribuiram de forma decisiva para a elaboração da presente dissertação.

À minha família, que sempre me apoiou.

Ao Instituto Superior Técnico e ao Centro de Sistemas Telemáticos e Computacionais, pela disponibilização dos meios técnicos e de enquadramento científico essenciais para a realização deste trabalho.

Lisboa, Dezembro 2007

Manuel Pedro Macara Coutinho

Aos meus pais.

Índice

1 Introdução **1**

2 Estado do Conhecimento **7**

2.1 Sistemas de Tempo-Real 7

2.2 Sistemas Tolerantes a Faltas 9

2.3 Sistemas de Tempo-Real Distribuídos 11

2.4 Arquitecturas Time-Triggered vs Event-Triggered 12

2.4.1 Arquitectura Time-Triggered 12

2.4.2 Escalonamento na Arquitectura Time-triggered 13

2.4.3 Arquitectura Event-Triggered 14

2.4.4 Escalonamento na Arquitectura Event-triggered 15

2.4.5 Comparação 18

2.5 Suporte de Sistema Operativo 19

2.5.1 Especificações Existentes 19

2.5.2 Sistemas Operativos de Tempo-Real 20

3 Escalonamento em Sistemas de Controlo de Tempo-Real **23**

3.1 Motivação 26

3.2 Modelo de Sistema 29

3.3 Análise RTA de Tarefas Periódicas Síncronas 30

3.4 Análise RTA de Tarefas Periódicas Assíncronas 33

3.4.1 Determinação do Último Instante de Idle 36

3.4.2 Cálculo do Tempo de Resposta 39

3.4.3 Exemplo de uma Aplicação 39

3.5 Integração de Tarefas Esporádicas . . . 42
3.5.1 Instante Crítico Assíncrono . . . 44
3.5.2 Análise RTA de Tarefas Periódicas Assíncronas . . . 46
3.5.3 Análise RTA de Tarefas Esporádicas . . . 49
3.6 Optimização da Análise de Escalonamento . . . 50
3.6.1 Exploração do Último Instante de Idle . . . 51
3.6.2 Lidar com Grandes Hiperperiodos . . . 52
3.6.3 Algoritmo de Escalonamento Distribuído . . . 54
3.7 Sumário . . . 60

4 Compartimentação e Composição em Sistemas de Controlo de Tempo-Real 63
4.1 Conceito de Partição . . . 65
4.1.1 Segregação Temporal . . . 66
4.1.2 Segregação Espacial . . . 68
4.1.3 APEX - APplication EXecutive . . . 68
4.2 Implementação da Especificação ARINC 653 . . . 70
4.2.1 Utilização de COTS RTOS . . . 70
4.2.2 Escolha da Arquitectura . . . 71
4.2.3 Arquitectura MEC . . . 75
4.2.4 Protecção Temporal . . . 76
4.2.5 Protecção Espacial . . . 81
4.2.6 Integração no RTEMS . . . 85
4.3 Escalonamento de Aplicações ARINC 653 . . . 88
4.3.1 Modelo de Sistema . . . 90
4.3.2 Escalonamento de Tarefas . . . 92
4.3.3 Optimizações . . . 94
4.4 Sumário . . . 95

5 Controlo Temporal de Eventos 97
5.1 Arquitectura Event-Triggered . . . 99
5.2 Arquitectura Time-Triggered . . . 101
5.3 Integração das Arquitecturas Time-Triggered e Event-Triggered . . . 102

5.4 Caracterização da Carga Computacional 103
5.5 Filtros Discretos 104
5.5.1 Implementação dos Filtros Discretos 107
5.6 Resultados Experimentais 109
5.7 Sumário 114

6 Conclusões & Perspectivas Futuras **115**

Índice de Figuras

2.1 Utilidade dos sistemas de tempo-real em sistemas (a) *hard real-time* (b) *firm real-time* (c) *soft real-time* (d) *no real-time* 8
2.2 Sistema de válvulas tolerante a faltas por replicação 10
2.3 Sistema de processamento tolerante a faltas por votação 11
2.4 Escalonamento de tarefas periódicas num sistema TT 13

3.1 Tarefas assíncronas periódicas sem interferência entre si 27
3.2 Modelação do Atraso em Sistemas de Controlo 28
3.3 Iterações do cálculo do tempo de resposta - equação 3.7 33
3.4 Função de Trabalho Estendida 37
3.5 Pseudo-código para determinar $L_i(t)$ 39
3.6 Iterações do pseudo-código para calcular $L_i(t)$ 40
3.7 Escalonamento nos primeiros instantes do hiperperiodo do Exemplo 1 42
3.8 Tempos de resposta para cada execução da tarefa (a) Γ_6; (b) Γ_7; (c) Γ_8; (d) Γ_8 com $D_8 = 90$ 43
3.9 Candidatos ao instante crítico assíncrono 45
3.10 Pseudo-código para determinar os candidatos ao *verdadeiro instante crítico* no intervalo $]t_{min}, t_{max}]$ 46
3.11 Exemplo das iteracções da função $\varphi_i(t)$ 47
3.12 Candidatos ao pior instante para a activação das tarefas esporádicas 48
3.13 Cálculo da função $\tilde{L}_i(last,t)$ que recebe como argumento adicional o último instante de idle conhecido prévio a t 52
3.14 Determinação de κ_i no intervalo $]t_{min}, t_{max}]$ recorrendo à função optimizada $\tilde{L}_i(last,t)$ 53
3.15 Determinação de ${}^{l_{initial}}R_i, \ldots, {}^{l_{final}}R_i$ recorrendo à função optimizada $\tilde{\kappa}_i(t_{min}, t_{max}, last)$ 54

3.16 As duas situações no cálculo de $L_i({}^{l}r_2)$: Γ_1 está idle em ${}^{l}r_2$; Γ_1 está a executar em ${}^{l}r_2$ 55
3.17 Cálculo rápido de $L_i(t)$ procurando para trás no tempo 56
3.18 Pseudo-código para o cálculo rápido do último instante idle das tarefas $\Gamma_1, ..., \Gamma_i$ antes de t 57
3.19 Comparação do tempo de análise para cada um dos métodos para determinar o último instante de idle 58
3.20 Determinação dos candidatos κ_i dentro do intervalo $[t_{initial}; t_{final}[$ 58
3.21 Determinação de ${}^{l_{initial}}R_i, \ldots, {}^{l_{final}}R_i$ recorrendo à função optimizada $\tilde{\kappa}_i(t_{min}, t_{max}, last)$ 59

4.1 Exemplo de aplicações com diferentes graus de criticalidade/funcionalidade 64
4.2 Arquitectura da especificação ARINC 653 65
4.3 Exemplo do escalonamento de partições 67
4.4 Arquitectura Single-Executive Core 71
4.5 Arquitectura Multi-Executive Core 73
4.6 ISR do *clock tick* 78
4.7 Pseudo-código do handler de *clock tick* 79
4.8 Transições de estado dos processos 80
4.9 Mecanismos de protecção de memória por segmentação (translação) da arquitectura Intel IA-32 82
4.10 Mecanismos de protecção de memória da arquitectura ATMEL AT697E SPARC LEON2 84
4.11 Mecanismos de protecção de memória por paginação (translação) da arquitectura Gaisler SPARC LEON3 84
4.12 Módulos RTEMS que são incluídos/excluídos na integração na arquitectura MEC . . 85
4.13 Produção de aplicações usando o RTEMS 86
4.14 Produção de aplicações na arquitectura MEC usando as ferramentas nativas do RTEMS 86
4.15 Produção de aplicações na arquitectura MEC usando as ferramentas nativas do RTEMS e um filtro adicional 87
4.16 Excerto da Makefile da construção da partição PX na arquitectura MEC 88
4.17 Excerto da Makefile da construção do PMK e linkagem com as restantes partições na arquitectura MEC 88
4.18 Exemplo do escalonamento de uma partição 90

4.19 Exemplo do escalonamento de tarefas periódicas dentro de uma partição 94

5.1 Comportamento por Histerese do Sistema: Arquitecturas TT e ET 98
5.2 *Workload* devido às interrupções periódicas 101
5.3 Carga Computacional 103
5.4 Pseudo-código da actualização do filtro IIR e tomada de decisão sobre a arquitectura 108
5.5 Vector em Anel contendo a memória do filtro FIR 108
5.6 Evolução de y[n] no filtro IIR sem os mecanismos de protecção 110
5.7 Evolução de y[n] no filtro IIR com os mecanismos de protecção 111
5.8 Evolução de y[n] no filtro FIR sem e com os mecanismos de protecção - a) e b) . . . 112
5.9 Evolução de y[n] no filtro FIR sem e com os mecanismos de protecção - a) e b) . . . 112
5.10 Evolução de $y[n]$ num sistema ethernet (a) sem protecções ; (b) com protecções . . . 113

Índice de Tabelas

3.1 Parâmetros do Exemplo 1. A análise de escalonabilidade foi realizada com o método melhorado descrito na Secção 3.6.1, Figura 3.15 41

4.1 Comparação das arquitecturas SEC e MEC 74

4.2 Comparação do escalonamento de processos nas arquitecturas SEC e MEC 75

5.1 Ritmos de interrupções máximos de alguns dispositivos (extraído parcialmente de (Regehr & Duongsaa, 2005)) 100

Lista de Acrónimos

AGC	Apollo Guidance Computer
AIR	ARINC 653 In RTEMS
APEX	APlication EXecutive
API	Application Programming Interface
CAN	Controller Area Network
COTS	Commercial Off-The-Shelf
CPU	Central Processing Unit
DARIO	Distributed Agency for Reliable Input/Output
DSP	Digital Signal Processor
E/S	Entrada/Saída
EDF	Earliest Deadline First
ESA	European Space Agency
ET	Event-Triggered
FIR	Finite Impulse Response
FPU	Floating Point Unit
GNU	GNU's Not Unix

I/O	Input/Output
IIR	Infinite Impulse Response
ISR	Interrupt Service Routine
MCU	Micro-Controller Unit
MDC	Máximo Divisor Comum
MEC	Multiple Executive Core
MIT	Minimum Inter-arrival Time
MMC	Mínimo Múltiplo Comum
MMU	Memory Management Unit
MPU	Micro-Processor Unit
MTF	Major Time Frame
OS	Operating System
PMK	Partition Management Kernel
POSIX	Portable Operating System Interface
RMS	Rate Monotonic Scheduler
RTEMS	Real Time Executive for Multiprocessor Systems
RTOS	Real Time operating Systems
SEC	Single Executive Core
SO	Sistema Operativo
TT	Time-Triggered
WCET	Worst Case Execution Time

1

Introdução

Nas últimas décadas temos vindo a assistir a um desenvolvimento crescente nas tecnologias de sistemas distribuídos de tempo-real e aplicações correspondentes. Apesar do progresso verificado, continuam a subsistir alguns nichos de aplicações onde a evolução tem sido mais lenta. Encontram-se nesta situação os denominados sistemas distribuídos embebidos de tempo-real, sistemas onde a par da distribuição existem requisitos adicionais de tolerância a faltas e restrições temporais, que frequentemente necessitam de ser concretizados em ambientes computacionais com recursos moderados (e.g. velocidade de processamento, memória). Encontram-se neste caso, muitas aplicações de controlo onde, para além das exigências resultantes da sua natureza eminentemente descentralizada, é ainda necessário proporcionar uma interface com o "mundo real", por exemplo através de sensores e actuadores.

A natureza concorrente das aplicações, i.e. a capacidade de se executarem várias tarefas numa mesma plataforma computacional de uma forma "pseudo-paralela", permite ao arquitecto de sistema reduzir os custos de construção devido à disponibilidade de produtos COTS (Commercial Off-The-Shelf) de grande maturidade. É o caso dos Sistemas Operativos de Tempo-Real. Contudo o paradigma de engenharia de sistemas a partir de produtos COTS não está isenta de desafios. Esta dissertação aborda parte dos problemas que se colocam nesta área, nomeadamente no que diz respeito a:

- **Análise de escalonamento das tarefas** - Estabelecimento de condições analíticas que permitem decidir se as tarefas cumprem (ou não) as restrições temporais impostas pela aplicação.
  - Esta dissertação compila os requisitos actuais dos sistemas de controlo distribuído em tempo-real e os avanços teóricos mais recentes nesta área, construindo modelos de escalonamento mais próximos da realidade e com resultados mais interessantes.

- **Compartimentação de tarefas** - Inclusão de um nível de protecção adicional para impedir interferências indesejáveis entre tarefas. Como exemplo, tem-se as protecções a nível temporal, onde uma tarefa não pode ocupar mais tempo do que o previsto, caso contrário, outras tarefas de alta criticicalidade podem não chegar a ser executadas. Outro exemplo refere-se ao acesso indevido a zonas de memória devido a ponteiros incorrectamente inicializados.

  – Esta dissertação analisa as especificações existentes para a compartimentação de tarefas. Discute ainda como implementar de forma modular e extremamente versátil uma especificação em particular: a norma ARINC 653.

- **Tratamento dos eventos externos** - Em muitos sistemas de controlo, a notificação de que ocorreu um evento externo (e.g. pacote que chega da rede de comunicação) interfere temporalmente com as restantes tarefas em execução. Se o número de eventos externos for demasiado alto, devido por exemplo, a uma falha do dispositivo de E/S (Entrada/Saída), a execução atempada das restantes tarefas pode estar comprometida.

  – Esta dissertação apresentada um mecanismo que permite impedir que sobrecargas de eventos externos (notificados a partir de interrupções) impeçam a execução das restantes tarefas no sistema. Discute ainda como integrar este mecanismo de uma forma modular e com o mínimo de alterações possíveis no sistema existente.

Em relação ao escalanamento de tarefas, é proposto um modelo mais complexo do sistema com um mapeamento directo na realidade dos sistemas de controlo actuais. Tal como o novo modelo proposto, a derivação de condições de escalonabilidade também se torna mais complexa. De facto, a complexidade da determinação da escalonabilidade com o modelo clássico é pseudo-polinomial enquanto que com o modelo proposto passa para *co-NP-hard* no sentido estrito. No entanto, à medida que os sistemas computacionais ficam sobrecarregados com novas funcionalidades, este novo modelo permite que sistemas que eram considerados não escalonáveis passem a sê-lo. Desta forma reduzem-se os custos de compra e certificação de equipamento com maior poder computacional. Esta análise encontra-se discutida no capítulo 3.

A compartimentação de tarefas é um mecanismo extremamente atractivo em sistemas complexos de tempo-real com requisitos de confiabilidade elevados. Como foi dito, é extremamente desejável

que o sistema contenha mecanismos que impedem uma dada tarefa (ou conjunto de tarefas) de interferir de forma "imprevista" noutra tarefa de maior criticalidade. Por exemplo, um ponteiro incorrectamente inicialidado de uma dada tarefa não deve aceder a zonas de memória (pelo menos em modo de escrita) de outras tarefas. Outro exemplo refere-se ao aspecto temporal: uma tarefa (ou conjunto de tarefas) não deve poder ocupar o processador por mais tempo que o previsto. Esta dissertação apresenta uma especificação muito utilizada em vários sistemas de controlo de tempo-real na área de aviónica e discute possíveis métodos da sua implementação e extensão a outros domínios computacionais. O capítulo 4 discute estes mecanismos em pormenor.

Por último, o tratamento de eventos introduz em vários sistemas de controlo uma certa incerteza temporal pois não é conhecido *a priori* quando são despoletados. Em caso de falha do dispositivo, é possível que haja uma sobrecarga de eventos e que o sistema não consiga executar as tarefas de grau elevado de criticalidade. Esta dissertação apresenta um mecanismo que, a partir de um conjunto de métricas definidas pelo programador, limita o número de eventos processados, sendo possível assegurar o tempo necessário para processar as restantes tarefas. Estes mecanismos encontram-se discutidos no capítulo 5.

O trabalho desenvolvido na presente dissertação integra-se nos projectos:

- DARIO - "Distributed Agency for Reliable Input/Output"
- AIR - "ARINC 653 In RTEMS"

O projecto DARIO é financiado pela FCT (Fundação para a Ciência e Tecnologia), através do programa POSC/EIA/56041/2004. Pretende integrar uma intensa investigação produzida nos últimos anos no domínio da utilização de redes dispositivos, em concreto na utilização da rede CAN (Controller Area Network), com a funcionalidade e os recursos necessários à concretização de aplicações distribuídas que necessitem de lidar directamente com sensores e actuadores.

O projecto AIR é financiado pela ESA (European Space Agency) através do programa de Iniciativa Triângular de Inovação. Corresponde a um estudo a nível funcional de como adequar uma especificação de compartimentação de tarefas (especificação ARINC 653) existente no domínio da aeronáutica para o Sistema Operativo de Tempo-Real RTEMS (Real-Time Executive for Multiprocessor Systems).

Resumo da Organização do Documento

- Segundo capítulo - São introduzidas noções fundamentais de tempo-real, fiabilidade e escalonamento. Realiza-se uma breve descrição do núcleo de sistema operativo de tempo real utilizado, o RTEMS (Real Time Executive Multiprocessor System), e da especificação da norma ARINC 653.
- Terceiro capítulo - É discutido o problema da análise de escalonamento de tarefas num sistema de tempo-real.
- Quarto capítulo - É discutida a implementação dos mecanismos de adequação de um RTOS nativo, em particular, o RTEMS, à norma ARINC 653.
- Quinto capítulo - São apresentados mecanismos que garantem o cumprimento das metas temporais na presença de sobrecargas de eventos.
- Conclusões & Perspectivas Futuras - Apresentam-se as conclusões e perspectivas de trabalho futuro.

Disseminação de Resultados

- *Response Time Analysis of Asynchronous Periodic and Sporadic Tasks Scheduled by a Fixed-Priority Preemptive Algorithm*, M. Coutinho, J. Rufino, C. Almeida (submetido para publicação)
- *Securing The Timeliness of I/O Event Handling in Real-Time Kernels*, C. Almeida, M. Coutinho, J. Rufino (submetido para publicação)
- *ARINC 653 in RTEMS*, J. Rufino, S. Filipe, M. Coutinho, S. Santos, J. Windsor, Data Systems In Aerospace (DASIA), Napoles, Itália, Maio 2007
- *VITRAL - A text mode window manager for real-time embedded kernels*, M. Coutinho, C. Almeida, J. Rufino, Emerging Technologies and Factory Automation (ETFA), Prague, Czech Republic, Setembro 2006

- *Control of Event Handling Timeliness in RTEMS*, M. Coutinho, C. Almeida, J. Rufino, Parallel and Distributed Computing Systems (PDCS), Phoenix, Estados Unidos da América, Novembro 2005

- *VITRAL - A text mode window manager for RTEMS*, M. Coutinho, C. Almeida, J. Rufino, Jornadas de Engenharia de Electrónica e Telecomunicações e de Computadores (JETC), Lisboa, Portugal, Novembro 2005

Relatórios Técnicos de Projecto

- ESA/ITI AIR Project Deliverable. WP3 - AIR Design Results and Proof of Concept. M. Coutinho, E. Pascoal, S. Santos and J. Rufino. AIR Consortium: Skysoft Portugal/FCUL. May 2007.

- ESA/ITI AIR Project Deliverable. WP2 - AIR Overall System Specification. M. Coutinho, J. Rufino, S. Santos and E. Pascoal. AIR Consortium: Skysoft Portugal/FCUL. January 2007.

- ESA/ITI AIR Project Deliverable. WP1 - AIR Requirements, Architecture and Services. S. Santos, M. Coutinho and J. Rufino. AIR Consortium: Skysoft Portugal/FCUL. November 2006.

1 Introdução

2

Estado do Conhecimento

Os sistemas de controlo distribuído são muitas vezes elementos críticos que podem por em causa a perda de equipamento ou até mesmo de vidas humanas. Dadas estes condicionantes, possuem restrições a níveis temporal e de confiabilidade extremamente exigentes. Como abordagem comum a estes requisitos encontra-se na literatura a definição de *sistemas de tempo-real* e *tolerantes a faltas*. Sistemas de tempo-real abordam as restrições no domínio temporal, i.e., fornecem condições teóricas de modo a garantir que o sistema fornece uma resposta a tempo. Sistemas tolerantes a faltas são desenhados de forma a que uma falha de um componente não se propague para outros, e.g., através de redundância.

2.1 Sistemas de Tempo-Real

Sistemas de tempo-real são normalmente classificados sobre diferentes perspectivas consideradas ortogonais (Kopetz, 1997).

- função utilidade
- *fail-safe* versus *fail-operational*
- *guaranteed-timeliness* versus *best-effort*
- *resource-adequate* versus *resource-inadequate*
- *event-triggered* (ET) versus *time-triggered* (TT)

A primeira categoria classifica-os em termos das diferenças na sua função de utilidade *versus* tempo

- *hard real-time*
- *firm real-time*
- *soft real-time*
- *no real-time*

Para melhor compreender a diferença na designação de cada uma destas categorias encontra-se ilustrada na Figura 2.1 a função de utilidade. Como se pode constatar, nos sistemas *hard real-time* o não cumprimento de uma meta temporal reduz a utilidade do seu processamento posterior para $-\infty$, que traduz que o não cumprimento da meta temporal pode danificar o equipamento ou inclusive causar a perda de vidas humanas. Como exemplo, tem-se o sistema de arrefecimento de uma central nuclear ou o controlo de velocidade/altitude de um avião. Nos sistemas *firm real-time* o não cumprimento de uma meta temporal reduz a utilidade para um valor negativo (ou zero), o que é o mesmo que dizer que o processamento atrasado prejudica o sistema de forma limitada. Por outro lado, os sistemas *soft real-time* possuem sempre uma utilidade positiva mas que se reduz ao longo do tempo. Como exemplo tem-se o processamento de vídeo/voz. Por último, os sistemas *no real-time* não definem uma meta temporal, sendo a sua utilidade constante.

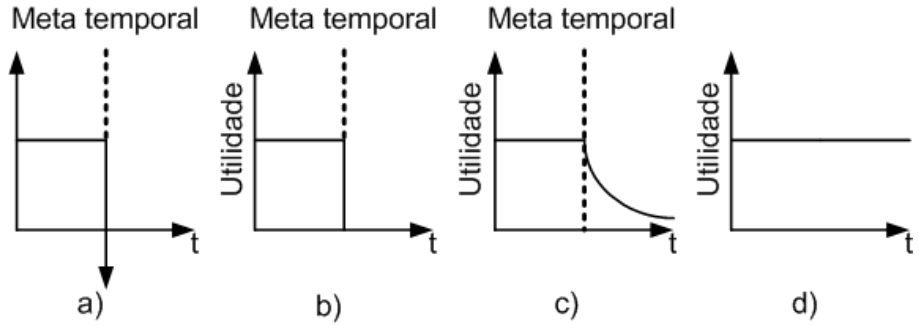


Figura 2.1: Utilidade dos sistemas de tempo-real em sistemas (a) *hard real-time* (b) *firm real-time* (c) *soft real-time* (d) *no real-time*

A segunda classificação, *fail-safe* versus *fail-operational*, especifica o modo como uma falha é tratada. No caso de ser *fail-safe* o sistema entra num estado bem conhecido e seguro, e.g., a paragem automática de um comboio quando encontra um sinal vermelho. Em sistemas *fail-operational* no

caso de uma falha o resto do sistema é capaz de recuperar automaticamente, mesmo que reduzindo a funcionalidade.

Enquanto que as primeiras duas classificações dependem sobretudo das características da aplicação, i.e., de factores externos ao sistema computacional, as restantes dependem do desenho e implementação, i.e. de factores inerentes ao sistema computacional:

- *guaranteed-timeliness* versus *best-effort*
- *resource-adequate* versus *resource-inadequate*
- *event-triggered* (ET) versus *time-triggered* (TT)

A primeira reflecte o grau de confiança a nível temporal, i.e., se permite estabelecer ou não garantias temporais em relação a uma determinada actividade. Para isso é necessário especificar as situações de pior caso do sistema.

A segunda reflecte se o sistema se encontra dimensionado de tal forma que mesmo as situações de sobrecarga estão contabilizadas e o sistema não sofre perdas de desempenho por causa desta fase transiente. Sistemas *resource-inadequate* são usados muitas vezes devido aos baixos custos que encorre. No entanto, não podem ser utilizados em sistemas *hard real-time* (Kopetz, 1997).

Por último tem-se a diferenciação entre a arquitectura *event-triggered* (ET) e *time-triggered* (TT). A arquitectura ET segue o princípio da reacção a um pedido, i.e., um evento. Desta forma, os eventos externos são rapidamente processados. Por outro lado, a arquitectura TT estabelece limites temporais rígidos sobre o processamento de cada componente. Assim, se o processamento de um dado componente não cumprir uma dada meta temporal, o processamento dos restantes componentes não é afectado (Kopetz, 2002).

2.2 Sistemas Tolerantes a Faltas

Sistemas Tolerantes a Faltas possuem níveis de criticalidade altos, i.e., são sistemas cuja falha de um componente não deve ser propagada para o resto do sistema. Como exemplo, tome-se o sistema de

válvulas ilustrado na Figura 2.2. Se uma válvula falhar, isto é, fica presa no estado aberto ou fechado, as restantes válvulas conseguem manter o funcionamento correcto do sistema. Este tipo de tolerância a faltas designa-se por replicação.

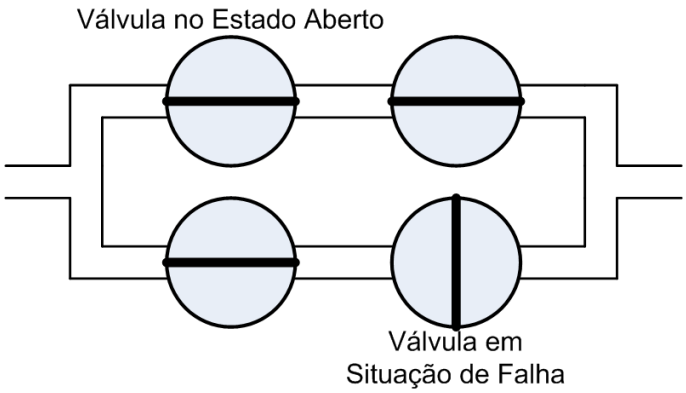


Figura 2.2: Sistema de válvulas tolerante a faltas por replicação

Outro tipo de tolerância basea-se na votação de elementos replicados. Supondo que um componente efectua uma operação específica. É possível que esse componente seja construído por um número ímpar de construtores independentes, de tal forma que no sistema alvo co-exista o mesmo componente replicado. O resultado desse grupo será realizado por votação. Por exemplo, caso dois componentes decidam fechar a válvula e o restante componente decidir o contrário, vence a decisão de fechar a válvula, Figura 2.3. A desvantagem deste método em relação ao anterior consiste na fragilidade do elemento de votação: se este elemento falhar, todo o sistema falha. Portanto, encontra-se dependente da operação correcta de um único elemento. No entanto, este componente é normalmente feito de tal forma que a probabilidade de erro seja extremamente baixa. Esta técnica é usada, por exemplo, na construção de processadores robustos a radiações (*radiation hardened*) como e o caso do processador SPARC LEON que iremos encontrar no capítulo 4.

Num sistema de controlo distribuído existem vários componentes susceptíveis de falharem. Dado que estes sistemas são usualmente compostos por um conjunto de sistemas computacionais interligados por uma rede de comunicação, cada elemento é passível de falhar. Como exemplos, tem-se a transmissão de um pacote pela rede de comunicação, uma leitura errónea do sensor, o processamento incorrecto do valor lido, a acção indevida do actuador, etc.

Num sistema computacional os elementos mais predispostos a falharem constituem os elementos

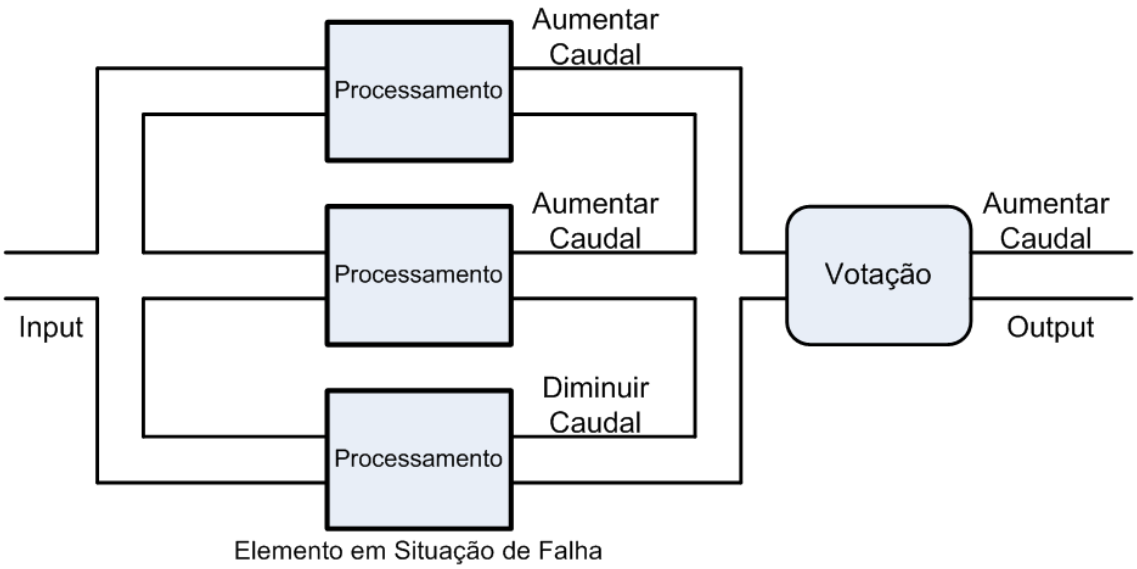


Figura 2.3: Sistema de processamento tolerante a faltas por votação

de software da aplicação. Dado que o RTOS e hardware em que se apoiam são usualmente expostos a rigorosos testes de certificação, o programador final é muito mais passível de ser um elemento introdutor de um erro. Como exemplo, tem-se o acesso indevido a zonas de memória (ponteiros não inicializados) ou *starvation* de outras tarefas. Para reduzir o efeito deste tipo de falhas é necessário dotar o sistema de mecanismos de validação de endereços e de protecção contra o uso excessivo de recursos do CPU.

2.3 Sistemas de Tempo-Real Distribuídos

Os sistemas de tempo-real seguem, geralmente, uma arquitectura distribuída, i.e., são compostos por um número variável de sistemas computacionais interligados através de uma plataforma de comunicação por onde interagem através da troca de mensagens. Esta arquitectura permite a utilização de sistemas embebidos, onde é sacrificada a velocidade e capacidade de processamento; a disponibilidade de uma interface com o operador sofisticada; a utilização de memória em massa; de forma a possibilitar a construção de uma solução eficaz e de custo moderado.

De forma a explorar os recursos fornecidos, a plataforma computacional executa várias actividades segundo um “pseudo-paralelismo”, i.e., embora só possa existir uma actividade em execução

em cada instante, alternam tão rapidamente que surge a noção que estão a decorrer simultaneamente. A cada uma destas actividades corresponde uma tarefa. Como exemplo destas operações tem-se a análise do estado dos sensores, o processamento dos dados, o accionamento dos actuadores, a comunicação com outros nós na rede, a interacção com o operador, etc. Devido à capacidade reduzida de cada sistema embebido face ao sistema que se quer implementar, estas operações são distribuídas através dos vários elementos da rede. De forma a garantir um comportamento coerente, i.e., as acções tomadas correspondem aos valores analisados, é necessário assegurar que todos os nós operam sobre a mesma versão das observações.

2.4 Arquitecturas Time-Triggered vs Event-Triggered

Como já foi dito anteriormente, existem duas filosofias no desenho de sistemas de tempo-real: ET e TT. Na arquitectura ET todas as actividades que decorrem do processamento dos dados são iniciadas quando o sistema detecta uma mudança significativa no estado do mundo exterior, i.e, um evento. Na arquitectura TT as actividades são iniciadas periodicamente e em instantes predefinidos no tempo (Kopetz, 1991).

2.4.1 Arquitectura Time-Triggered

Os sistemas TT são desenhados com base no princípio de que existem recursos suficientes para operar no pior cenário, mesmo que todas as falhas previstas no modelo ocorram simultaneamente (Kopetz, 1993). Enquanto que nos sistemas ET a ocorrência dos eventos é disseminada imediatamente através da rede, o problema principal num sistema TT relaciona-se com a rápida difusão das mensagens para todos os nós do sistema distribuído. Esta difusão é realizada periodicamente, com o período relacionado com a dinâmica dos elementos intervenientes (Kopetz, 1993).

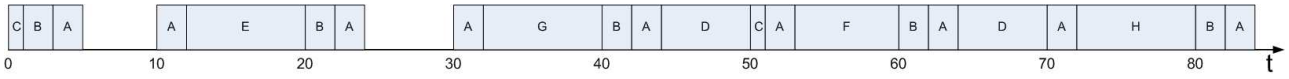
Em sistemas distribuídos, as mensagens enviadas através do arquitectura TT pela infra-estrutura de comunicação são produzidas periodicamente e em instantes bem conhecidos no tempo. Desta forma, a transmissão de mensagens é também temporalmente rígida. Dadas as características temporais dos sistemas TT, é necessária a utilização de um relógio de alta precisão em cada elemento da rede. Dado que o tempo é uma grandeza contínua e os relógios físicos não são exactamente “iguais”,

é preciso estampilhar a ocorrência dos eventos quantificando o tempo em unidades temporais de dimensão fixa (na ordem das dezenas de microsegundo). Assim, com uma dada precisão é possível ordenar em todo o sistema a ocorrência dos eventos.

2.4.2 Escalonamento na Arquitectura Time-triggered

O escalonamento de tarefas numa arquitectura TT é bastante simples. Embora possua uma complexidade elevada, i.e., pode demorar muito tempo a ser calculado, a sua implementação é trivial (Liu, 2000).

Este tipo de arquitectura apenas permite primariamente tarefas periódicas, pois responde somente à passagem do tempo, Figura 2.4. Assim, cada tarefa possui uma janela temporal pré-definida onde é executada. O escalonamento reduz-se a determinar quais as janelas temporais de cada tarefa. A decisão se uma dada escolha de janelas temporais torna o sistema escalonável é imediata dado que é conhecido o instante em que cada tarefa termina.



Task i	T_i	C_i	D_i
A	10	2	10
B	20	2	5
C	50	1	2
D	50	6	50
E	100	8	100
F	2000	7	100
G	2000	8	100
H	2000	8	2000

Figura 2.4: Escalonamento de tarefas periódicas num sistema TT

Em geral, é necessário mudar a tarefa em execução apenas em múltiplos dos períodos (Liu, 2000). Desta forma define-se o *clock tick* como um múltiplo do máximo divisor comum (MDC) dos períodos das tarefas. Assim, em cada *clock tick* é decidido qual a tarefa que é executada. Geralmente, segue-se o escalonamento *deadline monotonic* para decidir qual a tarefa que deve executar num dado momento (Burns & Wellings, 2001). Com este escalonador, a tarefa que possui a meta temporal mais próxima deve ser executada primeiro. A garantia de que todas as tarefas são escalonáveis é imediata, dado que se sabe exactamente em que momentos é que elas terminam.

Dado que o escalonamento é estático, a incorporação de mais tarefas requere refazer toda a análise. Assim, é extremamente exigente a adição de novas funcionalidades a um sistema existente. Por outro lado, tem-se a garantia de que uma tarefa mal-comportada não interfere temporalmente com as restantes tarefas.

A complexidade deste escalonamento é extremamente elevada dado que analisa todo o hiper-período em passos definidos pelo MDC dos tempos de execução das tarefas (Liu, 2000). Existem métodos que melhoram a velocidade de análise, como a definição de *frames* (Burns & Wellings, 2001; Liu, 2000), mas os métodos continuam ainda dependentes da dimensão do hiperperíodo e com iterações muito lentas.

2.4.3 Arquitectura Event-Triggered

Os sistemas ET são geralmente notificados de um acontecimento externo (evento) através da despoletação de uma interrupção (Burns & Wellings, 2001). Estas interrupções são processadas imediatamente após a sua activação, independentemente da contexto de execução do CPU. Regra geral, as tarefas em execução sofrem uma interferência temporal correspondente ao tempo de processamento da ISR (*Interrupt Service Routine*). Dado que os eventos tipicamente não possuem uma natureza intrinsecamente determinista, o seu processamento através deste mecanismo pode comprometer os pressupostos de pior caso do sistema, particularmente em situações de falhas, i.e., sobrecarga de eventos gerados.

Em sistemas distribuídos, o envio de mensagens na arquitectura ET é apenas conhecido pelo emissor. Desta forma, de modo a garantir a entrega de mensagens dentro de uma meta temporal, é necessário dotar a rede de mecanismos adicionais, como por exemplo, a associação de níveis de prioridade a cada mensagem. A abordagem comum é a utilização de “field-buses”, em particular do CAN (Controller Area Network). Esta infra-estrutura estabelece níveis de prioridade para cada mensagem, sendo que a mensagem mais prioritária será a enviada de seguida. Esta funcionalidade permite um tratamento do pior cenário em termos temporais (pior tempo de transmissão de uma mensagem) (Tindell *et al.* , 1995).

2.4.4 Escalonamento na Arquitectura Event-triggered

Dada a sua natureza dinâmica, o escalonador de tarefas em sistemas *event-triggered* é caracterizado por um conjunto de parâmetros que afectam a complexidade da análise de escalonabilidade, em contraposição com a facilidade da sua implementação. Os escalonadores na arquitectura ET são baseados em prioridades, i.e., quando é necessário mudar de tarefa, escolhe-se a tarefa em estado executável com a maior prioridade. Os escalonadores são caracterizados pela escolha dos elementos:

- prioridades fixas *vs* dinâmicas
- escalonador preemptivo *vs* não-preemptivo

A escolha de prioridades fixas/dinâmicas ou preempção/não-preempção revela como o escalonador decide qual a tarefa em execução num dado momento. Em geral, o escalonador atribui à tarefa de mais alta prioridade (num estado executável) o controlo do CPU. No caso de prioridades fixas, esta decisão é imediata, enquanto que para prioridades dinâmicas o escalonador tem que primeiro determinar a prioridade das tarefas em cada momento para depois poder escolher a de maior prioridade. No caso de escalonadores preemptivos, assim que uma tarefa de maior prioridade se tornar executável, o controlo do CPU é entregue imediatamente a tarefa. Escalonadores não-preemptivos apenas cedem o controlo do CPU a tarefas de mais alta prioridade quando a tarefa em execução liberta voluntariamente o controlo do CPU, o que diminui o número de mudanças de contexto mas também não permite um aproveitamento tão bom dos recursos.

Para além das propriedades do escalonador, é também necessário analisar os atributos das tarefas:

- tarefas síncronas *vs* assíncronas
- tarefas periódicas *vs* esporádicas
- relação da meta temporal (D_i) com o período (T_i) de cada tarefa
  - $D_i = T_i$
  - $D_i \leq T_i$
  - D_i ? T_i

As tarefas síncronas não definem o instante de activação da primeira vez que são executadas. Como tal, a análise de escalonabilidade é bastante mais simples, pois é assumido que são activadas no pior instante possível: instante crítico[1]. Tarefas assíncronas, por outro lado, são activadas em instantes definidos e conhecidos pelo arquitecto de sistema. A análise de escalonabilidade de tarefas assíncronas é extremamente mais complexa pois, em geral, é necessário analisar todo o hiperperiodo.

Tarefas periódicas são, como o nome indica, activadas periodicamente. Como tal, conhecido o instante da primeira activação podem ser determinados todos os instantes de activação futuros. Para as tarefas esporádicas é estabelecido apenas um tempo mínimo entre duas activações consecutivas: MIT (Minimum Inter-arrival Time). É possível estabelecer as quatro combinações possíveis entre o sincronismo e a periodicidade das tarefas:

- periódicas síncronas
- periódicas assíncronas
- esporádicas síncronas
- esporádicas assíncronas

Enquanto que existe uma diferença substancial entre tarefas periódicas síncronas e assíncronas, existe pouca utilidade na especificação do primeiro instante de activação das tarefas esporádicas. De facto, não é usual esta especificação nas aplicações de tempo-real. Como tal, as tarefas esporádicas assíncronas não são normalmente contabilizadas.

É de salientar também a semelhança a nível da análise de escalonamento entre as tarefas periódicas síncronas e esporádicas (síncronas). Muitas vezes (tanto quanto o nosso conhecimento permite dizer, em todos os testes existentes de escalonabilidade), as tarefas esporádicas são tratadas da mesma forma que as tarefas periódicas síncronas, com o mesmo instante de activação (instante crítico) e com o menor período possível (MIT). A semelhança é de tal forma que em (Liu, 2000) o autor confunde as tarefas periódicas com esporádicas[2], dizendo que as tarefas periódicas[3] podem

[1] Todas as tarefas são lançadas simultaneamente.
[2] Em (Liu, 2000) as tarefas esporádicas possuem outra definição.
[3] Em (Liu, 2000) o autor não distingue tarefas periódicas síncronas de assíncronas, considerando apenas o primeiro caso.

ser interpretadas como esporádicas, i.e., duas activações consecutivas podem estar separadas por um intervalo de tempo maior que o período. Do ponto de vista do escalonamento, é portanto necessário distinguir claramente se se trata de tarefas periódicas síncronas ou assíncronas.

Por último, a meta temporal associada a cada tarefa é também de grande importância para a análise de escalonabilidade. Em geral, os testes onde a meta temporal coincide com o período (ou MIT no caso de tarefas esporádicas) são os mais simples (Burns & Wellings, 2001). Quando é permitido que a meta temporal seja inferior ao período, a complexidade da escalonabilidade aumenta. Quando não existe relação entre a meta temporal e o período, os testes são ainda mais complexos.

Para além destas características, existem ainda outros elementos que têm que ser incorporados na análise de escalonabilidade: sincronização entre tarefas (acesso a um recurso comum); interferência das interrupções; mudanças de contexto; etc.

Dada a grande variedade de parâmetros, os testes que produzem condições necessárias e suficientes para concluir se uma tarefa é escalonável variam entre complexidade $O(N)$ até testes co-NP-hard no sentido forte. Tome-se como exemplo o escalonador EDF (Earliest Deadline First) preemptivo, prioridades dinâmicas, tarefas periódicas síncronas e $D_i = T_i$. O teste necessário e suficiente para que todas as tarefas sejam escalonáveis consiste em verificar a condição

$$\sum_{i=1}^{N} \frac{c_i}{T_i} \leq 1 \tag{2.1}$$

onde c_i corresponde ao pior tempo de execução (WCET - Worst Case Execution Time) e T_i ao período da tarefa Γ_i. Permitindo apenas que as metas temporais sejam inferiores ao período, a complexidade da mesma análise torna-se co-NP-hard no sentido forte (Leung & Whitehead, 1982).

A análise da transmissão de mensagens por uma rede *field-bus* como o CAN requer conhecer as características da rede (Ferreira *et al.* , 2002; Pedreiras *et al.* , 2002). Neste caso, trata-se de um escalonador de prioridades fixas não-preemptivo (a transmissão de uma mensagem de baixa prioridade não é interrompida por outra mensagem de alta prioridade) (Almeida & Fonseca, 2001). As mensagens são tratadas como tarefas, podendo ser síncronas/assíncronas; periódicas/esporádicas; e qualquer tipo de metas temporais (menores/iguais/maiores que o período).

2.4.5 Comparação

A arquitectura TT revela um comportamento temporal extremamente rígido, levando a sistemas bem comportados temporalmente mas com uma grande inflexibilidade para incorporar novos serviços, já que requer refazer todo o escalonamento. Este escalonamento requer um tempo significativo de análise off-line, bem como um espaço de memória substancial para poder armazenar todos os pontos de preempção durante um hiperperiodo.

Nos sistemas ET não é necessário um planeamento tão minucioso, dado que a execução das tarefas é função dos requisitos dinâmicos do sistema. No entanto, a verificação do comportamento temporal correcto de um sistema ET requer, em geral, um estudo mais aprofundado (Lonn & Axelsson, 1999).

Os sistemas distribuídos podem ser realizados através de uma arquitectura ET ou TT. Num sistema TT, dada uma especificação do pior comportamento a nível de execução, é necessário um estudo cuidadoso durante a fase de desenho. Como resultado, o sistema acaba por ser temporalmente extremamente determinístico. Por outro lado, os sistemas ET não requerem este estudo minucioso mas possuem um comportamento temporal mais dinâmico.

Do ponto de vista da utilização de recursos, os sistemas ET possuem uma vantagem clara: apenas são processados os eventos despoletados; enquanto que os sistemas TT analisam sempre o sistema, mesmo que não seja necessário qualquer tipo de processamento. Tome-se o caso do accionamento do airbag de um automóvel: o evento correspondente é activado apenas uma vez (para efeitos práticos) enquanto que necessita de uma resposta extremamente rápida. Num sistema ET este accionamento seria modelado como uma tarefa esporádica de alta prioridade. Como resultado, no pior caso, as tarefas de menor prioridade sofrem a interferência temporal de apenas **uma** execução desta tarefa. Por outro lado, num sistema TT é necessário uma tarefa de *polling* de muito alta frequência, levando a que as tarefas de menor prioridade sejam constantemente interrompidas para que possa ler o estado do sensor.

Os sistemas TT são intrinsecamente pessimistas: são desenhados baseados para o pior cenário possível, i.e., quando todas as falhas ocorrem simultaneamente. Como tal, o sistema acaba muitas vezes sobre-dimensionado. Por outro lado, os sistemas ET podem ser desenhados com base no pressuposto de um número variável de falhas simultâneas. Usualmente, utiliza-se o esquema de falha

única pois, sendo a probabilidade de uma falha reduzida, a probabilidade da ocorrência simultânea de duas falhas é extremamente baixa.

É possível a co-existência das duas filosofias no mesmo sistema (Kopetz, 1997), por exemplo, sendo os sistemas embebidos governados por uma filosofia ET, de forma a maximizar as suas capacidades, enquanto que a plataforma de comunicação segue uma arquitectura TT, de maneira a isolar temporalmente os componentes intervenientes. Um outro exemplo utiliza as duas filosofias no mesmo sistema computacional: a norma ARINC 653. Nesta especificação aparece o conceito de partição. Uma partição é escalonada temporalmente através da arquitectura TT, i.e., é definido off-line os períodos de execução de cada partição. Dentro de cada partição existem um conjunto de tarefas governadas por um escalonador preemptivo de prioridades fixas (arquitectura ET).

2.5 Suporte de Sistema Operativo

A maior parte dos sistemas ET tem como suporte um Sistema Operativo de Tempo-Real (RTOS - Real-Time Operating System). A inclusão deste componente reduz drasticamente o custo da construção da aplicação, uma vez que fornece um amplo conjunto de serviços, tais como a capacidade de sincronização/comunicação entre tarefas, gestores de dispositivos de E/S (*device drivers*). Actualmente, constituem peças fundamentais na construção de sistemas de tempo-real.

2.5.1 Especificações Existentes

De modo a aumentar a portabilidade de aplicações para diversos Sistemas Operativos foram criadas normas que estabelecem uma interface bem definida. Como exemplos tem-se a especificação POSIX ou mesmo a ARINC 653. Enquanto que a primeira define primordialmente um conjunto de primitivas pelas quais o programador deve aceder ao Sistema Operativo, a segunda estabelece, para além das primitivas, um comportamento temporal e espacial que os componentes devem obedecer.

POSIX

A interface POSIX (Portable Operating System Interface) é baseada nos sistemas UNIX (ANSI/IEEE, 1993). Foi criada com o objectivo de aumentar a portabilidade das aplicações ao nível do código fonte (IEE, 2004; IEE, 2003). É patrocinado pela IEEE e pelo "The Open Group".

A interface standard do POSIX é pesada demais para poder ser utilizada em sistemas embebidos, pelo que foi criada uma alternativa mais leve: POSIX.13. Esta nova norma foi especificada para sistemas com poucos recursos computacionais e com características de tempo-real, como por exemplo, a definição de prioridades das tarefas ou a criação de semáforos com mecanismos de herança/tecto de prioridade, etc.

ARINC 653 e APEX

A especificação ARINC 653 foi criada no seio da indústria aeronáutica como suplemento à construção de aplicações críticas. Um conceito fundamental desta norma é a partição: uma partição corresponde a um conjunto de tarefas que executa num espaço de memória próprio e em períodos de tempo bem definidos. Assim, uma partição não pode interferir nem temporalmente nem espacialmente com outra, diminuindo o risco de propagação de falhas de um componente para outro.

Esta norma estabelece uma interface rígida com o *kernel* do Sistema Operativo denominada APEX (APplication EXecutive). Esta interface possui um conjunto de primitivas de sincronização/comunicação entre tarefas, comunicação entre partições, etc.

2.5.2 Sistemas Operativos de Tempo-Real

Como exemplo de sistemas operativos de tempo-real encontram-se o eCos e o RTEMS (Real-Time Executive for Multiprocessor Systems).

eCos

O sistema operativo de tempo-real eCos é um sistema *open-source* no âmbito da GNU. Como tal, os programadores possuem acesso a todo o código fonte do sistema operativo.

O eCos é desenhado de forma a ser facilmente portável para várias plataformas computacionais, incluindo arquitecturas de 16, 32 e 64 bits, MPUs, MCUs e DSPs. Actualmente o eCos suporta dez arquitecturas computacionais, incluindo a Intel x86 e a SPARC, incluindo as variantes destas arquitecturas utilizadas na indústria espacial (LEON, ERC32, etc).

O eCos encontra-se desenhado para suportar aplicações de tempo-real, suportando, por exemplo, um escalonador preemptivo, tendo a preocupação de providenciar uma latência de interrupções baixa, várias primitivas de sincronização/comunicação entre tarefas, tratamento de interrupções, etc. O eCos suporta também um grande leque de gestores de dispositivos (*device drivers*), bibliotecas adicionais, tratamento de excepções, gestão de memória dinâmica, etc. Para além do suporte em tempo de execução, as ferramentas existentes para o desenvolvimento de aplicações embebidas, nomeadamente a configuração, compiladores, assemblers, linkers, debuggers e simuladores, são também fornecidas ou pela GNU ou pelo próprio eCos.

O eCOs suporta as API ITRON e POSIX, para além da interface nativa. Possui também um grande leque de protocolos de comunicação, tais como o stack TCP/IP, SNMP, FTP, HTTP, etc.

RTEMS

O RTEMS é um sistema operativo de tempo-real open-source e license-free mantido pela OAR (RTE, 2003a). Foi desenhado inicialmente pelo Departamento de Defesa dos EUA onde adquiriu a designação de Real-Time Executive for Missile Systems. Cedo se concluiu que a sua aplicação podia ser estendida para além dos mísseis, pelo que se renomeou para Real-Time Executive for Military Systems. Depois da sua manutenção ter sido transferida para a OAR tomou a sua designação actual: Real-Time Executive for Multiprocessor Systems.

De acordo com (Straumann, 2001), possui aspectos temporais adequados para sistemas de tempo-real, e.g., tempos de comutação de tarefas, latência de interrupções[4], etc. Oferece também um vasto leque de serviços de gestão de tarefas, dispositivos de E/S, tratamento de interrupções, gestão de memória e comunicação multiprocessador.

Em particular, a gestão de tarefas inclui primitivas que acedem a semáforos com características

[4]Tempo entre o despoletamento da interrupção e a execução da primeira instrução da ISR da aplicação.

(opcionais) de herança/tecto de prioridade, comunicação através da passagem de mensagens por filas de espera, estabelecimento de timers/watchdogs, tratamento de sinais (signals), etc.

Para além de ter sido recentemente certificado para aplicações do domínio espacial (Seronie-Vivien & Cantenot, 2005), existem também outras aplicações de tempo-real que o utilizam, sendo exemplos os sistemas

- Avenger Forward Air Defense System
- Surrey Satellite Solid State Data Recorder, SurreySSDR
- EPICS

A construção do RTEMS revela uma arquitectura modular, facilitando a integração de novas arquitecturas computacionais. Existem actualmente 14 famílias de CPU suportadas, cada uma das quais com diferentes BSP, incluindo a conhecida Intel-PC386 (RTE, 2003b), utilizada nos computadores pessoais e útil para testes rápidos, e a SPARC-ERC32 (RTE, 2003d), utilizada em sistemas espaciais.

Para além da interface nativa, o RTEMS proporciona as interfaces POSIX (RTE, 2003c) e ITRON, podendo ser programado nas linguagens C/C++ ou ainda em Ada (RTE, 2003a).

3

Escalonamento em Sistemas de Controlo de Tempo-Real

Em sistemas de controlo distribuído o cálculo de resultados correctos é tão importante como apresentá-los dentro de uma meta temporal. Como exemplo, tome-se o sistema de refrigeração de uma central nuclear: caso a acção de controlo seja retardada, a central pode ficar em perigo.

Sistemas que reagem a eventos (sistemas *event-triggered*) necessitam de uma análise de escalonabilidade mais complexa para garantir que as tarefas cumprem sempre as metas temporais correspondentes.

A escalonabilidade de um conjunto de tarefas não é, em geral, trivial. Existe um conjunto de factores que determinam a complexidade da análise (Liu, 2000; Burns & Wellings, 2001; Jeffay *et al.* , 1991):

- prioridades fixas ***versus*** dinâmicas
- escalonador preemptivo ***versus*** não-preemptivo
- tarefas assíncronas ***versus*** síncronas
- tarefas periódicas ***versus*** esporádicas
- meta temporal ($D_i = T_i$ ***versus*** $D_i \leq T_i$ ***versus*** D_i ? T_i)

Por exemplo, uma condição necessária e suficiente para um conjunto de tarefas escalonadas pelo algoritmo EDF preemptivo, com tarefas periódicas, síncronas e com $D_i = T_i$ é trivial (Liu & Layland, 1973; Lehoczky *et al.* , 1989; Audsley *et al.* , 1991)

$$\sum_{i=1}^{N} U_i \leq 1 \tag{3.1}$$

onde U_i corresponde à percentagem de utilização da tarefa. Por outro lado, o mesmo tipo de condição para o mesmo escalonador mudando apenas a última condição para $D_i \leq T_i$ necessita de uma análise necessária e suficiente com complexidade co-NP-hard no sentido forte (Leung & Whitehead, 1982; Leung & Merrill, 1980a; Baruah *et al.* , 1990). Para além destas cinco características, é também necessário tomar em conta outros factores associados a cada implementação específica: tempos de mudanças de contexto entre tarefas, interferência temporal de interrupções, interacções entre tarefas, acessos a recursos partilhados, etc, (Liu, 2000).

Actualmente, a maioria dos sistemas de controlo distribuído são compostos por um escalonador com os seguintes parâmetros:

- prioridade fixa
- preemptivo

Escalonadores de prioridades fixas são muitas vezes escolhidos em detrimento dos de prioridade dinâmica pois são mais fáceis de implementar e com o comportamento mais determinístico, i.e., uma tarefa apenas sofre interferências de outras tarefas de prioridade superior (à parte de eventuais situações de inversão de prioridade devido a acessos a recursos partilhados).

Escalonadores preemptivos, embora necessitem geralmente de mais mudanças de contexto, permitem um melhor aproveitamento dos recursos e maior separação temporal entre as tarefas, isto é, quando uma tarefa de alta prioridade se torna executável passa imediatamente para o estado em execução. No caso de sistemas não preemptivos poderia ter que esperar que uma tarefa de baixa prioridade em execução terminasse. Como consequência, em sistemas preemptivos uma tarefa apenas sofre interferência temporal das tarefas de maior prioridade, enquanto que sistemas não preemptivos uma tarefa pode também sofrer interferência de tarefas de menor prioridade[1].

Em relação às características das tarefas, os sistemas de controlo distribuído necessitam geralmente de tarefas com os seguintes parâmetros:

[1] No pior caso sofre a interferência da tarefa de menor prioridade com maior tempo de execução

- tarefas periódicas e esporádicas
  - tarefas periódicas assíncronas
  - tarefas esporádicas síncronas
- meta temporal: $D_i \leq T_i$

Não existem vantagens em analisar tarefas esporádicas assíncronas pelo que se considera que são síncronas.

Os sistemas de controlo são compostos por tarefas, tanto periódicas como esporádicas. Estes dois tipos de tarefas são extremamente úteis para um sistema de controlo. Como exemplos têm-se o controlo de admissão de combustível num automóvel, realizado por tarefas periódicas, e o despoletamento do airbag, tipicamente activado por uma tarefa esporádica.

Tarefas assíncronas, ao contrário do que o nome possa indicar, possuem uma caracterização rígida sobre o instante da **primeira activação** de cada tarefa. Por outro lado, as tarefas síncronas **não definem o primeiro instante de activação**. No caso das tarefas síncronas, dado que podem ser activadas em qualquer instante, assume-se que o são no pior instante possível: instante crítico[2]. Como tal, em sistemas síncronos é necessário analisar apenas um caso, reduzindo consideravelmente a complexidade da análise de escalonamento mas produzindo resultados pessimistas. Por outro lado, as tarefas periódicas assíncronas requerem uma análise para todas as activações dentro do hiperperiodo. No entanto, os sistemas de controlo actuais requerem longos períodos de construção, o que permite que uma análise mais demorada de escalonamento possa ser feita. Dado que as tarefas assíncronas não introduzem o pessimismo imposto pelo pressuposto de um instante crítico, produzem resultados melhores, levando a que mais sistemas se afirmem como escalonáveis. Como não existem vantagens em modelar tarefas esporádicas assíncronas, opta-se por modelar tarefas periódicas assíncronas e esporádicas síncronas.

Por último, as metas temporais são normalmente inferiores ao período. No caso de tarefas esporádicas tem-se a meta temporal inferior ao tempo mínimo entre duas activações consecutivas. Em sistemas de controlo distribuído este tipo de metas temporais é muitas vezes utilizado para poder

[2]Todas as tarefas são activadas simultaneamente

garantir um tempo de resposta extremamente baixo, como por exemplo, despoletamento do airbag, controlo de sistemas com uma dinâmica rápida, etc.

Em resumo, os sistemas de controlo são usualmente compostos por:

- escalonador de prioridades fixas preemptivo
- tarefas periódicas e esporádicas
  - tarefas periódicas assíncronas
  - tarefas esporádicas síncronas
- meta temporal: $D_i \leq T_i$

A diferença principal entre este sistema e os discutidos na literatura reside no tratamento de tarefas assíncronas periódicas em vez de síncronas periódicas. No entanto, como se verá mais adiante, é trivial a sua implementação num sistema de controlo enquanto que produz resultados consideravelmente melhores em termos de escalonamento. Note-se, no entanto, que esta "pequena" alteração modifica a complexidade da análise de escalonabilidade de pseudo-polinomial para co-NP-hard no sentido estrito.

Este capítulo trata o escalonamento deste tipo de sistemas através de um método vulgarmente conhecido como RTA (Response Time Analysis). Este método calcula o tempo de resposta de cada tarefa individualmente. Assim, se o tempo de resposta for menor que a meta temporal, é garantida a escalonabilidade. Caso contrário, o conjunto de tarefas não é escalonável.

3.1 Motivação

A especificação dos tempos de activação das tarefas assíncronas periódicas reduz o pessimismo introduzido pelo pressuposto do instante crítico. A abolição desta restrição conduz a resultados consideravelmente melhores, pois, por exemplo, permite que a análise de escalonamento estabeleça que algumas tarefas nunca interfiram entre si, mesmo com períodos diferentes, Figura 3.1.

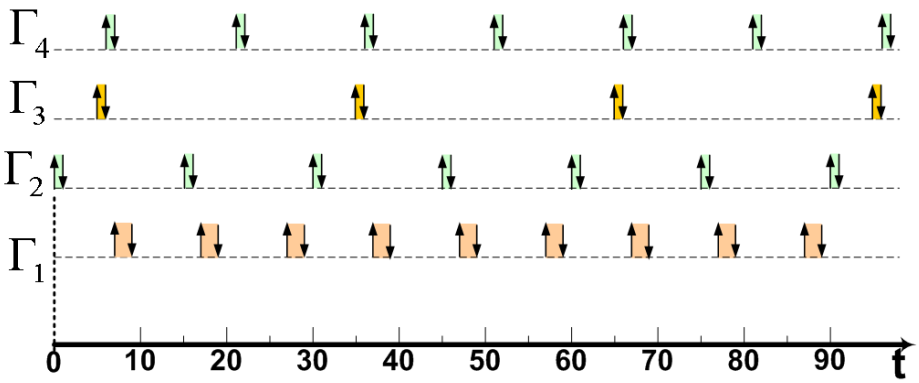


Figura 3.1: Tarefas assíncronas periódicas sem interferência entre si

Sob o ponto de vista de implementação num RTOS, é relativamente imediata a especificação dos instantes de activação das tarefas periódicas. A esmagadora maioria dos RTOS possui uma funcionalidade mínima que permite forçar a activação de uma tarefa num instante pré-definido de tempo (Audsley, 1991; Audsley, 2001; Grenier *et al.* , 2006; Devillers & Goossens, 1999; Goossens, 1999; Goossens & Devillers, 1999) .

É de salientar que o determinismo que advém do conhecimento dos instantes de activação das tarefas periódicas produz resultados comparáveis aos obtidos com a arquitectura *time-triggered*. As diferenças que ainda se mantêm advêm da interferência temporal causada pela aplicação: cenários de herança de prioridade, interrupções, etc. No entanto, vários estudos foram realizados para integrar estas interferências na análise do escalonamento (Sha *et al.* , 1990; Sandtrom *et al.* , 1998; Burns & Wellings, 2001). Como vantagem adicional em relação aos sistemas TT, tem-se o reduzido número de parâmetros necessários para caracterizar uma tarefa periódica (prioridade; período), enquanto que um sistema TT necessita de uma tabela de escalonamento construída na fase de inicialização, a indicar qual a tarefa que vai entrar em execução em cada instante de preempção. Mesmo para um número reduzido de tarefas, esta tabela pode facilmente atingir tamanhos consideráveis dado que depende do tamanho do hiperperiodo.

Embora a especificação dos instantes de activação crie um determinismo acrescido, com as vantagens que daí advêm, implica também um número significativamente maior de cálculos, pois é necessário a determinação do tempo de resposta de cada activação dentro de um hiperperiodo, isto é, até que o sistema se repita no tempo. Como tal, não é esperado que este possa ser um teste

realizado em tempo de execução, mas predominantemente numa fase prévia de implementação (**?**; **?**). Apesar dos recursos temporais necessários à análise de escalonamento, a esmagadora maioria das aplicações de tempo-real necessitam de longos períodos de desenho/implementação, tornando o tempo necessário à análise de escalonabilidade desprezável. Como exemplo de domínios de tempo-real com estas propriedades têm-se os sistemas espaciais, ramo automóvel, sistemas aeronáuticos, sistemas fabris, etc.

A análise proposta determina o tempo de resposta de cada tarefa que, para além da análise de escalonabilidade, permite também estabelecer outras considerações sobre o sistema:

- Determinação da dispersão (*jitter*) das respostas de uma tarefa ao longo do hiperperiodo
- Suporte a sistemas que permitem falhas ocasionais da meta temporal

Um sistema de controlo distribuído possui vários elementos que causam interferências temporais desde o instante de leitura do(s) sensor(es) até que a resposta correspondente é colocada no(s) actuador(es). O atraso imposto deve-se ao tempo do processamento inicial da leitura, de comunicação da rede, de cálculo da acção e finalmente ao de actuação. Usualmente, o atraso do sistema computacional é modelado no sistema de controlo por um intervalo de tempo múltiplo do período de amostragem, Figura 3.2. No entanto, dada a variabilidade temporal dos componentes computacionais, a resposta à leitura do sensor pode sofrer atrasos que não são contabilizados no modelo de controlo, podendo mesmo conduzir à instabilidade do sistema (Franklin *et al.* , 2001). É extremamente útil determinar qual a dispersão (*jitter*) provocada pelo sistema computacional e determinar como interfere com o dinamismo da aplicação (Shin & Chui, 1995; Cervin, 2001).

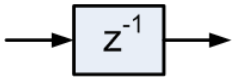


Figura 3.2: Modelação do Atraso em Sistemas de Controlo

Em sistemas de controlo distribuído (mas não só), existem aplicações não críticas cuja falha ocasional de uma meta temporal é permitida. A análise proposta permite dizer com precisão quais as execuções propensas ao não cumprimento da meta temporal (se o tempo de execução for menor que

o WCET, então o sistema real pode cumprir as metas). No entanto, esta análise apenas se mantêm correcta se o tempo de resposta for menor que o período, caso contrário, a próxima execução da tarefa tem uma interferência adicional da anterior, que não se encontra contabilizada nos métodos propostos (assume-se que as metas temporais são inferiores ao período).

A inclusão de tarefas esporádicas acrescenta um novo grau de complexidade. Como se tornará claro mais adiante, o escalonamento das tarefas assíncronas periódicas sofre ligeiras alterações que contabilizam a interferência das tarefas esporádicas de maior prioridade. Por outro lado, o escalonamento das tarefas esporádicas é mais complexo na medida em que o pior instante de activação é desconhecido. Dado que as tarefas esporádicas podem ser activadas em qualquer instante, é necessário determinar qual o pior instante possível e analisar se essa execução cumpre a meta temporal. Este pior instante de activação imposto pelas tarefas assíncronas periódicas de maior prioridade é denominado de *instante crítico assíncrono*.

Uma outra vantagem fundamental deste modelo consiste na aproximação entre o determinismo temporal dos sistemas *event-triggered* e sistemas *time-triggered*. Desta forma é possível uma análise de escalonamento de um sistema que interligue ambos: ARINC 653. Esta análise encontra-se detalhada no capítulo seguinte.

3.2 Modelo de Sistema

O modelo de sistema considerado inclui tarefas assíncronas periódicas e esporádicas, escalonadas por um algoritmo de prioridades fixas preemptivo e com metas temporais gerais (menores que o período). Como uma primeira aproximação, é assumido que as tarefas são independentes, co-existem num único processador e o tempo de mudança de contexto é nulo. Cada tarefa possui uma prioridade única, i.e., não existem duas tarefas com a mesma prioridade. No entanto, tarefas periódicas e esporádicas podem entre-cruzar as suas prioridades, i.e., a prioridade de uma dada tarefa esporádica pode estar no meio das prioridades de duas tarefas periódicas e vice-versa.

No que diz respeito ao conjunto de tarefas periódicas, cada tarefa é caracterizada pelos parâmetros $\Gamma_i = \{c_i, T_i, r_i, D_i\}$ onde c_i representa o WCET (Worst Case Execution Time), T_i o período da tarefa, r_i o instante da primeira activação da tarefa e D_i a meta temporal relativa ao instante de activação. A

condição de metas temporais estabelece que a meta temporal é menor ou igual ao período, logo os parâmetros estão condicionadas a $0 \leq c_i \leq D_i \leq T_i$ e $0 \leq r_i$.

Da mesma forma, cada tarefa esporádica também é caracterizada por $\tau_m = \{e_m, MIT_m, H_m\}$ onde e_m representa o WCET, MIT_m o Minimum Inter-Arrival Time entre duas activações consecutivas e H_m a meta temporal relativa ao instante de activação. Tal como as tarefas periódicas, os parâmetros das tarefas esporádicas estão condicionados a $0 \leq e_m \leq H_m \leq MIT_m$. Note-se a semelhança entre os dois conjuntos de tarefas, sendo as diferenças oriundas da sua periodicidade (T_i *vs* MIT_m) e pela definição dos instantes de activação para as tarefas assíncronas periódicas.

Definições Adicionais

- $\overline{R}_i$ - Tempo (instante) de resposta da tarefa síncrona periódica (a contra-parte da tarefa assíncrona periódica Γ_i)
- ${}^{l}r_i = r_i + l\,T_i$ - Instante de activação da execução l da tarefa periódica assíncrona Γ_i
- ${}^{l}R_i$ - Instante de resposta da execução l da tarefa assíncrona periódica Γ_i
- R_m - Instante de resposta da tarefa esporádica τ_m
- ${}^{l}R_i - {}^{l}r_i$ - Tempo de resposta da execução l da tarefa assíncrona periódica Γ_i
- Λ_i - Hiperperiodo da tarefa assíncrona periódica Γ_i

3.3 Análise RTA de Tarefas Periódicas Síncronas

A análise realizada neste capítulo parte do método RTA - *Response Time Analysis* (Joseph & Pandya, 1986; Leung & Merrill, 1980b). Esta secção descreve-o em pormenor para tarefas síncronas para estabelecer um ponto de entrada para o restante trabalho.

A análise RTA descrita nesta secção é destinada a sistemas com os seguintes parâmetros

- escalonadores de prioridades fixas e conhecidas

- escalonadores preemptivos
- tarefas periódicas e esporádicas
  - tarefas periódicas síncronas
  - tarefas esporádicas síncronas
- $D_i \leq T_i$

Como se pode concluir, a diferença entre este modelo e o modelo que se pretende estudar reflecte-se na mudança entre tarefas periódicas síncronas e periódicas assíncronas. Esta "ligeira" alteração é, no entanto, suficiente para alterar a complexidade de pseudo-polinomial para co-NP-hard no sentido estrito.

Na análise de tarefas síncronas, é assumido que são todas activadas simultaneamente (instante crítico). Este instante é normalizado para zero, de forma a facilitar os cálculos, logo é assumido que $r_1 = r_2 = ... = r_N = 0$. Como tal, o tempo de resposta de uma tarefa periódica síncrona, $\overline{R}_i$, é a soma do seu tempo de execução, c_i, com a soma da interferência causada por cada uma das tarefas de mais alta prioridade no intervalo $[0, \overline{R}_i[$, dado por $w_{i-1}(\overline{R}_i)$. Logo, a resposta é dada por

$$\overline{R}_i = c_i + w_{i-1}\left(\overline{R}_i\right) \tag{3.2}$$

A interferência das tarefas de mais alta prioridade torna-se o elemento mais complexo de determinar, pois depende do tempo de resposta. Isto é, se o tempo de resposta for maior, também é aumentada a interferência pois algumas tarefas podem se ter activado novamente. Sendo $n_i(t)$ o número de vezes que a tarefa Γ_i tornou-se activa dentro do intervalo $[0, t[$, tem-se

$$n_i(t) = \left\lceil \frac{t}{T_i} \right\rceil \tag{3.3}$$

onde o operador não-linear "tecto" $\lceil t \rceil$ retorna o menor inteiro acima de t. Assim, tem-se por exemplo $\lceil 5/10 \rceil = 1$ ou $\lceil 15/10 \rceil = 2$, ou seja, para $t = 5$ houve um activação da tarefa com período 10 e duas activações para $t = 15$. Assim, tem-se o trabalho pedido pela tarefa Γ_i no intervalo $[0, t[$, $I_i(t)$, dado por

$$I_i(t) = n_i(t)c_i = \left\lceil \frac{t}{T_i} \right\rceil c_i \tag{3.4}$$

Somando a interferência das tarefas tem-se

$$w_i(t) = \sum_{j=1}^{i} I_j(t) = \sum_{j=1}^{i} \left\lceil \frac{t}{T_j} \right\rceil c_j \tag{3.5}$$

Interligando esta equação com a 3.2 obtêm-se

$$\overline{R}_i = c_i + \sum_{j=1}^{i-1} \left\lceil \frac{\overline{R}_i}{T_j} \right\rceil c_j \tag{3.6}$$

O teste de escalonabilidade fica reduzido a verificar se $\overline{R}_i \leq D_i$. Se a condição for verdadeira, a tarefa é escalonável qualquer que seja o instante de activação. Caso contrário, a tarefa não é escalonável.

Dada a presença do operador não-linear "tecto", é complexo resolver esta equação numa forma fechada. Em vez disso, é utilizado um método iterativo de ponto-fixo (Joseph & Pandya, 1986):

$$\begin{cases} \overline{R}_i^{(0)} = c_i \\ \overline{R}_i^{(n+1)} = c_i + \sum_{j=1}^{i-1} \left\lceil \frac{\overline{R}_i^{(n)}}{T_j} \right\rceil c_j \end{cases} \tag{3.7}$$

Quando $\overline{R}_i^{(n+1)} = \overline{R}_i^{(n)} = \overline{R}_i$, o algoritmo pára. Este método converge para o tempo de resposta da tarefa Γ_i e possui uma complexidade pseudo-polinomial (Joseph & Pandya, 1986). A Figura 3.3 mostra um exemplo da progressão das iterações segundo o método descrito pela equação 3.7.

Basicamente o método de ponto-fixo calcula em cada iteração o instante onde todo o trabalho pedido pelas tarefas acaba. No entanto, entre a última iteração e a próxima, algumas tarefas podem ter requisitado mais trabalho. O método continua a incrementar o tempo de resposta até que o trabalho pare de aumentar.

A integração de tarefas esporádicas é trivial: são tratadas como tarefas periódicas com período igual ao MIT. Desta forma existe pouca diferença entre a análise de escalonabilidade de tarefas es-

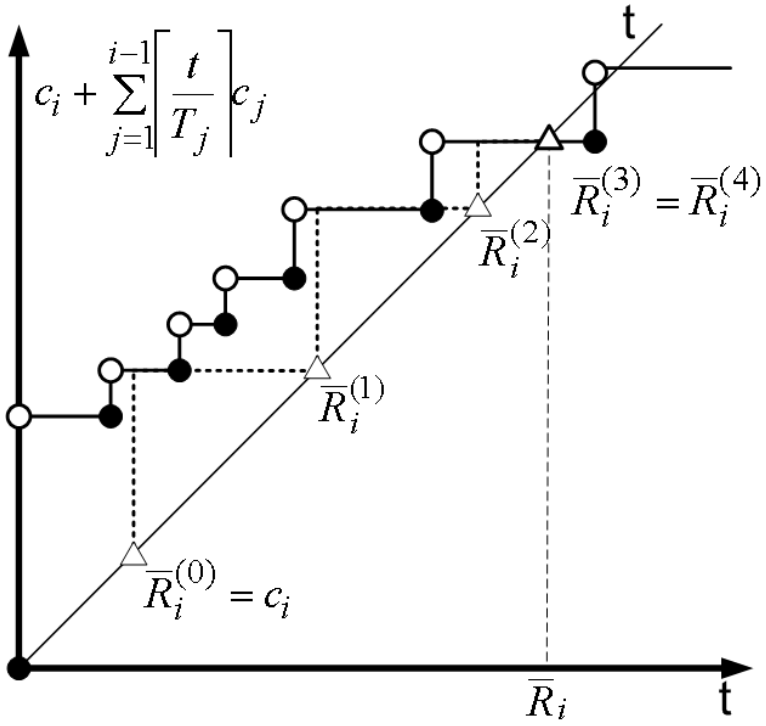


Figura 3.3: Iterações do cálculo do tempo de resposta - equação 3.7

porádicas e periódicas síncronas. Advêm daí a análise destes dois tipos de tarefas como um só (Liu, 2000).

3.4 Análise RTA de Tarefas Periódicas Assíncronas

Esta secção apresenta um novo método para determinar os tempos de resposta de um sistema composto por tarefas periódicas assíncronas. Para facilitar a análise, a integração de tarefas esporádicas ainda não é considerada. O sistema é portanto caracterizado pelos parâmetros

- escalonadores de prioridades fixas e conhecidas
- escalonadores preemptivos
- tarefas periódicas assíncronas
- $D_i \leq T_i$

A introdução dos instantes de activação das tarefas periódicas tem que ser incorporada na função de trabalho, $w_i(t)$. Esta função retorna o tempo de processamento requerido pelas tarefas $\Gamma_1,\ldots,\Gamma_i$ no intervalo $[0,t[$.

$$w_i(t) = \sum_{j=1}^{i} \left\lceil \frac{t-r_j}{T_j} \right\rceil_0 c_j \tag{3.8}$$

onde o operador $\lceil x \rceil_0$ representa a função $max\{0;\lceil x \rceil\}$. Este operador é introduzido para especificar que o número de vezes que uma tarefa é activada não pode ser menor que zero.

O instante de resposta de uma tarefa periódica assíncrona activada em ${}^l r_i$ é obtido pela soma do seu tempo de execução com a interferência devido a tarefas de maior prioridade, $I_{i-1}({}^l r_i, {}^l R_i)$, com o instante onde foi activada, ${}^l r_i$.

$${}^l R_i = {}^l r_i + c_i + I_{i-1}({}^l r_i, {}^l R_i) \tag{3.9}$$

A interferência das tarefas de maior prioridade é o único elemento desconhecido. Esta interferência pode ser dividida entre a interferência das tarefas de maior prioridade que ainda falta executar em ${}^l r_i$, que designamos por $I^b_{i-1}({}^l r_i)$, e a interferência que surge desde o instante de activação até ao instante de resposta, $I^a_{i-1}({}^l r_i, {}^l R_i)$, correspondente ao intervalo $[{}^l r_i, {}^l R_i[$:

$$I_{i-1}({}^l r_i, {}^l R_i) = I^b_{i-1}({}^l r_i) + I^a_{i-1}({}^l r_i, {}^l R_i) \tag{3.10}$$

O termo $I^b_{i-1}({}^l r_i)$ pode ser calculado sabendo o último instante de idle antes de ${}^l r_i$, denotado $L_{i-1}({}^l r_i)$. O último instante de idle é definido como o último instante onde todo o trabalho pedido pelas tarefas de maior prioridade foi processado. Assim, desde $L_{i-1}({}^l r_i)$ até ${}^l r_i$ o sistema está a processar as tarefas $\Gamma_1,\ldots,\Gamma_{i-1}$. A interferência que resta em ${}^l r_i$ é calculada facilmente a partir do trabalho pedido no intervalo $[L_{i-1}({}^l r_i), {}^l r_i[$ e o trabalho processado.

$$I^b_{i-1}({}^l r_i) = \underbrace{w_{i-1}({}^l r_i) - w_{i-1}(L_{i-1}({}^l r_i))}_{\text{trabalho pedido}} - \underbrace{({}^l r_i - L_{i-1}({}^l r_i))}_{\text{trabalho processado}} \tag{3.11}$$

O termo da interferência que surge depois de ${}^{l}r_i$ é mais complexo de se determinar pois depende de ${}^{l}R_i$. Esta interferência contabiliza todo o trabalho pedido no intervalo $[{}^{l}r_i, {}^{l}R_i[$.

$$I^a_{i-1}({}^{l}r_i, {}^{l}R_i) = w_{i-1}({}^{l}R_i) - w_{i-1}({}^{l}r_i) \tag{3.12}$$

Juntando as equações 3.11 e 3.12 com a 3.10 e 3.9 tem-se o instante de resposta dada por

$${}^{l}R_i = L_{i-1}\left({}^{l}r_i\right) + c_i + w_{i-1}\left({}^{l}R_i\right) - w_{i-1}\left(L_{i-1}\left({}^{l}r_i\right)\right) \tag{3.13}$$

A solução mais pequena desta equação (mas maior que ${}^{l}r_i$) coincide com o instante de resposta da tarefa Γ_i. Esta equação começa no último instante de idle, ao invés de ${}^{l}r_i$ como na equação 3.9. Adiciona depois o tempo de execução da tarefa, c_i, e o trabalho pedido no intervalo $[L_{i-1}({}^{l}r_i), {}^{l}R_i[$.

Esta equação tem que ser verificada para cada execução dentro do hiperperiodo. Em (Audsley, 1991) o hiperperiodo de Γ_i corresponde a todas as activações dentro do intervalo

$$[\max(r_1,...,r_i), \max(r_1,...,r_i) + MMC(T_1,...,T_i)[$$

onde MMC denota o Mínimo Múltiplo Comum. O hiperperiodo possui a dimensão mínima até que o sistema se comece a repetir no tempo. Este hiperperiodo tem que começar depois da fase transitória inicial, onde nem todas as tarefas estão "activas", isto é, ainda não foram activadas uma única vez. Por este motivo, o hiperperiodo pode começar em qualquer instante após $\max(r_1,...,r_i)$ desde que mantenha a mesma dimensão. Devido à interferência de tarefas esporádicas, define-se o hiperperiodo começando em $S_i = \max(r_1,...,r_i) + T_i$

$$\Lambda_i = [S_i, S_i + MMC(T_1,...,T_i)[\tag{3.14}$$

Este novo intervalo mantém a mesma dimensão e começa depois da fase transitória, pelo que mantém as mesmas propriedades que o anterior. A Secção 3.5.2 explica em pormenor as razões que levam a aumentar o ponto inicial do intervalo. No entanto, como explicação breve, quando se integram tarefas esporádicas é necessário analisar o intervalo $]{}^{l-1}r_i, {}^{l}r_i]$ para determinar ${}^{l}R_i$, logo é

necessário que este intervalo não esteja na fase transitória.

Os valores mínimo e máximo de l de forma a que ${}^{l}r_i \in \Lambda_i$ são dados por

$$\begin{aligned} {}^{l}r_i = r_i + l\,T_i &\geq S_i \Leftrightarrow \\ \Leftrightarrow l_{min} &= \left\lceil \frac{S_i - r_i}{T_i} \right\rceil \end{aligned}$$

$$\begin{aligned} {}^{l}r_i = r_i + l\,T_i &< S_i + MMC(T_1,...,T_i) \Leftrightarrow \\ \Leftrightarrow l_{max} &= \left\lceil \frac{S_i + MMC(T_1,...,T_i) - r_i}{T_i} \right\rceil - 1 \end{aligned}$$

A análise de escalonabilidade pode ser resumida na condição

$$\begin{cases} \forall_{l:{}^{l}r_i \in \Lambda_i} & {}^{l}R_i - {}^{l}r_i \leq D_i \quad , \quad \Gamma_i \textbf{ é escalonável} \\ \exists_{l:{}^{l}r_i \in \Lambda_i} & {}^{l}R_i - {}^{l}r_i > D_i \quad , \quad \Gamma_i \textbf{ não é escalonável} \end{cases} \tag{3.15}$$

onde ${}^{l}R_i$ é dado pela equação 3.13.

3.4.1 Determinação do Último Instante de Idle

A análise RTA descrita anteriormente necessita da determinação do último instante de idle das tarefas de maior prioridade para um dado instante, $L_i(t)$.

O objectivo consiste em estender a função de trabalho, $w_i(t)$ com a incorporação do tempo de idle passado até t. Desta forma a função $w_i^{ext}(t)$ é criada, Figura 3.4. Quando o sistema se encontra em *idle*, a função $w_i^{ext}(t)$ é igual a t (instante t_1 na Figura 3.4). Desta forma, a $w_i^{ext}(t) - t$ resulta no trabalho que ainda falta executar em cada instante t.

Analisando a função de trabalho estendida os períodos de idle podem ser determinados e o último instante de idle determinado. Em resumo, quando o sistema se encontra em computação, o método avança no tempo com iterações semelhantes à RTA; quando o sistema está em idle, uma nova função, $\rho_i(t)$ determina qual o próximo instante de computação.

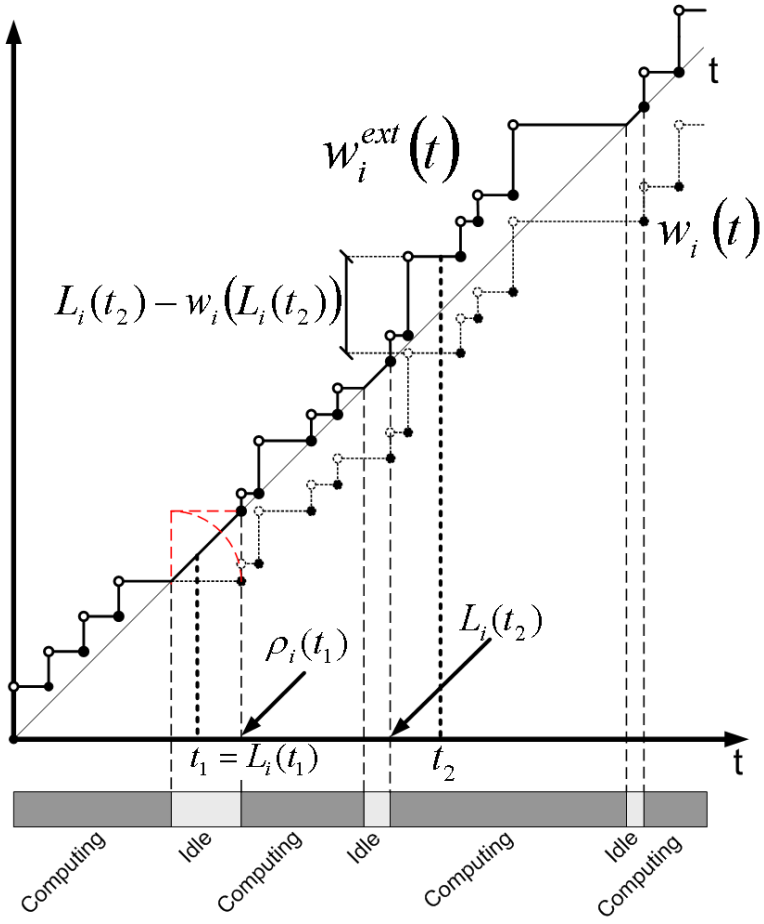


Figura 3.4: Função de Trabalho Estendida

A transição entre um período de idle para um de computação requere a determinação do trabalho pedido num instante em particular. Para isso, uma função auxiliar, $\overline{w}_i(t)$, é introduzida

$$\overline{w}_i(t) = \sum_{j=1}^{i} \left\lfloor 1 + \frac{t - r_j}{T_j} \right\rfloor_0 c_j \tag{3.16}$$

onde o operador $\lfloor x \rfloor$ representa a função "chão". Esta função retorna o trabalho pedido no intervalo $[0,t]$. Note-se que este intervalo é fechado, pelo que o trabalho num determinado instante t pode ser dado por $\overline{w}_i(t) - w_i(t)$.

O método avança pelos períodos de computação através de um método de ponto-fixo. Denote-se Υ o tempo total de idle até um determinado instante. Em cada iteração, o método de ponto-fixo determina o instante onde o trabalho pedido foi executado. Como desde a iteração anterior até à presente mais trabalho pode ter sido pedido, o método continua a acrescentar o novo trabalho pedido.

$$w_i^{(n+1)} = \sum_{j=1}^{i} \left\lceil \frac{w_i^{(n)} - r_j}{T_j} \right\rceil_0 c_j + \Upsilon \tag{3.17}$$

Quando $w_i^{(n+1)} = w_i^{(n)}$ o algoritmo pára e $w_i^{(n)}$ corresponde ao instante onde o período de computação terminou, que coincide com o início do período de idle. A Figura 3.6 ilustra as iterações até encontrar um instante de idle.

Após encontrar o instante onde o período de computação termina e o de idle começa, a função $\rho_i(t)$ determina o próximo instante de computação, que corresponde à activação mais próxima de t

$$\rho_i(t) = \min_{j=1,\ldots,i} \left\{ \left\lceil \frac{t - r_j}{T_j} \right\rceil_0 T_j + r_j \right\} \tag{3.18}$$

Sendo t o último instante de computação, da diferença entre $\rho_i(t)$ e t obtém-se a dimensão do período de idle entre os dois períodos de computação. Adicionando consecutivamente os períodos de idle a $w_i(t)$ obtém-se $w_i^{ext}(t)$, Figura 3.4.

$$w_i^{ext}(t) = w_i(t) + \underbrace{L_i(t) - w_i(L_i(t))}_{\text{tempo total de idle}} \tag{3.19}$$

Assim é construído um método iterativo que alterna entre os períodos de idle e de computação. A transição de um período de idle para um de computação é feita através da adição do trabalho pedido no instante do começo da computação que, como foi visto anteriormente, pode ser dado por $\overline{w}_i(\rho(t)) - w_i(\rho(t))$. A Figura 3.5 ilustra o pseudo-código que implementa este método.

Um exemplo das iterações deste método é apresentado na Figura 3.6.

Como se pode ver, as iterações para calcular $L_i(t)$ são semelhantes às da análise RTA, com a particularidade de somarem o tempo de idle que encontram. O método alterna entre os períodos de computação, onde as iterações são construídas partindo do trabalho pedido em cada instante, e os períodos de idle, onde a iteração toma o valor do instante seguinte de computação (mais o trabalho pedido neste instante). Quando as iterações forem maiores do que t, o método termina, retornando o último instante de idle conhecido.

```
function L_i(t){
    if( i = 0 ) return t; // there are no higher priority tasks
    last_idle_instant = 0;
    total_idle_time = 0;
    iterator = 0;
    while(true) {
        last_iterator = iterator;
        iterator = w_i(iterator) + total_idle_time;
        if(iterator > t)
            return last_idle_instant;
        if(last_iterator = iterator) { // arrived at an idle instant
            if(iterator ≤ t ≤ ρ_i(iterator))
                return t;
            last_idle_instant = ρ_i(iterator);
            total_idle_time += ρ_i(iterator) − iterator;
            iterator = ρ_i(iterator) + w̄_i(ρ_i(iterator)) − w_i(ρ_i(iterator));
} } }
```

Figura 3.5: Pseudo-código para determinar $L_i(t)$

3.4.2 Cálculo do Tempo de Resposta

Tal como as iterações do método RTA, para determinar o tempo de resposta de uma dada activação l de uma tarefa, também se usa o método do ponto-fixo:

$$\begin{cases} {}^{l}R_i^{(0)} &= c_i + {}^{l}r_i \\ {}^{l}R_i^{(n+1)} &= L_{i-1}({}^{l}r_i) + c_i + w_{i-1}({}^{l}R_i^{(n)}) - w_{i-1}\left(L_{i-1}\left({}^{l}r_i\right)\right) \end{cases} \tag{3.20}$$

Este método converge para o tempo de resposta pois, tal como o método clássico, adiciona o novo trabalho pedido pelas tarefas de maior prioridade até que o sistema estabilize (${}^{l}R_i^{(n+1)} = {}^{l}R_i^{(n)} = {}^{l}R_i$).

3.4.3 Exemplo de uma Aplicação

De seguida apresenta-se um exemplo (Exemplo 1) com os parâmetros dados pela Tabela 3.1. Na Figura 3.7 mostra-se o escalonamento das tarefas para os primeiros instantes. Este exemplo possui uma percentagem de utilização do processador de $\sum \frac{c_i}{T_i} = 98,67\%$ e $\sum \frac{c_i}{D_i} = 282,54\%$. De notar que o maior hiperperiodo, Λ_{10}, possui uma dimensão bastante grande, acima das 60 milhões de unidades de tempo. Este resultado foi propositado de forma a mostrar que mesmo para grandes hiperperiodos o

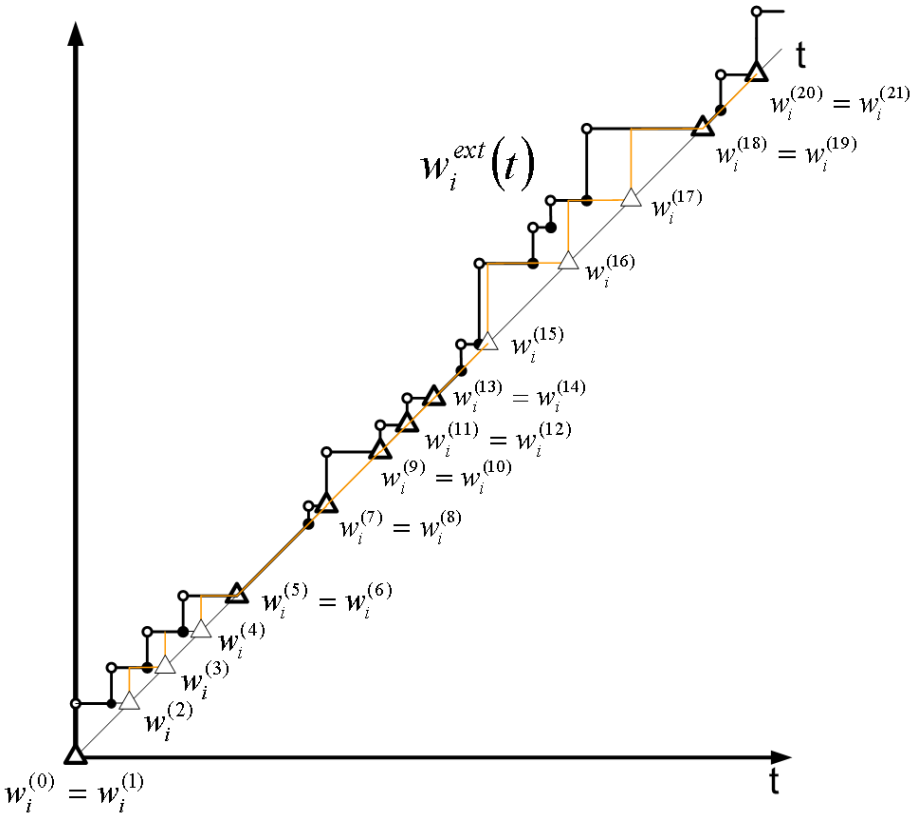


Figura 3.6: Iterações do pseudo-código para calcular $L_i(t)$

tempo necessário à escalonabilidade é ainda aceitável. O método proposto até agora de escalonabilidade é extremamente exigente para longos hiperperiodos[3], pelo que os tempos de análise demonstrados na Tabela 3.1 foram realizados segundo um método melhorado, discutido na secção 3.6.1.

A partir da Tabela 3.1 podemos ver que, calculando o tempo de resposta através do instante crítico dado pela equação 3.6, $\overline{R}_i$, quatro tarefas são classificadas como não escalonáveis: $\Gamma_{2,6,7,8}$. No entanto, a partir da Figura 3.7 conclui-se que Γ_2 é escalonável e que nunca sofre interferência de nenhuma outra tarefa. Para as restantes tarefas, $\Gamma_{6,7,8}$, ilustram-se na Figura 3.8 os seus tempos de resposta, ${}^{l}R_i - {}^{l}r_i$ para todas as execuções dentro do hiperperiodo. Como se pode constatar, os tempos de resposta são inferiores à meta temporal, o que indica que as tarefas são escalonáveis.

Este método permite determinar o pior tempo de resposta de cada activação. Como tal, é possível estabelecer qual o pior jitter no tempo de resposta das tarefas. Por exemplo, através da Figura 3.7

[3]Uma estimativa da duração do teste a Γ_{10} conclui que seriam precisas mais de 14000 horas (2 anos) para calcular todas as respostas com o método apresentado anteriormente

Parâmetros do Sistema					Análise de Escalonabilidade				
Γ_i	c_i	T_i	r_i	D_i	$\overline{R}_i$	$\max\left({}^{l}R_i - {}^{l}r_i\right)$	$LCM(T_1,\ldots,T_i)$	$\frac{LCM(T_1,\ldots,T_i)}{T_i}$	Analysis Time (s)
Γ_1	2	10	17	2	2	2	10	1	0.015
Γ_2	1	15	0	2	**3**	1	30	2	0.047
Γ_3	5	22	1	10	8	8	330	15	0.062
Γ_4	5	33	6	20	15	15	330	10	0.078
Γ_5	5	42	1	42	28	21	2 310	55	0,172
Γ_6	7	57	19	47	**58**	44	43 890	770	2,094
Γ_7	2	90	34	90	**98**	89	131 670	1 463	5,406
Γ_8	3	120	36	120	**148**	101	526 680	4 389	19,78
Γ_9	17	345	0	340	329	329	12 113 640	35 112	305,9
Γ_{10}	2	700	0	700	660	622	60 568 200	86 526	1196

Tabela 3.1: Parâmetros do Exemplo 1. A análise de escalonabilidade foi realizada com o método melhorado descrito na Secção 3.6.1, Figura 3.15

verifica-se que Γ_2 nunca sofre interferência de Γ_1. Como tal, o seu jitter encontra-se unicamente limitado pelo melhor e pior tempo de execução: $[c_2^-, c_2]$. Para um caso geral, é necessário analisar os tempos de resposta assumindo os melhores tempos de execução, $c_1^-,\ldots,c_i^-$, e os piores $c_1,\ldots,c_i$. O intervalo entre o melhor tempo de resposta no primeiro caso com o pior tempo de resposta do segundo fornece o pior jitter da tarefa.

Este método também é útil para sistemas que suportam falhas ocasionais das metas temporais. Diminuindo a meta temporal de Γ_8 para $D_8 = 90$, o resultado obtém-se na Figura 3.8. Como se pode concluir, existem várias execuções que não cumprem a meta temporal, enquanto que a esmagadora maioria cumpre. Tarefas de controlo não críticas, e.g., *firm real-time* ou *soft real-time*, podem utilizar esta análise para estabelecer regras quanto ao funcionamento correcto do sistema. Note-se, que as respostas são inferiores ao período (${}^{l}R_8 \leq {}^{l+1}r_8$). Caso contrário, a análise apresentada é insuficiente pois não contabiliza o tempo de execução restante de activações anteriores.

Dado que é necessário o cálculo do último instante de idle para todas as activações das tarefas, a complexidade total para determinar se uma tarefa Γ_i é escalonável corresponde a

$$O(E\,MMC(T_1,\ldots,T_i)^2)$$

onde E representa o número médio de iterações para calcular o próximo período de idle a partir da

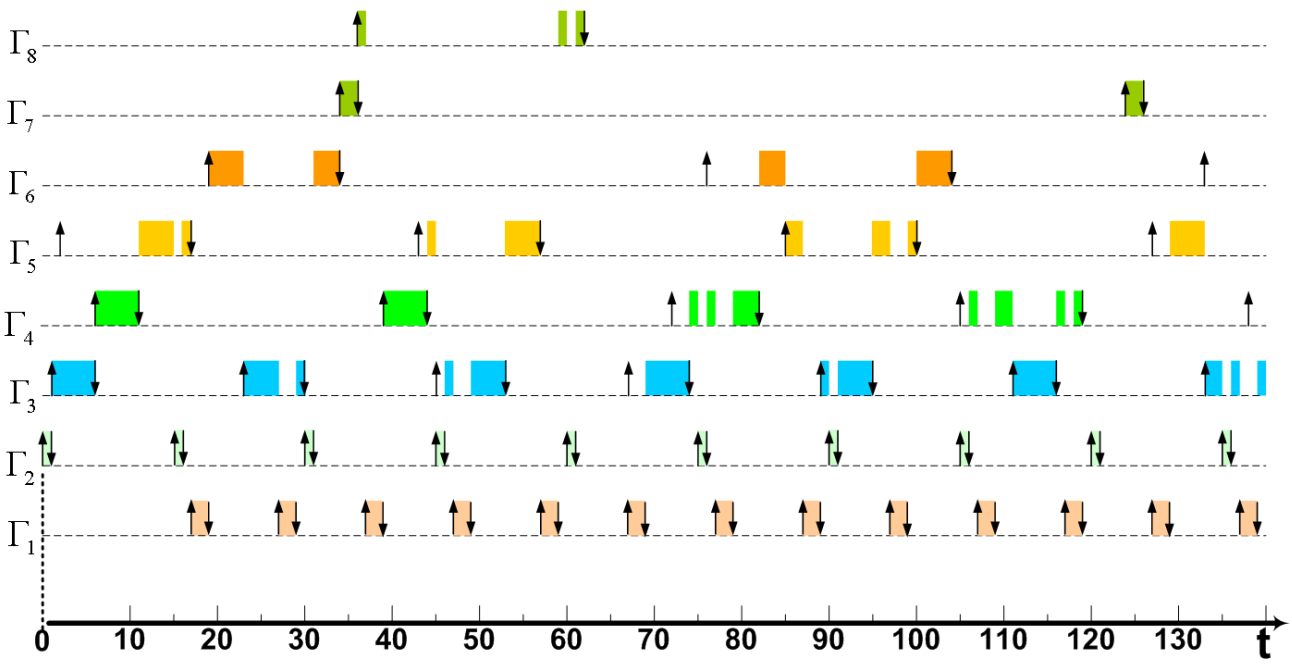


Figura 3.7: Escalonamento nos primeiros instantes do hiperperiodo do Exemplo 1

anterior. O algoritmo depende do quadrado de MMC pois o método recomeça desde o início para todas as execuções. Assim, para um número $MMC(T_1,...,T_i)/T_i$ de execuções, o número total de iterações necessárias corresponde a

$$E\,1+E\,2+\ldots+E\,\frac{MMC(T_1,\ldots,T_i)}{T_i}=E\frac{MMC(T_1,\ldots,T_i)^2/T_i+MMC(T_1,\ldots,T_i)}{2T_i}$$

Uma secção posterior descreve um método que torna a complexidade proporcional ao MMC dos períodos.

3.5 Integração de Tarefas Esporádicas

Uma das grandes vantagens dos sistemas *event-triggered* é a integração de tarefas com diferentes características. Entre elas, encontram-se as tarefas esporádicas. Este tipo de tarefas é extremamente útil para detectar situações "anómalas" no sistema, e.g., para-choques de um automóvel, sensor de pressão de uma fábrica acima de um dado limite, pacotes que chegam da rede, botão de *fail-safe*.

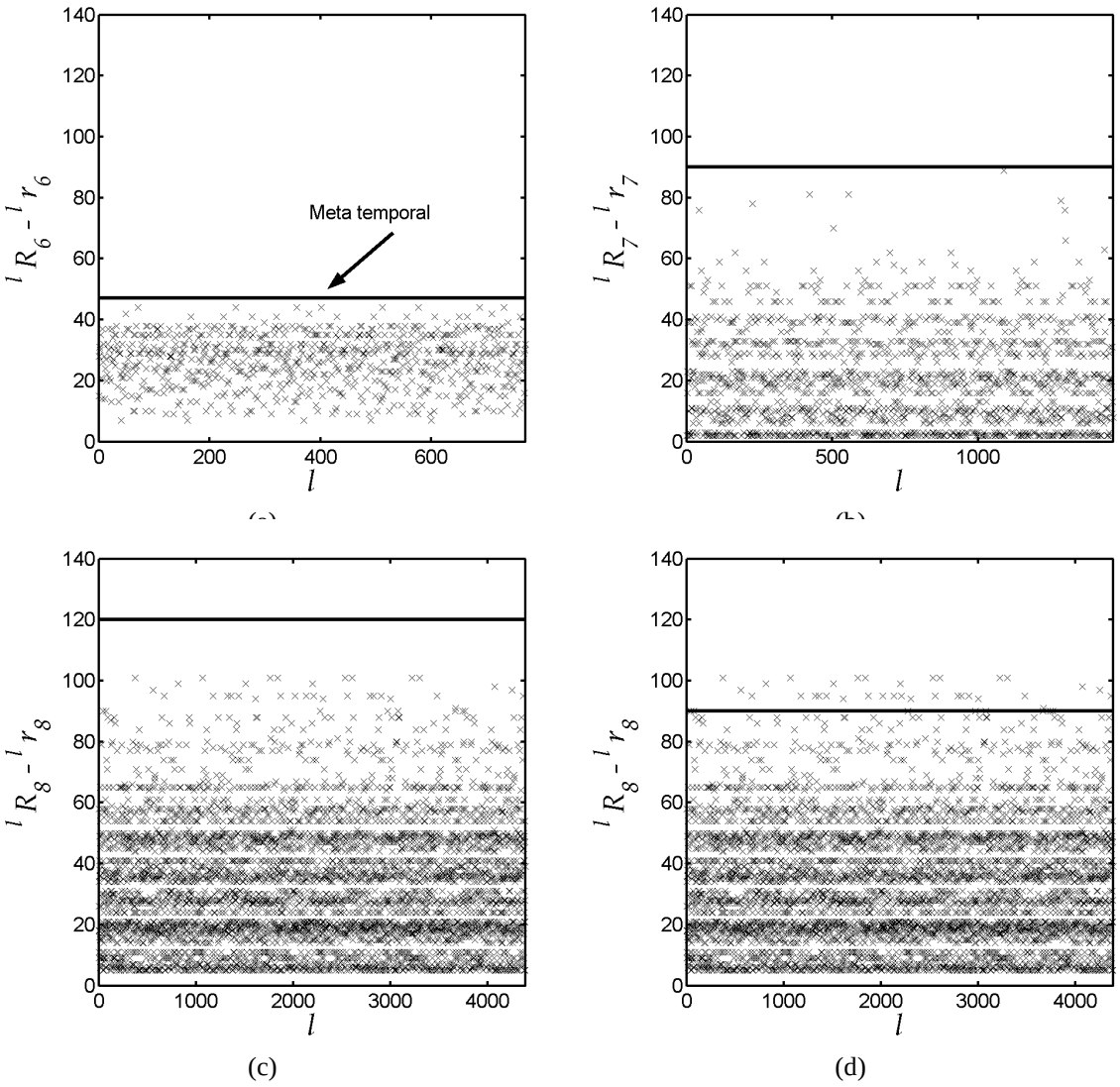


Figura 3.8: Tempos de resposta para cada execução da tarefa (a) Γ_6; (b) Γ_7; (c) Γ_8; (d) Γ_8 com $D_8 = 90$

Muitos destes eventos necessitam de uma resposta extremamente rápida, como o pára-choques para disparar o *airbag*. No entanto, dada a sua natureza pouco frequente, acontecem raramente. Um sistema *time-triggered*, para corresponder aos requisitos temporais, necessita de uma tarefa de *polling* de frequência muito alta, o que provoca um desperdício dos recursos da CPU. Em sistemas *event-triggered* tal não acontece devido à activação da tarefa apenas quando é despoletado o evento correspondente. Como a tarefa possui tipicamente um MIT bastante alto, no pior caso a interferência nas restantes tarefas é normalmente apenas de uma execução.

No modelo de tarefas síncronas assume-se que todas as tarefas, periódicas ou esporádicas, são lançadas no mesmo instante (instante crítico). No entanto, no modelo proposto, as tarefas assíncronas

periódicas são lançadas em instantes fixos, criando uma interferência bem conhecida nas restantes tarefas. É necessário saber, portanto, qual o pior instante onde as tarefas esporádicas podem ser activadas, *instante crítico assíncrono*, de forma a calcular qual a pior interferência que podem sofrer de tarefas periódicas de alta prioridade mas também qual a pior interferência que podem produzir nas tarefas periódicas de baixa prioridade.

Para além de saber qual o pior instante onde as tarefas esporádicas podem ser activadas, é também necessário incorporar o trabalho por elas pedido. Assim, assumindo que as tarefas são activadas em $t = 0$, o trabalho pedido pelas tarefas $\tau_1, \ldots, \tau_m$ é dado por $\omega_m(t)$ e é igual a

$$\omega_m(t) = \sum_{k=1}^{m} \left\lceil \frac{t}{MIT_k} \right\rceil_0 e_k \tag{3.21}$$

Se as tarefas esporádicas forem activadas em $t = t_0$ então a sua função de trabalho é $\omega_m(t - t_0)$.

3.5.1 Instante Crítico Assíncrono

Como já foi mencionado, o pior instante da activação das tarefas esporádicas é desconhecido *a priori*. Esta secção apresenta um método que calcula um conjunto de candidatos ao *instante crítico assíncrono*, κ_i, com base nas tarefas periódicas de mais alta prioridade. Como se tornará claro mais adiante, não é possível estabelecer um único instante crítico pois está intimamente ligado ao tempo de execução da tarefa esporádica. O instante crítico corresponde ao instante de activação das tarefas esporádicas que conduz ao seu pior tempo de resposta.

A Figura 3.9 permite ilustrar como obter os instantes candidatos ao instante crítico assíncrono. Esta Figura ilustra a função $w_i^{ext}(t) - t$ que permite estabelecer os períodos de computação e de idle. Como se pode concluir, os piores instantes para o lançamento de tarefas menos prioritárias coincidem com o instante do começo dos períodos de computação/fim do período de idle. Este facto pode ser demonstrado assumindo o contrário: se o pior instante estivesse no período de idle imediatamente anterior a tarefa começaria a executar-se nesse momento, indo terminar mais cedo, o que no pior caso mantém o tempo de resposta, Figura 3.9; se o pior instante fosse dentro do período de computação imediatamente posterior, então o instante de resposta da tarefa mantinha-se, mas o tempo de resposta seria menor, Figura 3.9. Logo os piores instantes para a activação de uma tarefa esporádica coincidem

quando o sistema inicia o período de computação. Note-se que o caso do instante crítico (quando todas as tarefas começam ao mesmo tempo) é um exemplo em particular do instante crítico assíncrono.

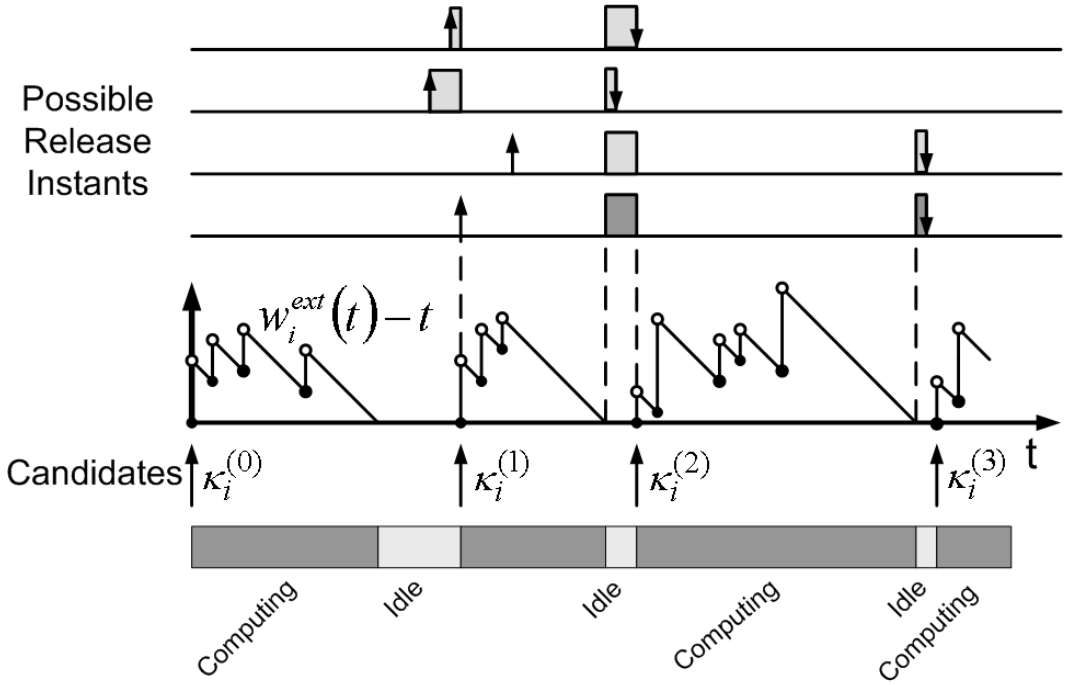


Figura 3.9: Candidatos ao instante crítico assíncrono

Estabelecidos os candidatos para o instante crítico assíncrono, resta construir um método que os calcule automaticamente. Para determinar o **próximo** instante de idle é construída a função $\varphi_i(t)$. Este novo método é semelhante às iterações do algoritmo $L_i(t)$, Figura 3.5. Como ilustrado na Figura 3.10 a função $\varphi_i(t)$ recebe como argumento o instante onde o período de computação começa. A primeira iteração do método $\varphi_i(t)$ adiciona o trabalho pedido nesse instante. As iterações seguintes são semelhantes aos outros métodos, adicionando em cada iteração o novo trabalho pedido, Equação 3.17. Um exemplo das iterações da função $\varphi_i(t)$ encontra-se na Figura 3.11.

A Figura 3.10 descreve também um método, $\kappa_i(t_{min}, t_{max})$ que retorna todos os candidatos dentro do intervalo $]t_{min}, t_{max}]$. Este método determina alternadamente o próximo instante de computação, através da função $\rho_i(t)$, ou o próximo instante de idle, através da função $\varphi_i(t)$. Quando o próximo instante de computação for maior que t_{max} o método chega ao fim.

A complexidade de determinação dos candidatos aos instantes críticos assíncronos em Λ_i é semelhante à determinação do último instante de idle: $O(E\,LCM(T_1, \ldots, T_i))$.

```
function κ_i(t_min, t_max){
    next_work_instant = L_i(t_min);
    work_instants = {};
    while(true) {
        next_idle_instant = φ_i(next_work_instant);
        next_work_instant = ρ_i(next_idle_instant);
        if( next_work_instant > t_max )
            return work_instants;
        work_instants = work_instants ∪ {next_work_instant};
} }
function φ_i(work_instant){ //returns the next idle instant
    total_idle_time = work_instant − w_i(work_instant);
    iterator = work_instant + w̄_i(work_instant) − w_i(work_instant);
    while(true) {
        last_iterator = iterator;
        iterator = w_i(iterator) + total_idle_time;
        if( iterator = last_iterator )
            return iterator;
} }
```

Figura 3.10: Pseudo-código para determinar os candidatos ao *verdadeiro instante crítico* no intervalo $]t_{min}, t_{max}]$

3.5.2 Análise RTA de Tarefas Periódicas Assíncronas

A pior interferência que as tarefas esporádicas podem produzir nas tarefas periódicas de baixa prioridade pode ser calculada com base no conhecimento dos candidatos ao instante critíco assíncrono.

Supondo uma tarefa periódica activada em ${}^{l}r_i$ e τ_K a tarefa de mais baixa prioridade mas com prioridade superior a Γ_i. Na Figura 3.12 encontra-se ilustrada a função de trabalho que ainda falta executar pelas tarefas periódicas de alta prioridade para cada instante, $w_{i-1}^{ext}(t) - t$. Como se pode ver, existem quatro candidatos no intervalo $]{}^{l-1}r_i, {}^{l}r_i]$: $\kappa_i^{(1)}, \ldots, \kappa_i^{(4)}$.

O pior instante onde as tarefas esporádicas podem ser activadas na Figura 3.12 é $\kappa_i^{(2)}$. Sendo as tarefas activadas neste instante, o período de idle entre $\kappa_i^{(2)}$ e ${}^{l}r_i$ desaparece, ficando um período de computação maior. De facto, se existir um período de idle entre o instante de activação e ${}^{l}r_i$ então esse não pode ser o pior instante de activação pois todo o trabalho foi, num dado instante, executado e logo não é adicionado ao próximo período de computação. Como exemplo, tome-se a activação em $\kappa_i^{(1)}$ na Figura 3.12.

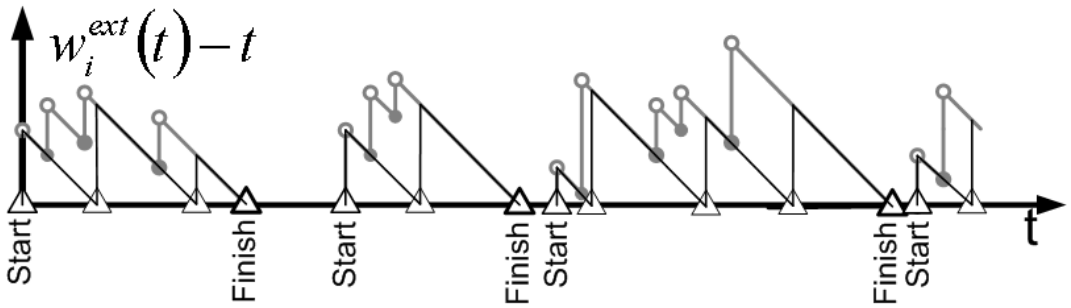


Figura 3.11: Exemplo das iteracções da função $\varphi_i(t)$

Dado que desde o pior instante de activação até ${}^{l}r_i$ não podem existir períodos de idle, se o pior instante for antes de ${}^{l-1}r_i$ então a execução $l-1$ não pode ser executada (dado que o CPU está sempre em computação) e falha a sua meta temporal. Logo, assumindo que todas as execuções anteriores cumpriram a meta temporal, os candidatos ao pior instante de activação encontram-se no intervalo $]{}^{l-1}r_i, {}^{l}r_i]$. Se não existirem candidatos neste intervalo então a tarefa não é escalonável dado que não existe nenhum momento onde o trabalho pedido tenha sido cumprido e portanto a execuçao $l-1$ não cumpriu a meta temporal.

O intervalo $]{}^{l-1}r_i, {}^{l}r_i]$ não pode estar dentro da fase transitória inicial. Logo, o hiperperiodo tem que começar T_i após a fase inicial ter terminado. Daqui advém a necessidade de fazer $S_i = \max(r_1, ..., r_i) + T_i$ na Equação (3.14). Esta alteração no hiperperiodo não afecta substancialmente o tempo de análise requirida dado que o MMC mantém-se como o factor mais significativo.

Dado que os períodos de idle desde o pior instante de activação, κ_i, até ${}^{l}r_i$ estão agora ocupados a processar as tarefas esporádicas de maior prioridade (caso contrário κ_i não seria o pior instante), têm que ser subtraídos quando se calcula o instante de resposta da tarefa. O tempo total de idle desde κ_i até ${}^{l}r_i$ é dado por

$$\underbrace{[L_{i-1}({}^{l}r_i) - w_{i-1}(L_{i-1}({}^{l}r_i))]}_{\text{tempo total de idle até } {}^{l}r_i} - \underbrace{[L_{i-1}(\kappa_i) - w_{i-1}(L_{i-1}(\kappa_i))]}_{\text{tempo total de idle até } \kappa_i}$$

Como em κ_i as tarefas de maior prioridade estão em idle, Figura 3.9, tem-se $L_{i-1}(\kappa_i)) = \kappa_i$. O instante de resposta de uma tarefa periódica assíncrona é encontrado adicionando à Equação 3.13 o trabalho pedido pelas tarefas esporádicas de maior prioridade e subtraindo o tempo de idle

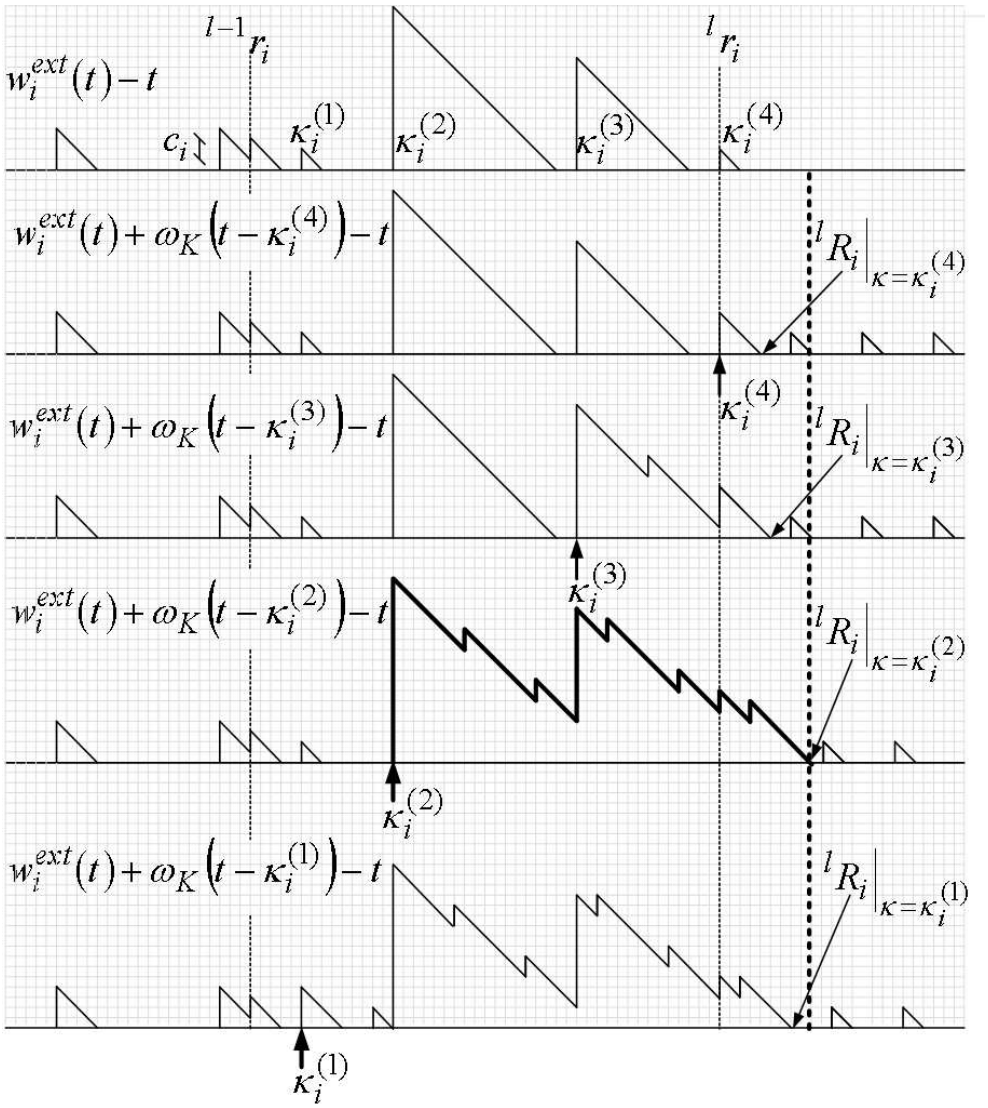


Figura 3.12: Candidatos ao pior instante para a activação das tarefas esporádicas

$$^{l}R_i = \kappa_i + c_i + w_{i-1}(^{l}R_i) - w_{i-1}(\kappa_i) + \omega_K(^{l}R_i - \kappa_i) \tag{3.22}$$

onde a menor solução (maior que $^{l}r_i$) corresponde ao instante de resposta. Esta equação pode ser resolvida usando o método de ponto-fixo com $^{l}R_i^{(0)} = {}^{l}r_i + c_i$. Esta equação tem que ser determinada para cada $\kappa_i \in]^{l-1}r_i, {}^{l}r_i]$ e o pior instante de resposta utilizado na condição 3.15 para determinar se Γ_i é escalonável.

Se um candidato κ_i não é o pior instante de activação e não remove completamente os períodos

de idle, então o instante de resposta encontrado é menor do que deveria ser (pois está-se a subtrair todos os períodos de idle), mas como este não é o pior caso não vai ser considerado no teste final de escalonabilidade.

No caso particular de não existirem tarefas esporádicas de maior prioridade, então o único candidato κ_i que não possui períodos de idle entre κ_i e ${}^{l}r_i$ coincide com $L_{i-1}({}^{l}r_i)$ (instante $\kappa_i^{(4)}$ na Figura 3.12), tornando a Equação 3.22 equivalente à Equação 3.13.

3.5.3 Análise RTA de Tarefas Esporádicas

Para determinar o pior instante de resposta de uma tarefa esporádica é necessário considerar todos os candidatos ao instante crítico assíncrono. Seja Γ_i a tarefa periódica de menor prioridade com prioridade maior que τ_m. O pior instante de resposta de uma tarefa esporádica é encontrado com todas as tarefas esporádicas de maior prioridade foram activadas no mesmo candidato ao instante crítico assíncrono. Logo, é a menor solução (maior que κ_i) de

$$R_m = \kappa_i + e_m + w_i(R_m) - w_i(\kappa_i) + \omega_{m-1}(R_m - \kappa_i) \tag{3.23}$$

Esta equação pode ser resolvida com o método de ponto-fixo, com $R_m^{(0)} = e_m + \kappa_i$. A análise de escalonabilidade pode ser resumida na condição

$$\begin{cases} \forall_{\kappa_i \in \Lambda_i} & R_m - \kappa_i \leq H_m \quad , \quad \tau_m \text{ is } \textbf{schedulable} \\ \exists_{\kappa_i \in \Lambda_i} & R_m - \kappa_i > H_m \quad , \quad \tau_m \text{ is } \textbf{unschedulable} \end{cases} \tag{3.24}$$

Como exemplo tome-se a interferência provocada pelas tarefas $\Gamma_1, ..., \Gamma_i$. Os candidatos para o instante crítico assíncrono deste conjunto são (primeiros 10): 37; 45; 57; 60; 67; 75; 77; 87; 89; 97. No total existem 55 candidatos em Λ_3. Os tempos de resposta para uma tarefa de menor prioridade com $WCET = 1$ e activada nestes instantes são respectivamente: 3; 9; 3; 2; 8; 2; 3; 9; 7; 3. Para uma tarefa com $WCET = 10$ os tempos de resposta são: 20; 21; 23; 21; 20; 21; 20; 23; 21; 20. Note-se, em particular, que se uma tarefa for activada em $t = 57$, para um $e_1 = 1$ o seu tempo de resposta é dos mais baixos. Por outro lado, quando $e_1 = 10$ o seu tempo de resposta é o maior. Logo, todos os candidatos têm que ser analizados mesmo que para WCET baixos forneça muito bons tempos de

resposta. Este facto deriva de que, para um WCET baixo, o pior candidato corresponde ao maior período de computação ininterrupto; para um WCET alto corresponde à maior cadeia de períodos de computação com períodos de idle pequenos.

Como exemplo adicional tome-se a análise de escalonabilidade de uma tarefa esporádica τ_1 com prioridade menor que Γ_8 e maior que Γ_9 com $e_1 = 6$; $MIT_1 = 200$ e $H_1 = 150$. Os candidatos para o instante crítico assíncrono são (primeiros 5): 105; 124; 127; 237; 287. No total existem 15703 candidatos dentro de Λ_8. Os tempos de resposta para uma tarefa esporádica, assumindo que foi activada nos candidatos, são (primeiros 5): 127 ; 111; 109; 155; 106. O pior tempo de resposta corresponde a 168 quando a tarefa é activada nos candidatos 2175; 27255; 39975; 69495; etc (existem 22 candidatos onde a resposta é igual). Dado que o pior tempo de resposta é maior que a meta temporal (150), a tarefa não é escalonável. Se a meta temporal fosse maior que 168 então a tarefa seria escalonável.

3.6 Optimização da Análise de Escalonamento

Como já foi referido, a análise do método proposto depende rigidamente com o hiperperiodo imposto pelas tarefas periódicas. Se os períodos forem co-primos entre si, então o hiperperiodo explode e mesmo para um número reduzido de tarefas, a análise torna-se rapidamente intratável. Esta secção discute métodos que aliviam a dependência do hiperperiodo e aumentam a rapidez dos cálculos.

Como um teste extremamente rápido à escalonabilidade de uma tarefa, podem-se considerar tarefas periódicas síncronas (Audsley, 1991). No entanto, dado o pessimismo introduzido por estes testes, é de esperar que não sejam suficientes para garantir a escalonabilidade de várias tarefas. Nos casos onde esta análise não permite dizer que a tarefa é escalonável, deve-se recorrer ao método proposto, a menos que se queira alguma outra análise, e.g., dispersão (*jitter*), número de falhas de metas temporais, etc.

3.6.1 Exploração do Último Instante de Idle

Como já foi referido anteriormente, a complexidade da determinação das respostas de uma dada tarefa Γ_i é

$$O\left(E\,MMC(T_1,...,T_i)^2\right)$$

A dependência da segunda potência de MMC deriva de reinicialização do método para calcular o último instante de idle, $L_{i-1}(t)$ para cada execução da tarefa. No entanto, mantendo o último instante idle conhecido cada iteração começa a partir do último ponto calculado, em vez de fazer a mesma análise do começo. Assim, a complexidade fica reduzida a

$$O\left(E\,MMC(T_1,...,T_i)\right)$$

Dado que o MMC é claramente o factor limitativo, este melhoramento produz resultados significativos.

A Figura 3.13 ilustra o pseudo-código optimizado para determinar o último instante de idle a partir do último instante de idle conhecido. As alterações efectuadas concentram-se na fase de inicialização, em particular no cálculo inicial do tempo total de idle e da primeira iteração. A Figura 3.14 descreve o pseudo-código optimizado para determinar os candidatos κ_i usando a função $\tilde{L}_i(last,t)$.

Finalmente, a Figura 3.15 descreve o pseudo-código optimizado para calcular os instantes de resposta para as execuções $l_{initial},\ldots,l_{final}$. Neste algoritmo, o último instante de idle é actualizado a partir do maior candidato conhecido visto que, como dito anteriormente, os candidatos são eles próprios instantes de idle.

Se o tempo de resposta for maior que o período da tarefa em análise, esta análise não é suficiente para calcular os tempos de resposta das execuções posteriores, dado que terão uma interferência adicional. Quando é detectada esta situação, o método pára. Note-se que neste caso já foi estabelecido que a tarefa não é escalonável pois a meta temporal é inferior ao período.

```
function L̃_i(last_known_idle, t){
    if( i = 0 ) // there are no higher priority tasks
        return t;
    last_idle_instant = last_known_idle;
    total_idle_time = last_known_idle − w_i(last_known_idle);
    iterator = last_known_idle;
    while(true) {
        last_iterator = iterator;
        iterator = w_i(iterator) + total_idle_time;
        if(iterator > t)
            return last_idle_instant;
        if(last_iterator = iterator) { // arrived at an idle instant
            if(iterator ≤ t ≤ ρ_i(iterator))
                return t;
            last_idle_instant = ρ_i(iterator);
            total_idle_time += ρ_i(iterator) − iterator;
            iterator = ρ_i(iterator) + w̄_i(ρ_i(iterator)) − w_i(ρ_i(iterator));
} } }
```

Figura 3.13: Cálculo da função $\tilde{L}_i(last,t)$ que recebe como argumento adicional o último instante de idle conhecido prévio a t

Como um exemplo da optimização feita, a determinação das respostas de Γ_7 necessita de aproximadamente uma hora com o método original, enquanto que com o novo demorou 5 segundos[4] - Tabela 3.1.

3.6.2 Lidar com Grandes Hiperperiodos

Dado que a análise de escalonamento das tarefas periódicas depende rigidamente com o hiperperiodo por elas definido, para um grande número de tarefas esta análise pode ser intratável, especialmente se as tarefas possuirem períodos primos entre si.

No entanto, conhecendo os instantes críticos assíncronos impostos pelas tarefas periódicas, a análise pode englobar apenas um conjunto reduzido de tarefas periódicas de alta prioridade, assumindo que as tarefas de menor prioridade são activadas nos piores instantes (instantes críticos assíncronos). Assim, é encontrado um compromisso entre o pessimismo do pressuposto dos instantes

[4]Usando MatLab sobre Windows XP num processador PIII a 1300 MHz

function $\tilde{\kappa}_i(t_{min}, t_{max}, last\_idle\_instant)\{$
    $next\_work\_instant = \tilde{L}_i(last\_idle\_instant, t_{min})$;
    $work\_instants = \{\}$;
    while(true) {
        $next\_idle\_instant = \varphi_i(next\_work\_instant)$;
        $next\_work\_instant = \rho_i(next\_idle\_instant)$;
        if($next\_work\_instant > t_{max}$)
            return *work_instants*;
        $work\_instants = work\_instants \cup \{next\_work\_instant\}$;
} }

Figura 3.14: Determinação de κ_i no intervalo $]t_{min}, t_{max}]$ recorrendo à função optimizada $\tilde{L}_i(last, t)$

críticos assíncronos e a complexidade da análise.

Por exemplo, supondo um conjunto de 40 tarefas periódicas e 40 tarefas esporádicas. Se o hiper-periodo das 40 tarefas periódicas for demasiado grande, o arquitecto de sistema pode determinar os candidatos aos instantes críticos assíncronos das 20 tarefas periódicas de maior prioridade. Um teste suficiente pode assim ser realizado, assumindo que todas as tarefas de menor prioridade, quer sejam esporádicas ou periódicas, são activadas nos instantes candidatos. Embora o número de candidatos possa ser relativamente alto, é um número fixo, tornando o tempo de análise para cada tarefa de menor prioridade relativamente igual. À medida que a potência computacional aumenta, cada vez mais tarefas periódicas podem ser incorporadas na análise, reduzindo o pessimismo introduzido.

Como um exemplo concreto tome-se a análise de escalonabilidade de Γ_8 do Exemplo 1. A determinação de todos os instantes candidatos para o instante crítico assíncrono das tarefas $\Gamma_1, \ldots, \Gamma_7$ demorou 4.28s e a determinação dos tempos de resposta de Γ_8 assumindo que foi activada nestes instantes candidatos demorou 9.2s (estes tempos não estão presentes na Tabela 3.1). O pior tempo de resposta corresponde aos instantes de activação $925; 49435; 97945$ e é igual a 110. Dado que a meta temporal é de 120, esta análise é suficiente para concluir que Γ_8 é escalonável. Se o pior tempo de resposta for maior que a meta temporal nada pode ser concluído.

function $\tilde{R}_i(l_{initial}, l_{final})$ {
  $idle = 0$; // last known idle instant
  for$\big(l = l_{initial}\ ;\ l \leq l_{final}\ ;\ l{+}{+}\big)\{$
    ${}^{l}R_i = {}^{l}r_i + c_i$; // smallest possible value of ${}^{l}R_i$
    $\kappa_i = \tilde{\kappa}_i({}^{l-1}r_i, {}^{l}r_i, idle)$;
    $idle = max(\kappa_i)$; // update last known idle instant
    foreach$(\kappa_i^{(x)} \in \kappa_i)\{$
      $R = {}^{l}r_i + c_i$; // initial iteration
      do {
        $last\_R = R$;
        $R = \kappa_i^{(x)} + c_i + w_{i-1}(R) - w_{i-1}(\kappa_i^{(x)}) + \omega_K(R - \kappa_i^{(x)})$;
        if$(R\ >\ {}^{l+1}r_i)$
          return “cannot determine response instant”;
      } while$(last_R \neq R)$;
      if$(R\ >\ {}^{l}R_i)$ // update ${}^{l}R_i$ with largest response instant
        ${}^{l}R_i = R$;
  } }
  return ${}^{l_{min}}R_i, \ldots, {}^{l_{max}}R_i$;
}

Figura 3.15: Determinação de ${}^{l_{initial}}R_i, \ldots, {}^{l_{final}}R_i$ recorrendo à função optimizada $\tilde{\kappa}_i(t_{min}, t_{max}, last)$

3.6.3 Algoritmo de Escalonamento Distribuído

Como problema comum associado aos métodos iterativos, cada iteração precisa da anterior para poder ser calculada. Desta forma, para que se conheça a resposta para uma determinada execução em particular, são necessárias todas as iterações desde o início do sistema. Em geral, é extremamente complexo ou mesmo impossível sub-dividir um problema iterativo de forma a poder distribuir o seu cálculo por vários computadores. Felizmente, é possível calcular o último instante de idle, $L_i(t)$, sem analisar o sistema desde o início.

Em vez de começar na origem do tempo, o último instante de idle pode ser encontrado procurando para trás no tempo, assumindo que todas as metas temporais anteriores foram cumpridas. Os métodos apresentados para calcular $L_i(t)$ apenas procuram para a frente no tempo, logo são necessárias algumas modificações. Em particular, o novo método proposto começa por procurar para trás no tempo até um determinado instante a partir do qual, começa de novo a procurar para a frente até t.

Para simplificar a explicação do método, consideramos primeiro em apenas duas tarefas e de seguida generalizamos para um número variável. O objectivo centra-se em encontrar $L_2({}^l r_2)$. Na Figura 3.16 encontram-se ilustradas duas situações: a tarefa mais prioritária, Γ_1, não está a executar em ${}^l r_2$; a tarefa Γ_1 está a executar em ${}^l r_2$. Como se pode concluir, para o primeiro caso o último instante de idle das tarefas $\Gamma_{1,2}$ em ${}^l r_2$, dado por $L_2({}^l r_2)$, é ${}^l r_2$. Como Γ_1 encontra-se em idle nesse instante, também se pode dizer que $L_2({}^l r_2) = L_1({}^l r_2)$. Para o segundo caso, a tarefa Γ_1 encontra-se em execução em ${}^l r_2$, portanto o último instante de idle corresponde ao início da execução da tarefa Γ_1, dado por ${}^x r_1$. Tem-se portanto, $L_2({}^l r_2) = L_1({}^x r_1) = {}^x r_1$. Portanto, em ambos os casos tem-se $L_2({}^l r_2) = L_1({}^l r_2)$. Este resultado deve-se a que, no instante ${}^l r_2$, a tarefa Γ_2 já concluiu todo o trabalho pedido; caso contrário, teria falhado a sua meta temporal. Assim, em ${}^l r_2$ o último instante de idle depende apenas de Γ_1. Generalizando para várias tarefas, tem-se $L_i({}^l r_i) = L_{i-1}({}^l r_i)$.

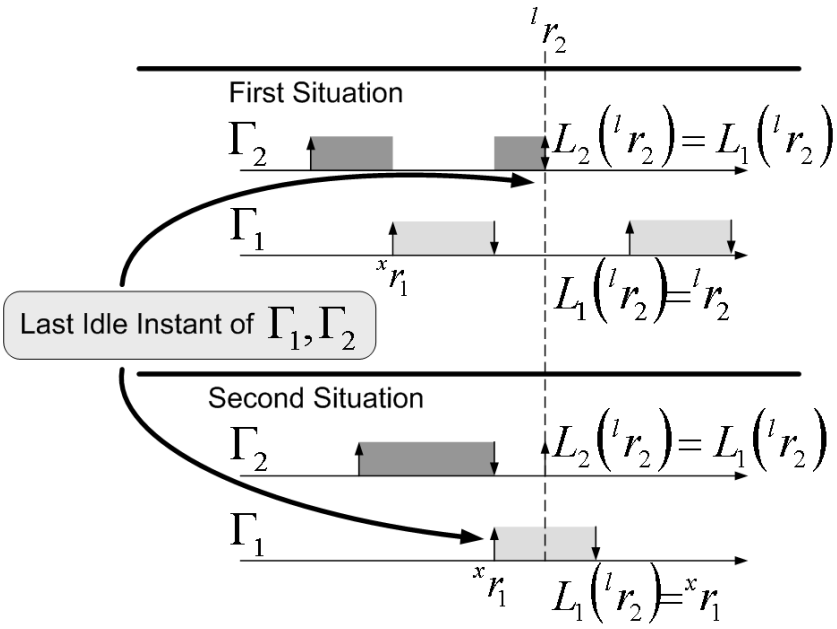


Figura 3.16: As duas situações no cálculo de $L_i({}^l r_2)$: Γ_1 está idle em ${}^l r_2$; Γ_1 está a executar em ${}^l r_2$

A Figura 3.17 ilustra como o método proposto para calcular, com poucas iterações, o último instante de idle é aplicado. Como se pode observar, o algoritmo parte do instante t e vai percorrendo para trás no tempo para descobrir a activação mais próxima da tarefa de prioridade imediatamente acima, ${}^x r_3$. O algoritmo percorre desta forma até chegar à activação da primeira tarefa, ${}^z r_1$. Este instante corresponde a $L_1({}^z r_1)$. Tendo um instante de idle, é possível calcular os seguintes através da função $\tilde{L}_i(last, t)$ descrita anteriormente. Assim tem-se $L_1({}^y r_2) = \tilde{L}_1({}^x r_1, {}^y r_2)$. Este instante corre-

sponde, como foi descrito, a $L_2({}^{y}r_2)$. Tendo $L_2({}^{y}r_2)$ é possível calcular o último instante de idle até ${}^{x}r_3$: $L_2({}^{x}r_3) = \tilde{L}_2(L_1({}^{y}r_2), {}^{x}r_3)$. Continuando desta forma, encontra-se $L_3(t) = \tilde{L}_3(L_2({}^{x}r_3), t)$.

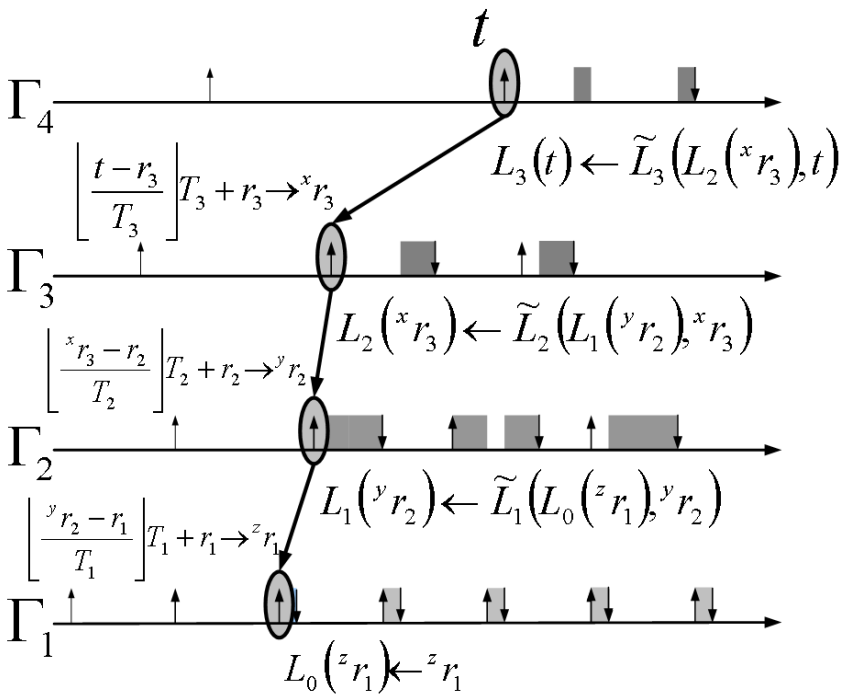


Figura 3.17: Cálculo rápido de $L_i(t)$ procurando para trás no tempo

A Figura 3.18 ilustra o pseudo-código que implementa o método necessário ao cálculo do último instante de idle antes de t. Como se pode observar, esta função necessita de andar para trás no tempo para descobrir qual o último instante de idle da primeira tarefa para andar para a frente a partir daí. Se a função andar para trás no tempo de tal forma que não existem activações anteriores, então recorre ao método original, $L_i(t)$.

Na Figura 3.19 encontra-se demonstrado um exemplo do tempo necessário à análise recorrendo a cada uma das três funções que calculam o último instante de idle: $L_i(t); \tilde{L}_i(last, t); \check{L}_i(t)$. O exemplo calcula $L_{10}(t)$ do Exemplo 1 - Tabela 3.1. Como esperado, a função original, $L_i(t)$, é a mais lenta e aumenta proporcionalmente com t. A função $\tilde{L}_i(last, t)$ é geralmente a mais rápida dado que conhece o último instante de idle. Note-se, no entanto, que para $t = 5000$ requer aproximadamente o mesmo tempo que a função original, devido a não conhecer nenhum instante de idle prévio (primeira vez que é executada, $last = 0$). A função $\check{L}_i(t)$ é rápida para calcular t mas não tanto como $\tilde{L}_i(last, t)$ pois necessita de determinar o último instante de idle para cada uma das tarefas de maior prioridade.

function $\breve{L}_i(t)\{$
    if $(i = 0)$
        return t;
    //calculate the previous higher priority task release
    $r_i^{\cdot} = \left\lfloor \frac{t-r_i}{T_i} \right\rfloor T_i + r_i$;
    if$(r_i^{\cdot} < r_i)$
        return $L_i(t)$;
    else
        return $\tilde{L}_i\left(\breve{L}_{i-1}\left(r_i^{\cdot}\right), t\right)$;
$\}$

Figura 3.18: Pseudo-código para o cálculo rápido do último instante idle das tarefas $\Gamma_1, ..., \Gamma_i$ antes de t

A partir destes resultados é possível assumir que a função $\breve{L}_i(t)$ deve ser usada para o primeiro cálculo do último instante de idle e $\tilde{L}_i(last, t)$ para os restantes cálculos.

A Figura 3.20 ilustra o pseudo-código da função $\breve{\kappa}_i(t_{min}, t_{max})$ que calcula os candidatos ao instante crítico assíncrono dentro do intervalo $]t_{min}, t_{max}]$. Note-se na alteração somente no cálculo do primeiro instante de idle que usa a nova função $\breve{L}_i(t)$.

De igual modo, a Figura 3.21 ilustra o pseudo-código da função $\breve{R}_i(l_{initial}, l_{final})$ que também utiliza $\breve{L}_i(t)$ para determinar o primeiro instante de idle.

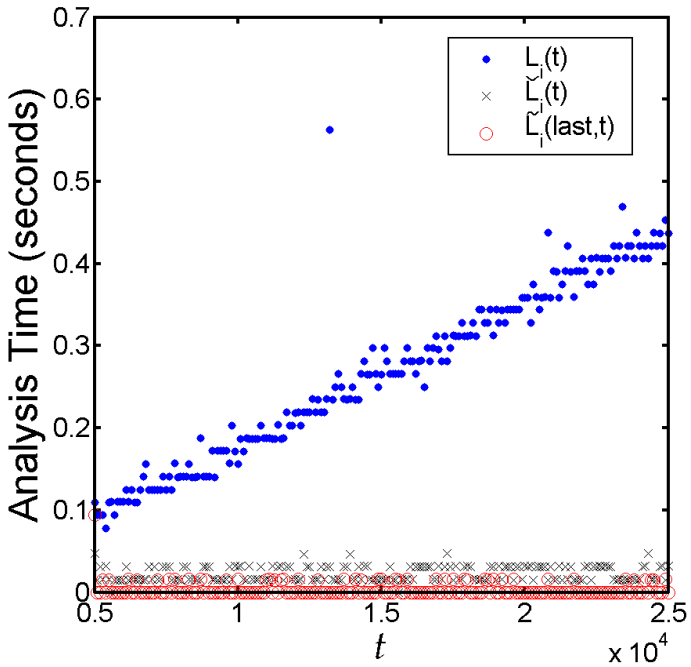


Figura 3.19: Comparação do tempo de análise para cada um dos métodos para determinar o último instante de idle

function $\check{\kappa}_i(t_{initial}, t_{final})\{$
    $next\_work\_instant = \check{L}_i(t_{initial})$;
    $work\_instants = \{\}$
    while(true) {
        $next\_idle\_instant = \varphi_i(next\_work\_instant)$;
        $next\_work\_instant = \rho_i(next\_idle\_instant)$;
        if($next\_work\_instant > t_{final}$)
            return $work\_instants$;
        $work\_instants = work\_instants \cup \{next\_work\_instant\}$;
} }

Figura 3.20: Determinação dos candidatos κ_i dentro do intervalo $[t_{initial}; t_{final}[$

function $\breve{R}_i(l_{initial}, l_{final})$ {
  $idle = \breve{L}_i(^{l_{initial}})$; // optimized last known idle instant
  for$\big(l = l_{initial}\ ;\ l \leq l_{final}\ ;\ l{+}{+}\big)\{$
    $^{l}R_i = {}^{l}r_i + c_i$; // smallest possible value of $^{l}R_i$
    $\kappa_i = \tilde{\kappa}_i(^{l-1}r_i, {}^{l}r_i, idle)$;
    $idle = max(\kappa_i)$; // update last known idle instant
    foreach$(\kappa_i^{(x)} \in \kappa_i)\{$
      $R = {}^{l}r_i + c_i$; // initial iteration
      do {
        $last\_R = R$;
        $R = \kappa_i^{(x)} + c_i + w_{i-1}(R) - w_{i-1}(\kappa_i^{(x)}) + \omega_K(R - \kappa_i^{(x)})$;
        if$(R\ >\ {}^{l+1}r_i)$
          return “cannot determine response instant”;
      } while$(last_R \neq R)$;
      if$(R\ >\ {}^{l}R_i)$ // update $^{l}R_i$ with largest response instant
        $^{l}R_i = R$;
  } }
  return $^{l_{min}}R_i, \ldots, ^{l_{max}}R_i$;
}

Figura 3.21: Determinação de $^{l_{initial}}R_i, \ldots, ^{l_{final}}R_i$ recorrendo à função optimizada $\tilde{\kappa}_i(t_{min}, t_{max}, last)$

3.7 Sumário

A maioria dos métodos existentes assumem que todas as tarefas, quer sejam periódicas ou esporádicas, são síncronas, i.e., são activadas no pior instante - instante crítico. Desta forma, é necessária apenas a análise do sistema neste instante, produzindo testes de complexidade pseudo-polinomial. Por outro lado, em sistemas assíncronos as tarefas periódicas são caracterizadas por activações em instantes fixos que podem não corresponder ao instante crítico. Estes sistemas produzem melhores resultados em termos de escalonabilidade, pois não assumem o pessimismo induzido pelo instante crítico. Por outro lado, a análise de sistemas assíncronos possui uma complexidade co-NP-hard no sentido forte.

À medida que os sistemas de controlo vão ficando sobrecarregados com tarefas, subsistem dois métodos para lidar com esta carga adicional:

- Comprar sistemas mais rápidos
- Analisar mais detalhadamente o escalonamento

A primeira opção é contraproducente, pois resulta não só em custos de compra mas também de certificação do novo material. A segunda opção não acarta custos adicionais (relevantes), mas requer um conhecimento teórico mais profundo sobre a escalonabilidade. À medida que o sistema vai ficando sobrecarregado, esta análise torna-se cada vez mais determinística às custas de uma complexidade crescente. Aqui surge uma necessidade da passagem de sistemas síncronos para assíncronos.

Este capítulo apresenta um novo método que permite concluir se uma dada **tarefa periódica assíncrona é escalonável**, aumentando o modelo do sistema com a integração dos instantes das suas activações. Uma nova função, **função de trabalho estendido**, é introduzida para assistir na determinação dos tempos de resposta das tarefas. O teste apresentado permite que mais sistemas se tornem escalonáveis, pois elimina a restrição das tarefas lançadas no instante crítico. No entanto, necessita de um número significativo de cálculos adicionais em comparação com os métodos clássicos.

É apresentado também um método que permite a **integração de tarefas esporádicas**. As tarefas esporádicas são semelhantes às periódicas com um diferença fundamental para esta análise: podem ser activadas em qualquer instante. O escalonamento das tarefas periódicas sofre ligeiras alterações,

mas a determinação do pior tempo de resposta das tarefas esporádicas está condicionado ao instante onde são activadas. É descrito um método que, a partir da interferência das tarefas periódicas assíncronas, produz uma série de candidatos ao pior instante onde qualquer tarefa de menor prioridade pode ser activada: candidatos ao ***instante crítico assíncrono***. A escalonabilidade das tarefas esporádicas fica reduzida à determinação do seu tempo de resposta quando são activadas nestes instantes. Tanto quanto o nosso conhecimento permite dizer, não existe outro teste que integre tarefas periódicas assíncronas e esporádicas.

Dada a complexidade do teste apresentado, $O(MMC^2)$, torna-se necessário apresentar alternativas para reduzir o tempo necessário da análise. O teste de escalonabilidade mais básico consiste em tratar as tarefas assíncronas como síncronas, i.e., são activadas no instante crítico. Dado que se um sistema síncrono é escalonável então o contra-posto assíncrono também o é, tratando-se de um teste suficiente para determinar se uma dada tarefa é escalonável.

De forma a reduzir a dependência do quadrado do MMC, foi também apresentado outro método que **reduz a complexidade** para $O(MMC)$. Dado que o hiperperiodo é o elemento limitativo no tempo de análise, este melhoramento é bastante significativo no tempo de análise: num exemplo em particular, reduziu uma hora de cálculos para 5 segundos.

Outro resultado de grande importância conduz à **redução da dependência do hiperperiodo** do tempo de análise através da **introdução de um certo pessimismo** na análise: tratar as tarefas periódicas assíncronas de menor prioridade como síncronas, i.e., são activadas nos piores instantes. A análise das tarefas esporádicas necessita da determinação de uma série de candidatos para o *instante crítico assíncrono*. Tratando as tarefas periódicas de menor prioridade como síncronas limita o hiperperiodo, embora introduza um certo grau de pessimismo. À medida que os sistemas de análise tornam-se mais potentes, cada vez mais tarefas periódicas assíncronas podem ser incorporadas na análise inicial.

Um outro resultado significativo deste capítulo revela que o método iterativo utilizado para determinar o parâmetro essencial para as análises acima referidas, o último instante de idle, pode ser calculado facilmente (com poucas iterações). Assim, divindindo o hiperperiodo em N intervalos é possível a atribuição da análise de escalonabilidade de cada um deles a uma máquina diferente. Desta forma, é possível **distribuir** o cálculo computacional sobre N máquinas diferentes, diminuindo o

tempo de análise. Este resultado pode ser aplicado tanto à determinação dos tempos de resposta como para o cálculo dos candidatos do *instante crítico assíncrono*.

Como possíveis aplicações directas deste método, encontra-se o escalonamento de sistemas de controlo nos ramos automóvel, espacial, aero-náutico, fabril, etc. Para além de reduzir o pessimismo, estes métodos criam um determinismo que também pode ser aplicado noutros domínios, nomeadamente, no cálculo da dispersão (*jitter*) da resposta de uma tarefa e em sistemas que permitem **falhas ocasionais da meta temporal**.

4

Compartimentação e Composição em Sistemas de Controlo de Tempo-Real

Nas indústrias aeronáutica, espacial, etc., a confiança no funcionamento e a segurança são factores vitais. À parte a perda de equipamento devido a uma falha do sistema, a perda de vidas humanas no caso de uma falha do sistema é imensurável. Assim, o software crítico presente nos sistemas embebidos tem que seguir regras bem delineadas, tanto em termos de implementação como de certificação.

A compartimentação de tarefas em sistemas de controlo é uma solução extremamente atractiva, uma vez que permite separar a nível espacial e temporal o seu funcionamento. Desta forma é possível prevenir acessos indevidos a zonas de memória que pertencem a outra tarefa. Do mesmo modo, a protecção a nível temporal define qual a janela temporal de execução de cada tarefa. Assim, se uma tarefa mal-dimensionada não libertar o processador dentro de uma janela temporal pré-definida, o escalonador força a mudança da tarefa em execução.

Por outro lado, a compartimentação de tarefas necessita de um acréscimo temporal de modo a decidir qual a tarefa em execução com base nas janelas temporais associadas a cada uma delas e a actualizar os registos de protecção de memória. Com base nestes argumentos, as tarefas são compostas em grupos ou "partições" de modo a que cada partição esteja protegida das restantes. Dentro de cada partição, as tarefas competem pelos recursos de uma forma livre e sem protecções.

A Figura 4.1 ilustra uma possível divisão de uma aplicação por partições. Como se pode observar, a divisão em partições permite que um erro em uma partição não seja propagado para as restantes partições. Esta protecção é particularmente vital nos sistemas de controlo, onde a zona de código crítica é tipicamente pequena e sujeita a testes extensos enquanto que as restantes zonas são usualmente grandes extensões de código menos testadas e portanto mais propensas a erros.

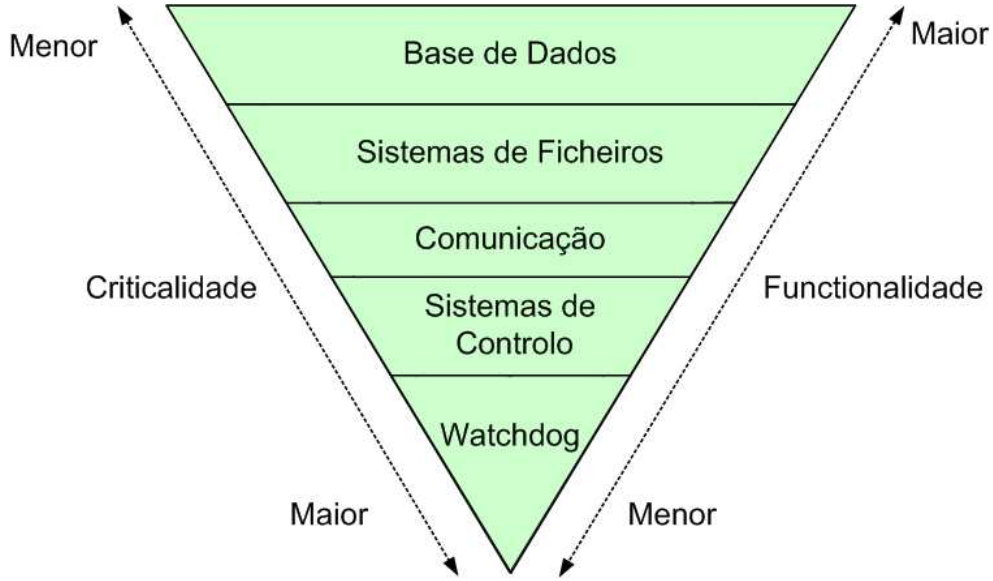


Figura 4.1: Exemplo de aplicações com diferentes graus de criticalidade/funcionalidade

Um exemplo funcional deste compartimentação e composição de aplicações encontra-se na especificação da norma ARINC 653. Esta especificação foi criada pela APEX Working Group (ARI, 2003; ARI, 1991) e tem como origem a indústria de aviação civil, com o objectivo de fornecer um suporte ao desenvolvimento de aplicações. Surge como um componente que permite abstrair ao programador de aplicações abstrair-se da maquinaria e do Sistema Operativo, permitindo aceder a serviços e recursos através de uma interface abstracta bem definida e independente da linguagem de programação.

A primeira versão da especificação ARINC 653 foi publicada em 1997 e tem como origem o relatório ARINC 651. Foi pela primeira vez posta à prova no Boeing 777. Actualmente encontra-se presente nas aeronaves Airbus A380, A400M e Boeing 787. É actualmente evidente a aposta da indústria aeronáutica na especificação ARINC 653.

A especificação ARINC 653 define o modo como as tarefas devem ser agrupadas dentro de partições e quais as regras de escalonamento e protecção de memória que devem ser cumpridas.

A especificação ARINC 653 estabelece ainda uma interface com o programador (API) independente da linguagem designada por APEX (APplication EXecutive) dentro do contexto do IMA (Integrated Modular Avionics). Esta API estabelece como a informação deve ser transferida (estati-

camente através da configuração ou dinamicamente através de serviços) e qual o comportamento dos serviços fornecidos pelo Sistema Operativo, Figura 4.2.

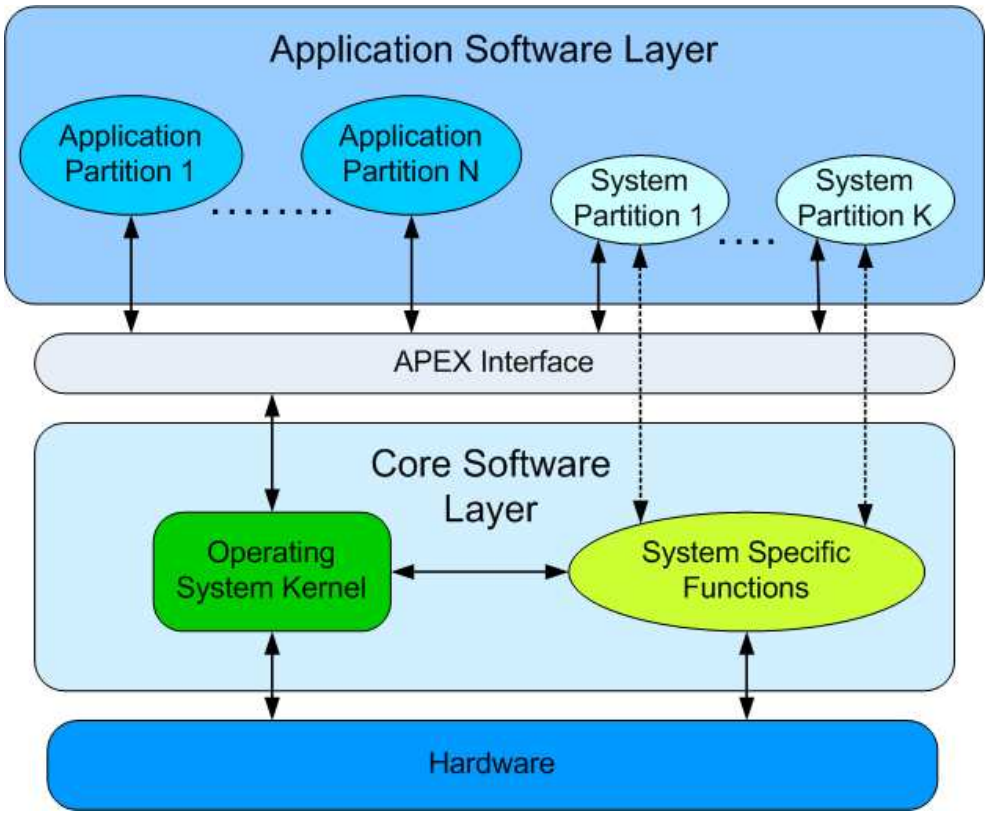


Figura 4.2: Arquitectura da especificação ARINC 653

Este capítulo apresenta resumidamente a especificação ARINC 653 e discute os mecanismos necessários para adequar os Sistemas Operativos já existentes à norma ARINC 653. De forma a reduzir os custos de construção, é abordado como implementar estes mecanismos recorrendo a Sistemas Operativos de Tempo-Real COTS. Em particular, é discutido como podem ser integrados no RTEMS.

4.1 Conceito de Partição

O conceito de partição dentro do contexto do Integrated Modular Avionics surge como um dos mais importantes, sendo fundamental para assegurar uma separação funcional e protecção entre aplicações. O conceito de partição tem como objectivo implementar mecanismos de tolerância a

faltas, prevenindo que a falha de um componente se propague a outros. Também é útil aos procedimentos de verificação, validação e certificação.

A especificação ARINC 653 descreve uma partição como sendo, em termos gerais, um programa num ambiente de aplicação única, isto é, tendo os seus espaços de código e dados, os seus atributos de configuração e contexto de execução, onde um ou mais processos se executam concorrentemente acedendo aos recursos da infra-estrutura computacional.

A especificação exige que as funções num módulo sejam compartimentadas com base no espaço (segregação espacial) e no tempo (segregação temporal). Uma partição é, portanto, um programa da aplicação desenhada para satisfazer estes requisitos. Um núcleo (*core*), cf. Figura 4.2, é responsável por assegurar estes dois tipos de segregação e pela gestão das partições individualmente.

A segregação espacial é obtida através da alocação de zonas pré-determinadas de espaço (memória) para cada partição. Assim, é estritamente proibido o acesso de uma partição a zonas de memória fora do seu espaço.

A segregação temporal define que cada partição é executada em intervalos de tempo pré-determinados. Dentro de cada partição, o fluxo de execução pode alternar entre várias tarefas.

A configuração das zonas de memória, janelas de tempo, etc., necessárias para a definição de cada partição é da responsabilidade do integrador de sistemas e mantida em tabelas de configuração.

Dado que o sistema se encontra dividido em várias aplicações, é necessário impedir que duas aplicações acedam ao mesmo dispositivo causando incoerências no acesso ao espaço de E/S. Desta forma é criado o conceito de partição sistema, Figura 4.2. A cada partição sistema é-lhe atribuída um conjunto de funções adicionais que lhe permitem comunicar com o mundo exterior. Salvo raras excepções, cada partição de sistema está encarregue de um (ou mais) dispositivos (e.g. motor de um avião, rede de comunicação).

4.1.1 Segregação Temporal

A especificação ARINC 653 define quais as janelas temporais onde cada partição é executada. Esta atribuição é feita *off-line* e é uma função do integrador de sistemas.

Observando a Figura 4.3 nota-se uma semelhança entre o escalonamento das partições e a arquitectura *time-triggered*. De facto, as partições seguem uma filosofia TT. Dado que as mudanças de partições apenas se podem fazer em determinados instantes, esta arquitectura possui algumas das desvantagens associadas com a arquitectura TT:

- É possível a existência de intervalos de tempo sem nenhuma tarefa para executar
- Não permite uma resposta tão rápida a eventos

Caso a janela temporal de uma dada partição esteja sobre-dimensionada a partição está a gastar recursos que poderiam ser aproveitados por outras partições.

A resposta a um evento necessita que a partição que o processa esteja em execução. No pior caso, o sistema pode demorar um ciclo inteiro (cf MTF na Figura 4.3) até que execute a partição associada ao evento. Logo, a resposta ao evento pode demorar algum tempo, mesmo que a tarefa que o processe seja de alta prioridade.

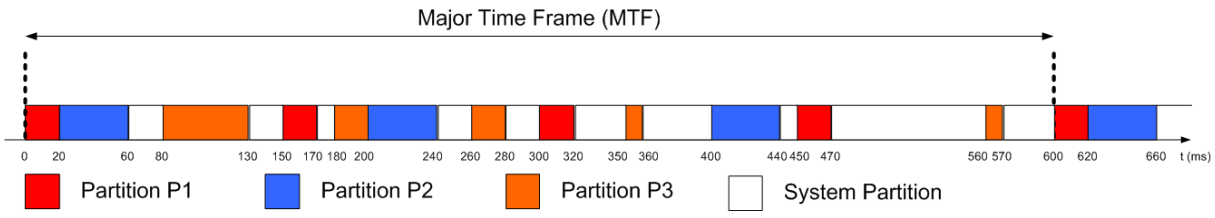


Figura 4.3: Exemplo do escalonamento de partições

Contudo, existem vantagens extremamente relevantes na segregação temporal por meio de uma arquitectura TT:

- Análise independente de cada partição
- Maior tolerância a faltas, através do controlo da interferência entre partições
- Menores custos de verificação, validação e certificação

Como já foi referido no Capítulo 2, a arquitectura TT permite separar temporalmente os seus componentes, dado que não é possível que uma partição independente interfira temporalmente com

outra. Este facto permite diminuir os custos de validação do sistema, uma vez que não é necessário, neste caso, analisar o sistema como um todo. A arquitectura TT também possui intrinsecamente uma maior tolerância a faltas, uma vez que impede que um dado componente ocupe mais tempo do que devia. Impõe, assim, um controlo adicional sobre os aspectos temporais.

Em relação aos domínios alvo da especificação ARINC 653, nomeadamente os ambientes aeronáutico e espacial, ambos suportam as desvantagens da utilização da arquitectura TT, isto é, possuem uma dinâmica lenta[1] e não exigem muita rapidez ao processamento de eventos externos (Kopetz, 1993).

4.1.2 Segregação Espacial

Da mesma forma que é impedido que uma partição seja executada fora da janela temporal que lhe é atribuída, também não pode aceder a posições de memória fora do seu espaço de endereçamento pré-definido. Qualquer acesso de leitura/escrita/execução é estritamente proibido e caso aconteça (e.g. devido a ponteiros mal inicializados) é despoletado um evento e o núcleo é notificado da falta originada nessa partição. Desta forma, é impedida a propagação deste tipo de faltas entre partições.

A segregação espacial é tipicamente feita através de maquinaria especializada, e.g. MMU (Memory Management Unit). Como tal, é normalmente realizado em paralelo ao processamento normal da CPU, não acarretando custos a nível temporal. Os custos a nível de operação temporal restringem-se à computação entre partições mas mesmo estes podem, regra geral, ser minimizados por uma gestão adequada dos mecanismos de protecção de memória. Desta forma as desvantagens que acarreta centram-se num ligeiro aumento do custo (mais maquinaria) e maiores necessidades energéticas. No entanto, estas desvantagens não são comprometedoras nos ambientes aeronáutico e espacial.

4.1.3 APEX - APplication EXecutive

A especificação ARINC 653 estabelece um conjunto de primitivas através das quais cada partição comunica com o Sistema Operativo. Esta API é independente da linguagem utilizada (C, Ada) de

[1] Tarefas periódicas de controlo possuem tipicamente baixas frequências (no máximo rondam os 60 Hz).

forma a criar uma maior portabilidade para vários sistemas. O conjunto de serviços oferecido por esta API permite, por exemplo, criar tarefas periódicas, comunicação entre tarefas, comunicação entre partições, serviços de configuração do sistema, etc. Os serviços especificados encontram-se divididos nas seguintes categorias:

- Gestão de Tarefas
- Gestão do Tempo
- Gestão de Memória
- Comunicação intra-partição
- Comunicação entre partições
- Controlo de Sanidade
- Configuração

Os serviços requiridos à gestão de tarefas prendem-se principalmente com a inicialização das tarefas e o seu escalonamento. A especificação ARINC 653 impõe um escalonamento de tarefas por prioridades estáticas e preemptivo.

Os serviços de gestão do tempo permitem estabelecer o escalonamento das partições/tarefas e criam um relógio global para todas as partições.

Os serviços de memória estabelecem quais as zonas de memória que cada partição pode aceder livremente.

Os serviços de comunicação intra-partição centram-se em mecanismos de sincronização e comunicação entre tarefas dentro da mesma partição. Por exemplo, têm-se os semáforos, buffers, eventos, etc.

Os serviços de entre partições estão associados a mecanismos de comunicação (a sincronização de partições não é necessária). Como exemplo, encontam-se os *sampling* e *queueing ports*.

Os serviços de controlo de sanidade (*health monitoring*) permitem detectar erros (e.g. falha de uma meta temporal associada a tarefas periódicas, acessos de memória inválidos, etc) e a sua recuperação através de primitivas fornecidas pela aplicação.

Os serviços de configuração permitem estabelecer *off-line* todos os objectos que são necessários a cada partição e os mecanismos de comunicação entre partições. Por exemplo, quais os canais de comunicação entre uma partição e outra e quais os seus atributos, o número de tarefas de uma partição

e quais os seus atributos, o número de semáforos de uma partição, etc.

4.2 Implementação da Especificação ARINC 653

Esta secção discute como implementar a especificação ARINC 653 num sistema computacional. A solução óbvia de construir todo o SO de raíz implica grandes custos de desenvolvimento e certificação. Por outro lado, e tendo em consideração o conjunto de serviços especificados pela APEX, torna-se claro que o reaproveitamento de Sistemas Operativos já disponíveis é uma solução menos dispendiosa, uma vez que a sua certificação pode inclusive ter sido realizada.

Desta forma, é proposta a extensão dos serviços oferecidos por um COTS RTOS (Real-Time Operating System) de forma a implementar a especificação ARINC 653.

4.2.1 Utilização de COTS RTOS

A especificação ARINC 653 requer um conjunto de serviços cuja funcionalidade fundamental se encontra presente na grande maioria dos RTOS. Por exemplo: o uso de escalonamento de tarefas baseado em prioridades fixas e preemptivo; o suporte a mecanismos de comunicação e sincronização entre tarefas; gestão do tempo; etc.

Como tal, a implementação da especificação ARINC 653 acarreta custos consideravelmente menores quando se parte de um RTOS. Numa primeira aproximação poder-se-ia pensar que esta solução requer o conhecimento do funcionamento interno do *kernel*, nomeadamente sobre o escalonamento, gestão das pilhas associadas às tarefas, gestão de interrupções, etc. Como tal, nem todos os RTOS disponíveis no mercado poderiam ser considerados como solução à implementação da especificação ARINC 653. Em particular, seria necessário conhecer o funcionamento interno do Sistema Operativo e aceder/modificar funções internas que não estão normalmente disponíveis na API fornecida.

Como iremos demonstrar isto não é necessariamente verdade e a extensão de um RTOS pode ser feita em larga medida de uma forma modular, de forma a evitar alterações estruturais quando se muda de versão ou mesmo de RTOS.

4.2.2 Escolha da Arquitectura

Partindo de um COTS RTOS, propõem-se duas arquitecturas para a implementação da especificação ARINC 653:

- Single-Executive Core (SEC)
- Multi-Executive Core (MEC)

Single-Executive Core

A arquitectura SEC encontra-se ilustrada na Figura 4.4. Nesta arquitectura o núcleo do RTOS controla o fluxo de todos os processos de todas as partições. Desta forma existe apenas uma camada APEX e um núcleo RTOS para todas as partições.

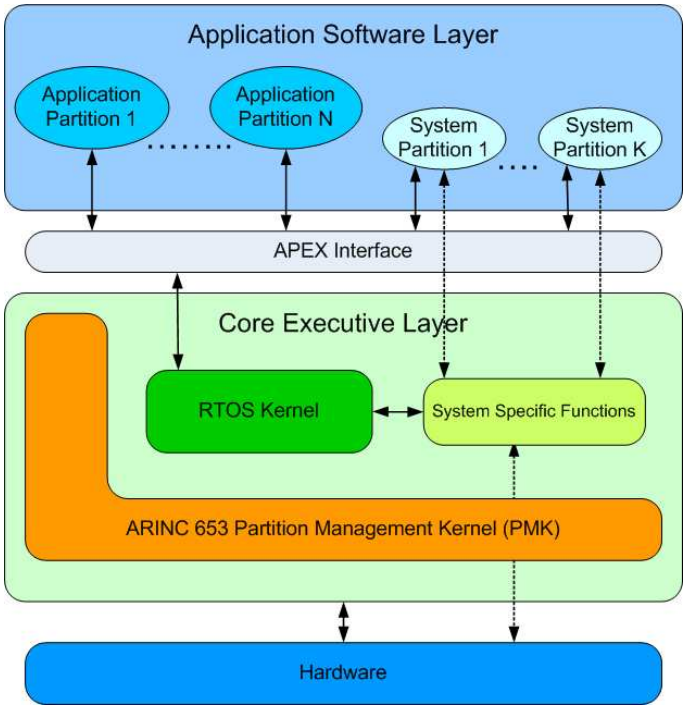


Figura 4.4: Arquitectura Single-Executive Core

Um dos principais óbices à concretização desta solução rezide nas desvantagens que apresenta em relação ao escalonamento dos processos, em particular ao tempo requerido durante uma mudança da partição em execução:

- Existindo um escalonador para gerir todas as tarefas de todas as partições, é necessário garantir que apenas as tarefas da partição em execução podem ser executadas. Para isso é necessário suspender as tarefas das restantes partições. Quando é decidido que é necessário mudar a partição em execução para a partição "herdeira", é necessário suspender todas as tarefas da partição em execução e activar todas as tarefas da partição herdeira. Este processo está intimamente ligado com o número de tarefas presentes em cada partição, pelo que o tempo gasto por esta operação depende do número de objectos do sistema, o que compromete as características de tempo-real.

Note-se, no entanto, que não é necessário a existência de um escalonador único para gerir todas as tarefas de todas as partições. É possível a criação de um escalonador separado para cada partição. Este conceito é, no entanto, difícil de concretizar uma vez que requer mudanças extensas no código do núcleo do RTOS. Em particular, é necessário analisar todos os locais onde o escalonador é invocado, criar "barreiras" de modo a que uma tarefa apenas comunique directamente com outras tarefas da mesma partição e apontar para o escalonador da partição em execução.

Para além da operação de mudança de partição, a gestão do tempo também pode comprometer a segregação temporal entre partições imposta pela especificação ARINC 653:

- As tarefas periódicas são activadas tipicamente pelo despoletamento de uma interrupção de relógio de sistema (*clock tick*). Logo, é necessário em cada interrupção de relógio anunciar a passagem do tempo a todas as tarefas e determinar quais as que mudaram de estado (e.g. a passagem do estado suspenso e bloqueado para o estado suspenso ou do estado bloqueado para executável). No entanto, o procedimento de mudança de estado das tarefas depende normalmente do número de tarefas que é necessário alterar, i.e., é necessário mudar o estado das tarefas de todas as partições individualmente. Logo, a complexidade das mudanças de estado causadas pela passagem do tempo depende não só do número de tarefas presentes na partição em execução mas também de todas as outras partições.

Em relação à implementação da arquitectura SEC, esta solução é relativamente fácil pois não requer mudanças extensas nem um conhecimento de baixo nível do RTOS que se pretende estender. A construção do executável final também não requer mudanças estruturais no ambiente de desenvolvimento, i.e., o uso das ferramentas de compilação nativas não sofre alterações.

Multi-Executive Core

Em contra-posição com a arquitectura SEC, tem-se a arquitectura MEC, Figure 4.5. Nesta arquitectura existe um kernel RTOS por cada partição. A gestão das partições é feita por um PMK (Partition Management Kernel) especializado. Dentro de cada partição as tarefas são escalonadas através do RTOS local.

Desta forma, aumenta-se a tolerância a faltas, pois se um dado RTOS local possuir um erro, este não se propaga às restantes partições. Aumenta-se também a portabilidade pois a arquitectura não está dependente de uma versão particular de um RTOS ou mesmo de diferentes RTOS.

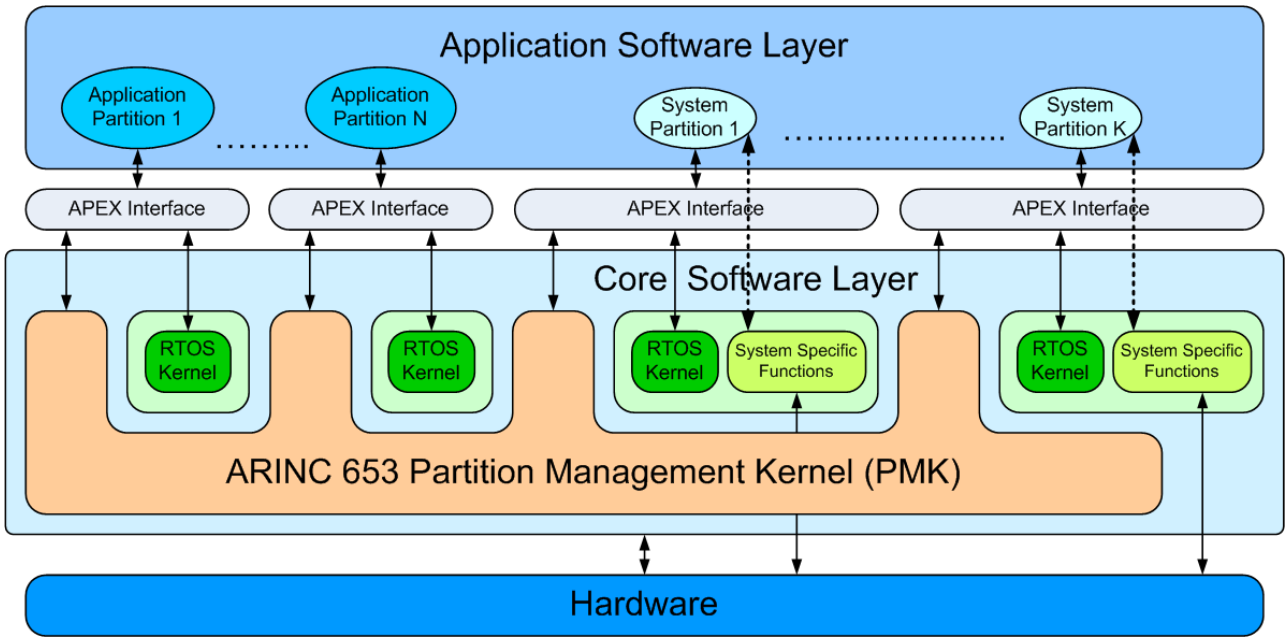


Figura 4.5: Arquitectura Multi-Executive Core

Por outro lado, esta arquitectura exige uma maior dimensão da imagem final, dado que é necessário um RTOS por cada partição. requer ainda modificações estruturais na utilização das ferramentas de compilação pois os símbolos de cada partição (nomes de variáveis, funções, etc) em cada partição podem ser iguais e dão origem a erros de compilação.

No RTOS nativo existe um escalonador para todas as tarefas. Dentro da arquitectura MEC, existe um escalonador de tarefas por cada partição. Desta forma, existem N escalonadores de tarefas para N partições. A integração das partições estabelece um novo nível de escalonamento, originando um

escalonamento hierárquico de dois níveis.

O escalonador de partições escolhe, em momentos específicos, qual a partição que deve ser executada. O controlo da CPU é depois dado ao escalonador de tarefas dessa partição que escolhe qual a tarefa de maior prioridade num estado executável.

Durante uma mudança de contexto de partição, o escalonador de tarefas de um dado RTOS é activado. Todas as operações de escalonamento, incluindo o processamento de *clock ticks*, encontram-se restringidos à partição correspondente. Logo, a interferência temporal entre partições é reduzida ao mínimo.

O tempo extra necessário ao processamento dos *clock ticks* e actualização do estado das tarefas que daí derivam encontra-se durante a mudança de contexto das partições. Este intervalo é uma função do número de *clock ticks* que são necessários processar e do número de processos que mudam de estado. Contudo, a maior parte dos RTOS permite uma optimização desta operação.

Esta técnica permite ainda a co-existência de diferentes RTOS (e.g RTEMS e eCOS) em diferentes partições. Desta forma é possível, por exemplo, que os processos de uma partição sejam escalonados pelo *rate monotonic scheduler*, os de outra partição pelo *earliest deadline first scheduler* ou *least laxity first scheduler*, etc. Contudo, é de referir que apenas o escalonamento preemptivo de prioridades fixas está em conformidade com a especificação ARINC 653.

Comparação entre SEC *versus* MEC

As Tabelas 4.1 e 4.2 estabelecem um resumo da comparação dos atributos da arquitectura e do escalonamento de processos dos modelos SEC e MEC.

Atributos da Arquitectura								Decisão
Arquitectura	Versatilidade	Modularidade	Configurabilidade	Integridade	Tolerância a Faltas	Dimensão	Ferramentas de Desenvolvimento	
SEC	fraco	fraco	fraco	razoável	razoável	óptima	canónico	×
MEC	bom	bom	bom	bom	bom	sub-óptima	canónico mais ferramentas adicionais	√

Tabela 4.1: Comparação das arquitecturas SEC e MEC

Como se pode observar pelas Tabela 4.1 e 4.2, a arquitectura MEC apresenta diversas vantagens sobre a arquitectura SEC. A única desvantagem consiste no tamanho do executável final, onde a opção MEC exige uma dimensão maior. Esta desvantagem não é, no entanto, comprometedora da solução

Atributos do Escalonamento de Processos					Decisão
Arquitectura	Versatilidade	Modularidade	Tempo de mudança de partições	Interferência Temporal	
SEC	fraco	fraco	alta	ligeira	×
MEC	bom	bom	ligeira	mínima	√

Tabela 4.2: Comparação do escalonamento de processos nas arquitecturas SEC e MEC

uma vez que os sistemas alvo possuem uma grande capacidade de memória (cerca de 1000 KiB) em relação às dimensões dos kernels RTOS actuais (podem chegar a 30 KiB). É de mencionar que se uma dada partição não necessitar de um grande conjunto de serviços disponíveis pela APEX (e.g. sincronização por semáforos) é possível diminuir a dimensão da imagem dessa partição através da configuração do RTOS correspondente.

Dadas as vantagens inerentes à **arquitectura MEC**, opta-se por esta solução, embora seja mais difícil de concretizar que a opção SEC.

4.2.3 Arquitectura MEC

Esta secção discute os detalhes fundamentais a arquitectura escolhida: MEC (Multi-Executive Core), ilustrada na Figura 4.5. Iremos descrever em maior pormenor:

- Os requisitos fundamentais do hardware e das propriedades do escalonamento RTOS para um implementação fácil do escalonamento de partições e processos dentro da arquitectura MEC
- Como implementar estes recursos, focando num RTOS particular: RTEMS

Em relação à arquitectura inicial da especificação ARINC 653, Figura 4.2, a arquitectura MEC introduz o módulo Partition Management Kernel (PMK). O PMK é constituído pelos componentes:

- HAL (Hardware Abstraction Layer) - implementa um nível de abstração com acesso aos recursos materiais (*hardware*)
- Inicialização do sistema - inicializa o sistema (hardware) e as partições e os respectivos RTOS
- Gestor de tempo - gere o relógio interno do PMK e actualiza as restantes partições

- Expedidor de partições - responsável pela salvaguarda do contexto de execução da partição que vai ser desactivada e restauração do contexto da partição que vai ser executada de seguida (partição "herdeira"). Também actualiza os mecanismos de protecção de memória
- Escalonador de partições - selecciona a partição que deve ser executada em cada *clock tick* de acordo com um esquema cíclico fixo
- Comunicação inter-partições - permite a troca de informação entre as partições sem violar os conceitos de segregação espacial e temporal

O PMK é responsável pela inicialização do sistema e das partições, da comunicação com o hardware, da comunicação entre as partições e do escalonamento das partições. O escalonamento das tarefas é da responsabilidade do escalonador interno de cada partição.

4.2.4 Protecção Temporal

A protecção temporal na especificação ARINC 653 exige que as partições sejam executadas apenas em janelas temporais pré-definidas. Este comportamento assegura que cada partição possui uma janela temporal onde se pode executar, mesmo quando as restantes partições possam falhar. As partições são escalonadas segundo um janela temporal cíclica. Dentro de cada partição, os processos são escalonados segundo um algoritmo preemptivo de prioridades fixas.

O componente do PMK responsável pela decisão de qual a partição a ser executada (partição herdeira) é o *escalonador de partições*. O escalonador de partições toma as decisões com base no tempo actual e com a ajuda de uma tabela pré-definida. O *expedidor de partições* é responsável por efectuar a mudança da partição em execução pela partição herdeira.

Como previamente discutido, a arquitectura MEC necessita de um escalonador hierárquico de dois níveis. Logo, cada partição deve fornecer o seu próprio escalonador de processos. Supondo que o RTOS possui um escalonador que obedece à especificação ARINC 653 (escalonamento preemptivo de prioridades fixas), este assumirá este papel dentro da arquitectura. Qualquer RTOS com o escalonador de processos com estas características é passível de assumir este papel. Desta forma é possível a utilização de diferentes RTOS, um por partição.

Escalonamento de Partições

O escalonamento de partições definido pela especificação ARINC 653 é temporalmente estritamente determinístico, i. e., cada partição possui uma janela temporal na qual possui o controlo da plataforma computacional. Cada partição é escalonada segundo um esquema cíclico fixo. O RTOS kernel mantém uma MTF (Major Time Frame) de duração fixa e pré-definida, que é repetida periodicamente, como ilustrado na Figura 4.3.

Como se pode constatar pela Figura 4.3, a partição em execução apenas muda em instantes bem definidos no tempo. Estes instantes são definidos pelo MDC (Maior Divisor Comum) dos instantes de mudanças de partição. No exemplo da Figura 4.3, o MDC corresponde a 10 ms. Logo, se o escalonador de partições for activado com um intervalo de 10 ms, apanha todos os instantes de mudanças de partição.

Definindo um MDC como um múltiplo de um *clock tick*, apenas é necessário que o escalonamento de partições seja executado em cada *clock tick* (Rufino *et al.* , 2007). No exemplo da Figura 4.3 o *clock tick* pode ser 1 ms (ou 2, 5 ou 10 ms). O *clock tick* corresponde à execução de uma interrupção periódica despoletada pelo Timer interno da CPU.

O despoletamento da interrupção derivada do *clock tick* é usual em praticamente todas as arquitecturas computacionais, e. g., Intel IA-32 ou SPARC. Logo, através da ISR (Interrupt Service Routine) associada ao *clock tick* o escalonador de partições pode decidir qual a próxima partição a ser executada (partição herdeira), Figura 4.6.

A Figura 4.7 ilustra o pseudo-código que contém o handler chamado em cada *clock tick*. De acordo com este pseudo-código, a variável global $\_table$ contém uma tabela com os instantes de mudanças de partição e qual a partição herdeira nesses instantes. Esta tabela é preenchida durante a inicialização tal como as restantes variáveis globais. A passagem do tempo é reflectida na variável $\_ticks$, que contém o número total de *clock ticks* desde que o sistema arrancou. Quando esta variável é actualizada, é verificado se nesse *clock tick* é necessário uma mudança de partição. Em caso afirmativo, o escalonador muda a partição herdeira e o iterador da tabela, $\_table\_iterator$, é incrementado seguindo um movimento circular pela tabela.

A vantagem deste algoritmo reside na sua complexidade $O(1)$, i.e., o processo de decisão da

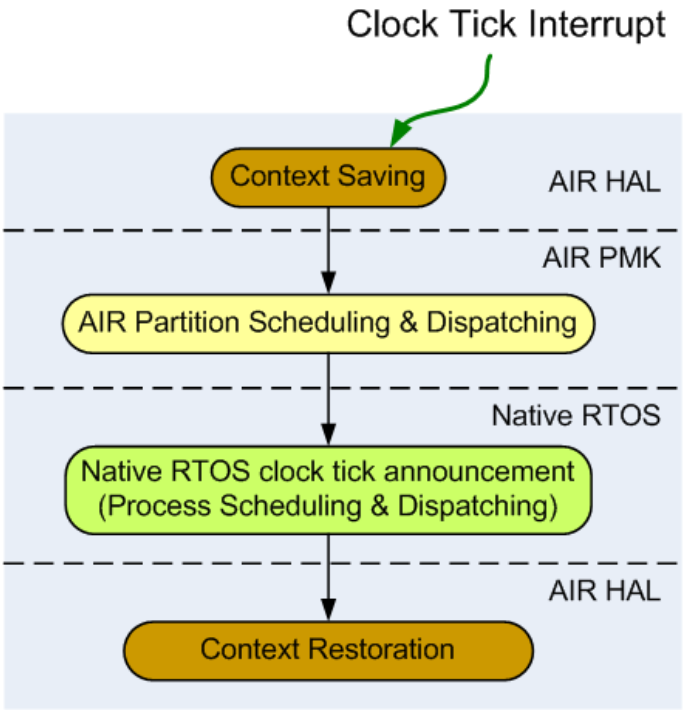


Figura 4.6: ISR do *clock tick*

partição herdeira não depende do número de partições ou outros objectos do sistema. Esta propriedade é um requisito dos sistemas de tempo-real.

Escalonamento de Processos

Dentro de cada partição podem co-existir vários processos escalonados em função do seu estado e prioridade. Estes processos possuem um grau elevado de inter-operacionalidade fornecida através da interface APEX. Esta funcionalidade exige um controlo rígido sobre as transições de estado dos processos, tal como ilustrado na Figura 4.8.

As transições ilustradas na Figura 4.8 são despoletadas através de chamadas a primitivas APEX. Por exemplo, a chamada à primitiva STOP_SELF transita o processo em execução para um estado suspenso.

Dada a funcionalidade básica fornecida pelo RTEMS, o seu escalonador de tarefas assume o papel de escalonador de processos dentro do contexto da arquitectura MEC.

```
/* Function only called when a clock tick has occurred */
void AIRPartitionScheduler()
{
    _ticks++;
    /* check if this is a partition preemption point */
    if(_table.tick[_table_iterator] == _ticks%MTF)
    {
        /* change the heir partition */
        _heirPartition = _table.partition[_table_iterator];
        /* increase the table iterator through a cyclic fashion*/
        _table_iterator++;
        if(_table_iterator == NUMBER_PARTITON_PREEMTIVE_POINTS)
            _table_iterator = 0;
    }
}
```

Figura 4.7: Pseudo-código do handler de *clock tick*

A ligação entre o PMK e o escalonador de processos do RTOS é realizada durante o processamento da interrupção *clock tick*, como ilustrado na Figura 4.6. Quando o escalonador de partições decide que é necessário mudar de partição, é invocado o expedidor de partições que promove a partição herdeira para ser a próxima partição em execução.

Esta mudança de partição envolve, para além de outras coisas, o anúncio para a partição herdeira do tempo desde que processou o último *clock tick* até ao *clock tick* actual. Isto é feito através de uma primitiva RTOS designada rtos_clock_tick. Normalmente, esta rotina é apenas invocada dentro de uma rotina de interrupção para anunciar a passagem de um *clock tick*. Dentro da arquitectura MEC, esta rotina é agora invocada um número variável de vezes correspondentes ao número de *clock ticks* que passaram (Rufino *et al.* , 2007). Uma versão optimizada desta rotina pode receber como argumento o número de *clock ticks* passados, de forma a ser invocada apenas uma vez.

No RTEMS, uma rotina semelhante é fornecida através da API nativa, designada por rtems_clock_tick. Apenas anuncia a ocorrência de um único *clock tick*. Embora a complexidade desta função esteja associada ao número de processos à espera do *clock tick* (se N processos forem activados no *clock tick* actual são necessárias N activações individuais dos processos), pode ser optimizada para anunciar um número variável de *clock ticks*. No entanto, no caso do RTEMS, esta rotina também invoca o expedidor de processos, o que implica que o PMK perca o controlo do fluxo de

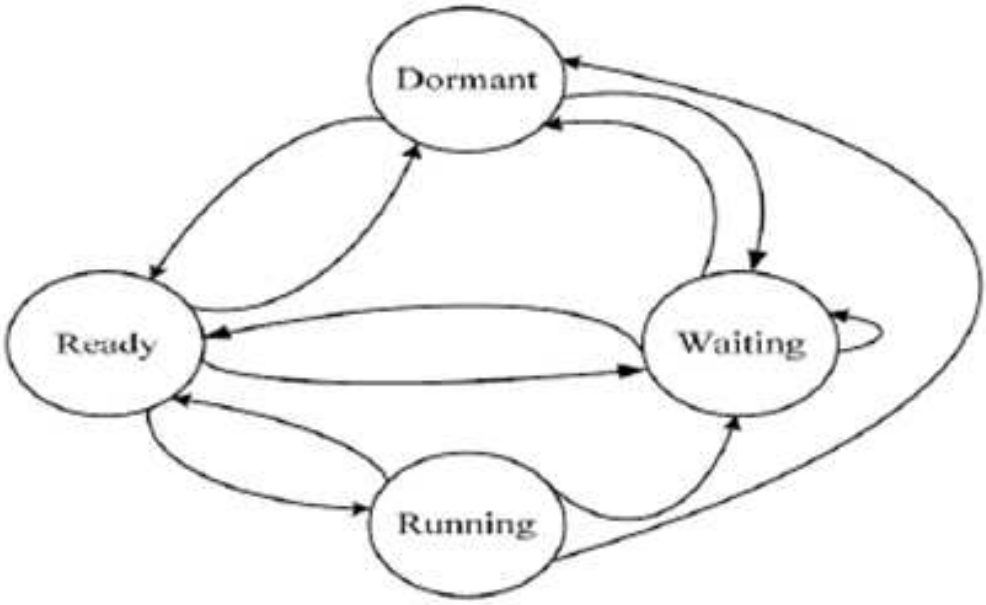


Figura 4.8: Transições de estado dos processos

execução, não podendo chamar a função mais vezes de forma a anunciar mais *clock ticks*. Para prevenir isto, esta função tem que ser modificada de forma a, ou não invocar o expedidor de processos, ou a receber como argumento um número variável de *clock ticks* e de seguida invocar o expedidor de processos. Após uma análise do código do RTEMS, concluiu-se que a primeira solução é mais simples de implementar, dado que as mudanças requeridas são desprezáveis.

Logo, os requisitos fundamentais que o RTOS nativo tem que fornecer para uma implementação simples da arquitectura MEC estão sujeitos a uma de duas opções:

1. fornecer um serviço que:
    - anuncia um número variável de *clock ticks* ao kernel RTOS e invoca de seguida o expedidor de processos
2. fornecer um serviço que:
    - anuncia a ocorrência de um único *clock tick* ao kernel RTOS
    - invoca o expedidor de processos

Em geral, nenhuma das opções está disponíveis na API nativa, o que requer modificações ligeiras ao RTOS. A primeira opção é preferível, uma vez que permite optmizar o processamento de *clock ticks* e as modificações necessárias são desprezáveis (não é necessário modificar o kernel, apenas a adição de uma nova função).

4.2.5 Protecção Espacial

A especificação ARINC 653 impõe que os componentes de software numa dada partição, com a excepção para os serviços de comunicação inter-partições, não podem aceder ao espaço de endereçamento de outras partições (ou pelo menos com privilégios de escrita). Este mecanismo é conhecido como segregação espacial e implica o uso de mecanismos de protecção no acesso a um dado endereço. Estes mecanismos podem envolver a protecção tanto de endereços de memória como de entrada/saída (E/S).

De modo a serem eficientes, estes mecanismos devem ser suportados directamente em maquinaria (*hardware*), tipicamente usando a MMU (Memory Management Unit). Existem dois mecanismos distintos de protecção de memória:

1. Monitorização - este método simplesmente verifica o endereço que se pretende aceder com uma tabela de endereços válidos. Este método é simples de implementar e é inerentemente menos vulnerável a erros (bit-flips) devidos a falhas únicas (single-event upsets - SEU). Em particular, este método é usado na implementação no processador SPARC LEON2 dos mecanismos de protecção de escrita de memória do processador ATMEL AT697E.

2. Translação - este método traduz o endereço através de um descritor da MMU. Este método possui uma complexidade semelhante ao anterior, embora possua um maior grau de flexibilidade no que diz respeito ao mapeamento dos endereços lógicos nos endereços físicos. No entanto, é mais vulnerável a erros de bit-flops em cenários de falha única (SEU). Este mecanismo é usado no modelo segmentado utilizado na arquitectura IA-32.

As partições são executadas em modo utilizador e cada acesso de memória é verificado (e possivelmente traduzido) em tempo de execução pela MMU, através de tabelas de descritores para

endereços de memória e E/S. Um sinal de excepção é gerado quando é detectado um acesso não autorizado.

Dentro da arquitectura MEC, estes mecanismos de protecção de memória são actualizados pelo expedidor de partições, i.e., quando se muda a partição em execução actualizam-se os descritores da MMU. A comunicação com o a maquinaria é delegada ao HAL (Hardware Abstraction Layer).

Dentro do contexto da especificação ARINC 653, não existe segregação espacial ao nível do processo dentro de cada partição.

4.2.5.1 Mecanismos de Protecção de Memória na Arquitectura IA-32

A arquitectura IA-32 fornece directamente os mecanismos de protecção de memória por segmentação (translação e a sua validação), como ilustrado na Figura 4.9.

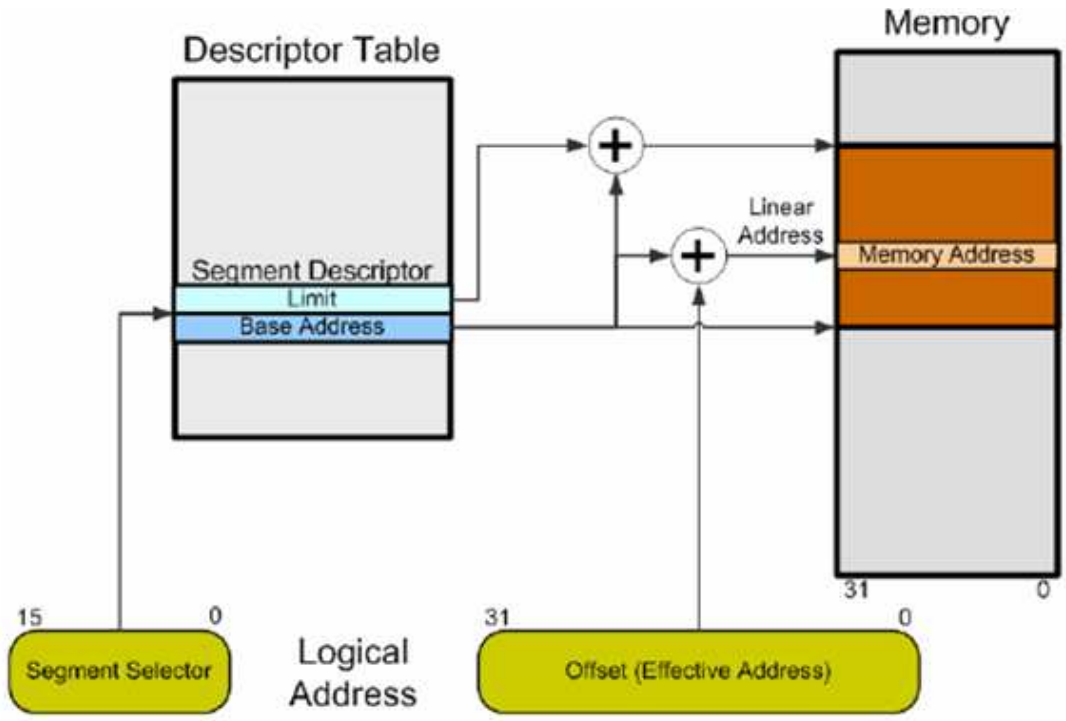


Figura 4.9: Mecanismos de protecção de memória por segmentação (translação) da arquitectura Intel IA-32

Usando a nomenclatura da arquitectura IA-32, um segmento é uma zona protegida de endereçamento que pode ser usado para suportar as partições ARINC 653. Os mecanismos de gestão

de memória são essencialmente tradução de endereços e validação de espaços de endereçamento. Uma excepção é despoletada quando é detectado um acesso a um endereço inválido.

O mecanismo de segmentação de memória na arquitectura IA-32 funciona de uma forma simples. O descritor da MMU possui a base e o limite do segmento. O endereço físico é calculado somando o deslocamento do endereço lógico ao endereço base do descritor. Quando se muda de partição o expedidor de partições actualiza o descritor da MMU que deve ser utilizado. Esta operação é rápida uma vez que apenas o índice na tabela de descritores é alterado (consiste na alteração do registo CS do CPU). Caso o endereço lógico seja superior ao limite especificado no descritor, a MMU detecta que o endereço é inválido.

4.2.5.2 Mecanismos de Protecção de Memória na Arquitectura SPARC

Apesar da arquitectura base SPARC V8 especificar a implementação de uma MMU rudimentar, alguns produtos comuns incluem mecanismos mais elaborados. Como exemplos, discutem-se de seguida as implementações ATMEL e GAISLER.

4.2.5.3 Processador ATMEL SPARC V8 AT 697E LEON2-FT

Esta implementação do processador SPARC LEON2 inclui dois registos para controlo dos mecanismos de protecção de memória, permitindo protecção de escrita sobre blocos de 32 KiB, como ilustrado na Figura 4.10. O mecanismo especifica dois campos:

1. um endereço que define os quinze bits mais significativos do bloco de endereço a ser protegido
2. uma máscara que especifica quais os bits do endereço definido anteriormente que devem ser monitorizados
    - Podem-se especificar dois modos de protecção: activo dentro do bloco; activo fora do bloco

Se uma violação de acesso é detectada, a escrita é parada e uma excepção é despoletada.

Figura 4.10: Mecanismos de protecção de memória da arquitectura ATMEL AT697E SPARC LEON2

4.2.5.4 Processador Gaisler SPARC V8 LEON3-FT

Esta implementação da arquitectura SPARC LEON3 inclui um componente facultativo que implementa a MMU descrita na especificação SPARC V8. No essêncial, especifica um esquema de páginas de três níveis para traduzir um endereço virtual de 32 bits para um endereço físico de 36 bits em modo paginado. O tamanho de cada página é de 4 KiB. O funcionamento deste mecanismo encontra-se ilustrado na Figura 4.11.

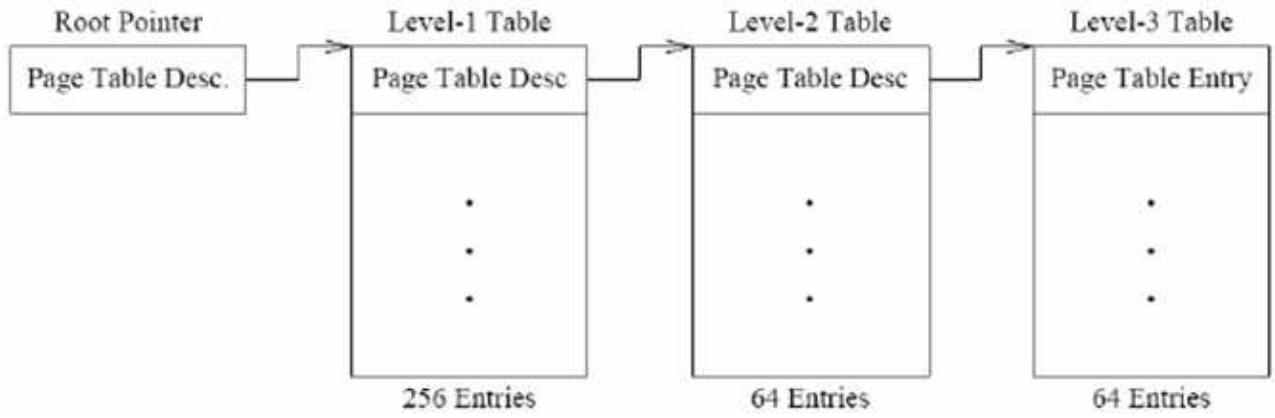


Figura 4.11: Mecanismos de protecção de memória por paginação (translação) da arquitectura Gaisler SPARC LEON3

Embora estes mecanismos tenham sido desenhados tendo em atenção a tolerância a faltas, o número de níveis de translação de endereços de memória é exagerado para as necessidades de sistemas ARINC 653. Uma implementação mais simples pode tornar o sistema mais robusto em relação a falhas da MMU mantendo a conformidade com a especificação ARINC 653.

4.2.6 Integração no RTEMS

Esta secção discute os aspectos de maior importância relativos à implementação da arquitectura MEC no RTEMS.

4.2.6.1 Integração da APEX

A API definida pela especificação ARINC 653, designada APEX, requer um conjunto de primitivas que fornecem um serviço de gestão de tarefas, sincronização/comunicação de tarefas, etc, discutido na secção 4.1.3. Estes serviços podem ser mapeados quase numa relação um-para-um com os serviços fornecidos pelo RTEMS. No entanto, dado que a interface nativa do RTEMS é consideravelmente mais complexa que a da APEX (e.g. memória dinâmica), é possível retirar módulos do RTEMS de forma a criar uma imagem mais reduzida. A Figura 4.12 resume quais os módulos do RTEMS que devem ser incluídos/excluídos para construir a interface APEX.

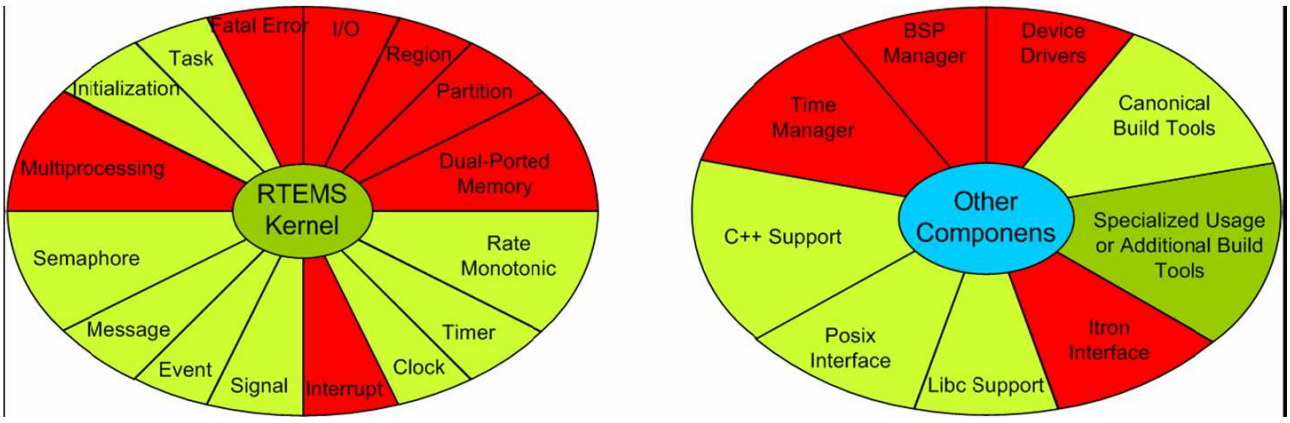


Figura 4.12: Módulos RTEMS que são incluídos/excluídos na integração na arquitectura MEC

4.2.6.2 Ferramentas de Desenvolvimento

Como discutido anteriormente, a construção de aplicações usando a arquitectura MEC requer modificações no uso das ferramentes de desenvolvimento da aplicação. Esta secção discute as mudanças necessárias e como solucioná-las de uma forma eficiente e a baixo custo.

O processo de desenvolvimento de aplicações usando somente o RTEMS encontra-se ilustrado na Figura 4.13. Como se pode observar, o cross-compiler cria uma biblioteca RTEMS que pode ser ligada (*linked*) com a aplicação para produzir uma imagem final. Esta imagem final é então carregada no ambiente alvo, e.g. SPARC, IA-32, etc.

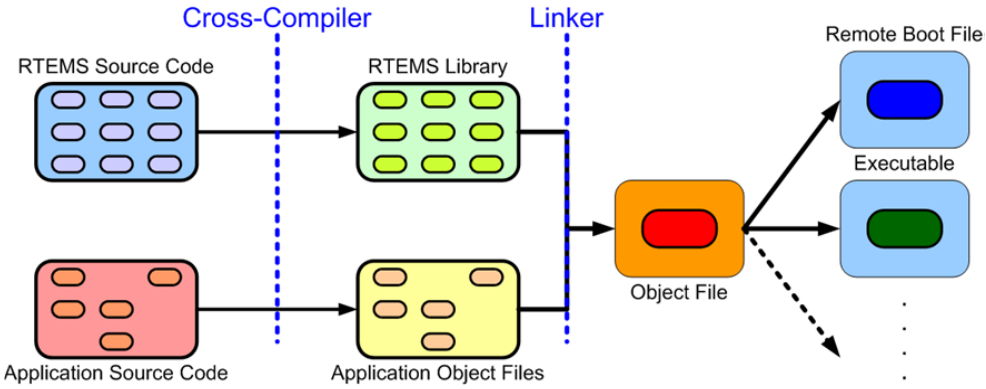


Figura 4.13: Produção de aplicações usando o RTEMS

Na arquitectura MEC é necessário juntar diversas aplicações como ilustra a Figura 4.14. Como se pode concluir, a biblioteca RTEMS encontra-se presente em cada uma das partições, o que origina erros de ligação (*linkage*) pois os mesmos símbolos (e.g. nomes de funções, variáveis, etc) estão declarados em mais que um módulo.

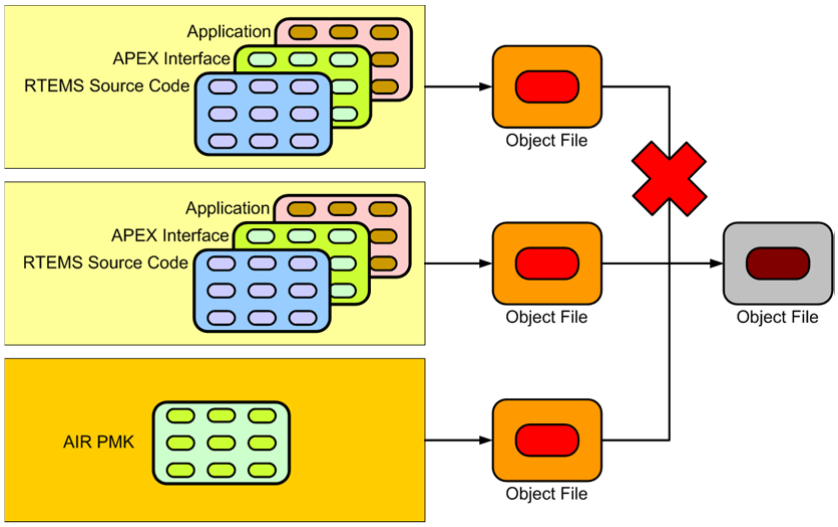


Figura 4.14: Produção de aplicações na arquitectura MEC usando as ferramentas nativas do RTEMS

A solução adoptada para este problema consiste na utilização de um filtro que altera o nome dos símbolos de cada uma das partições de forma a serem únicos, Figura 4.15. Em particular, é adicionado um prefixo diferente por partição. Assim, por exemplo, na partição "1" é adicionado o prefixo "P1". Desta forma eliminam-se os símbolos repetidos.

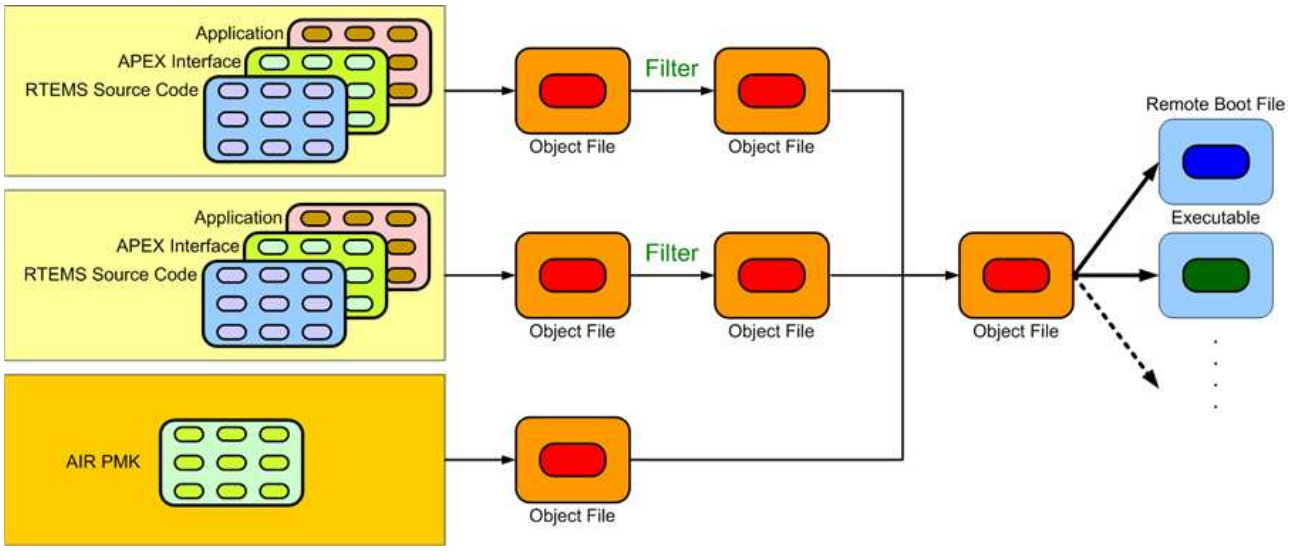


Figura 4.15: Produção de aplicações na arquitectura MEC usando as ferramentas nativas do RTEMS e um filtro adicional

Este filtro pode ser construído aproveitando as ferramentas da GNU (linker e objcopy), já presentes no desenvolvimento de aplicações no RTEMS, e modificando o seu uso. Em particular, a construção das partições requer a adição da opção -r (relocatable) durante a fase de ligação. Esta opção força o linker a não calcular os endereços finais dos símbolos, determinando apenas os endereços relativos. A Figura 4.16 ilustra a Makefile utilizada durante a construção da partição PX. Como se pode observar, a opção –prefix-symbols adiciona o prefixo PX a todos os símbolos do ficheiro objecto "o-optimize/object.obj", produzindo um outro ficheiro objecto "**PX**_PATH/**PX**object.obj".

Como existem funções dentro do PMK que todas partições necessitam de aceder, como por exemplo a gestão de partições ou a comunicação inter-partições, as duas últimas linhas da Makefile da Figura 4.16 retiram o prefixo adicionado na opção anterior, de forma a manter a coerência entre o nome dos símbolos.

Embora não seja necessário, o PMK também pode ser construído partindo do RTEMS e retirando-lhe módulos (grande parte deles). Em particular, os módulos mais importantes são relativos ao ataque

```
$(PGM.c): $(OBJS)
        $(LINK.c) $(AM_FLAGS) $(AM_LDFLAGS) -W1 -r -W1 \
                -o $(basename $@).obj $(LINK_OBJS) $(LINK_LIBS)
        objcopy –prefix-symbols=PX o-optimize/object.obj PX_PATH/PXobject.obj
        objcopy –redefine-sym PX_PMK_f1=PMK_f1 PX_PATH/PXobject.obj
        objcopy –redefine-sym PX_PMK_f2=PMK_f2 PX_PATH/PXobject.obj
```

Figura 4.16: Excerto da Makefile da construção da partição PX na arquitectura MEC

ao hardware (inicialização do CPU, ISR, etc). Desta forma a sua construção é muito semelhante à das restantes partições. Existem, no entanto, alterações: a construção do PMK é conjunta com a imagem final, i.e., são ligadas (*linked*) todas as partições e o PMK de forma a produzir a imagem final. Como o PMK é um componente especial, não requer a passagem pelo filtro (desde que os símbolos não comecem por PX). Assim, a Figura 4.17 ilustra a Makefile utilizada para produzir o PMK juntamente com a ligação com as restantes partições e consequente construção da imagem final.

```
$(PGM.c): $(OBJS)
        $(LINK.c) $(AM_FLAGS) $(AM_LDFLAGS) -W1, -Ttext, $(RELOCADDR) \
                -o $(basename $@).obj \
                P1_PATH/P1_object.obj \
                P2_PATH/P2_object.obj \
                P3_PATH/P3_object.obj \
                $(LINK_OBJS) $(LINK_LIBS)
        $(OBJCOPY) -O elf32-i386 \
                –remove-section=.rodata \
                –remove-section=.comment \
                –remove-section=.note \
                –strip-unneeded $(basename $@).obj $@
```

Figura 4.17: Excerto da Makefile da construção do PMK e linkagem com as restantes partições na arquitectura MEC

4.3 Escalonamento de Aplicações ARINC 653

As características temporais das aplicações ARINC 653 possuem elementos retirados da arquitectura *time-triggered* (escalonamento das partições) e da *event-triggered* (escalonamento das tarefas

dentro de uma partição). Neste cenário, um problema fundamental do escalonamento resume-se à pergunta: dado o escalonamento das partições, será que as tarefas cumprem a meta temporal?

Um artigo recente ((Turja *et al.* , 2005)) responde a uma questão semelhante através da modelação das restantes partições como tarefas de alta prioridade com relações de translação temporal (*offset*) (Mäki-Turja & Nolin, 2004; Mäki-Turja & Nolin, n.d.). Embora não seja directamente aplicável ao caso das partições ARINC 653, este modelo permite criar condições de escalonabilidade para tarefas periódicas síncronas.

Como foi dito anteriormente, os sistemas alvo da especificação ARINC 653 não exigem uma grande rapidez de resposta a eventos externos. Como tal, a especificação de tarefas síncronas, periódicas ou esporádicas, impõe um pessimismo acrescido pois assume que as tarefas são activadas nos piores instantes, do ponto de visto do escalonamento. Como consequência, muitos sistemas são caracterizados como não escalonáveis.

Desta forma, a modolação das restantes partições como tarefas de alta prioridade com relações de offset feita em (Turja *et al.* , 2005) impõe um pessimismo acrescido. É portanto, desejável a modolação de tarefas periódicas assíncronas em vez de síncronas.

O modelo apresentado neste capítulo considera tarefas assíncronas periódicas e tarefas esporádicas. As tarefas síncronas periódicas normalmente consideradas podem ser modeladas como uma tarefa esporádica com período igual ao MIT correspondente.

Partindo dos resultados obtidos no capítulo anterior, é possível modelar o escalonamento das partições através de um número variável de tarefas periódicas assíncronas. Desta forma o escalonamento de tarefas é semelhante ao discutido no capítulo anterior, i.e., a um modelo onde existem apenas tarefas assíncronas periódicas e tarefas esporádicas.

Dado que as partições são escalonadas segundo uma arquitectura TT, é possível analisar cada partição separadamente pois não é permitido que interfiram temporalmente, sendo cada uma completamente definida, do ponto de vista temporal, pelas janelas temporais em que pode executar.

4.3.1 Modelo de Sistema

O sistema é composto por partições que internamente possuem tarefas. As tarefas possuem metas temporais que, caso sejam sempre cumpridas, tornam a tarefa escalonável. Caso contrário, i.e., basta um caso onde não cumpra a meta temporal, a tarefa não é escalonável. As secções seguintes apresentam como as partições são modeladas e dentro delas, quais as características das tarefas.

Modelo de Partições

Cada partição possui um conjunto de janelas temporais onde se executa. Estas janelas repetem-se ao fim de uma MTF (Major Time Frame).

É possível modelar as janelas temporais de uma partição considerando os períodos onde não se executa como tarefas periódicas assíncronas de alta prioridade. Desta forma o conceito de partição é removido e o sistema é apenas composto, do ponto de vista do escalonamento, por tarefas periódicas assíncronas e tarefas esporádicas.

Tome-se o exemplo de escalonamento de uma partição ilustrado na Figura 4.18. Neste exemplo, os intervalos de tempo onde a partição não executa, $\{[40,80[;[130,180[;[240,300[\}$, podem ser modelados como três tarefas periódicas assíncronas com o período igual ao MTF da partição e com tempos de execução correspondentes à dimensão do intervalo respectivo:

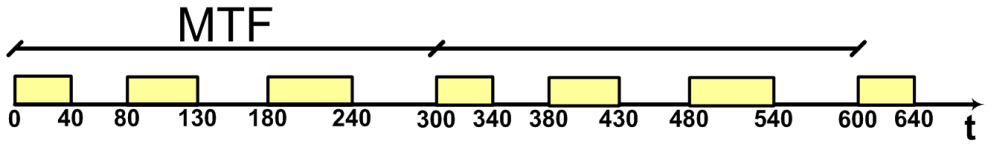


Figura 4.18: Exemplo do escalonamento de uma partição

$$\begin{cases} \Gamma_1 = \{c_1 = 40 \ , \ T_1 = MTF \ , \ r_1 = 40 \ , \ D_1 = c_1\} \\ \Gamma_2 = \{c_2 = 50 \ , \ T_2 = MTF \ , \ r_2 = 130 \ , \ D_2 = c_2\} \\ \Gamma_3 = \{c_3 = 60 \ , \ T_3 = MTF \ , \ r_3 = 240 \ , \ D_3 = c_3\} \end{cases}$$

Desta forma, define-se P como o número de tarefas periódicas assíncronas necessárias para modelar o escalonamento da partição em análise. Estas tarefas possuem a maior prioridade possível, pois

o escalonamento das partições é mais prioritário do que as restantes tarefas.

Modelo de Tarefas

O Modelo de Sistema considerado em cada partição é semelhante ao considerado no capítulo anterior. Assim, cada partição é composta por tarefas assíncronas periódicas e esporádicas, escalonadas por um algoritmo de prioridades fixas preemptivo e com metas temporais gerais (menores ou iguais ao período). Como uma primeira aproximação, é assumido que as tarefas são independentes, co-existem num único processador e o tempo de mudança de contexto é nulo. Cada tarefa possui uma prioridade única, i.e., não existem duas tarefas com a mesma prioridade. No entanto, tarefas periódicas e esporádicas podem entre-cruzar as suas prioridades, i.e., a prioridade de uma dada tarefa esporádica pode estar no meio das prioridades de duas tarefas periódicas e vice-versa.

No que diz respeito ao conjunto de tarefas periódicas, cada tarefa é caracterizada pelos parâmetros $\Gamma_i = \{c_i, T_i, r_i, D_i\}$ onde c_i representa o WCET (Worst Case Execution Time), T_i o período da tarefas, r_i o instante da primeira activação da tarefa e D_i a meta temporal relativa ao instante de activação. A condição de metas temporais estabelece que a meta temporal é menor ou igual ao período, logo os parâmetros estão condicionadas a $0 \leq c_i \leq D_i \leq T_i$ e $0 \leq r_i$. De forma a integrar o escalonamento das tarefas que pertencem à partição com as tarefas que modelam o escalonamento das partições, define-se Γ_{P+1} como a tarefa periódica assíncrona de maior prioridade pertencente à partição, onde P representa o número de tarefas "auxiliares" necessárias para modelar o comportamento da partição em análise.

Cada tarefa esporádica também é caracterizada por $\tau_m = \{e_m, MIT_m, H_m\}$ onde e_m representa o WCET, MIT_m o Minimum Inter-Arrival Time entre duas activações consecutivas e H_m a meta temporal relativa ao instante de activação. Tal como as tarefas periódicas, os parâmetros das tarefas esporádicas estão condicionados a $0 \leq e_m \leq H_m \leq MIT_m$. Tal como para as tarefas periódicas, a tarefa esporádica de maior prioridade possui prioridade inferior a Γ_P.

Definições Adicionais

${}^{l}r_i = r_i + l\,T_i$	–	Instante de activação da execução l da tarefa periódica assíncrona Γ_i
${}^{l}R_i$	–	Instante de resposta da execução l da tarefa assíncrona periódica Γ_i
R_m	–	Instante de resposta da tarefa esporádica τ_m
${}^{l}R_i - {}^{l}r_i$	–	Tempo de resposta da execução l da tarefa assíncrona periódica Γ_i
Λ_i	–	Hiperperiodo da tarefa assíncrona periódica Γ_i

4.3.2 Escalonamento de Tarefas

Devido à modelação do escalonamento das partições como tarefas periódicas assíncronas de alta prioridade, o escalonamento de tarefas num ambiente ARINC 653 é extremamente semelhante ao apresentado no capítulo anterior. Como tal, esta secção resume os resultados encontrados anteriormente e aplica-os ao ambiente em estudo.

4.3.2.1 Escalonamento de Tarefas Assíncronas Periódicas

Seguindo a análise feita no capítulo anterior, tem-se que o tempo de resposta de uma tarefa periódica assíncrona é a solução mais pequena (mas maior que ${}^{l}r_i$) de

$$ {}^{l}R_i = \kappa_i + c_i + w_{i-1}({}^{l}R_i) - w_{i-1}(\kappa_i) + \omega_K({}^{l}R_i - \kappa_i) \tag{4.1} $$

onde

- $i > P$
- K - índice da tarefa esporádica de menor prioridade mas com maior prioridade que Γ_i
- κ_i - candidato ao instante crítico assíncrono das tarefas $\Gamma_1, \ldots, \Gamma_i$
- $w_{i-1}(t)$ - trabalho pedido pelas tarefas periódicas de maior prioridade no intervalo $[0, t[$

- $\omega_K(t)$ - trabalho pedido pelas tarefas esporádicas de maior prioridade no intervalo $[0,t[$ assumindo que foram activadas em $t_0 = 0$

Esta equação pode ser resolvida com base nos métodos descritos no capítulo anterior, nomeadamente as Figuras 3.15 e 3.21.

A análise de escalonabilidade das tarefas $\Gamma_1,\ldots,\Gamma_P$ não é necessária visto que nunca sofrem interferência de nenhuma outra tarefa mais prioritária. A análise de escalonabilidade das tarefas $\Gamma_{P+1},\Gamma_{P+2},...$ pode ser resumida na condição

$$\begin{cases} \forall_{l:{}^l r_i \in \Lambda_i} & {}^l R_i - {}^l r_i \leq D_i & , & \Gamma_i \textbf{ é escalonável} \\ \exists_{l:{}^l r_i \in \Lambda_i} & {}^l R_i - {}^l r_i > D_i & , & \Gamma_i \textbf{ não é escalonável} \end{cases} \tag{4.2}$$

onde ${}^l R_i$ é dado pela equação 4.1.

4.3.2.2 Escalonamento de Tarefas Esporádicas

A integração de tarefas esporádicas também se mantém inalterada, com o senão de que todas as tarefas esporádicas possuem menor prioridade que as tarefas $\Gamma_1,\ldots,\Gamma_P$. Sendo τ_K a tarefa esporádica de menor prioridade mas com prioridade maior que Γ_i (com $i > P$), o instante de resposta de Γ_i é a menor solução (maior que ${}^l r_i$) de

$$R_m = \kappa_i + e_m + w_i(R_m) - w_i(\kappa_i) + \omega_{m-1}(R_m - \kappa_i) \tag{4.3}$$

sendo Γ_i a tarefa periódica de menor prioridade mas com maior prioridade que τ_m e sendo κ_i um candidato a instante crítico assíncrono. A solução desta equação pode ser encontrada com o método de ponto-fixo com $R_m^{(0)} = e_m + \kappa_i$. A condição de escalonabilidade é dada por

$$\begin{cases} \forall_{\kappa_i \in \Lambda_i} & R_m - \kappa_i \leq H_m & , & \tau_m \textbf{ é escalonável} \\ \exists_{\kappa_i \in \Lambda_i} & R_m - \kappa_i > H_m & , & \tau_m \textbf{ não é escalonável} \end{cases} \tag{4.4}$$

4.3.2.3 Exemplo de uma Aplicação

Na Figura 4.19 encontra-se um exemplo do escalonamento de cinco tarefas periódicas assíncronas dentro de uma partição. Como se pode concluir pela figura, todas as tarefas cumprem a meta temporal (igual ao período). Todos estes tempos de resposta podem ser calculados com o método proposto.

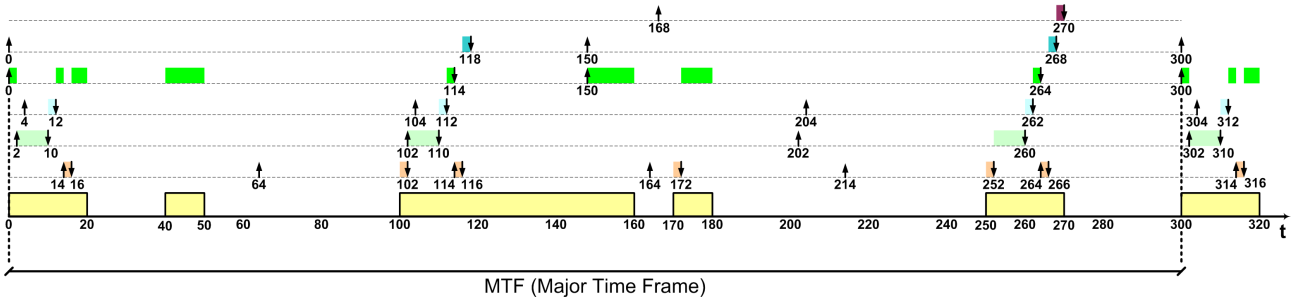


Figura 4.19: Exemplo do escalonamento de tarefas periódicas dentro de uma partição

4.3.3 Optimizações

Dado que as tarefas se encontram separadas temporalmente pelo conceito de partição, a dependência da complexidade do escalonamento com o hiperperiodo imposto pelas tarefas periódicas assíncronas é fortemente levantada. Como cada partição apenas possui parte do conjunto global de tarefas, a sua análise de escalonamento torna-se mais fácil, embora também dependa do período da própria partição (MTF).

De igual forma, sendo as partições modoladas como tarefas periódicas assíncronas, os melhoramentos propostos no capítulo anterior mantêm-se inalterados, isto é, é possível:

- Aplicar o conhecimento do último instante de inactividade (*idle*) para diminuir o tempo de análise
- Lidar com grandes hiperperiodos introduzindo um certo pessimismo
- Distribuir a análise por diversos computadores

4.4 Sumário

Este capítulo apresenta a especificação ARINC 653. Esta especificação permite que múltiplas aplicações (partições) com diferentes graus de criticalidade e funcionalidade co-existam numa única plataforma computacional sem criarem interferências prejudiciais entre si. Por exemplo, o sistema de base de dados não pode prejudicar o sistema de controlo de navegação e vice-versa. Para isso são apresentados os conceitos segregação temporal e espacial:

- segregação temporal - assegurar que as actividades de uma partição não afectam temporalmente as actividades de outra partição
- segregação espacial - impossibilitar que as actividades de uma partição escrevam na zona de memória de outra partição

Para assegurar a segregação temporal é usada a arquitectura *time-triggered* no escalonamento das partições. Dentro de cada partição as tarefas são escalonadas segundo um algoritmo preemptivo de prioridades fixas.

Para assegurar a segregação espacial são usados os mecanismos de hardware para validação e/ou translação de endereços, presentes na MMU do CPU em causa.

Este capítulo discute como a especificação ARINC 653 pode ser implementada usando COTS (Commercial Off-The-Shelf) RTOS (Real Time Operating Systems). Esta solução permite custos de desenvolvimento consideravelmente mais reduzidos uma vez que a sua maioria concentra-se no desenvolvimento dos serviços intra-partição (escalonamento e comunicação/sincronização entre tarefas) já existentes em vários *open-source* RTOS. São apresentadas duas arquitecturas para a implementação da especificação ARINC 653:

- single-executive core (SEC)
- multi-executive core (MEC)

Na arquitectura SEC existe apenas um único RTOS ao qual é acrescentado o conceito de partição. Na arquitectura MEC existe um RTOS por cada partição (Rufino *et al.* , 2007). Após comparar as

duas arquitecturas conclui-se que para os sistemas alvo, a arquitectura MEC é preferível uma vez que aumenta a fiabilidade e tolerância a faltas, juntamente com melhores características temporais (tempos de comutação entre partições, por exemplo) embora necessite de uma maior dimensão.

Discutiu-se um método inovador e estruturado de concretizar a arquitectura MEC que possibilita a integração de núcleos RTOS na arquitectura introduzindo apenas alterações pontuais na estrutura destes núcleos.

Como exemplo de um RTOS em particular, é discutido como o RTEMS (Real-Time Executive Multiprocessor System) pode ser utilizado como ponto de partida para a implementação da especificação ARINC 653. Tomando como base este RTOS, discute-se quais os serviços mínimos que este deve fornecer de modo a estabelecer uma ligação entre o PMK (Partition Management Kernel) e as partições. Em particular, é necessário fornecer primitivas de

- inicialização do RTOS sem começar o multi-tasking
- actualização de um (ou mais) *clock ticks*
- invocar o expedidor de tarefas

Embora possua várias vantagens, a arquitectura MEC obriga ao uso de ferramentas de desenvolvimento com propriedades adicionais. São também apresentados e discutidos os mecanismos adicionais necessários e a sua integração no ambiente de desenvolvimento centrado no RTEMS.

Este capítulo discute ainda métodos para estudar o escalonamento das tarefas dentro de cada partição. Em resumo, é necessário apenas considerar um conjunto de tarefas periódicas assíncronas adicionais para modelar o escalonamento das partições. Desta forma o escalonamento é extremamente semelhante ao discutido no capítulo anterior.

5

Controlo Temporal de Eventos

Existem dois modos no universo dos sistemas de tempo-real para analisar o estado do mundo real e tomar decisões: *time-triggered*; *event-triggered*. A primeira é estritamente determinística, pois analisa o sistema em instantes predefinidos. Desta forma os sensores são periodicamente analisados e, caso seja detectado algum evento, o sistema toma uma decisão. Por outro lado, a arquitectura *event-triggered* analisa o mundo apenas quando um evento é despoletado. Desta forma o sistema torna-se mais eficiente, pois apenas necessita de processamento quando ocorre um evento, e aumenta a rapidez com que reage a um evento, embora torne o sistema temporalmente mais incerto.

Como já foi referido, em sistemas de tempo-real a importância de um resultado correcto é igual à sua concretização num tempo limitado, i.e., não interessa que o sistema de controlo mande o avião subir depois de ele se ter despenhado. Logo, a incerteza temporal associada aos sistemas *event-triggered* tem que ser conhecida e limitada. Em particular, este capítulo dedica-se a analisar situações de sobrecarga de eventos.

Em sistemas TT os eventos são periodicamente analisados através de *polling*, i.e., a aplicação lê o estado do um recurso que pode ter mudado conforme tenha ocorrido um evento. Em sistemas ET os eventos despoletam uma interrupção que possui uma prioridade máxima. Desta forma, as tarefas em execução são atrasadas. Numa tentativa de minimizar este atraso, as rotinas de interrupção (ISR - *Interrupt Service Routine*) efectuam o mínimo processamento possível. Tipicamente, a interrupção apenas activa uma tarefa que processará o evento despoletado e informa o *hardware* que a interrupção foi processada.

Num sistema *time-triggered* a sobrecarga de um tipo de eventos, e.g., pacotes a chegarem da rede de comunicação, apenas afecta temporalmente um componente: a interface de rede. Como esta é

tipicamente analisada periodicamente, uma sobrecarga não interfere com os restantes componentes. Por outro lado, num sistema *event-triggered* uma sobrecarga de eventos produz uma interferência em todos os componentes devido à geração de interrupções que afectam temporalmente todas as tarefas em execução.

A junção destas duas filosofias seria preferível, sendo a arquitectura ET ideal para situações onde os eventos produzem uma interferência denominada “aceitável”, enquanto que para situações de sobrecarga a TT é desejável devido ao seu comportamento temporalmente determinístico.

Como tal, este capítulo destina-se a apresentar um mecanismo que detecta se uma dada fonte de eventos está a sobrecarregar o sistema, devido possivelmente a uma falha, inibe o tratamento desses eventos segundo uma arquitectura ET e activa o processamento TT. Quando for detectado que a situação de sobrecarga transitória se encontra ultrapassada, activa de novo o processamento de eventos segundo a arquitectura ET. Este tipo de arquitectura híbrida age segundo um ciclo de histerese - Figura 5.1.

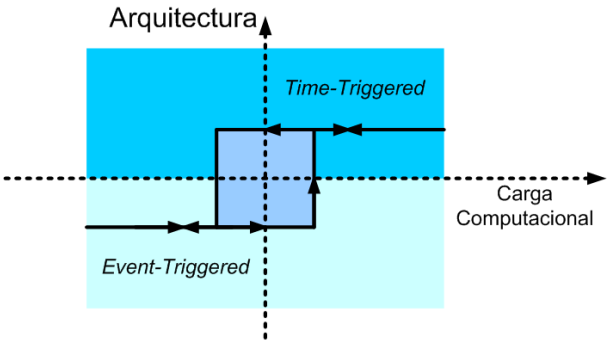


Figura 5.1: Comportamento por Histerese do Sistema: Arquitecturas TT e ET

Devido ao comportamento temporalmente estático da arquitectura TT, é substancialmente mais fácil implementar o mecanismo proposto sobre uma arquitectura ET. De outra forma seria necessário implementar todo o algoritmo de escalonamento num sistema TT, o que na prática se traduziria em construir um próprio sistema ET, o que se deseja evitar. Desta forma o trabalho apresentado parte de um RTOS: RTEMS (Real Time Executive Multiprocessor System), como exemplo prático de implementação. Note-se, no entanto, que o RTEMS é apenas um exemplo em particular, sendo os mecanismos propostos facilmente extensíveis a outros RTOS.

5.1 Arquitectura Event-Triggered

Segundo uma arquitectura ET, a notificação ao sistema de que ocorreu um evento é feita tipicamente através de interrupções. Estas interrupções são imediatamente processadas pelo sistema computacional, atrasando o processamento em curso até que a interrupção seja tratada. Por causa da interferência temporal causada nas actividades que estão a decorrer no processador, as interrupções têm requisitos a nível temporal exigentes e não devem ocupar muito tempo do processador. Tipicamente, uma interrupção apenas notifica o hardware de que recebeu o evento e activa uma tarefa com a informação de que ocorreu um evento de um determinado tipo.

Para permitir um maior controlo temporal sobre o processamento de eventos críticos (e.g. botão de emergência) é possível estabelecer um grau hierárquico, reflectido de certa maneira nos sistemas operativos de tempo-real na prioridade associada a cada tarefa. Usando as condições de escalonamento conhecidas, e.g., Rate Monotonic Scheduler (RMS), as prioridades são função de parâmetros do sistema como por exemplo, o período das tarefas. No entanto, a prioridade das tarefas não fornece, por si só, a garantia que os eventos mais críticos são processados sem interferência temporal dos restantes. Por exemplo, dado que o evento que surge de um botão de emergência tem tipicamente um período muito baixo, segundo o escalonamento RMS a tarefa que o trata tem também uma baixa prioridade. Mesmo que a prioridade reflicta o grau de criticalidade da tarefa, dado que outros eventos podem interferir temporalmente com a tarefa em execução, por forma de interrupções, existe sempre um grau de interferência temporal.

Se o ritmo a que as interrupções são despoletadas for maior do que o sistema computacional é capaz de suportar, o sistema entra em sobrecarga, ocupando grande parte do tempo durante o processamento de interrupções em vez de acabar as restantes tarefas do sistema. Mesmo que o ritmo das interrupções seja tal que não ocupe uma grande percentagem de tempo do CPU, o processamento de que daí advém, por parte das tarefas, pode ser tal que o sistema não consegue responder aos eventos a tempo. Como exemplo refira-se um sistema de aterragem lunar (AGC - *Apollo Guidance Computer*) que ficou comprometido devido à geração descontrolada de interrupções provenientes de um dispositivo com uma ligação errónea. Felizmente o sistema estava sobre-dimensionado para poder tratar situações de falha única, pelo que a aterragem decorreu sem mais incidentes.

Em muitos sistemas o próprio hardware limita o ritmo das interrupções, Tabela 5.1. Para que

se possam obter garantias de que o sistema suporta este ritmo de interrupções conjuntamente com as tarefas, pressupõem-se várias condições do sistema. A mais usual é a integração da pior interferência possível das interrupções (maior ritmo possível) nas condições de escalonamento. Para isso determina-se, através da especificação do *hardware*,um tempo mínimo entre a chegada de duas interrupções consecutivas (MIT - Minimum Inter-arrival Time - corresponde ao inverso da frequência da Tabela 5.1) e o tempo máximo dispendido na ISR. Assim, é possível interpretar um fonte de interrupção como uma tarefa de alta prioridade no esquema de escalonamento. Como por exemplo, a integração das interrupções no cálculo do tempo de resposta das tarefas é dada por (Regehr & Duongsaa, 2005):

$$R_i = c_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil c_j + \sum_{k \in Interrupcoes} \left\lceil \frac{R_i}{MIT_k} \right\rceil e_k \tag{5.1}$$

Neste caso é somado a interferência das interrupções assumindo que são lançadas no "instante crítico" e com a maior frequência possível, sendo e_k o pior tempo de execução da ISR. A Figura 5.2 demonstra o *workload* pedido pelas interrupções no pior cenário (assumindo que chegam periodicamente)

Dispositivo	**Frequência máxima de interrupções (Hz)**
Teclado	33
Fio solto	500
Bounce de um interruptor	1300
Porta série a 115 Kbps	11500
Ethernet a 10 Mbps	14880
CAN bus a 1 Mbps	15000
I2C bus	50000
USB	90000
Ethernet a 100 Mbps	148800
Ethernet a 1 Gbps	1488000

Tabela 5.1: Ritmos de interrupções máximos de alguns dispositivos (extraído parcialmente de (Regehr & Duongsaa, 2005))

Esta abordagem possui, no entanto, várias desvantagens:

- Cálculo da interferência temporal das interrupções pode ser demasiado pessimista

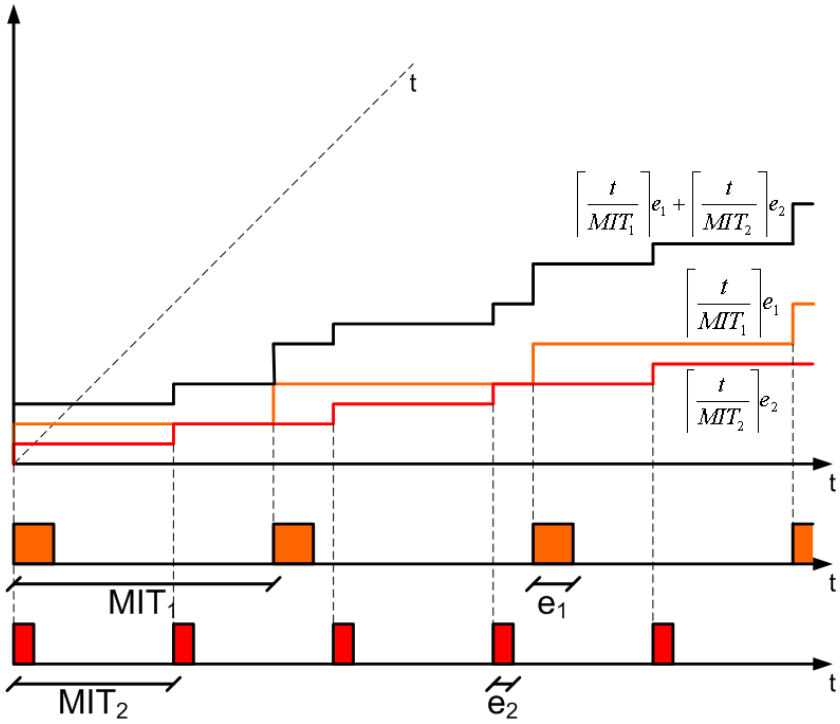


Figura 5.2: *Workload* devido às interrupções periódicas

- Em situações de falha (geração descontrolada de interrupções) é impossível estabelecer garantias

Pela Tabela 5.1 verifica-se que existem dispositivos que podem gerar um grande número de interrupções mas que são de uma natureza *bursty*, isto é, podem surgir num intervalo de tempo muitos eventos mas a maior parte do tempo o dispositivo está parado. Desta forma o pressuposto de que as interrupções surgem no pior caso com um intervalo definido pelo MIT durante toda a vida do sistema é demasiado pessimista, levando à implementação desnecessária de sistemas mais potentes. Indo mais além, em situações de falha do dispositivo, como por exemplo, um fio solto, a geração descontrolada de interrupções não está prevista no modelo pelo que pode comprometer o sistema.

5.2 Arquitectura Time-Triggered

Na arquitectura TT todos as tarefas são activadas em instantes precisos e conhecidos no tempo. Como tal, não existe a interferência de interrupções pois estas não são admitidas no sistema. Em

vez disso, a interacção com o exterior é feita através de *polling*. Assim, se houver uma falha de um dispositivo levando à geração de um grande número de eventos, estes são apenas conhecidos durante a janela temporal da tarefa que interage com esse dispositivo. Deste modo não é criada interferência temporal com os restantes componentes do sistema.

A arquitectura TT é desta forma, preferível para tratar situações de sobrecarga de eventos pois assegura que a falha de um componente não é propagada temporalmente para o restante sistema.

5.3 Integração das Arquitecturas Time-Triggered e Event-Triggered

Dadas as vantagens/desvantagens das arquitecturas TT e ET torna-se evidente que devem ser utilizadas dependendo do número de eventos processados, Figura 5.1. Quando o sistema se encontra em sobrecarga de eventos, deve adoptar um controlo mais rígido usando a arquitectura TT. Dado que existe um grande número de eventos para processar, este método não gasta desnecessariamente os recursos do CPU. Quando o número de eventos despoletados é relativamente baixo, o sistema pode adoptar a arquitectura ET uma vez que a interferência que produz nas restantes tarefas também é baixa.

Como o comportamento das interrupções na arquitectura ET é tal que interfere temporalmente com qualquer tarefa do sistema, quando se comuta para o modo TT é natural que a tarefa que leia o sensor também seja de alta prioridade. Desta forma esta tarefa não possui interferência das restantes tarefas (ou uma interferência muito limitada) e é executada em condições bem definidas. Desta forma, o sistema comuta entre a admissão de interrupções (modo ET) ou a tarefa de *polling* (modo TT).

O elemento caracterizador da carga provocada pelo processamento dos eventos é responsável pela comutação entre os dois modos de funcionamento.

5.4 Caracterização da Carga Computacional

O conceito de carga computacional pode ser definido de várias maneiras. O sistema pode considerar a percentagem de tempo de *idle* do CPU, o ritmo das interrupções, o tempo entre interrupções consecutivas (de uma mesma fonte), o tempo de processamento de tarefas a partir de uma determinada prioridade, etc, ou inclusive uma combinação de condições. Na Figura 5.3 encontram-se ilustrados vários conceitos que podem definir a carga computacional.

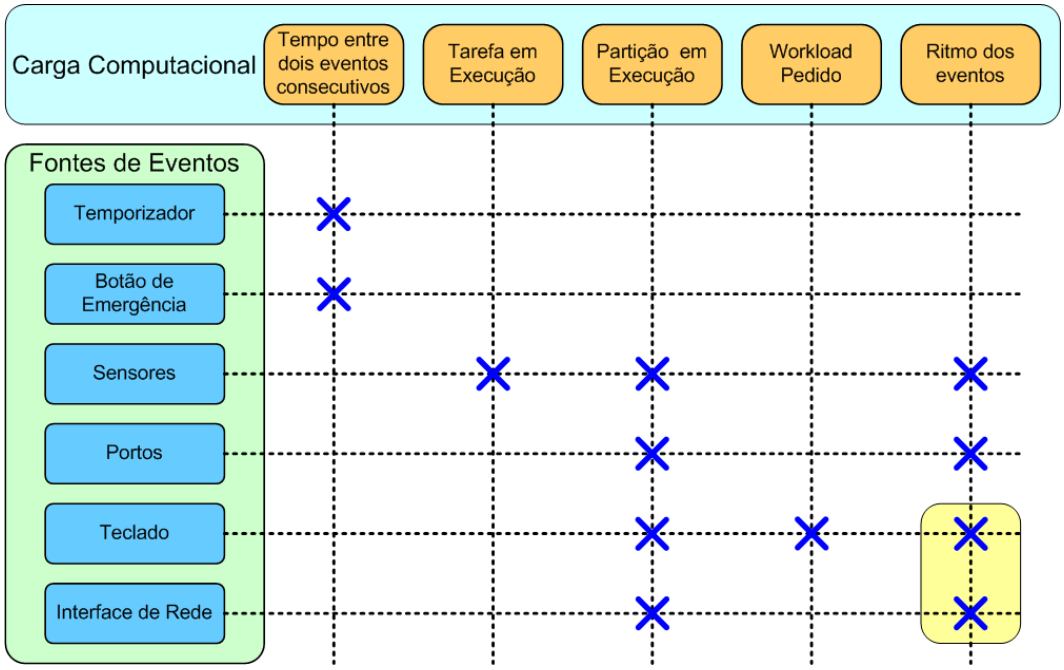


Figura 5.3: Carga Computacional

No exemplo da Figura 5.3 o temporizador e botão de emergência são caracterizados pelo tempo mínimo entre duas interrupções consecutivas: se for demasiado pequeno então uma situação de sobrecarga/falha é detectada. No caso do temporizador, se não gerar interrupções com uma precisão dentro da especificada, o sistema pode assumir que o temporizador encontra-se defeituoso (assume-se que o relógio do sistema tem uma probabilidade de falha muito inferior ao temporizador) e passar para outro esquema de processamento. Por outro lado, o botão de emergência não deve ser pressionado sem que um intervalo mínimo de tempo passe, caso contrário, o sistema encontra-se sempre a reinicializar e não tem tempo para informar os outros elementos da rede do evento (se for possível fazê-lo).

Os eventos de pressão encontram-se afectados por três elementos caracterizadores do sistema:

tarefa em execução; partição em execução e ritmo dos eventos. Por exemplo, pode-se definir que o evento é apenas admitido dentro da janela temporal de uma partição mas não dentro de uma tarefa dessa partição. É ainda definido um ritmo que não pode ser ultrapassado caso contrário o sistema assume que estão a ser gerados demasiados eventos e que se encontra numa situação de sobrecarga. O sensor de temperatura é semelhante ao de pressão mas não possui nenhuma restrição em relação à tarefa em execução.

Os eventos provenientes do teclado são caracterizados pela partição em execução, pelo workload pedido e pelo ritmo conjunto das teclas premidas e dos eventos provenientes da interface de rede. O workload pedido prende-se com o número de tarefas em execução. É possível associar a cada tarefa uma tempo máximo de execução e calcular dinamicamente quanto tempo falta executar. Logo, é possível determinar quanto "trabalho" ou workload está a ser pedido ao CPU. Se este workload for demasiado grande, o sistema pode optar por inibir esta interrupção pois é de baixa criticalidade. Se os eventos conjuntos das teclas e da rede ultrapassar um patamar predefinido, o sistema assume que estes dois componentes em conjunto estão a gerar mais eventos do que o sistema consegue suportar.

5.5 Filtros Discretos

Como referido anteriormente, é desejável a existência de um mecanismo que permita limitar o número de interrupções processadas embora não se baseie directamente no intervalo mínimo entre duas interrupções consecutivas. Existem muitos dispositivos que geram eventos de uma forma *bursty*, isto é, são geradas muitas interrupções numa pequena janela de tempo mas na maior parte do tempo o dispositivo não detecta eventos exteriores. É necessário portanto uma métrica diferente do intervalo mínimo para poder caracterizar este tipo de fontes (Coutinho *et al.* , 2005).

Este trabalho toma o ritmo a que os eventos são despoletados como métrica fundamental para caracterizar fontes de eventos tipicamente *bursty*. Se o ritmo ultrapassar um limiar pré-definido, as interrupções são inibidas e o sistema passa para um modo próximo do *time-triggered*. Da mesma maneira, estando no modo TT, se o ritmo dos eventos for menor do que outro limiar também pré-definido, o sistema volta ao modo *event-triggered*.

Para detectar o ritmo a que os eventos chegam é utilizado um filtro passa-baixo discreto. Estes fil-

tros são utilizados nos sistemas computacionais tipicamente para tarefas de controlo e processamento de dados, mas também podem ser utilizados para outros fins. O primeiro passo na construção deste tipo de filtros é estabelecer um período de amostragem T_s. Este intervalo de tempo caracteriza os instantes onde é realizado o *sample & hold* da entrada do filtro. A entrada do filtro é simbolicamente representada pelo sinal $x[n]$ onde n é a representação discreta do tempo $n = \lceil t/T_s \rceil$. Tem-se $x[n] = 1$ nos instantes onde foi detectado o despoletamento de um evento (qualquer) e $x[n] = 0$ caso contrário.

Tome-se a função z[n] como a soma de todos os eventos ocorridos desde o início de vida do sistema ($n = 0$). O sinal z[n] é dado por

$$z[n] = z[n-1] + x[n] \tag{5.2}$$

Este sistema é conhecido como um integrador discreto. Soma simplesmente todos os eventos ocorridos até $n-1$ ($z[n-1]$) com a ocorrência ou não do evento no tempo n ($x[n]$). Pondo esta equação numa forma não recursiva, tem-se

$$z[n] = \sum_{k=1}^{n} x[k] \tag{5.3}$$

assumindo que $x[n] = 0$ para qualquer $n < 1$. Nestas condições, o ritmo das interrupções pode assim ser descrito como uma outra função discreta $y[n]$

$$y[n] = \frac{z[n]}{n} \tag{5.4}$$

que providencia o número médio de eventos no tempo n. Para uma forma recursiva de y[n] pode-se encontrar, a partir das equações 5.2 e 5.4, a expressão

$$y[n] = \frac{z[n]}{n} = \frac{z[n-1] + x[n]}{n} = \frac{(n-1)y[n-1] + x[n]}{n} = \left(1 - \frac{1}{n}\right) y[n-1] + \frac{1}{n} x[n] \tag{5.5}$$

Esta solução para o cálculo de $y[n]$ é impraticável e sem utilidade pois, se o sistema estiver em funcionamento durante vários anos e n for muito grande, então para que $y[n]$ cresça até a um patamar razoável é necessário um enorme número de eventos. Toma assim, em igual importância todos os

eventos que ocorreram, enquanto que os mais recentes têm maior relevância. Deseja-se que $y[n]$ apresente não o número médio de eventos desde a origem dos tempos, mas tanto quanto possível, o ritmo instantâneo da sua geração. A solução óbvia seria considerar apenas os eventos ocorridos dentro de uma janela temporal (D) (Coutinho *et al.* , 2005). Este sistema é conhecido na literatura como um filtro FIR (Finite Impulse Response) onde $z[n]$ passa a ser dado por

$$z[n] = \sum_{k=n-D+1}^{n} x[k] \tag{5.6}$$

O parâmetro D é conhecido mais geralmente como a dimensão do filtro, pois necessita de guardar informação sobre as últimas D amostras de $x[n]$. O seu nome provém de que uma entrada $x[n]$, num dado instante n, apenas afecta a saída do sistema até $n+D$ amostras, sendo que a partir deste tempo, a saída é inalterada, quer o evento tenha ocorrido ou não. O ritmo das interrupções é dado pela média das últimas D amostras

$$y[n] = \frac{z[n]}{D} = \frac{1}{D} \sum_{k=n-D+1}^{n} x[k] \tag{5.7}$$

Repara-se que este sistema é equivalente ao primeiro quando se faz $D = n$.

Outra solução consiste em pesar as amostras com base no tempo da sua ocorrência. Um filtro IIR (Infinite Impulse Response) de primeira ordem realiza a pesagem de cada evento baseada numa exponencial decrescente a partir da sua ocorrência (Coutinho *et al.* , 2005). O ritmo é dado por

$$y[n] = \alpha y[n-1] + (1-\alpha)x[n] \tag{5.8}$$

onde o parâmetro α se encontra no intervalo $]0,1[$ para que o sistema seja estável e $y[n]$ positivo. De novo, este filtro torna-se equivalente ao primeiro fazendo α variante com o tempo ($\alpha = 1/n$) por comparação com a equação 5.5. Dado que $\alpha \neq 0$, a ponderação de um evento é realizada através de uma exponencial decrescente (de base $1-\alpha$) e portanto terá sempre uma componente decrescente mas sempre diferente de zero, sendo daqui derivado o seu nome (IIR).

5.5.1 Implementação dos Filtros Discretos

Os filtros discretos possuem uma desvantagem quando são portados para a arquitectura *event-triggered*: não são actualizados periodicamente. Como se pode constatar da equação 5.8, a actualização do filtro é feita em todos os intervalos de amostragem, o que levaria a um processamento excessivo (em princípio, todo o processamento do sistema estaria ocupado com a actualização do filtro pois T_s é tipicamente pequeno). Como tal, os filtros discretos apenas podem ser actualizados durante o processamento de um evento (designado processamento assíncrono), quer pela arquitectura ET (dentro de uma ISR) ou pela TT (dentro da tarefa de *polling*).

A partir da equação do filtro IIR 5.8 é possível concluir que para dois instantes de amostragem, n e n', em que tenham sido processados dois eventos consecutivos, com $n > n'$, tem-se $y[n]$ dado por

$$\begin{aligned} y[n] &= \alpha y[n-1] + (1-\alpha)x[n] = \\ &= \alpha^2 y[n-2] + (1-\alpha)x[n] = \ldots = \\ &= \alpha^{n-n'} y[n'] + (1-\alpha)x[n] \end{aligned} \tag{5.9}$$

É portanto possível calcular a saída $y[n]$ do filtro IIR tendo como base a manutenção do valor de $y[n']$ e n', ou seja, o valor do filtro no instante do último evento processado e esse próprio instante. O pseudo-código da Figura 5.4 ilustra a actualização do filtro quando é processado um evento.

Para o filtro FIR descrito, a actualização assíncrona é mais complexa e requer mais cálculos. Quando um evento é processado, é guardado o instante de amostragem da sua ocorrência numa tabela contendo as interrupções mais recentes - Figura 5.5.

O filtro decide quais os eventos que ainda estão na sua janela temporal (D). Caso a estrutura de dados fosse uma lista simplesmente ligada, o algoritmo de pesquisa teria que percorrer todas as posições desde a interrupção mais recente até à mais antiga, o que tomaria uma complexidade $O(D)$. Tomando a estrutura de dados uma forma de Vector em Anel (*Ring Buffer*), a pesquisa pode aceder instantaneamente a qualquer posição da tabela, tornando possível dividir sequencialmente em dois grupos as interrupções para tomar uma complexidade $O(log(D))$. Quando encontrar a última

```
newISR(){
        // Determinação do ritmo dos eventos
    t = getTime();
    n = sampleHold(t);
    y = α^(n−last_n) y + (1-α);
    last_n = n;
        // Detecção de sobrecarga e mudança de modo
    if( y > M ){
        state = overload;
        disableInterrupt();
        switchPollMode();
    }
        // Processamento original do evento
    originalISR();
}
```

Figura 5.4: Pseudo-código da actualização do filtro IIR e tomada de decisão sobre a arquitectura

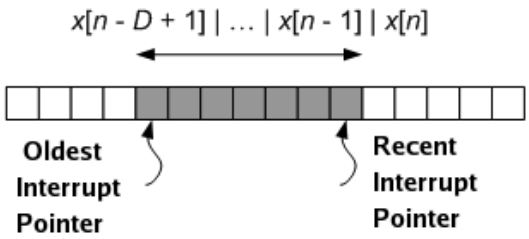


Figura 5.5: Vector em Anel contendo a memória do filtro FIR

interrupção válida, avança o ponteiro. O somatório da equação 5.7 pode ser calculado pela diferença entre os dois ponteiros da Figura 5.5 pois representam o número de interrupções que ocorreram dentro da janela temporal do filtro. Caso as entradas tivessem pesos diferentes consoante a sua posição, a determinação do somatório teria complexidade $O(D)$, o que pode ser incomportável para a maior parte dos sistemas. Comparativamente com o filtro IIR, com complexidade $O(1)$, necessita de mais tempo para calcular a saída mas é ainda comportável. Veja-se o caso em que $D = 1024$, apenas são necessárias 10 operações no pior caso para determinar quais as interrupções dentro da janela temporal.

A implementação destes filtros discretos dentro da tarefa de *polling* é relativamente imediata, mas dentro de uma interrupção é mais complexa pois os RTOS não permitem normalmente operações que usem a FPU (*Floating Point Unit*) dentro de uma ISR. Isto deve-se ao peso computacional de salvaguardar/restaurar o contexto dos registos da FPU para apenas uma interrupção. Note-se que o contexto da FPU é de aproximadamente 100 registos na arquitectura IA32. Para contornar este

obstáculo, é acrescentado um factor de escala para que operações com inteiros sejam permitidas e com pouca perca de resolução. No caso do filtro IIR em particular, nota-se através do pseudo-código da Figura 5.4 que é necessário o cálculo de uma exponencial α^k. Dado que a FPU não é utilizada, este cálculo necessita de recorrer a uma tabela construída durante a inicialização do sistema, de forma a diminuir o tempo ocupado dentro da ISR. Esta tabela contém, para o índice j, o valor α^j. Como $\alpha < 1$, tem-se, a partir de um certo índice, $\alpha^j \simeq 0$. Assim, a dimensão da tabela fica limitada.

5.6 Resultados Experimentais

Esta secção apresenta os resultados referentes à implementação do método que elimina a sobrecarga das interrupções através da determinação do seu ritmo com o filtro IIR de primeira ordem e realizando a técnica de polling para o tratamento de eventos de uma forma controlada (Kopetz, 2002). O teclado foi o dispositivo testado, devido ao trabalho prévio com o VITRAL (desenvolvimento do sistema de janelas VITRAL para o Sistema Operativo de Tempo-Real RTEMS) (Coutinho *et al.* , 2006a; Coutinho *et al.* , 2006b). Este dispositivo alcança as 33 interrupções por segundo (Coutinho *et al.* , 2005). Embora seja um valor extremamente baixo quando comparado com, por exemplo, interfaces de rede, constitui um exemplo representativo de dispositivos que interagem com o sistema através de interrupções.

O exemplo apresentado demonstra a evolução do sistema com e sem os mecanismos de protecção, na presença de um operador do teclado "rápido" e mais tarde forçando uma "tecla presa" (stuck key).

A implementação do filtro IIR contêm os parâmetros $\alpha = 0.999$; $T_{amostragem} = 1ms$; $M = 0,020$; $m = 0.002$; $T_{polling} = 300ms$. As Figuras 5.6 e 5.7 apresenta a mesma situação, com e sem os mecanismos de protecção. Nas primeiras 4000 amostras, um operador "rápido" pressiona as teclas normalmente e $y[n]$ tende a estabilizar para um valor relativamente baixo (abaixo do limiar M). Entre as amostras 7000 e 11000, uma "tecla presa" (stuck key) é experimentalmente simulada, resultando na subida de $y[n]$ até $1/33 = 0,0303$. Verifica-se que o operador não é rápido suficiente para sobrecarregar o sistema enquanto que, com $M = 0.02$, o ritmo da stuck key é mais do que suficiente.

Em aproximadamente $n = 4500$ e $n = 11000$, $y[n]$ decresce rapidamente devido à dimensão da tabela que calcula a função ser excedida (dimensão de 200 períodos de amostragem). Nestes casos,

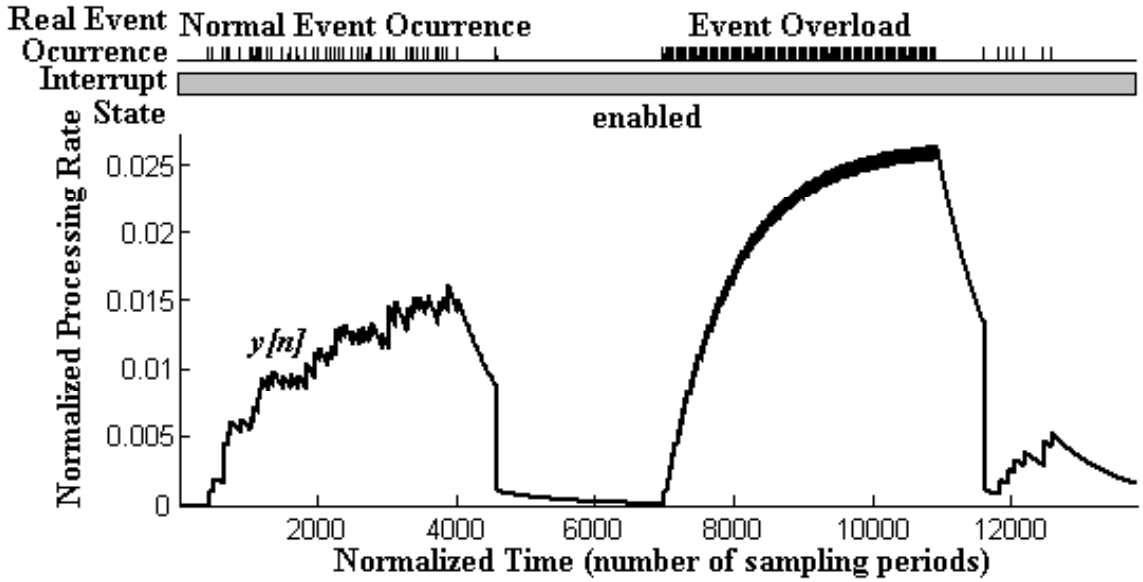


Figura 5.6: Evolução de y[n] no filtro IIR sem os mecanismos de protecção

na próxima interrupção $y[n]$ é colocada a $1-\alpha$. Para além de reduzir o tamanho da tabela, é também um mecanismo simples para decrescer rapidamente $y[n]$ quando um operador deixar de pressionar as teclas. A dimensão da tabela deve também ser tal que não comprometa o pior cenário do ritmo das interrupções admitido.

Com o mecanismo de protecção activado, quando $y[n]$ ultrapassa M, o sistema detecta uma sobrecarga, inibindo a interrupção e activando a tarefa de polling. O tratamento de eventos durante a sobrecarga por polling tende a estabilizar $y[n]$ em $1/300 = 0,0033$. Como m é menor que este valor, o sistema não volta ao normal enquanto tantos eventos forem despoletados. Logo que a sobrecarga é ultrapassada, $y[n]$ decresce para um valor abaixo de m e o sistema volta ao normal, activando de novas as interrupções.

Resultados semelhantes podem ser obtidos com o filtro FIR, com o parâmetro $D = 1024$ - Figuras 5.8 e 5.9. Possui o mesmo comportamento que o IIR mas estabiliza mais rapidamente, especialmente quando as interrupções estão inibidas, devido à forma linear como varia, em vez da curva exponencial decrescente do IIR, que varia muito lentamente quando próxima de zero.

Também foi analisado o comportamento do filtro num dispositivo mais "rápido" como a interface de rede. Como se pode observar pela Figura 5.10, usando o mecanismo de protecção o número de

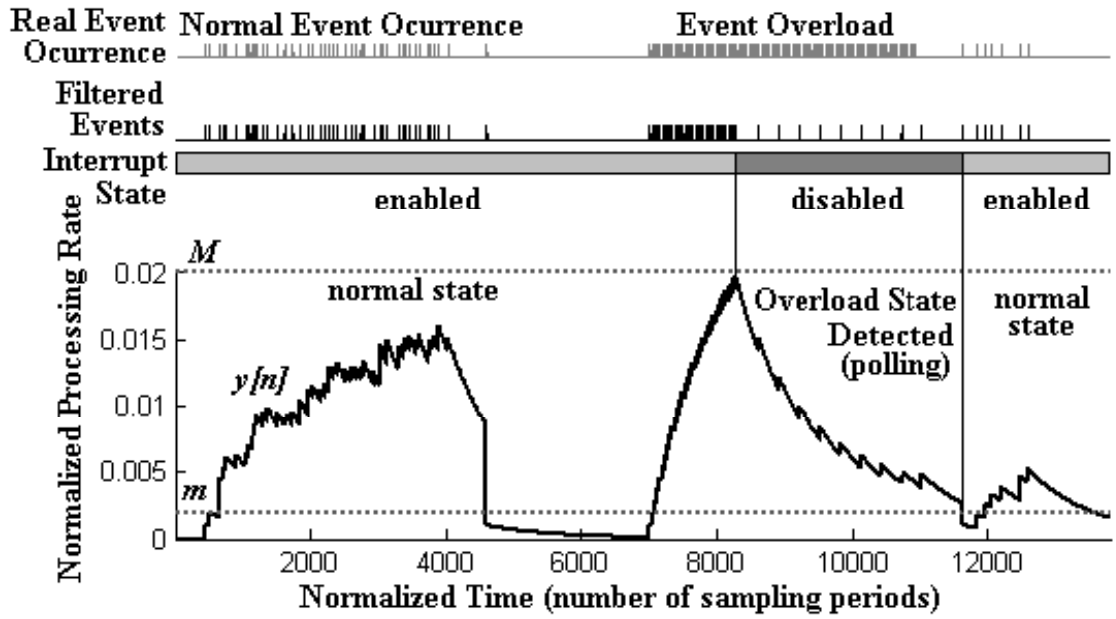


Figura 5.7: Evolução de y[n] no filtro IIR com os mecanismos de protecção

interupções processadas pelo sistema é limitado. Neste cenário em vez de uma tarefa de polling adicional é permitida a admissão de interrupções dentro de um certo intervalo de tempo. Esta solução não requer a utilização de uma tarefa especializada de leitura da interface de rede, pelo que encorre em menores custos de engenharia.

Neste cenário é possível utilizar as capacidades do hardware do dispositivo, nomeadamente do buffer, para ler de seguida um conjunto relativamente grande de eventos. Desta forma os pacotes são processados em conjunto pelo que o tempo requirido é menor.

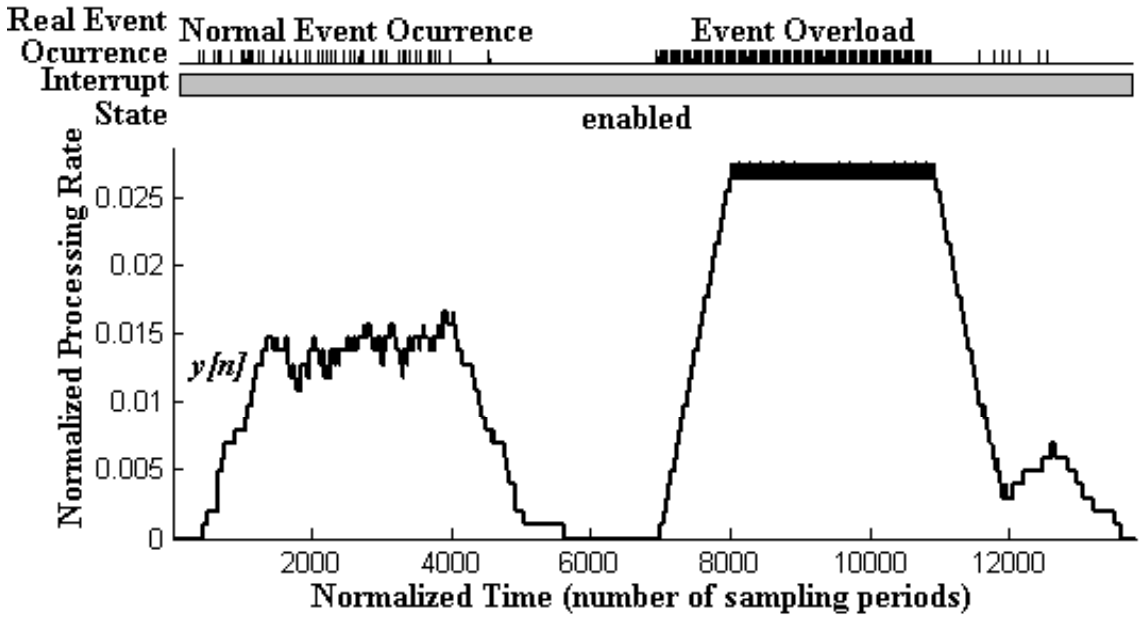


Figura 5.8: Evolução de y[n] no filtro FIR sem e com os mecanismos de protecção - a) e b)

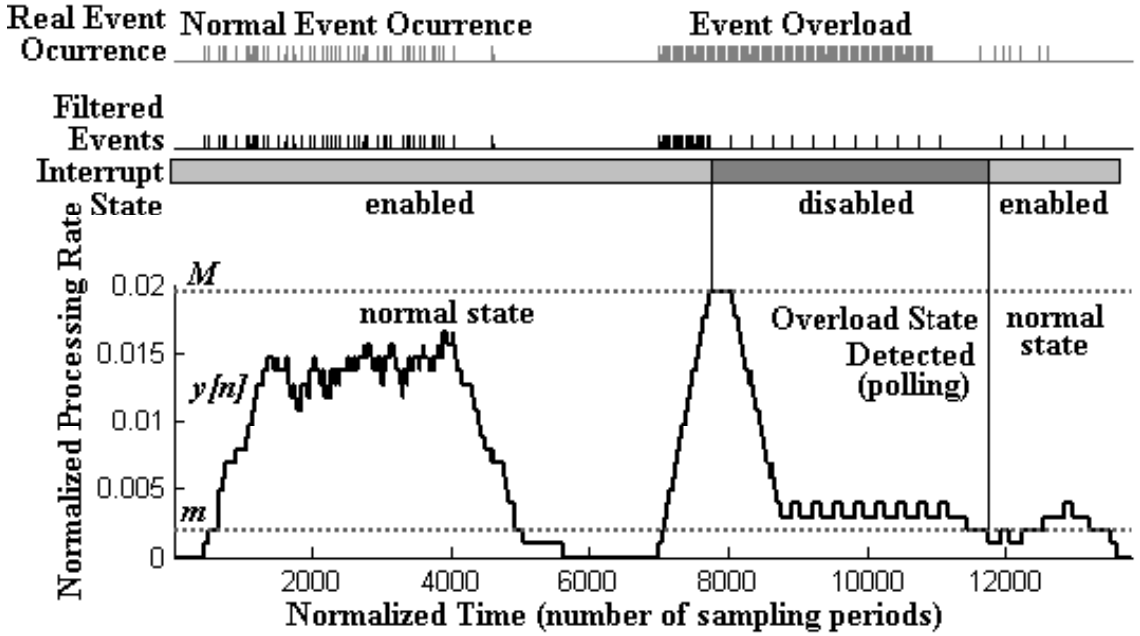


Figura 5.9: Evolução de y[n] no filtro FIR sem e com os mecanismos de protecção - a) e b)

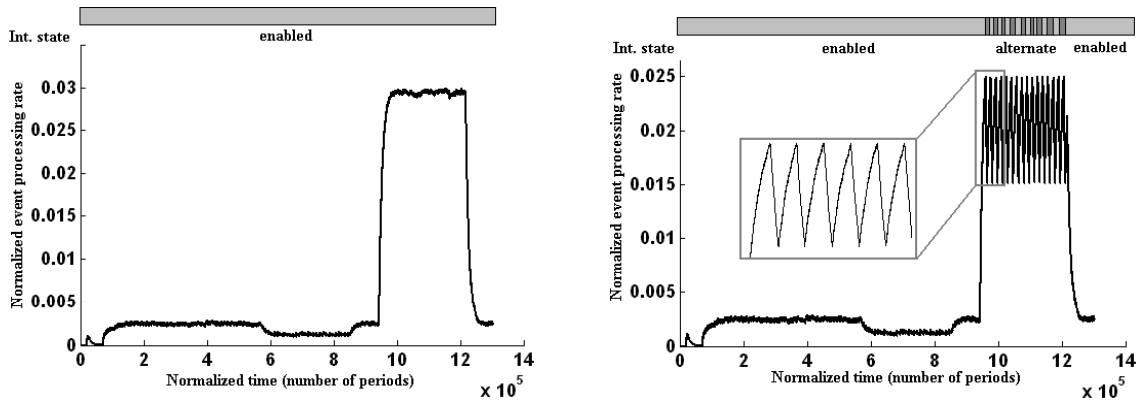


Figura 5.10: Evolução de $y[n]$ num sistema ethernet (a) sem protecções ; (b) com protecções

5.7 Sumário

Neste capítulo são discutidos os problemas temporais associados aos sistemas *event-triggered*. Como os eventos externos tipicamente despoletam interrupções que possuem prioridade superior a qualquer tarefa do sistema, a sua geração descontrolada pode causar o não cumprimento das metas temporais das tarefas. Na arquitectura *time-triggered* este problema não se coloca uma vez que os sensores são lidos periodicamente e em intervalos de tempo bem definidos.

Para controlar a geração dos eventos são apresentadas várias métricas que estabelecem um limiar a partir do qual é decido que o sistema se encontra em sobrecarga. Como exemplo, tem-se o intervalo mínimo entre duas interrupções (do mesmo dispositivo) consecutivas. Esta métrica pode ser utilizada para controlar os timers. Outra métrica determina o ritmo instantâneo da geração de interrupções, que pode ser aplicada em fontes com geração de eventos em rajada, tais como o teclado (*stuck key*) ou a rede de comunicação.

Quando o sistema se encontra em sobrecarga é utilizado um método semelhante ao *time-triggered* onde os sensores são analisados periodicamente através de uma tarefa especializada. Esta tarefa possui a prioridade mais alta uma vez que simula o comportamento das interrupções de uma forma mais controlada. Quando é determinado que a sobrecarga já passou, o sistema volta a processar os eventos segundo a arquitectura *event-triggered*.

São apresentados métodos para calcular o ritmo instantâneo da geração de interrupções com base em modelos matemáticos retirados da teoria do processamento digital de sinal. A sua implementação e os problemas de engenharia que lhe estão associados (por exemplo, operações de floating point dentro de uma ISR) são também discutidos.

Por último, é discutido como fazer a análise de escalonamento utilizando as métricas utilizadas. É analisado qual o pior caso e qual a interferência temporal que as interrupções causam nesse cenário.

6

Conclusões & Perspectivas Futuras

O presente trabalho tratou de uma forma abrangente o problema de proporcionar um ambiente de desenvolvimento de aplicações distribuídas de controlo em tempo-real executáveis em plataformas embebidas. A intervenção efectuada abordou as seguintes contribuições:

- Estudo do escalonamento de tarefas periódicas assíncronas juntamente com tarefas esporádicas utilizando escalonadores preemptivos de prioridades fixas.
- Definição de uma arquitectura que comporte os requisitos da especificação ARINC 653. Elaboração de um demonstrador que ilustra o conceito fundamental do funcionamento da segregação temporal imposto pela especificação ARINC 653 utilizando a arquitectura proposta.
- Controlo da pontualidade do sistema/aplicações em condições de sobrecarga de eventos.

No primeiro ponto, é estudado um método que determina o pior tempo de resposta de tarefas periódicas assíncronas e tarefas esporádicas. O estado do conhecimento concentrava-se apenas em tarefas síncronas (periódicas e esporádicas). Este novo estudo permite que mais sistemas sejam denominados escalonáveis mediante a atribuição apropriada dos parâmetros adicionais (instantes de activação).

No segundo ponto, é proposta uma arquitectura que permite satisfazer os requisitos da especificação ARINC 653, nomeadamente, nos aspectos de segregação temporal e espacial. A arquitectura MEC (Multi-Executive Core) estabelece que dentro de cada partição existe um núcleo RTOS que proporciona todos os serviços dentro da mesma partição. Um conjunto de serviços adicionais permite que haja uma interface entre com o PMK (Partition Management Kernel) e comunicação

com as restantes partições. Embora a dimensão do executável final seja sub-óptima, não constitui um obstáculo relevante nas aplicações alvo, enquanto que proporciona um acréscimo em termos de modularidade, configurabilidade, e confiabilidade. Foram também implementados dois demonstradores que ilustram os conceitos fundamentais do funcionamento da segregação temporal utilizando as arquitecturas MEC e SEC (Single-Executive Core). Foi ainda estudado o escalonamento das tarefas de uma aplicação ARINC 653 com base no estudo realizado anteriormente.

Por último, introduziram-se mecanismos que permitem garantir que as metas temporais são cumpridas, mesmo na presença de sobrecargas de eventos. Apenas um método foi construído, mas verificou-se que implementa o controlo necessário para limitar a interferência do tratamento de eventos com as tarefas. Realizou-se ainda uma análise matemática que estabelece garantias necessárias para verificação a priori do cumprimento (ou não) das metas temporais definidas.

Perspectivas Futuras

Em cada área abordada nesta dissertação existem novos estudos que podem ser realizados e com grande importância para os sistemas de controlo distríbuido.

O escalonamento pode ser estendido para sistemas não-preemptivos de prioridades fixas de modo a simular o comportamento de redes de comunicação do tipo *field-buses*. Como as tarefas periódicas assíncronas possuem instantes de activação bem definidos, pode também ser extremamente útil a modelação do jitter presente nestas redes. Novos melhoramentos podem ainda ser implementados de modo a aumentar a velocidade dos cálculos.

Embora o aspecto mais inovador da arquitectura proposta para a implementação da especificação ARINC 653 já tenha sido apresentado, existem novos aspectos, primordialmente em relação à segregação espacial e ao porte para diferentes arquitecturas computacionais, que ainda não foram completamente analisados e que requerem um estudo mais aprofundado.

A análise de escalonamento mostrada no controlo de interrupções pode ainda ser mais aprofundada, sendo integrada com o escalonamento de tarefas periódicas assíncronas e tarefas esporádicas, ou com escalonadores não-preemptivos e de prioridades dinâmicas. É ainda possível o estabelecimento de condições com o período de sampling definido pelo utilizador, i.e., não é estabelecido pelos

parâmetros do sistema.

Disseminação de Resultados

- *Response Time Analysis of Asynchronous Periodic and Sporadic Tasks Scheduled by a Fixed-Priority Preemptive Algorithm*, M. Coutinho, J. Rufino, C. Almeida (submetido para publicação)
- *Securing The Timeliness of I/O Event Handling in Real-Time Kernels*, C. Almeida, M. Coutinho, J. Rufino (submetido para publicação)
- *ARINC 653 in RTEMS*, J. Rufino, S. Filipe, M. Coutinho, S. Santos, J. Windsor, Data Systems In Aerospace (DASIA), Napoles, Itália, Maio 2007
- *VITRAL - A text mode window manager for real-time embedded kernels*, M. Coutinho, C. Almeida, J. Rufino, Emerging Technologies and Factory Automation (ETFA), Prague, Czech Republic, Setembro 2006
- *Control of Event Handling Timeliness in RTEMS*, M. Coutinho, C. Almeida, J. Rufino, Parallel and Distributed Computing Systems (PDCS), Phoenix, Estados Unidos da América, Novembro 2005
- *VITRAL - A text mode window manager for RTEMS*, M. Coutinho, C. Almeida, J. Rufino, Jornadas de Engenharia de Electrónica e Telecomunicações e de Computadores (JETC), Lisboa, Portugal, Novembro 2005

Relatórios Técnicos de Projecto

- ESA/ITI AIR Project Deliverable. WP3 - AIR Design Results and Proof of Concept. M. Coutinho, E. Pascoal, S. Santos and J. Rufino. AIR Consortium: Skysoft Portugal/FCUL. May 2007.
- ESA/ITI AIR Project Deliverable. WP2 - AIR Overall System Specification. M. Coutinho, J. Rufino, S. Santos and E. Pascoal. AIR Consortium: Skysoft Portugal/FCUL. January 2007.

- ESA/ITI AIR Project Deliverable. WP1 - AIR Requirements, Architecture and Services. S. Santos, M. Coutinho and J. Rufino. AIR Consortium: Skysoft Portugal/FCUL. November 2006.

References

. 1991. *Airlines Electronic Engineering Committee (AEEC), Design Guidance for Integrated Modular Avionics (ARINC Specification 651)*. ARINC, Inc.

. 2003. *Airlines Electronic Engineering Committee (AEEC), Avionics Application Software Standard Interface (ARINC Specification 653-1)*. ARINC, Inc.

. 2003. *IEEE Std 1003.13 - Standard for Information Technology - Standardized Application Environment Profile (AEP) - POSIX Real-Time and Embedded Application Support*.

. 2003a (Aug.). *RTEMS C Users Guide*. OAR - On-Line Applications Research Corporation. Edition 4.6.6, for RTEMS 4.6.6 edition.

. 2003b (Aug.). *RTEMS Intel i386 Applications Supplement*. OAR - On-Line Applications Research Corporation. Edition 4.6.6, for RTEMS 4.6.6 edition.

. 2003c (Aug.). *RTEMS POSIX API Users Guide*. OAR - On-Line Applications Research Corporation. Edition 4.6.6, for RTEMS 4.6.6 edition.

. 2003d (Aug.). *RTEMS SPARC Applications Supplement*. OAR - On-Line Applications Research Corporation. Edition 4.6.6, for RTEMS 4.6.6 edition.

. 2004. *IEEE Std 1003.1 - Standard for Information Technology Portable Operating System Interface (POSIX) System Interfaces*.

ALMEIDA, LUÍS, & FONSECA, JOSÉ ALBERTO. 2001. Analysis of a Simple Model for Non-Preemptive Blocking-Free Scheduling. *Pages 233– of: ECRTS*.

ANSI/IEEE. 1993. *1003.1b-1993 Portable Operating System Interface (POSIX) - Part 1: API C Language - Real-Time Extensions*. IEEE Standard. ISBN 1-55937-375-X.

AUDSLEY, N. 1991. *Optimal Priority Assignment and Feasibility of Static Priority Tasks With Arbitrary Start Times*. Tech. rept. Department of Computer Science, University of York.

AUDSLEY, N. 2001. On Priority Assignment in Fixed Priority Scheduling. *Information Processing Letters*, **79**(1), 39–44.

AUDSLEY, N., BURNS, A., RICHARDSON, M. F., & WELLINGS, A. J. 1991. Hard Real-Time Scheduling: The Deadline Monotonic Approach. *In: Proceedings 8th IEEE Workshop on Real-Time Operating Systems and Software*.

BARUAH, SANJOY K., ROSIER, LOUIS E., & HOWELL, RODNEY R. 1990. Algorithms and Complexity Concerning the Preemptive Scheduling of Periodic, Real-Time Tasks on One Processor. *Real-Time Systems*, **2**, 301–324.

BURNS, A., & WELLINGS, A. 2001. *Real-Time Systems and Programming Languages*. Addison-Wesley.

CERVIN, A. 2001 (Mar). Analyzing the Effects of Missed deadlines in control systems. *Pages 17–26 of: ARTES Real-Time Graduate student conference*.

COUTINHO, M., RUFINO, J., & ALMEIDA, C. 2005 (November). Control of Event Handling Timeliness in RTEMS. *In: Proceedings of the 17th IASTED International Conference on Paralel and Distributed Computing Systems - PDCS 2005*. IASTED, Phoenix, Arizona, USA.

COUTINHO, M., ALMEIDA, C., & RUFINO, J. 2006a. VITRAL - A text mode window manager for real-time embedded kernels. *Pages 1254–1260 of: ETFA*.

COUTINHO, M., ALMEIDA, C., & RUFINO, J. 2006b (November). VITRAL - A text mode window manager for RTEMS. *Pages 1254–1260 of: JETC - Jornadas de Engenharia Electrónica e de Computadores*.

DEVILLERS, R., & GOOSSENS, J. 1999. General response time computation for the deadline driven scheduling of periodic tasks. *Fundamenta Informaticae*, **40**(2–3), 199–219.

FERREIRA, JOAQUIM, PEDREIRAS, PAULO, ALMEIDA, LUÍS, & FONSECA, JOSÉ ALBERTO. 2002. The FTT-CAN Protocol for Flexibility in Safety-Critical Systems. *IEEE Micro*, **22**(4), 46–55.

FRANKLIN, G., POWELL, J., & WORKMAN, M. 2001. *Digital Control of Dynamic Systems*. Addison-Wesley Publishing Company.

GOOSSENS, J. 1999 (December). *Scheduling of Hard Real-Time Periodic Systems with Various Kinds of Deadline and Offset Constrains*. Ph.D. thesis, Faculté des Sciences.

GOOSSENS, J., & DEVILLERS, R. 1999 (December). Feasibility intervals for the deadline driven scheduler with arbitrary deadlines. *Pages 54–61 of:* SOCIETY, IEEE COMPUTER (ed), *The 6th International Conference on Real-Time Computing Systems and Applications (RTCSA'99)*.

GRENIER, M., GOOSSENS, J., & NAVET, N. 2006 (May). Near-Optimal Fixed Priority Preemptive Scheduling of Offset Free Systems. *In: Proceedings of the 14th International Conference on Network and Systems (RTNS'2006)*.

JEFFAY, K., STANAT, D., & MARTEL, C. 1991 (December). On Non-Preemptive Scheduling of Periodic and Sporadic Tasks. *Pages 129–139 of:* PRESS, IEEE COMPUTER SOCIETY (ed), *Proceeedings of the 12th IEEE Real-Time Systems Symposium*.

JOSEPH, M., & PANDYA, P. 1986. Finding the Response Times in a Real-Time System. *Computer Journal*, **29**(5), 390–395.

KOPETZ, H. 1997. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Kluwer Academic.

KOPETZ, HERMANN. 1991. *Event-Triggered versus Time-Triggered Real-Time Systems*. Research Report 8/1991. Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria.

KOPETZ, HERMANN. 1993. Should Responsive Systems be Event-Triggered or Time-Triggered ? *Institute of Electronics, Information, and Communications Engineers Transactions on Information and Systems*, **E76-D**(11), 1325–1332.

KOPETZ, HERMANN. 2002. Time Triggered Architecture. *ERCIM NEWS*, Jan., 24–25.

LEHOCZKY, JOHN, SHA, LUI, & DING, YE. 1989. The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior. *Pages 166–171 of: Proceedings IEEE Real-Time Systems Symposium*.

LEUNG, J., & WHITEHEAD, J. 1982. On the complexity of fixed-priority scheduling of periodic real-time tasks. *Performance Evaluation*, **2**, 237–250.

LEUNG, JOSEPH Y.-T., & MERRILL, M. L. 1980a. A Note on Preemptive Scheduling of Periodic, Real-Time Tasks. *Inf. Process. Lett.*, **11**(3), 115–118.

LEUNG, JOSEPH Y.-T., & MERRILL, M. L. 1980b. A Note on Preemptive Scheduling of Periodic, Real-Time Tasks. *Information Processing Letters*, **11**(3), 115–118.

LIU, C. L., & LAYLAND, JAMES W. 1973. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *Journal of the ACM*, **20**(1), 46–61.

LIU, J. 2000. *Real-Time Systems*. Prentice Hall.

LONN, H., & AXELSSON, J. 1999. *A Comparison of Fixed-Priority and Static Cyclic Scheduling for Distributed Automotive Control Applications*.

MÄKI-TURJA, JUKKA, & NOLIN, MIKAEL. *Tighter Response-Times for Tasks with Offsets*.

MÄKI-TURJA, JUKKA, & NOLIN, MIKAEL. 2004. Efficient Response-Time Analysis for Tasks with Offsets. *Page 462 of: 10th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'04)*.

PEDREIRAS, PAULO, ALMEIDA, LUÍS, & GAI, PAOLO. 2002. The FTT-Ethernet Protocol: Merging Flexibility,Timeliness and Efficiency. *Page 152 of: ECRTS '02: Proceedings of the 14th Euromicro Conference on Real-Time Systems*. Washington, DC, USA: IEEE Computer Society.

REGEHR, JOHN, & DUONGSAA, USIT. 2005 (June). Preventing Interrupt Overload. *In: Proc. of the ACM Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES 2005)*.

RUFINO, J., FILIPE, S., COUTINHO, M., SANTOS, S., & WINDSOR, J. 2007 (May). ARINC 653 in RTEMS. *In: DASIA*.

SANDTROM, K., ERIKSON, C., & G.FOHLER. 1998. Handling Interrups with Static Scheduling in a Automative Vehicle Control System. *In: 5th International Conference on Real-Time Computing Systems and Applications (RTCSA'98)*.

SERONIE-VIVIEN, J., & CANTENOT, C. 2005 (June). RTEMS Operating System Qualification. *In: Proceedings of the DASIA 2005 "DAta Systems In Aerospace" Conference*. EUROSPACE, Edinburgh, Scotland.

SHA, L., RAJKUMAR, R., & LEHOCZKY, J. P. 1990. Priority Inheritance Protocols: An Approach to Real-Time Synchronization. *IEEE Transactions on Computers*, **39**(9), 1175–1185.

SHIN, K., & CHUI, X. 1995. Computing time delay and its effects on real-time control systems. *IEEE Transactions on Control Systems Technology*, **3**(2), 218–224.

STRAUMANN, T. 2001. Open Source Real Time Operating Systems Overview. *In: International Conference on Accelerator and Large Experimental Physics Control Systems*.

TINDELL, K., BURNS, A., & WELLINGS, A. 1995. Calculating Controller Area Network (CAN) message response times.

TURJA, J., HANNINEN, K., & NOLIN, M. 2005 (Feb). *Response Times in Hybrid Scheduled Systems*. Tech. rept.