
UNIVERSIDADE TÉCNICA DE LISBOA

Instituto Superior Técnico

# Trabalho Final de Curso

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

## 157/2004/L

Manuel Coutinho, nº 49414, AE de Sistemas, Decisão e Controlo

Licenciatura em Engenharia Electrotécnica e de Computadores
**Relatório Final do Trabalho Final de Curso**

Prof. Orientador: Carlos Almeida
Prof. Acompanhante: José Rufino

Junho de 2005

# Agradecimentos

O autor agradece, em primeiro lugar, aos Professores Orientadores deste Trabalho Final de Curso, Carlos Almeida e José Rufino, sem os quais, certamente não estaria concluído. Gostaria também de salientar os preciosos comentários e sugestões realizados pelos colegas com os quais se realizaram várias reuniões de grupo, analisando e discutindo vários problemas e soluções. Por último, mas não menos importante, gostaria de agradecer à sua família que, com a uma paciência sem limites, aturaram as infinitas teimosias.

Este trabalho encontra-se integrado no projecto DARIO (*Distributed Agency for Reliable Input/Output*), financiado pela FCT (Fundação para a Ciência e Tecnologia), através do programa POSC/EIA/56041/2004.

# Resumo

Este trabalho final de curso aborda Sistemas de Tempo Real Distribuídos. No meio fabril, estes sistemas são componentes fundamentais pois distribuem as tarefas em várias células e operam de modo síncrono para garantir que, em condições de funcionamento normais, nem equipamento nem vidas humanas são postos em perigo.

Como núcleo de sistema operativo de tempo real, é utilizado o RTEMS (*Real-Time Executive for Multiprocessor Systems*), que reúne funcionalidades e documentação necessárias, para além de possuir uma filosofia *open-source*, apreciada por programadores que trabalham na área de investigação e/ou desenvolvimento.

O desenvolvimento de aplicações sobre o RTEMS leva à necessidade de um significativo conhecimento acerca da sua estrutura e funcionamento interno. A criação de um ambiente de separação entre os programadores que operam sobre o código fonte do RTEMS e os que constróem a aplicação é fundamental para impor um nível de abstracção maior, moderando o custo e aumentando a rapidez de desenvolvimento.

A implantação das aplicações na plataforma computacional alvo beneficia de um arranque rápido e flexível das aplicações, implementado num mecanismo de arranque remoto (*Remote Boot*), construído sobre a infra-estrutura computacional e de comunicação existente.

Um novo componente, o gestor de janelas VITRAL, colmata uma lacuna existente na apresentação dos dados no RTEMS. A construção de janelas em modo texto policromáticas permite um novo nível de clareza e organização. O VITRAL é, no entanto, construído de forma cuidada de modo a não interferir com as características de tempo real do sistema.

A integração de dispositivos de entrada/saída, tal como o VITRAL, com o sistema subjacente apresenta um desafio encontrado pela grande variedade de dispositivos existentes. Uma interface bem definida necessita de ser estabelecida para uma integração modular e escalonável entre os sensores/actuadores e sistema computacional.

Por último, mas não menos importante, a integração de dispositivos de entrada pode comprometer as metas temporais definidas, através da geração de eventos descontrolada. Este trabalho apresenta introduz mecanismos que impedem o processamento deste número alto de eventos, de modo a mantendo o cumprimento das metas temporais.

# Palavras-Chave

Sistemas Embebidos de Tempo Real, Entrada/Saída, Eventos, Interrupções, Sistema Distribuídos

# Abstract

The present work addresses Real Time Distributed Systems. In the industrial field, these are fundamental components for they distribute the several tasks into specialized cells that operate in a synchronous fashion to insure that, under normal conditions, nor equipment nor human lives are jeopardized.

The operating system core explored is the RTEMS (Real-Time Executive for Multiprocessor Systems), which encompasses the necessary functionalities and documentation, besides having an open-source philosophy, appreciated by developers, research related industries and academics.

The applications development under RTEMS brings the necessity for a significant knowledge about its internal structure and operating mode. The creation of an environment that separates the programmers which operate on the RTEMS source code and on the application side is very appealing to impose a greater level of transparency, moderating the cost and increasing the speed of the development.

The use of the Remote Boot to start the application on the computational platform improves the speed and flexibility of the application initialization, thus improving the environment of the programmers.

A new component, the window manager, VITRAL (Stained Glass Window in English), addresses the existent lack of quality in the data presentation under RTEMS. The use of multicolored windows allows a new level of clarity and organization. The VITRAL however, is built with great care so as not to interfere with the real time characteristics of the system.

The integration of input/output devices, such as VITRAL, with the underlying system provides a challenge found by the great diversity of existent devices. A well-defined interface must be established to allow a modular and scalable integration of sensors/actuators with the computational system.

For last, but not least, the devices integration may compromise the real-time deadlines, through an uncontrolled event generation. This work presents mechanisms that prevents the processing of such a high number of events, to maintain the fulfillment of the deadlines.

# Keywords

Real Time Embedded Systems, Input/Output, Events, Interruptions, Distributed Systems

# Índice

# Lista de Figuras

# Lista de Tabelas

# Lista de Siglas

| | |
|---|---|
| API | *Application Programming Interface* |
| ASR | *Asynchronous Signal Routine* |
| BIOS | *Basic Input/Output System* |
| CPU | *Central Processing Uni*t |
| EDF | *Earliest Deadline First* |
| EEPROM | *Electrically Erasable Programmable Read Only M*emory |
| EPROM | *Erasable Programmable Read Only Memory* |
| FIFO | *First In First Out* |
| FIR | *Finite Impulse Respo*nse |
| GUI | *Graphical Unit Interface* |
| IIR | *Infinite Impulse Response* |
| IRQ | *Interrupt ReQuest* |
| ISR | *Interrupt Service Routine* |
| IST | *Interrupt Service Task* |
| LTI | *Linear Time Invariant* |
| MIT | *Minimum Inter-arrival Time* |
| MLF | *Minimum Laxity First* |
| MUT | *Maximum Urgency First* |
| PROM | *Programmable Read Only Memory* |
| RMS | *Rate Monotonic Sche*duler |
| RTC | *Real Time Clock* |
| RTEMS | *Real Time Executive for Multiprocessor Systems* |

| | |
|---|---|
| RTOS | *Real Time Operating Syste*ms |
| SLIT | Sistema Linear Invariante no Tempo |
| SO | Sistema Operativo |
| TCB | *Task Control Block* |
| TFC | Trabalho Final de Curso |

# Capítulo 1

# Introdução

Nas últimas décadas temos vindo a assistir a um desenvolvimento progressivo das tecnologias de sistemas distribuídos e correspondentes aplicações. Apesar do progresso verificado, continuam a subsistir alguns nichos de aplicações onde a evolução tem sido mais lenta. Estão neste caso, os denominados sistemas distribuídos embebidos, sistemas onde a par da distribuição existem requisitos adicionais de tolerância a faltas e de operação de tempo-real, que frequentemente necessitam de ser concretizados em ambientes computacionais com recursos moderados (e.g. processamento, memória). Encontram-se neste caso, muitas aplicações de controlo em que 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.

Este Trabalho Final de Curso contribui para a resolução do problema de conceber e construir uma sistema embebido tolerante a faltas para aplicações de controlo distribuído em tempo-real. Esta contribuição é efectuada no âmbito do Projecto DARIO – "*Distributed Agency for Reliable Input/Output*" que 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.

Em concreto, o trabalho realizado desenvolveu-se segundo as linhas de actuação seguintes:

- Definição da plataforma computacional a utilizar incluindo núcleo de sistema operativo e instalação de uma cadeia de produção de aplicações para esta plataforma.
- Definição dos procedimentos associados à implantação das aplicações produzidas na plataforma computacional alvo.
- Enriquecimento das funcionalidades proporcionadas pelo núcleo de sistema operativo, nomeadamente aos seguintes aspectos:
  - Interface com utilizador.
  - Integração de dispositivos de entradas/saídas.
- Definição e concepção de mecanismos/procedimentos para preservação das características de tempo-real do sistema, incluindo o controlo da pontualidade perante condições de sobrecarga.

Como núcleo de sistema operativo de tempo real, é utilizado o RTEMS (*Real-Time Executive for Multiprocessor Systems*), que reúne funcionalidades e documentação necessárias, para além de possuir uma filosofia *open-source*, apreciada em meios académicos. A descrição de Sistemas de Tempo e Real e uma pequena introdução ao RTEMS encontram-se no terceiro capítulo.

O desenvolvimento de aplicações sobre o RTEMS leva à necessidade de um significativo conhecimento acerca da sua estrutura e funcionamento interno. A criação de um ambiente de separação entre os programadores que operam sobre o código fonte do RTEMS e os que

constróem a aplicação é fundamental para impor um nível de abstracção maior, moderando o custo e aumentando a rapidez de desenvolvimento. O Anexo A contem uma discussão sobre a criação deste ambiente.

A implantação das aplicações na plataforma computacional alvo beneficia de um arranque rápido e flexível das aplicações, implementado num mecanismo de arranque remoto (*Remote Boot*), construído sobre a infra-estrutura computacional e de comunicação existente. A descrição da engenharia do seu funcionamento e implementação encontra-se no Anexo B.

Um outro aspecto de igual ou maior relevância prende-se com a apresentação dos dados por parte de núcleos de Sistema Operativo, tal como o RTEMS. A interface nativa da distribuição do RTEMS é semelhante a um terminal em *Linux*. Embora seja útil em aplicações com um único fluxo de acções, em ambientes multitarefa, constitui um entrave na construção de uma apresentação clara e organizada dos dados.

Um novo componente, o gestor de janelas VITRAL, colmata esta lacuna, substituindo a interface original, permitindo que aplicações residentes permaneçam inalteradas no seu comportamento. A construção de janelas em modo texto policromáticas permite um novo nível de clareza e organização e preservando no entanto as características de tempo real do sistema.

O quarto capítulo explora a arquitectura e construção do VITRAL no RTEMS, definindo também uma camada de portabilidade para outros sistemas.

A integração de dispositivos de entrada/saída, tal como o VITRAL, com o sistema subjacente apresenta um desafio encontrado pela grande variedade de dispositivos existentes. Uma interface bem definida necessita de ser estabelecida para uma integração modular e escalonável entre os sensores/actuadores e sistema computacional. O quinto capítulo aborda a integração de dispositivos genéricos com o sistema operativo utilizado, com especial ênfase no RTEMS.

Por último, mas não menos importante, a integração de dispositivos de entrada pode comprometer as metas temporais definidas, através da geração de eventos descontrolada. O sexto capítulo descreve mecanismos de forma a colmatar este obstáculo à construção de ambientes de tempo real.

Este trabalho integra-se no âmbito do Projecto DARIO (*Distributed Agency for Reliable Input/Output*), financiado pela FCT (Fundação para a Ciência e Tecnologia), através do programa POSC/EIA/56041/2004.

## Resumo da Organização do Documento

- Segundo capítulo – identifica os problemas principais a resolver e apresenta de forma breve as respectivas soluções.
- Terceiro capítulo – são introduzidas noções fundamentais de tempo real e de sistemas de tempo-real e realiza-se uma breve descrição do núcleo de sistema operativo de tempo real utilizado, o RTEMS.
- Quarto capítulo – é discutida a arquitectura e implementação do gestor de janelas – VITRAL.

- Quinto capítulo – é discutida a integração modular de dispositivos de entrada/saída com o sistema computacional
- Sexto capítulo – são apresentados mecanismos que garantem o cumprimento das metas temporais na presença de sobrecargas de eventos
- Sétimo capítulo – apresentam-se as conclusões e perspectivas de trabalho futuro
- Os Anexos contêm informação adicional, relativa a questões práticas de engenharia:
    - Anexo A – Apresentação de um ambiente de desenvolvimento de aplicações que utilizam o RTEMS.
    - Anexo B – Discussão do processo de Arranque Remoto e a sua implementação na infraestrutura utilizada.
    - Anexo C – Análise matemática de um dos métodos apresentados para tratamento de sobrecargas de eventos
    - Anexo D - Análise matemática para escolha de parâmetros associados a um dos métodos apresentados para tratamento de sobrecargas de eventos

Capítulo 2

# Plataformas para Controlo Industrial

Os Sistemas de Controlo Distribuído são peças extremamente importantes na automação de processos industriais. São normalmente constituídos por sistemas computacionais embebidos interligados através de uma infra-estrutura de comunicação. Cada sistema embebido constitui um nó da rede e possui recursos escassos devido à tentativa de minimização de custos. A capacidade de armazenamento de dados através de memória em massa, a velocidade e a capacidade de processamento, a disponibilidade de interface sofisticada com o utilizador, são normalmente sacrificadas de modo a possibilitar a construção de uma solução eficaz mas de custo moderado.

A construção de Sistemas Embebidos para Controlo Distribuído tem, nos últimos anos, seguido duas filosofias que em certo sentido podem ser consideradas antagónicas. A primeira, adoptada em larga medida no desenvolvimento da TTA (*Time-Triggered Architecture*) [1], implica construir todo o sistema de raiz. Regra geral, os custos de desenvolvimento associados são extremamente elevados e as soluções alcançadas são frequentemente de natureza proprietária e especializada.

A segunda aproximação, seguida por exemplo em [2], privilegia a utilização de componentes padrão de origem comercial (COTS – *Commercial Off-The-Shelf*), amplamente disponíveis a custo reduzido. Na elaboração do presente trabalho, segue-se igualmente esta aproximação, explorando sempre que possível a disponibilidade de componentes "*off-the-shelf*" complementada, quando necessário, com a utilização de componentes específicos a cada aplicação (e.g. interfaces a sensores/actuadores específicos).

Nesta perspectiva, na definição de uma plataforma fiável para controlo distribuído em tempo-real considera-se a utilização dos seguintes componentes:

- Plataformas Computacionais: centradas na utilização de cartas de computador pessoal padrão, complementadas com componentes de interface a redes de tempo real, como por exemplo redes de dispositivos (*fieldbuses*), e componentes de interface a sensores e actuadores.
- Sistemas Operativos: onde se explora a utilização de sistemas do domínio público adequados à utilização em causa: Linux, como sistema de desenvolvimento; núcleos multi-tarefa de tempo-real, para as plataformas embebidas de controlo.
- Redes de Comunicação: para interligação dos nós da plataforma computacional entre si e ao mundo exterior (e.g. acesso remoto via Internet), como representado na Figura 2.1.
- Interfaces a Dispositivos de Entrada/Saída: interfaces de natureza eminentemente especializada para interligação ao "mundo-real".

A utilização de infra-estruturas de computador pessoal (PC) padrão em plataformas de sistemas embebidos visa fornecer uma plataforma de processamento poderosa a custo reduzido. Frequentemente, prescinde-se da utilização de alguns dispositivos clássicos no PC padrão, como por exemplo, os dispositivos de armazenamento em memória de massa.

Neste contexto, e dados os requisitos acrescidos de pontualidade por parte das aplicações de controlo, a utilização de tecnologias de sistemas operativos em plataformas embebidas de

controlo centra-se habitualmente no uso de núcleos de sistema operativo multi-tarefa de tempo-real. Estes fornecem a funcionalidade necessária à estruturação das aplicações de uma forma modular enquanto oferecem os mecanismos base para escalonamento de acções de forma atempada. A integração de componentes de tempo real no sistema operativo é imperativa, de forma a constituir uma base sólida para a construção de aplicações fiáveis. Frequentemente, os mecanismos base fornecidos pelo sistema operativo necessitam ser complementados de forma a reduzir o grau de incerteza temporal, garantindo o cumprimento de metas temporais, mesmo na presença de factores de perturbação (e.g. sobrecarga temporária).

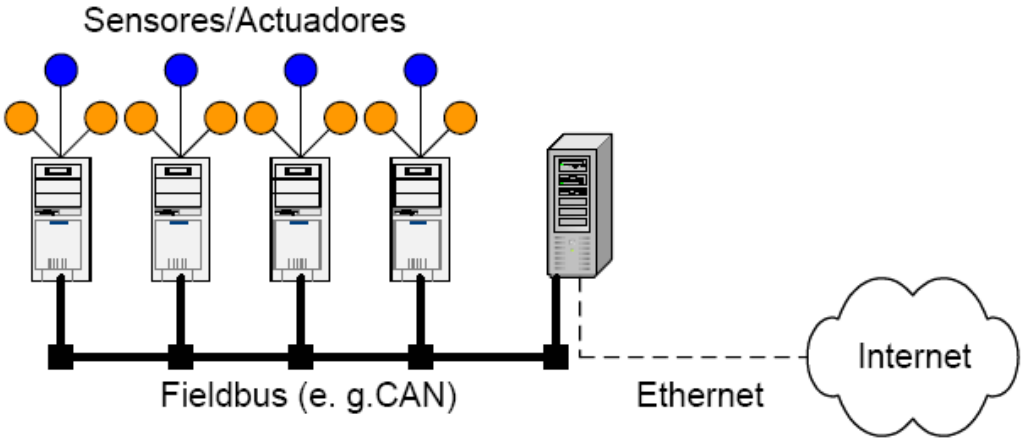


Figura 2.1 – Integração de Sistemas Distribuídos de Tempo Real com o Mundo Exterior

Os núcleos de sistema operativo existentes no domínio público apresentam-se como uma ferramenta poderosa, no sentido em que permitem obter um ciclo de desenvolvimento de curta duração, a baixo custo. Entre as ofertas disponíveis encontra-se o RTEMS (*Real Time Executive for Multiprocessor Systems*). A escolha do RTEMS como núcleo de sistema operativo base surge como consequência lógica dos requisitos de tempo real e recursos reduzidos encontrados nas plataformas embebidas de controlo. A sua filosofia *open-source* e uma grande portabilidade em relação a bibliotecas sistema existentes em ambientes como o Linux tornam-no um alvo preferencial em ambientes tanto académicos como comerciais. O RTEMS é considerado actualmente como uma ferramenta robusta, devido a uma permanente evolução através da OAR (*On-Line Applications Research Corporation*).

O desenvolvimento de uma plataforma computacional para controlo distribuído em tempo-real centrada numa infra-estrutura de computador pessoal, exige assim como primeira etapa a disponibilização de uma base de instalação do RTEMS obedecendo aos seguintes requisitos:

- Desenvolvimento cruzado de aplicações no ambiente Linux, tendo como alvo a plataforma embebida de controlo;
- Suporte à integração de componentes específicos (e.g. mecanismos para preservação da pontualidade em condições de sobrecarga, interfaces a dispositivos específicos entrada/saída) de forma modular e flexível.
- Independência deste processo de integração face às diferentes distribuições do RTEMS.

Integrada a aplicação com o RTEMS, o executável da aplicação pode ser introduzido de diversas formas no sistema alvo. A sua cópia para uma *diskette* e consequente arranque, através de um processo designado *grub*, constituí a solução mais simples, tendo no entanto, grandes

inconvenientes pela sua lentidão. A transferência da imagem do executável para uma EPROM e o seu arranque constitui uma solução alternativa, mas contendo também graves desvantagens durante a alteração do código. A sua inflexibilidade durante as fases de construção e depuração tornam-no uma alternativa mais demorada que o próprio carregamento através do *grub*.

Finalmente, apresenta-se o arranque através de um processo designado arranque remoto (*remote boot*, onde a imagem do executável é colocada num servidor, actuando o sistema alvo como um cliente (através de um programa colocado numa EPROM integrada na placa de rede) e realizando o pedido de transferência durante a sua inicialização. Este processo, para além de rápido e flexível, não acarreta a alteração da infra-estrutura, uma vez que a rede de comunicação é um elemento fundamental em plataformas distribuídas. Aliás, as infra-estruturas de computador pessoal padrão recentemente já integram na própria carta principal os mecanismos base para suporte ao processo de arranque remoto.

O desenvolvimento de uma plataforma computacional para controlo distribuído em tempo-real centrada numa infra-estrutura de computador pessoal, exige assim:

- Disponibilização de um ambiente de suporte ao processo de arranque remoto incluindo, sempre que necessário, a produção das EPROMs de arranque remoto;
- Disponibilização e definição de uma metodologia de configuração dos servidores/serviços necessários ao processo de arranque remoto.
- Integração do suporte ao processo de arranque remoto na cadeia de produção de aplicações do RTEMS.

Sempre que a infra-estrutura distribuída incluir a utilização de redes de dispositivos (e.g. CAN – *Controller Area Network*) a filosofia do processo de arranque remoto deve ser estendida a este ambiente – Figura 2.2, através da utilização de procedimentos padrão (e.g. funções de arranque remoto no protocolo CANopen) ou recorrendo a protocolos especializados que optimizam o processo para redes de dispositivos particulares (por exemplo – protocolo CANBoot [3])). Contudo, a engenharia destes procedimentos não foi abordada no presente trabalho.

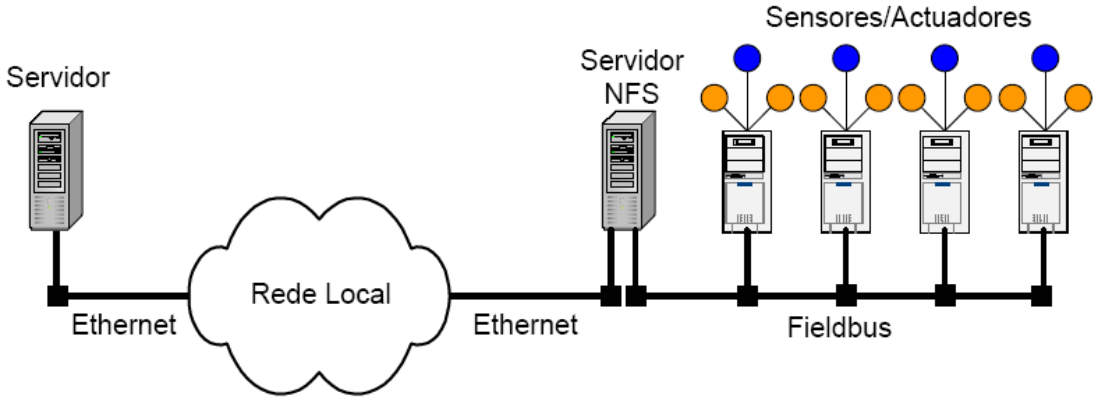


Figura 2.2 – Arranque Remoto dos sistema embebidos através da interligação do Sistema de Tempo Real com o exterior

A interface nativa do RTEMS com o operador da aplicação constitui uma grave lacuna na apresentação dos dados de uma forma clara e organizada. Pode ser comparável a um terminal em Linux, o que pode ser suficiente em aplicações estimuladas por um único fluxo de acções, mas em geral é insuficiente em ambientes multitarefa, onde se deseja uma representação separada e

clara dos dados relativos a cada tarefa. Embora seja simples e intuitiva de utilizar, não fornece o nível desejado de paralelismo esperado em sistemas multitarefa.

No sentido de colmatar esta lacuna, um novo gestor de janelas, VITRAL, é apresentado como uma solução para sistemas de tempo real. Uma versão preliminar do VITRAL representa um ponto de partida para a construção de um sistema robusto em ambientes com restrições temporais extremamente rigorosas. O gestor VITRAL é responsável tanto pela escrita de dados da aplicação no monitor, como reagir a eventos provenientes do teclado. O tratamento destes dois dispositivos introduz a necessidade da integração de dispositivos de entrada/saída com um carácter modular. A sua interligação com o sistema operativo (no caso, RTEMS) deve seguir uma interface bem conhecida e estabelecida *a priori*, de forma a aumentar a compatibilidade com as várias versões e mesmo com outros sistemas operativos.

A concepção e o desenvolvimento do VITRAL, visando a sua integração na plataforma computacional para controlo distribuído em tempo-real a utilizar, implica:

- Definição das características funcionais do gestor de janelas e identificação dos componentes activos intervenientes.
- Definição da arquitectura interna do gestor de janelas e das interacções necessárias entre os diferentes componentes constituintes, sem contudo comprometer a portabilidade do gestor de janelas e a preservação das características temporais do sistema.
- Implementação da arquitectura do gestor de janelas VITRAL e sua integração no sistema operativo explorando os mecanismos próprios do RTEMS.

A integração dos componentes do VITRAL no núcleo de sistema operativo deve ser efectuada através de um processo abrangente, modular e flexível que contemple igualmente os restantes dispositivos de entrada/saída – Figura 2.3.

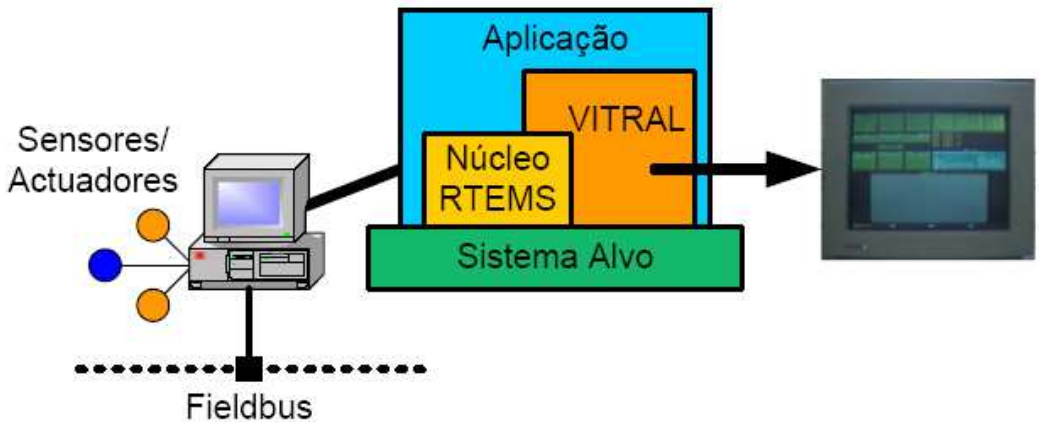


Figura 2.3 – Interligação do gestor de janelas VITRAL com o núcleo de Sistema Operativo RTEMS

À medida que a popularidade dos sistemas computacionais aumenta, tanto no sector industrial, comercial como privado, também cresce o número e diversidade de dispositivos. A sua interligação de uma forma modular e escalonável, não prescindindo da preservação das características de tempo real do sistema, é fundamental para minimizar o custo de desenvolvimento. A uniformização da interface com o sistema operativo provocou a divisão em classes dos dispositivos, encontrando assim uma solução de compromisso entre a definição de

uma interface bem definida com a variedade dos dispositivos disponíveis. Como exemplo de estudo da interligação de um dispositivo com o Sistema Operativo (RTEMS), apresenta-se uma prensa electro-pneumática.

A integração de componentes adicionais de entrada/saída na plataforma computacional para controlo distribuído em tempo-real é crucial e exige:

- Definição de uma arquitectura modular e flexível para integração uniforme de dispositivos de interface com o "mundo exterior", através de sensores/actuadores, preservando as características temporais do sistema.
- Avaliação do processo de separação hierárquica da arquitectura de entradas/saídas de forma a contemplar capacidades funcionais diversificadas, como por exemplo configuração de sensores inteligentes, transferência de aplicações robóticas residentes, acesso remoto dos dispositivos – Figura 2.4.
- Projecção desta arquitectura sobre os mecanismos tradicionais de entrada/saída existentes na maioria dos sistemas operativos, incluindo o RTEMS (*device drivers*).

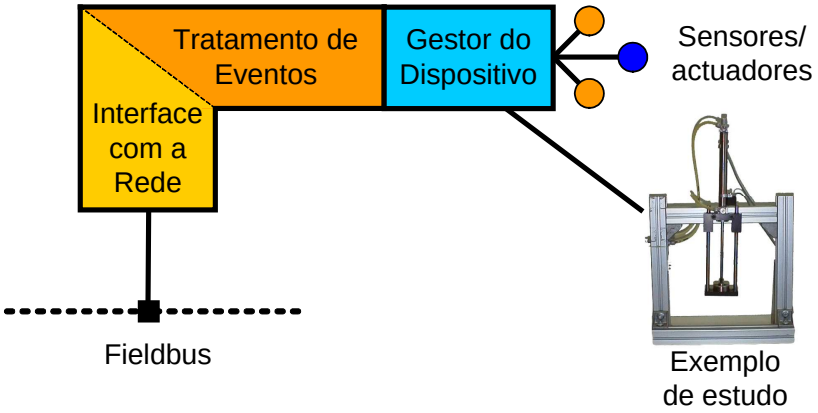


Figura 2.4 – Integração Modular de dispositivos de entrada/saída em Ambientes Distribuídos de Tempo Real

O tratamento de eventos assíncronos, tal como a leitura do teclado, recepção de pacotes, etc., representa um possível ponto de falha no cumprimento dos requisitos temporais. Em situações de sobrecargas de eventos, o sistema pode não conseguir garantir o seu processamento bem como a conclusão das tarefas em execução dentro dos prazos temporais pré-estabelecidos. Em geral, o número de eventos processados pode ser relaxado de maneira a ser possível garantir o cumprimento das metas temporais. Nestes casos, a imposição de um limite ao número de eventos processados introduz garantias relativamente à sua interferência com as tarefas. A introdução de um cenário de pior caso permite estabelecer (ou não) garantias de que o sistema suporta tanto os eventos admitidos, como o cumprimento das metas temporais da aplicação.

Finalmente, a última contribuição para a definição de uma plataforma computacional para controlo distribuído em tempo-real envolve:

- definição, desenvolvimento e integração de mecanismos modulares e flexíveis para avaliação de um conjunto relevante de métricas relativas ao comportamento do sistema e detecção de condições de sobrecarga.
- definição, desenvolvimento e integração de mecanismos que efectuem o controlo da pontualidade do sistema/aplicações em condições de sobrecarga.

## Identificação das Contribuições Técnicas

O presente trabalho trata 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:

- Definição e adaptação do ambiente de geração de aplicações para o sistema operativo RTEMS, tendo em visto a sua utilização específica em ambientes distribuídos envolvendo operações de entradas/saídas.
- Implantação dos mecanismos necessários a um arranque rápido e flexível da aplicação através de mecanismos de Arranque Remoto.
- Definição, concepção e integração de uma interface de gestão de janelas, denominada VITRAL, que visa a qualidade e organização da apresentação de dados na consola.
- Definição, concepção e integração de uma interface modular e flexível para incorporação de gestores de entrada/saída que comunicam com o "mundo exterior" através de sensores e actuadores.
- Controlo da pontualidade do sistema/aplicações em condições de sobrecarga de eventos.

## Abordagem Técnica das Contribuições

- O processo de engenharia desenvolvido sobre alguns temas encontra-se discutido em Anexo, devido à concretização de apenas alguns aspectos práticos já abordados noutros trabalhos
- A discussão sobre a implementação da infraestrutura necessária para o Arranque Remoto encontra-se no Anexo B. A integração modular do desenvolvimento, conquanto seja inovadora, também se prende significativamente com aspectos práticos de engenharia, pelo que também se encontra em Anexo – A.
- Os conceitos e implementação do gestor de janelas VITRAL encontram-se no quarto capítulo.
- Uma discussão aprofundada da concepção e integração de dispositivos de entrada/saída no sistema computacional encontra-se no quinto capítulo.
- A pontualidade do sistema de tempo real é tratada no sexto capítulo, sendo apresentados mecanismos de detecção e controlo em situações de sobrecarga de eventos.

## Disseminação de Resultados

Os resultados obtidos neste trabalho encontram-se disseminados no seguintes artigos:

- *VITRAL: A Text Mode Windows Manager for RTEMS*, JETC Jornadas de Engenharia Electrónica Telecomunicações e de Computadores. 2005. (submetido para publicação).
- *Securing the Timeliness of Input/Output Event Handling in Real-Time Kernels*, (em preparação).
- *Control of Event Handling Timeliness in RTEMS*, IASTED International Conference on Parallel and Distributed Computing Systems. 2005. (submetido para publicação).

Capítulo 3

# Sistemas de Tempo Real

Um sistema diz-se pertencente à classe de Tempo Real se possui um conjunto de regras estritas que regem a sua execução, sendo o factor mais importante as restrições temporais [4]. Se tomarmos sistemas de Tempo Real como uma "caixa preta", com um conjunto de entradas e saídas, então são objectos dos quais se espera um conjunto de resultados correctos e que produzam a saída desejada após um certo intervalo de tempo limitado de receber a entrada correspondente.

Grande parte dos sistemas de Tempo Real gerem o tratamento de várias tarefas simultaneamente, interagindo com diversos dispositivos. No caso de Programação Sequencial, Figura 3.1 a), o tratamento é feito, tal como o nome indica, sequencialmente, mas com a particularidade de que, se as tarefas forem independentes entre si, nenhuma tarefa poder bloquear o processador mais do que um certo intervalo de tempo, para que todas possam ser executadas. Em contrapartida, o paradigma de Programação Concorrente, Figura 3.1 b) é mais atractivo para o programador da aplicação, pois é-lhe permitido muito maior flexibilidade e transparência, tendo, no entanto, de possuir mecanismos de escalonamento que decidem qual a tarefa que deve ser executada [5].

Os núcleos de sistema operativo multitarefa incorporam um escalonador de modo a permitir mudanças de tarefa em execução, implementando assim um nível de *pseudo-paralelismo*. Para além do escalonador, os núcleos de sistema operativo (SO) também necessitam de incluir mecanismos de modo a que as tarefas possam coexistir sem perturbarem o seu funcionamento, já que podem ser interrompidas em qualquer sítio da sua execução, inclusive no acesso ao mesmo dispositivo ou região de memória. São necessários algoritmos de gestão de memória, entradas/saídas, sistemas de ficheiros, para além de meios que permitem que as tarefas possam comunicar entre si (por meio de variáveis partilhadas, mensagens, eventos, etc.) [6].

Os Sistemas Operativos de Tempo Real, para além de fornecerem todos (ou parte) estes mecanismos, possuem a particularidade de serem deterministas na sua execução, limitando a interferência temporal do próprio escalonador, colocando secções de preempção dentro das próprias chamadas de sistema e inibindo interrupções durante o mínimo intervalo de tempo possível [6].

## 3.1 Classificação dos Sistemas de Tempo Real

Os Sistemas de Tempo Real podem ser classificados em três tipos quanto às consequências resultantes do incumprimento da meta temporal (Figura 3.2) [6]:

1. *Hard Real Time*
2. *Firm Real Time*
3. *Soft Real Time*

No primeiro caso, o incumprimento da meta temporal da tarefa tem como consequência a falha catastrófica do sistema, que pode incluir a perda de vidas humanas ou de equipamento de

custo elevado – Figura 3.2. A utilidade da terminação da tarefa depois da meta temporal é considerada "-∞". Um exemplo de um sistema deste tipo seria o arrefecimento de uma central nuclear ou o controlo de um avião, onde se a resposta do sistema não for atempada, as consequências envolvem tanto a perda de vidas humanas como a de equipamento. Sistemas *Firm Real Time* também não possuem utilidade positiva passada a meta temporal, mas as suas consequências não são tão desastrosas, tendo a função de utilidade um valor nulo. O terceiro caso corresponde a um cenário *best-effort*, onde a utilidade é constante até à meta temporal, a partir da qual começa a decrescer até o valor nulo, resultando apenas numa perda de desempenho sem consequências de maior. A Figura 3.2 demonstra as funções de mérito relativas a cada caso.

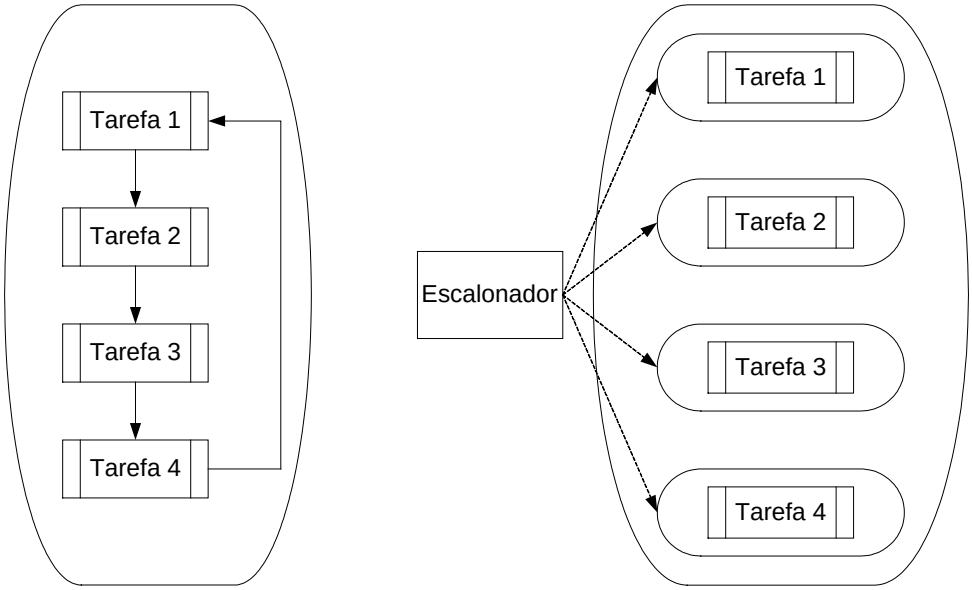


Figura 3.1 – Exemplo de Programação Multitarefa. a) Programação Sequencial. b) Programação Concorrente

## 3.2 Classificação das Tarefas

As tarefas podem ser classificadas, quanto às suas características temporais, em três grupos

1. Tarefas Periódicas
2. Tarefas Aperiódicas
3. Tarefas Esporádicas

O primeiro grupo atribui para cada tarefa $J_i$ um conjunto de parâmetros conhecidos $(c_i, T_i, D_i)$. O primeiro parâmetro, $c_i$, corresponde ao tempo máximo de execução. $T_i$ corresponde ao período da tarefa e $D_i$ ao intervalo de tempo entre o começo da tarefa e a sua meta temporal. Estas tarefas possuem o tratamento mais fácil devido ao conhecimento prévio de quando se tornam executáveis dado pelo seu período [7].

O segundo grupo corresponde a tarefas que reagem a eventos externos e desconhecidos *a priori*. São parametrizados pelo triplo $(c_i, a_i, D_i)$, onde $a_i$ corresponde ao período de tempo entre activações.

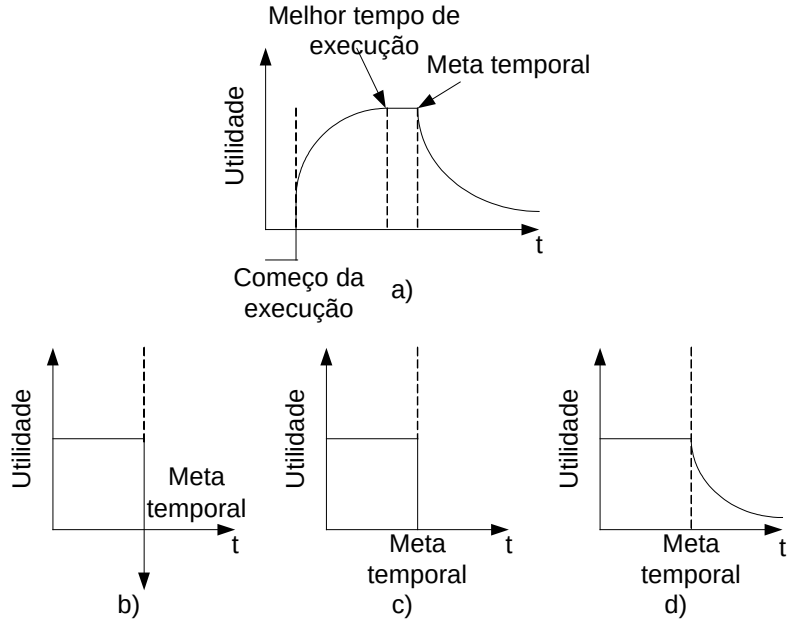


Figura 3.2 – Funções de Mérito para cada caso de Sistemas de Tempo Real. a) Sistemas genéricos. b) Sistemas *Hard Real Time*. c) Sistemas *Firm Real Time*. d) Sistemas *Soft Real Time*

Tarefas Esporádicas são semelhantes às aperiódicas reagindo a eventos, mas com um elemento adicional, *MIT*, sendo definidas por $(c_i, a_i, MIT, D_i)$. O parâmetro *MIT* (*Minimum Interarrival Time*) corresponde ao intervalo de tempo mínimo entre a ocorrência de dois eventos (Figura 3.3).

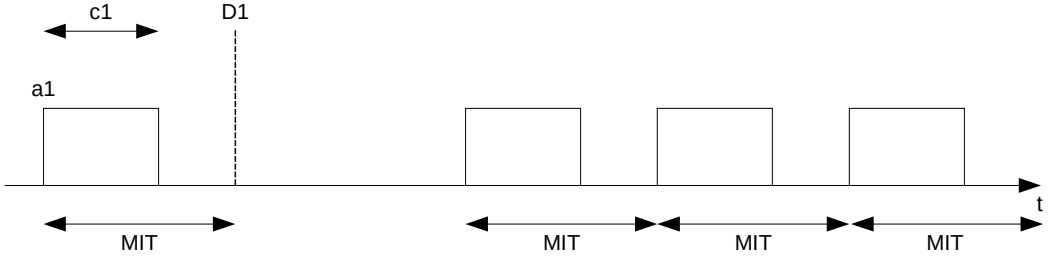


Figura 3.3 Escalonamento de tarefas esporádicas

Outro aspecto relacionado com a execução das tarefas tem um aspecto mais prático, devendo-se ao facto de existir um intervalo de tempo entre o instante onde a tarefa deveria de ter começado a sua execução e onde começa realmente [8].

Este atraso na execução de uma tarefa (*jitter*) prende-se com o processo de decisão do escalonador, com o tratamento de interrupções que surgem, etc.

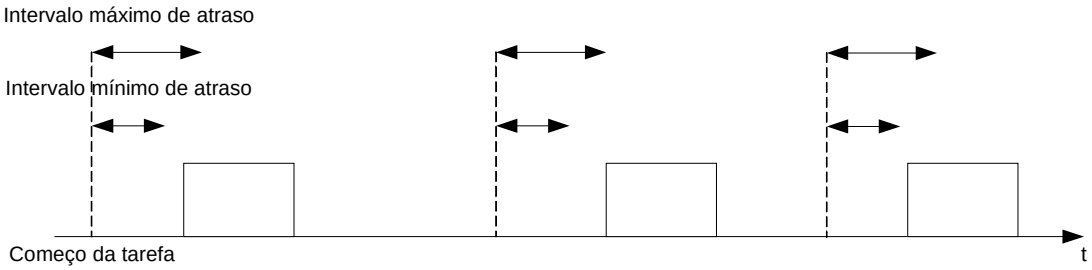

Figura 3.4 Escalonamento das tarefas na presença de *jitter*

## 3.3 Escalonamento

O escalonamento é o mecanismo pelo qual o núcleo do sistema operativo determina qual a tarefa que deve ser executada. O seu funcionamento de forma correcta e determinista é crucial para que as metas temporais das diversas tarefas sejam cumpridas. Um sistema com um escalonador errado pode resultar no não cumprimento de metas temporais, mesmo com o dobro da capacidade de processamento do que um com um escalonador correcto. Existem três parâmetros que podem intervir no processo de decisão de qual a tarefa a executar [6]:

1. Tempo
2. Peso das Tarefas
3. Prioridade das Tarefas

No primeiro parâmetro encontramos escalonadores *Clock-Driven*. São escolhidos instantes de tempo nos quais é feita a preempção de uma tarefa para a seguinte. O processo de decisão de qual a próxima tarefa a ser executada é realizado estaticamente, mas a informação sobre a escala respectiva tem de estar acessível em tempo de execução.

O peso das tarefas é utilizado pelos escalonadores *Weighted Round-Robin* em que a cada tarefa é atribuída um peso que corresponde a uma fatia temporal que pode usar o processador. Este tipo de escalonadores é utilizado em tráfego de comunicações, particularmente em *high-speed switches* de tempo real baseados em *Round-Robin*.

Por último, a classe mais divulgada e usada de escalonadores baseia-se na prioridade de cada tarefa, onde a tarefa de maior prioridade disponível se encontra em execução. Existem vários escalonadores que, sobre certas condições, conseguem garantir o cumprimento das metas temporais, entre os quais se destacam o RMS (*Rate Monotonic Scheduler*), DMS (*Deadline Monotonic Scheduler)*, EDF (*Earliest Deadline First*), MLF (*Minimum Laxity First*) entre outros. As condições necessárias que todos necessitam para que sejam cumpridas as metas são

1. Tarefas independentes – não comunicam entre si nem partilham recursos
2. Tarefas Periódicas
3. Metas temporais coincidentes com o período ($D_i = T_i$)
4. Escalonador preemptivo (se uma tarefa de maior prioridade se tornar executável, interrompe a que ocupa o processador)
5. Tempo máximo de execução conhecido e constante ($c_i$)
6. Tempo de
   - comutação entre tarefas nulo
   - envio de mensagens de CPU para CPU nulo (ou apenas um CPU)

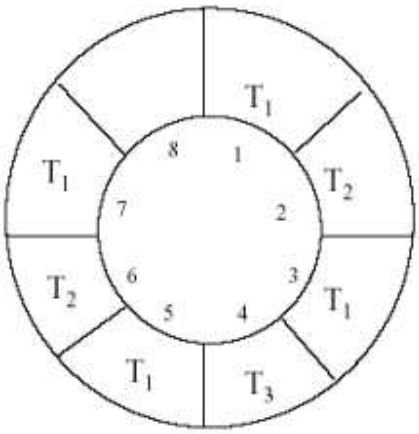

Figura 3.5 – Escalonamento *Clock-Driven*

Caso alguma destas condições não se verifique, os resultados que se seguem não são garantidos. Na última parte desta secção é abordado como garantir as metas com o relaxamento de algumas destas condições.

O primeiro escalonador, RMS [9], é usado em sistemas periódicos ou que lhe sejam redutíveis (com *MIT* pelo menos igual ao período). Baseia-se unicamente no período de cada tarefa para lhe atribuir a prioridade. É, desta forma, um escalonador estático. A prioridade de uma tarefa é atribuída de forma inversa ao seu período, tendo a de menor período uma maior prioridade. Para que o escalonamento garanta que todas as metas temporais são cumpridas é necessário que a condição,

$$\sum_{i=1}^{N} \frac{C_i}{T_i} \leq N\left(2^{1/N} - 1\right) \tag{3.1}$$

onde $N$ representa o número máximo de tarefas de tempo real, seja cumprida. O membro esquerdo da equação estabelece a percentagem de utilização do processador. Quando $N \to \infty$, o membro direito tende para um valor finito, 0.69, sendo portanto garantido que qualquer aplicação que utilize menos de 69% do tempo do processador é escalonável. É possível provar que este escalonador é óptimo no sentido em que, se não conseguir cumprir as metas, então nenhum outro escalonador estático o consegue. A condição apresentada é suficiente mas não necessária para que o escalonador garanta que as metas são cumpridas. Existem outros resultados que estabelecem condições necessárias, sendo a mais óbvia a de que o membro direito não pode ultrapassar a unidade, o que é o mesmo que dizer que o processador não pode estar ocupado mais do que 100% do tempo.

Ao contrário do RMS, o escalonador EDF é um algoritmo dinâmico [10], isto é, estabelece a prioridade imediatamente antes da execução da tarefa. Este método altera em tempo de execução a prioridade das tarefas com base na próxima meta temporal a ser cumprida. A tarefa com meta temporal mais próxima possui maior prioridade que as restantes. Este algoritmo permite uma restrição mais suave que os anteriores, bastando que a percentagem de utilização do processador não ultrapasse os 100%

$$\sum_{i=1}^{N} \frac{c_i}{T_i} < 1 \tag{3.2}$$

É, desta forma, considerado um escalonador dinâmico óptimo. Este escalonador tem, no entanto, a desvantagem de que se uma tarefa não cumprir uma meta temporal (por acções externas e imprevistas), todas as outras tarefas podem também não cumprir as suas metas.

O MLF também é um escalonador dinâmico, mas baseia-se na diferença de tempo entre a próxima meta temporal ($D_i$) e o tempo de processamento máximo ($c_i$) [11]

$$l_i = D_i - c_i \tag{3.3}$$

Quanto menor for $l_i$, maior será a prioridade da tarefa *i*. Deste modo é possível determinar se uma dada tarefa consegue cumprir a sua meta temporal ($l_i > 0$) e, em caso negativo, pode-se associar um mecanismo para que não prejudique as outras tarefas. Este escalonador possui características semelhantes ao EDF mas, em vez de partir da informação da próxima meta temporal, parte do tempo restante para a meta temporal, que fornece mais informação. A condição necessária e suficiente corresponde à mesma do EDF, equação 3.2.

Outros algoritmos de escalonamento como o MUF (*Maximum Urgency First*), que se baseia em métodos híbridos, com componentes estáticas e dinâmicas, podem ser encontrados na literatura [12]. As vantagens de sistemas multiprocessador também são exploradas por alguns escalonadores, que estabelecem em que processador é que as tarefas devem de ser executadas [13].

São necessárias várias restrições para que os algoritmos descritos na secção anterior sejam possíveis. No entanto, a vida real proporciona exemplos em que tais condições não são fornecidas.

No caso em que as tarefas tenham acesso aos mesmos recursos, ou seja, não são independentes – primeira condição –, é necessário a utilização de semáforos para assegurar a exclusão mútua. O escalonamento torna-se complexo devido à espera entre tarefas por uma região crítica, podendo mesmo, se não forem tomadas as devidas precauções, causar o não cumprimento das metas temporais, como foi o caso da *Mars Path Finder*. Neste sistema existiam três tarefas de diferentes prioridades. A de mais baixa prioridade acedia a uma região de exclusão mútua, deixando a de alta prioridade bloqueada. Enquanto procedia pela região de exclusão, foi preemptada por uma tarefa de média prioridade que executava durante um longo período de tempo, resultando no bloqueio da de alta prioridade pela de média. Este bloqueio da tarefa de alta prioridade pela de média designa-se por inversão de prioridade. Este resultado proporcionou um quebra-cabeças, que acabou por ser solucionado por mecanismos de herança de prioridade [14], que resulta na subida de prioridade da tarefa que detêm o recurso até à mais alta que o deseja aceder. Outro algoritmo obtém certas vantagens em termos de computação: “tecto” de prioridade (*priority ceiling*) [15]. A prioridade da tarefa é elevada até à tarefa de maior prioridade que possa vir a desejar o recurso. O escalonamento deste tipo de aplicações pode tomar complexidade não polinomial, ou seja, o tempo necessário para encontrar um escalonador que garanta que as metas temporais são cumpridas cresce com uma ordem superior a qualquer polinómio, como por exemplo, exponencial [6]. Em geral tenta-se encontrar uma condição necessária que, caso não seja verificada, demonstra que o escalonamento não é possível. Caso contrário, estabelece-se uma prova suficiente que, se cumprida, é garantido o cumprimento das metas.

Para sistemas esporádicos em vez de periódicos, o processo mais usual do seu tratamento é "torná-los periódicos", respeitando as condições relativas a cada escalonador, mas com o período dado pelo *MIT* – Figura 3.6. Em sistemas aperiódicos, o cumprimento das metas temporais é impossível de garantir dado que *MIT* = 0. Dado que estes sistemas não são *hard real time*, uma abordagem *best-effort* pode ser aplicada [6].

Para sistemas com metas temporais inferiores ao período – terceira condição $(D_i < T_i)$, é possível garantir as metas alterando artificialmente o período para igualar a meta temporal $(T_i = D_i)$. A atribuição da prioridade é realizada a partir do novo período.

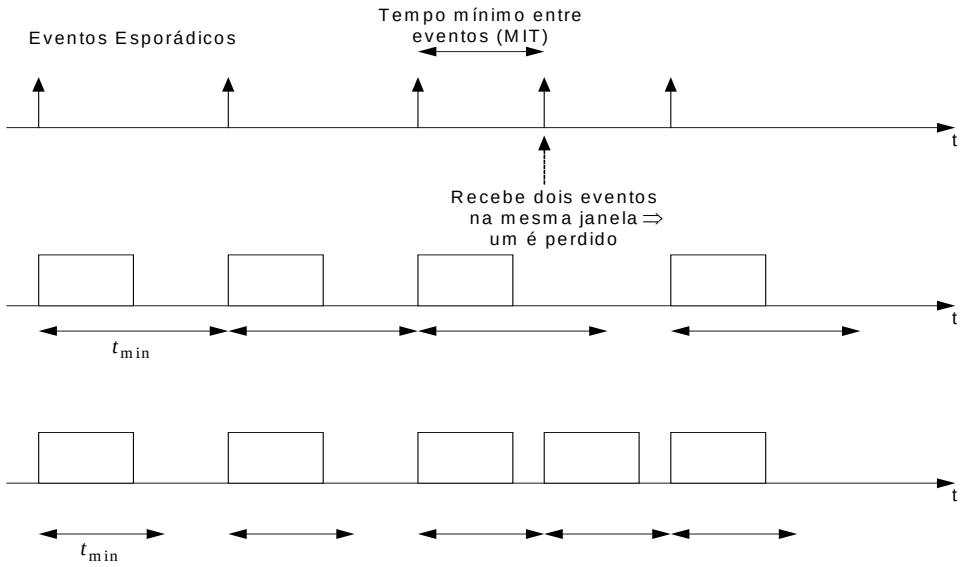


Figura 3.6 – Tratamento de eventos esporádicos em dois casos. No primeiro caso $t_{min} > MIT$ e no segundo $t_{min} = MIT$

Por último, relaxando a condição que obriga a utilização de escalonadores preemptivos, existem versões alternativas do estabelecimento de garantias dos algoritmos referidos para garantir metas temporais, mas tais restrições não são abordadas neste trabalho. Actualmente, a maioria dos sistemas operativos de tempo real fornece um escalonador preemptivo [16].

Como sistema operativo de tempo real utilizado no presente trabalho, é apresentado o RTEMS, incorporado em diversas aplicações de tempo real.

## 3.4 Sistema Operativo de Tempo Real – RTEMS

O RTEMS constitui o Sistema Operativo de Tempo Real base deste trabalho. Foi criado sob patrocínio do Departamento de Defesa dos EUA tendo em vista aplicações de Tempo Real para Controlo de Mísseis, razão pela qual foi originalmente chamado de *Real Time Executive for Missil Systems*. Cedo se percebeu que o interesse na sua utilização se estendia a um leque mais abrangente de aplicações pelo que o seu nome foi alterado para *Real Time Executive for Millitary Systems*. Finalmente, após o Departamento de Defesa dos EUA ter tornado o sistema público e *open-source*, o seu nome tomou a sua forma actual, *Real Time Executive for Multiprocessor*

*Systems*. O RTEMS é actualmente mantido pela *OAR Corporation* que coordena os esforços internos de desenvolvimento com os produzidos por uma vasta comunidade de utilizadores. A OAR também fornece apoio a aplicações comerciais sobre RTEMS.

O RTEMS foi desenhado para sistemas embebidos de tempo real, cujos recursos de memória, dispositivos, capacidade de processamento, etc., são moderadamente limitados. Para isso, possui uma arquitectura que permite definir quais os recursos (quer a nível de gestores específicos, como das bibliotecas disponíveis, como por exemplo interface *Posix*, emulação *Itron*, pilha de protocolos TCP/IP, etc.) que a aplicação necessita de modo a incluir apenas o código necessário. Usando esta filosofia de atribuição estática dos recursos, permite também definir quais os gestores de dispositivos (*device drivers*) que se deseja incorporar [17].

O RTEMS possui suporte para uma grande variedade de sistemas alvo, tais como a família *Intel* 80386 (e acima), *Intel* 80960, *sparc*, *powerpc*, AMD, Motorola MC68xxx, entre outros.

O RTEMS utiliza um escalonador baseado em prioridades, *event-driven*, e que permite definir se as tarefas são preemptivas ou não.

Tal como o seu nome (actual) indica, incorpora mecanismos que tratam sistemas multiprocessador tanto homogéneos como heterogéneos.

A arquitectura do RTEMS (Figura 3.7) incorpora então um conjunto muito diversificado de gestores que incluem entre outros os gestores típicos dos núcleos de sistemas operativo, como por exemplo: escalonamento de tarefas, sincronização e comunicação entre tarefas, gestão de memória.

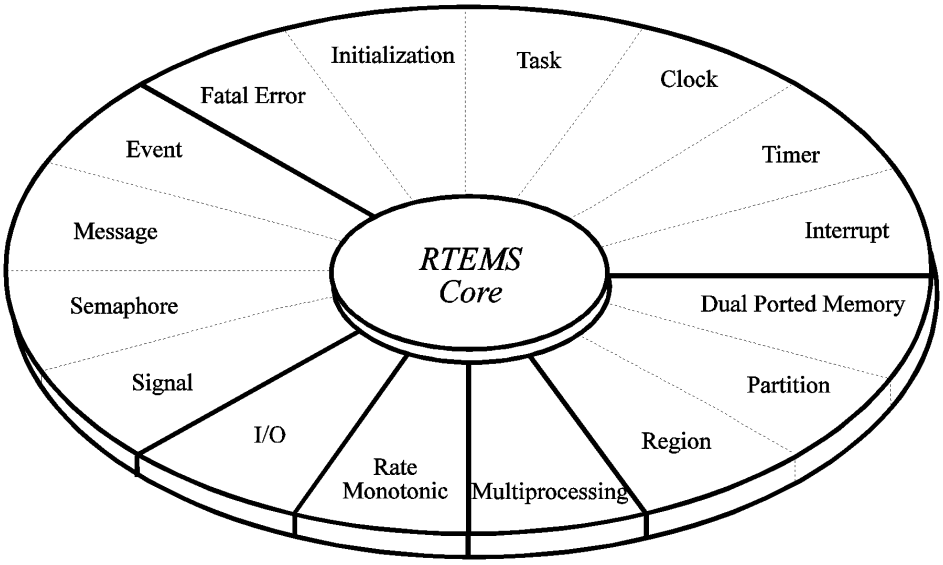


Figura 3.7 – Gestores do RTEMS

As funções fundamentais proporcionadas pelo núcleo de sistema operativo RTEMS incluem então no que diz respeito à comunicação e sincronização entre tarefas:

- **Message Queue** – suporta comunicação de dados bem como sincronização.
- **Sempahore** – suporta exclusão mútua através da sincronização do acesso a um ou mais recursos partilhados. Semáforos binários podem utilizar algoritmos de herança de prioridade para evitar o problema da inversão de prioridade. Também existe a

possibilidade da implementação de semáforos não binários, úteis em alguns cenários, como por exemplo, o produtor/consumidor.

- **Event** – sincronização através de associações isoladas ou agregadas, com eficiência superior à da *Message Queue*, mas sem transferência de dados.
- **Signal** – suporta comunicação assíncrona. É tipicamente usada no tratamento de excepções.

O desenvolvimento de aplicações de tempo real requer um medição da passagem do tempo. A unidade básica do tempo no RTEMS é conhecida como *tick*. A freqüência dos *ticks* é dependente da aplicação e determina a exactidão e resolução dos intervalos definidos e do calendário interno. O gestor **Clock** actualiza o número de *ticks* passados, através de uma função que é chamada dentro de uma rotina de serviço de interrupção (*interrupt service routine* – ISR) do dispositivo de relógio e fornece um calendário.

Através da medição dos *ticks* ocorridos, o sistema é capaz de suportar funções que implementam uma medida da passagem do tempo, como por exemplo as esperas temporizadas, *timeslicing*[1], temporizadores (**Timer**) e escalonamento periódico de tarefas (**Rate Monotonic**).

A gestão da memória é agrupada em duas classes:

- Alocação dinâmica de memória
- Tradução de endereços

A primeira é concretizada através dos gestores **Region** e **Partition**, sendo a única diferença entre eles a dimensão fixa ou variável do segmento de memória alocado. A tradução de endereços é concretizada através do gestor **Dual Ported Memory** e é apenas usada em sistemas multiprocessador, onde as aplicações partilham memória com diversos CPU (ou um dispositivo de entrada/saída "inteligente").

A incorporação de forma modular de operações de entrada/saída permite integrar os dispositivos através de uma interface bem definida: *device driver*. O gestor de **I/O** estabelece esta interface e permite realizar pesquisas de dispositivos no sistema através do seu nome (tal como em *Linux*).

Qualquer sistema de tempo real necessita de um mecanismo de resposta rápida a interrupções geradas externamente de modo a satisfazer as restrições temporais. O gestor **Interrupt** do RTEMS permite este tipo de resposta rápida, realizando o processamento crítico do escalonamento da próxima tarefa que deve executar, permitindo que a tarefa em execução seja preemptada caso a ISR provoque que uma tarefa de prioridade mais alta se torne executável.

O gestor permite que a aplicação ligue (indirectamente) uma função ao vector de interrupção de *hardware*. Quando uma interrupção ocorre, o processador chama automaticamente uma função do RTEMS (ISR interna do RTEMS) que salvaguarda o contexto do *hardware* que poderá ser modificado pela rotina da aplicação e que não foi automaticamente salvaguardado pelo processador. De seguida, invoca a ISR da aplicação. Após o seu processamento, é completado o tratamento do RTEMS, que determina se uma tarefa de mais alta prioridade se tornou executável.

[1] Tempo definido para cada tarefa processar antes de ser preemptada por outra tarefa de igual prioridade.

### 3.4.1 Desempenho do Sistema

Em sistemas operativos de tempo real, um dos parâmetros mais importantes consiste no tempo de execução de algumas das suas primitivas ou de determinadas acções, como por exemplo a latência no atendimento de interrupções ou os limites temporais de preempção de tarefas. Sendo o sistema alvo um processador Intel 486 de 16 Mhz, encontra-se na literatura as medições temporais para uma arquitectura semelhante [18].

Para uma plataforma i386 com 16 Mhz de processamento e um co-processador numérico i80387, foram efectuados testes para medir tempos máximos de comutação entre tarefas[1], regiões críticas[2] e latência de interrupções[3]. Com tarefas possuindo vírgula flutuante (*float-point*), o máximo tempo encontrado para mudanças de contexto foi de 83 μs (sem vírgula flutuante é de 34 μs, devido à não salvaguarda do contexto da FPU – *Floating Point Unit*). A maior região crítica inibe as interrupções durante um máximo de 13 μs enquanto o pior tempo medido entre a ocorrência de um evento e a salvaguarda de contexto feita pelo RTEMS foi de 12 μs, levando a uma latência de interrupções de no máximo $13+12=25\,\mu s$.

Tal como foi referido anteriormente, o RTEMS suporta apenas a medição do tempo com a resolução de um *clock tick*. Embora esta variável seja definida pela aplicação, é em geral muito grande para medições na casa dos microsegundos. Caso o programador coloque, por exemplo, um *tick* de 1 μs, o sistema fica inundado com interrupções do *Timer* interno.

Nas medições temporais para os piores casos apresentados, é utilizado *hardware* especializado para determinar o tempo em que ocorrem as instruções relacionadas com as tarefas ou interrupções. Não tendo o sistema alvo estas ferramentas, para aumentar a resolução temporal implementou-se uma função auxiliar que mede a contagem interna do Timer0 da placa pc386. Esta função fornece a granularidade de 1 μs. Para não contabilizar o próprio tempo de duração da função que lê o tempo, determinou-se a sua duração e a cada leitura subtraiu-se a sua duração.

### 3.4.2 Comparação do RTEMS com outros Núcleos de Sistema Operativo de Tempo Real

Para se ter uma ideia das vantagens/inconvenientes do RTEMS, é necessário uma comparação com outros sistemas operativos de tempo real populares. A prevalência da escolha do RTEMS sobre os outros sistemas operativos é mostrada, através de uma comparação com documentação existente, funcionalidades e tempos de comutação entre tarefas e latência de interrupções. Como sistemas operativos de comparação, apresentam-se o RT-Linux e VxWorks.

O RTEMS e RT-Linux têm uma documentação extensa e disponível em relação à sua operação, funcionalidade, desenho e implementação. O VxWorks apresenta uma documentação acerca da sua produção, configuração e testes [19]. Utilizadores do mundo industrial tendem a

[1] Intervalo de tempo entre a execução da última instrução de uma tarefa e o começo da primeira instrução da próxima tarefa. Inclui o tempo que o escalonador demora a decidir qual a tarefa a correr, tempo de salvaguarda do contexto da tarefa que libertou o processador e reposição do contexto da tarefa que irá ser executada

[2] Segmentos de código onde as interrupções são inibidas

[3] Tempo entre a ocorrência do evento (por exemplo, interrupção por hardware) até à execução da primeira instrução da ISR. Inclui o tempo necessário ao processador para associar o vector à interrupção e o tempo extra do SO no início de cada ISR para salvaguarda de contexto.

utilizar o VxWorks devido à apresentação de dados estatísticos sobre a seu desempenho e aplicações específicas do produto. Por outro lado, pesquisadores ligados ao meio académico preferem os dois primeiros devido ao conhecimento do seu funcionamento interno.

Em termos de desenvolvimento, o RTEMS e RT-Linux são vastamente superiores ao VxWorks, devido a uma apresentação clara, gratuita e total do seu código interno, de forma ao programador optimizar o seu sistema, explorando o *kernel* até aos seus limites. Enquanto que os dois primeiros são ambos *open-so*urce, apenas o RTEMS é *licence free*, o que implica que pode alterar, modificar ou integrá-lo como parte de uma sistema comercial.

Em relação às funcionalidades providenciadas, o RTEMS é na maior parte dos casos superior. Possui, por exemplo, memória dinâmica, ao contrário do RT-Linux, ou algoritmos de "tecto" de prioridade ausentes no VxWorks. Uma comparação mais aprofundada pode ser obtida através de [13].

Uma comparação dos parâmetros de latência de interrupções e o tempo de comutação entre tarefas para uma mesma arquitectura encontra-se na Tabela 3.1.

| Núcleo de Sistema Operativo | Latência de Interrupções | | Comutação entre Tarefas | |
|---|---|---|---|---|
| | máximo (μs) | média+σ (μs) | máximo (μs) | média+σ (μs) |
| | *Idle System* | | | |
| RT-*Linux* | 13.5 | 1.7±0.2 | 33.1 | 8.7±0.5 |
| RTEMS | 15.1 | 1.3±0.1 | 16.4 | 2.2±0.1 |
| *VxWorks* | 13.1 | 2.0±0.2 | 19.0 | 3.1±0.3 |
| | Sistema Carregado (*Loaded System*) | | | |
| RT-*Linux* | 196.8 | 2.1±3.3 | 193.9 | 11.2±4.5 |
| RTEMS | 20.5 | 2.9±1.8 | 51.3 | 3.7±2.0 |
| *VxWorks* | 25.2 | 2.9±1.5 | 38.8 | 9.5±3.2 |

Tabela 3.1 – Tempos de comutação e de regiões críticas para vários Sistemas Operativos de Tempo Real (extraído de [19])

Esta informação foi obtida para uma arquitectura PowerPC 604 CPU (300 Mhz). Como se pode verificar, o RT-*Linux* é o Sistema Operativo com pior desempenho, caindo na casa das centenas de microsegundos, tanto no tempo necessário para começar a processar uma interrupção, como no tempo de comutação entre tarefas. O sistema VxWorks apresenta o menor tempo máximo de comutação entre tarefas (38.8 μs), mas o segundo melhor na latência das interrupções (25.2 μs). O RTEMS é, tanto no caso médio como no pior cenário, em geral superior a ambos os sistemas, apresentando numa visão global, os melhores resultados.

Como última nota, é de referir como exemplo prático, que a própria NASA optou por substituir o VxWorks, que se encontrava nos seus sistemas de tempo real, para o RTEMS, devido aos requisitos de tempo real e ao ambiente fechado que enclausura o VxWorks que torna complexo o desenvolvimento de aplicações [20].

# Sumário

Este capítulo aborda a definição e concepção e implementação de sistemas de tempo real. Diferencia entre a programação Sequencial e a Multitarefa. Enquanto que a primeira provoca que a decisão de mudança de tarefa seja realizada pelo próprio programador, a programação Multitarefa introduz o uso de um escalonador para realizar a preempção das tarefas. A introdução dos Sistemas Operativos que implementam o escalonador e outras funcionalidades, necessita de limitar as suas interferências temporais com o sistema, de modo a ser possível garantir o cumprimento de metas temporais, surgindo assim os Sistemas Operativos de Tempo Real.

Classifica os Sistemas de Tempo Real existentes:

- *hard real time*,
- *firm real time*
- *soft real time*.

Divide em classes os tipos de tarefas existentes:

- periódicas,
- esporádicas
- aperiódicas.

Apresentando os parâmetros que definem cada uma destas classes (tempo de execução, período, tempo mínimo entre eventos, etc).

Discute os escalonadores *Clock-Driven*, *Weighted Round-Robin* e os que estabelecem prioridades para cada tarefa.

Apresenta vários escalonadores baseados em prioridade – RMS, EDF, MLF, MUT – e estabelece uma análise matemática que permite, sob certas condições, garantir o cumprimento das metas temporais definidas – equações 3.1.e 3.2.

Particulariza a descrição do núcleo sistema operativo de tempo real RTEMS, que constitui a base do trabalho efectuado no decorrer neste TFC, comparando-o com outros sistemas e mostrando quais as razões que conduziram à sua escolha.

# Capítulo 4

# O Gestor de Janelas VITRAL

O VITRAL é um gestor de janelas em modo texto policromático, construído para aumentar tanto a qualidade de apresentação dos dados da aplicação ao operador, como a transparência de desenvolvimento para o programador – Figura 4.1.

Figura 4.1 – Aspecto do VITRAL

Em sistemas multitarefa, a consola original é muitas vezes insuficiente para disponibilizar os dados de cada tarefa de uma forma clara e organizada ao operador. A escrita para o monitor, sendo comum às várias tarefas, constitui uma fonte de conflitos de acesso, sendo também objecto de preocupação adicional para o programador. Mais ainda, se várias tarefas fizerem operações de leitura do teclado, é impossível, *a priori*, prever qual a tarefa que vai obter a tecla premida[1]. É desejável a implementação de um sistema de leitura/escrita da consola de modo a aumentar tanto a clareza e organização dos dados, como a suportar leitura simultânea por parte de várias tarefas, sem adicionar conflitos que necessitem de ser resolvidos pelo programador.

A construção do VITRAL teve como base uma versão preliminar na qual foram desenvolvidos alguns conceitos fundamentais, mas cuja concretização revelou deficiências graves quer no domínio da portabilidade (mesmo entre diferentes versões do mesmo sistema) quer na preservação das garantias de tempo-real do sistema onde se integrava [21].

[1] Excepto em sistemas de tempo real e onde cada tarefa possui uma prioridade diferente. Neste caso, é normalmente possível dizer que a tarefa de mais alta prioridade receberá a tecla premida.

## 4.1 Conceitos Fundamentais

Esta secção inicia a discussão do VITRAL um gestor de janelas modo texto, apresentando quais os seus conceitos fundamentais.

- Coordenada - cada posição no monitor é caracterizada por uma coordenada de duas dimensões $(x, y) \in [0; C_{\max} - 1] \times [0; L_{\max} - 1]$, onde $C_{max}$ e $L_{max}$ correspondem ao número máximo de colunas e linhas respectivamente.

- Célula – cada uma das unidades atómicas correspondente a um caracter e correspondentes atributos (cor, cor de fundo, brilho). Cada célula é identificada por uma coordenada.

- Tela – corresponde à totalidade das células. Constitui um rectângulo com dimensões $C_{max}$ por $L_{max}$, sendo definido pelo conjunto de coordenadas $\{(x, y) \in [0; C_{\max} - 1] \times [0; L_{\max} - 1]\}$.

- Cursor – indicador da célula onde se está a fazer o eco, no dispositivo de saída, dos caracteres encontrados no dispositivo de entrada da consola. O cursor tem em geral o aspecto de um bloco ou sublinhado intermitente.

- Janela Base – A janela base representa uma sub-região da tela. É constituída pelo conjunto de células (*x*,*y*) tais que $\{(x, y) \in [pc, pl] \times [uc, ul]\}$, definindo um rectângulo, onde os dois pontos limitadores do conjunto representam vértices opostos, sendo

  - pc – Primeira Coluna
  - pl – Primeira Linha
  - uc – Última Coluna
  - ul – Última Linha

  A aplicação apresenta os seus dados através das janelas base.

- Janela – uma janela é constituída por duas janelas base. A primeira janela base apresenta somente o título – Figura 4.2. A segunda representa a zona principal de visualização (corpo). O seu funcionamento é semelhante a um terminal em *Linux*. Uma tarefa ao associar-se a uma janela, faz com que a escrita dos seus dados seja efectuada na janela base do corpo. A Figura 4.3 apresenta uma disposição de várias janelas na tela.

- Janela Base Corrente – consoante as tarefas em execução mudam, também é alterada a janela base que apresenta os dados. A janela base corrente é alterada dinamicamente consoante a tarefa que se encontra em execução. Este conceito permite estabelecer de forma dinâmica qual a janela base que está a ser usada pela tarefa em "execução".

- Janela Base Activa – consiste na janela base na qual se deseja realizar a leitura do teclado. Num dado momento, apenas uma janela base pode realizar a leitura do teclado, pois caso contrário, todas as tarefas em modo de leitura apanhariam teclas de forma

indiscriminada. A escolha da janela activa é realizada pelo operador do sistema através de um conjunto de teclas especiais (*hot keys*).

- *Hot keys* – teclas "especiais" que permitem alterar a apresentação das janelas e mudar a janela base activa. Permitem associar as teclas de função F1, F2... F12 a janelas de modo a mostrar/esconder o seu conteúdo. A combinação das teclas ALT-TAB permitem mudar a janela base activa, direccionando a leitura do teclado para a janela base correspondente.

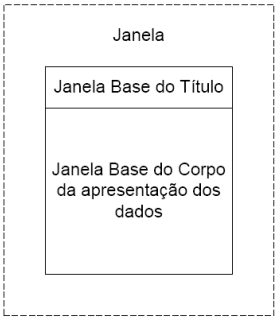


Figura 4.2 – Conceitos de Janela e Janela Base do VITRAL

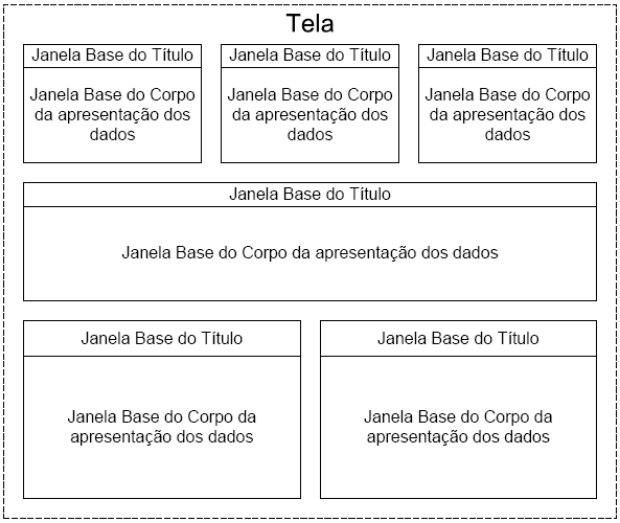


Figura 4.3 – Possível Disposição das Janelas na Tela

## 4.2 Descrição funcional

O VITRAL possui uma tarefa central, que coordena a criação de janelas e o processamento de *hot keys*. No entanto, ao contrário de outros ambientes gráficos, a apresentação dos dados através da escrita para o dispositivo de saída é realizada dentro do contexto da própria tarefa que os deseja apresentar. Esta opção apresenta várias vantagens

- Eficiência – reduz o número de comutações entre tarefas (entre a tarefa que deseja escrever e a tarefa central)

- Prioridade/urgência – cada escrita no dispositivo de saída possui um grau de urgência associado à prioridade da tarefa que esta a apresentar os dados. Com uma abordagem contendo uma tarefa central, o sistema necessita de mecanismos de comunicação que, caso não associem prioridades aos dados transmitidos, eliminam a urgência das escritas, tratando-as segundo uma ordem FIFO (*First In First Out*)
- Simplicidade – não necessita de comunicação entre tarefas para transmitir os dados que deseja apresentar

A introdução do grau de urgência associado a cada escrita é fundamental em sistemas de tempo real pois, caso não exista, uma tarefa de baixa prioridade pode inundar o sistema com pedidos e caso apareça uma mensagem urgente, como um alarme, o sistema demonstra uma grande latência na sua apresentação ao operador.

O processamento inicial dos eventos através de interrupções permite um grau de eficiência e fiabilidade não presente na leitura do dispositivo por *polling*. A ISR que apanha os eventos provenientes do teclado efectua um processamento preliminar fazendo a separação entre as teclas ditas "normais" e as *hot keys*.

Para o tratamento de *hot keys* e da criação das janelas é introduzida uma tarefa coordenadora – Figura 4.4. Cada mensagem transmitida para a tarefa coordenadora contém o tipo de conteúdo que as permite distinguir entre si e tratá-las de forma separada.

Estabelecendo a mais alta prioridade na tarefa central do VITRAL, um efeito indesejável no restante sistema é introduzido. As restantes tarefas ficam em espera pela criação de janelas e processamento de *hot keys* que, embora tenha uma frequência máxima de apenas 33 teclas por segundo, pode introduzir uma sobrecarga no processamento não desprezável[1]. Embora algumas tarefas de prioridade baixa, de *soft* ou *non real time*, possam permitir esta interferência, em geral, tarefas de *hard real time* não podem comportar este acréscimo temporal.

A descida de prioridade da tarefa central (de modo a não causar interferências devido ao processamento de *hot keys*) pode causar um problema de inversão de prioridades (similar com o da *Mars Path Finder*) – Figura 4.5. A resolução da inversão de prioridades encontra-se bem documentada na literatura, sendo constituída pela subida temporária de prioridade da tarefa central, de modo a que uma tarefa de média prioridade não interfira no processo de criação de janela e consequentemente, não aumente o tempo de espera da tarefa de mais alta prioridade. Os dois algoritmos conhecidos de inversão de prioridade (herança e tecto de prioridade) foram já discutidos no terceiro capítulo.

O mecanismo implementado consiste na especificação da aplicação de qual a prioridade base da tarefa central do VITRAL e na subida de prioridade através do algoritmo do tecto de prioridade.

A sincronização das tarefas que criam uma janela e o processamento de *hot keys* não pode ser resolvido apenas com mecanismos de herança de prioridade. Um conjunto adicional de características de implementação garante que o processamento de hot keys não interfere com as restantes tarefas, bem como permite que várias tarefas criem janelas ao mesmo tempo.

Sempre que uma *hot key* é processada pela tarefa central, deve diminuir a sua prioridade, para que não comprometa o funcionamento das restantes tarefas. O envio de mensagens urgentes de

[1] Para processar as teclas F1, F2... F12 que mostram/escondem janelas, podem ser necessários até 12 ms, medidos na infraestrutura computacional base. Com uma destas teclas a gerar eventos a um ritmo máximo de 33 Hz, a tarefa central ocupa 40% da capacidade total do sistema (desprezando o tempo do processamento das interrupções e comutação entre tarefas).

criação de uma janela possuem prioridade superior às mensagens de *hot keys*, pois as tarefas não devem de ter ficar à espera que sejam processados todas as *hot keys* que possivelmente estejam em fila de espera, o que causaria uma interferência temporal significativa.

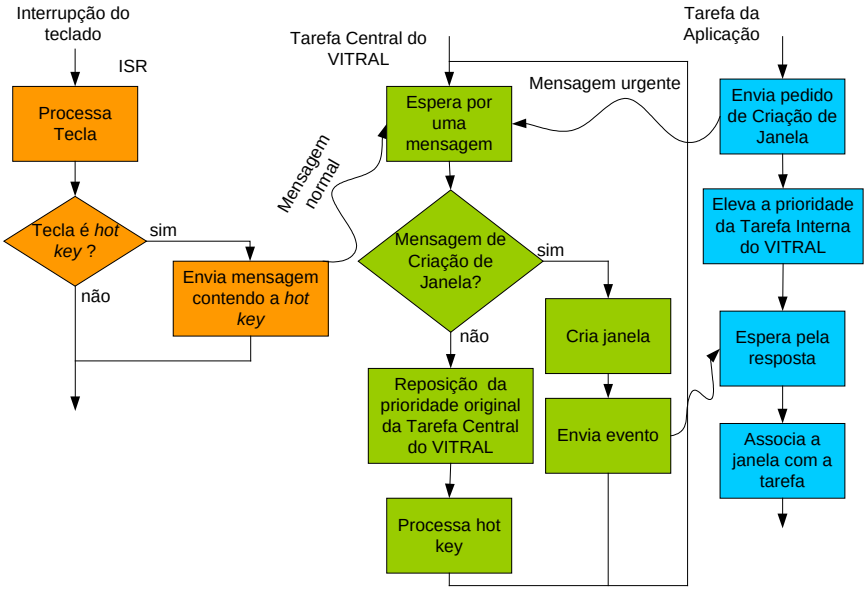


Figura 4.4 – Fluxograma do tratamento de *hot keys* e criação de janelas do VITRAL

## 4.3 Arquitectura Interna

O VITRAL substitui a consola original, mantendo a mesma interface e possibilitando que as aplicações construídas usando a biblioteca ANSI C possam ser executadas sem modificações – Figura 4.6.

A ligação entre os dispositivos de entrada/saída e o Sistema Operativo encontra-se sob a forma de um *device driver*, de forma a integrar o gestor de forma modular, através de uma interface bem definida – Figura 4.6.

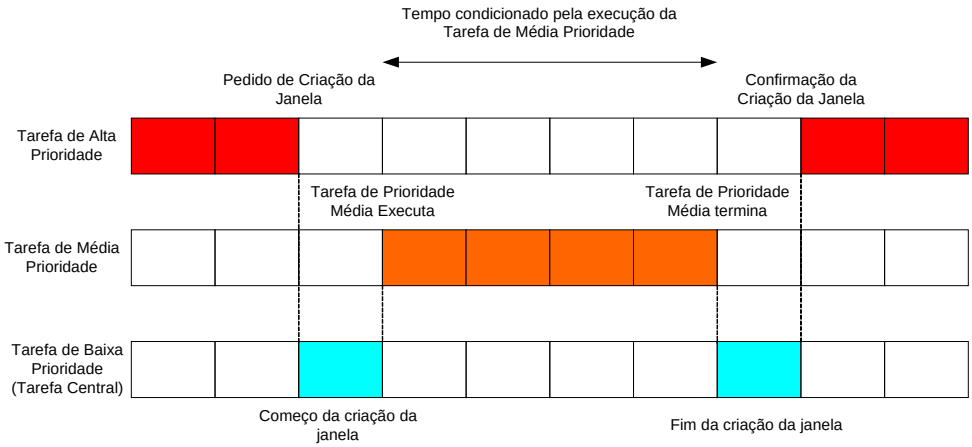


Figura 4.5 – Diagrama de Gantt exemplificando a inversão de prioridades tendo a tarefa central do VITRAL baixa prioridade

De modo a integrar o VITRAL num ambiente embebido de tempo real flexível, a aplicação deve definir certas variáveis para melhor eficiência de memória e de tempo – Figura 4.6. Entre outros, é definido qual a prioridade da tarefa central do VITRAL, o tamanho do *buffer* de entrada ou a prioridade máxima de todas as tarefas que possam criar uma janela.

A portabilidade do VITRAL foi aumentada acrescentando uma camada de *software* colocada de modo a torná-lo independente do Sistema Operativo – Figura 4.6. A portabilidade para outros adaptadores de vídeo e arquitecturas computacionais não foi abordada.

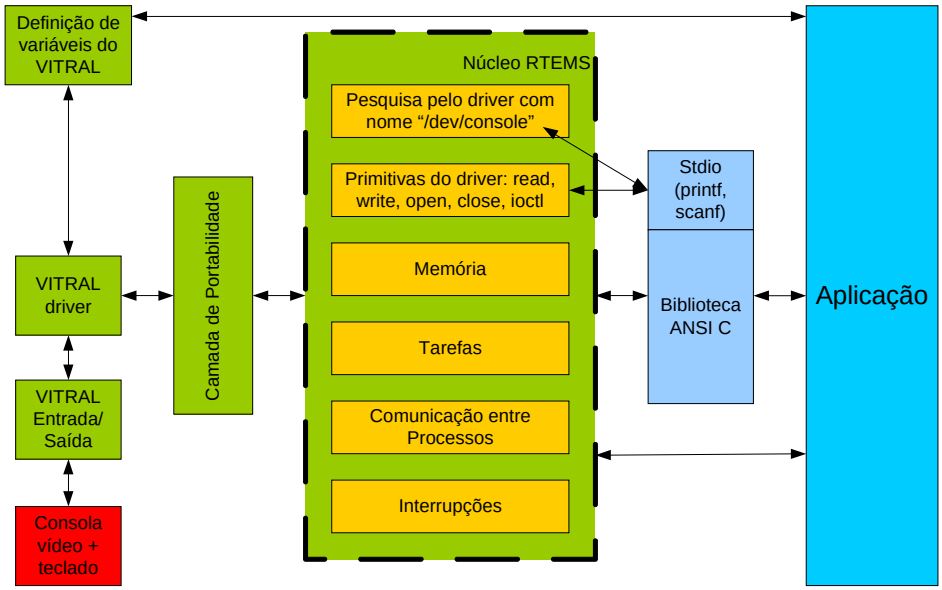


Figura 4.6 – Arquitectura do VITRAL

## 4.4 Escrita no adaptador de vídeo

A apresentação dos dados para o dispositivo de saída pode ser realizada através da manipulação directa da memória do adaptador de vídeo ou da utilização dos serviços disponibilizados pela BIOS através do INT 10h [22]. Embora o uso da BIOS apresente uma maior compatibilidade com os sistemas compatíveis com a arquitectura PC, implica uma sobrecarga temporal, devido à passagem dos modos de endereçamento do processador de protegido para real e vice-versa. O acesso directo à memória do adaptador de vídeo revela-se, deste modo, uma técnica mais rápida.

Nos adaptadores VGA ou EGA, o endereço b8000 (em hexadecimal) constitui a posição inicial de memória. Cada caracter é escrito como uma célula, constituído por dois *bytes*, o primeiro para os atributos – cor, cor de fundo e brilho – e outro para o caracter em si [22].

## 4.5 Escrita na janela base corrente

A escrita dos dados na janela base é realizada através do conhecimento da janela base corrente. O RTEMS, sistema operativo de suporte, permite estabelecer variáveis únicas para cada

tarefa, através do acesso à sua TCB (*Task Control Block*) [23]. Um campo próprio da TCB, definido como uma *Notepad Entry*, guarda a janela associada a cada tarefa. É inicializada durante a criação da tarefa recorrendo a outra facilidade do RTEMS: *Extension Manager*. Tal como foi descrito no terceiro capítulo, permite que primitivas da aplicação sejam executadas em pontos chave do escalonamento das tarefas. Durante a criação de uma tarefa, é-lhe associada uma janela base, criada logo durante a inicialização do sistema, designada fullscreen[1]. A tarefa pode alterar mais tarde a sua janela base recorrendo à *Notepad Entry*, através da interface fornecida pelo VITRAL. Quando a tarefa escreve algo para o dispositivo de saída, é pesquisado à *Notepad Entry* qual a sua janela base, correspondente à janela base corrente.

## 4.6 Leitura do dispositivo de entrada

A leitura de eventos do dispositivo de entrada é realizada por meio de interrupções. A ISR correspondente ao teclado, caso detecte uma *hot key*, envia uma mensagem para a tarefa central do VITRAL e, caso contrário, agulha o envio da tecla para uma fila de espera, através da janela base activa – Figura 4.7. Este processo elimina esperas activas (presentes na consola original) e permite uma multiplexagem dos eventos para cada janela base.

Enviando mensagens de uma ISR para uma fila de espera contendo o caracter torna ainda possível, graças aos mecanismos internos do RTEMS, atribuir dois níveis de prioridade às mensagens. Supondo que existe uma combinação de teclas (como CTRL–C) do qual se deseja efectuar um processamento prioritário (como eliminar a tarefa), é possível enviar uma mensagem urgente para a fila de espera, ultrapassando todos os caracteres que lá estejam contidos.

Assumindo que a grande maioria das janelas base não deseja processar eventos do teclado, para diminuir o número de filas de espera é especificado pela aplicação, durante o arranque do sistema, quantas janelas base desejam fazer processamento do teclado. Quando as janelas forem criadas, é definido se desejam estar em modo de leitura e em caso afirmativo, uma fila de espera é-lhe associada. Caso a janela base activa não possua um identificador válido de uma fila de espera, a tecla é perdida. A tarefa que deseje processar uma tecla fica bloqueada na fila de espera da janela base a que está associada (a sua janela base corrente), até que uma tecla seja pressionada.

Sendo a variável que indica qual a janela base activa lida durante uma ISR e acedida/modificada pela tarefa central (durante o processamento de algumas *hot keys*), é necessário inibir as interrupções durante a sua alteração[2] de modo a não causar conflitos de acesso. Este tempo deve ser o mínimo possível, de modo a não aumentar a latência das interrupções[3]. É desejável também a actualização do cursor à medida que se varia a janela base activa.

Enquanto que a concretização tem como recurso as filas de espera, é também possível a utilização de semáforos para sincronização das tarefas eliminando também esperas activas. A utilização mais comum dos semáforos é do modo binário, onde o seu valor interno apenas pode tomar o valor nulo ou unitário. A utilização directa deste tipo de semáforos provoca um controlo

[1] Esta janela base ocupa toda a tela e permite que aplicações que não utilizem o VITRAL possam ser executadas sem alterações no seu funcionamento.

[2] É impossível estabelecer uma região de exclusão mútua com base em semáforos, pois durante a ISR não pode ser chamada a operação semaphore_wait.

[3] A medição deste resultou em 6 µs, não aumentando assim a região crítica estabelecida pelo RTEMS.

adicional de sincronização no caso de se querer um *buffer* com mais de um posição. Semáforos não binários fornecem uma solução significativamente mais simples e eficaz. Quando ocorrer uma interrupção, a ISR correspondente realiza uma operação de semaphore_signal, aumentado-lhe a contagem interna, e colocando num *buffer* (*Ring Buffer*) o caracter – Figura 4.8.

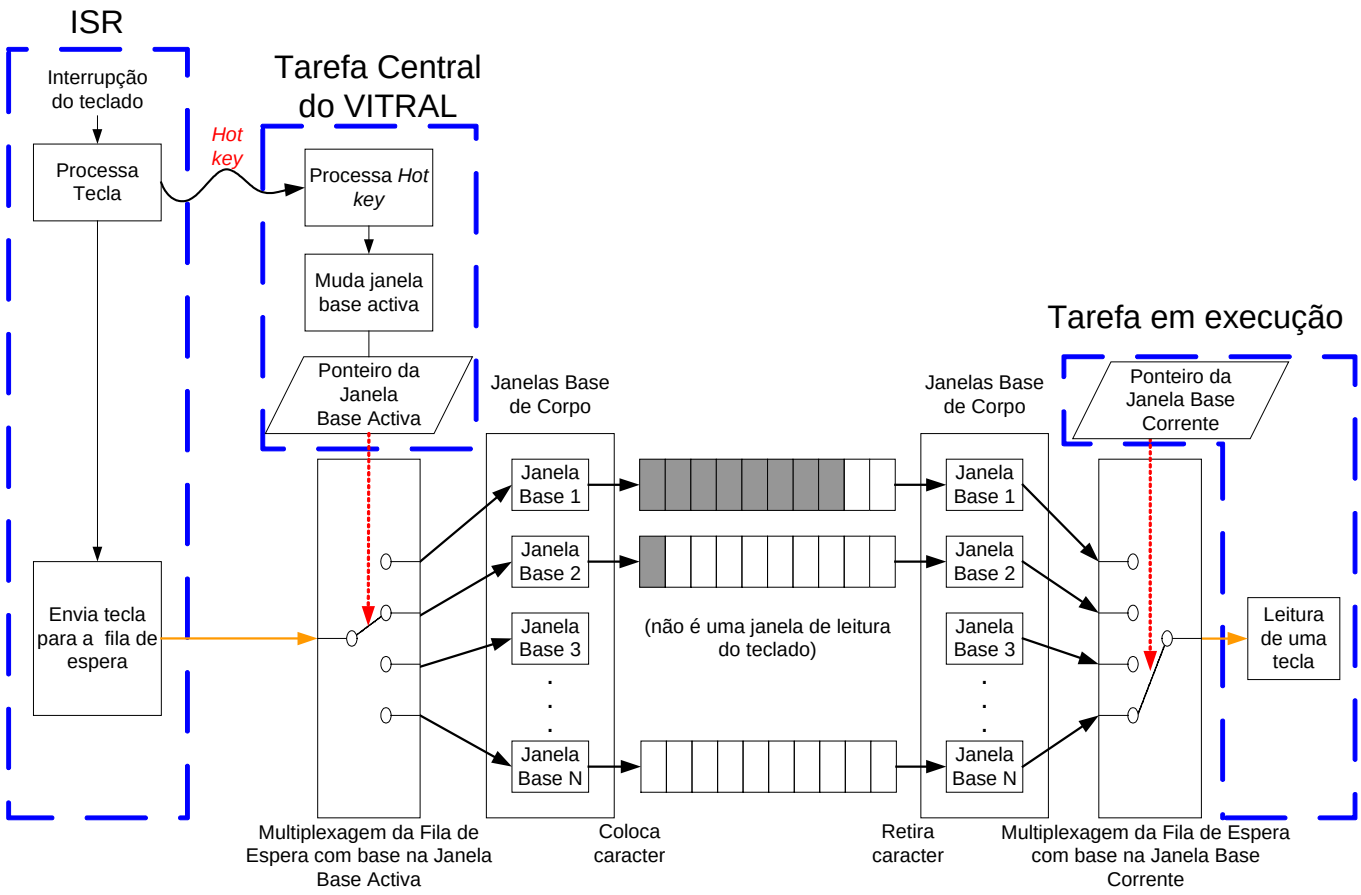


Figura 4.7 – Multiplexagem do *input* do teclado para a Fila de Espera de cada Janela base através da Janela base Activa

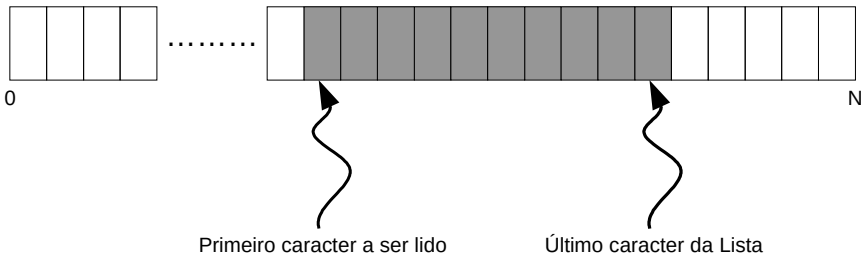


Figura 4.8 – Leitura de um Vector em Anel (*Ring Buffer*) com *N* posições e sob a forma de uma lista

A tarefa que deseja ler o teclado realiza uma operação de semaphore_wait pelo semáforo, que, enquanto existirem caracteres no *buffer*, não bloqueia. O caracter é então lido do *buffer* e o

apontador para a leitura do próximo é incrementado. Quando uma ISR ocorrer, é testado a contagem interna do semáforo e, caso seja igual à dimensão do *buffer*, o caracter não é inserido. A implementação usando semáforos é mais flexível, uma vez que é possível aceder directamente ao *buffer*, realizando um processamento que se achar adequado, como agrupar teclas (para minimizar o espaço ou detecção de uma tecla presa – *stuck key*), remover teclas repetidas, etc.

A solução escolhida, da utilização da fila de espera, embora ocupe mais espaço de memória (um semáforo ocupa 124 *bytes* enquanto que uma fila de espera ocupa 148 *bytes*) e seja menos flexível, possui uma maior facilidade de concretização e não reduz, em comparação com outras consolas, a funcionalidade. A memória do *buffer* utilizada pela fila de espera é alocada quando é criada, ocupando o mesmo espaço que o *buffer* da solução com o semáforo. É também definido pela aplicação qual a dimensão do *buffer*, para melhor controlo do espaço utilizado.

## 4.7 Funcionalidade das *Hot keys*

Os comandos possíveis definidos pelas *hot keys* são

- **Alt** – mostra qual a janela activa
- **Alt** + **Tab** – circula entre as várias janelas, modificando a janela base activa
- **Alt** + **Shift** + **Tab** – o mesmo funcionamento que a anterior mas circulando em sentido inverso
- **Alt** + **F1**, **F2** ... **F12** ou **Alt** + **Shift** + **F1**, **F2** ... **F12** – associam a janela activa com a tecla F1, F2, ... F12 pressionada
- **F1**, **F2** ... **F12** – teclas de acesso rápido. Quando pressionadas, mostram/escondem a janela que lhes está associada
- **Alt** + **Esc** ou **Alt** + **Shift** + **Esc** – mostram/escondem a janela activa

## 4.8 Comparação com outros ambientes gráficos

Ao contrário de outros ambientes gráficos mais evoluídos, o VITRAL suporta apenas a escrita de caracteres em modo policromático e em janelas não sobrepostas. No entanto, analisando outros gestores gráficos, como o Nano-X Window System[1], openGui, Qt/Embedded ou MiniGUI, verifica-se que a sua dimensão não é compatível com muitos sistemas embebidos e apenas o último aparenta ser de tempo real [24, 25, 26].

Conquanto o MiniGUI publicite que é desenhado para ambientes de tempo real, não possui, por exemplo, escrita para o monitor com a ordem estabelecida pelas prioridades das tarefas, isto é, a urgência na apresentação dos dados não é proporcional à prioridade da tarefa. A arquitectura

[1] antigamente designado Micro-Windows.

do MiniGUI, possui uma tarefa central que coordena as escritas para o monitor através do recebimento de mensagens de cada tarefa. Estas mensagens não possuem prioridade associada e portanto são tratadas numa ordem FIFO. Com uma tarefa de baixa prioridade continuamente a escrever para o ecrã, o sistema fica rapidamente sobrecarregado e as mensagens urgentes são apresentadas com grande latência.

Outros ambientes gráficos, como o NanoWindows, para além de não serem desenhados para sistemas de tempo real, possuem uma grande latência temporal devido aos mecanismos usados internamente (comunicação entre o cliente/servidor X-Windows através de *sockets*, que realizam acessos ao *kernel*, produzindo uma grande ineficiência).

## Sumário

Esta capítulo apresenta o gestor de janelas em modo texto policromático, VITRAL. Este gestor surge como alternativa à consola original (semelhante a um terminal em *Linux*), que apresenta os dados em sistemas multitarefa, de uma maneira geral, com pouca clareza e capacidade de organização.

Refere os conceitos fundamentais em gestores de janelas, definindo e caracterizando os operadores envolvidos.

Aborda a sua integração no núcleo de sistema operativo RTEMS, explorando os seus mecanismos internos. Apresenta a descrição funcional do VITRAL, introduzindo os elementos activos e passivos e a sua interligação, de modo a que as características de tempo real não sejam comprometidas.

Descreve a sua arquitectura interna, com ênfase na portabilidade para outros sistemas operativos.

Por fim, compara o gestor VITRAL com outros ambientes gráficos, concluindo-se que possui características de tempo real únicas, embora as suas funcionalidades não sejam tão evoluídas.

Capítulo 5

# Integração Modular de Dispositivos de Entrada/Saída

Os mecanismos de entrada/saída nos sistemas computacionais constituem o meio de ligação entre a aplicação e o mundo exterior. A diversidade de dispositivos existentes introduz a discussão de como integrá-los de forma uniforme e modular, de modo a ser compatível com vários sistemas, e escalável, para que o sistema comporte tanto uma grande diversidade como quantidade de dispositivos [5].

Enquanto que a interface entre sistema e sensores/actuadores se realiza através de estimulação eléctrica por hardware especializado, a ligação entre o núcleo de sistema operativo e a camada de hardware não é tão simples e imediata. Aplicações que utilizem um dispositivo (como o monitor/teclado), não devem ter que produzir alterações significativas para incorporar modificações relativas ao sistema computacional ou sistema operativo onde se encontram. A rivalidade entre marcas comerciais pode tornar este processo impraticável, mas algumas normas são aceites de modo a estabelecer uma camada de portabilidade subjacente.

Nos Sistemas Operativos é normalmente fornecido uma interface bem definida que permite interligar a aplicação com o dispositivo: o gestor de dispositivo (*device driver)*. Através da integração de componentes com recurso a *device drivers*, os dispositivos são integrados no sistema computacional com significativamente menos esforço, para além de o uniformizar para os vários sistemas [5, 27, 28].

## 5.1 Gestores de Dispositivos (*Device Drivers)*

Como foi dito anteriormente, vários Sistemas Operativos adoptaram a definição de uma interface bem definida na qual são mapeados os comandos da aplicação para o dispositivo. Esta interface passa pela especificação de funções mas também de parâmetros que possam identificar univocamente o dispositivo, tal como o *major* e o *minor number* discutidos de seguida, ou pela atribuição de nomes de ficheiros, como é o caso do Linux ou mesmo do RTEMS (onde é normalmente declarado com um nome, como por exemplo “/dev/press1”).

Nas restantes secções são discutidos aspectos relacionados com a interligação dos gestores de dispositivos (*device drivers*) com o sistema operativo, apresentando como exemplo prático o RTEMS.

### 5.1.2 Identificação do dispositivo

Como foi mencionado, existe a necessidade de identificar univocamente um dispositivo, de modo a que as operações lhe sejam transmitidas. Na quase totalidade dos sistemas operativos, existem dois números, o *major* e o *minor numbers*, que identificam univocamente o dispositivo, sendo que o primeiro estabelece o tipo (por exemplo a porta série) e o segundo o número (COM1 ou COM2, etc.) [28]. Enquanto que o *minor number* é utilizado apenas pelo *driver* para poder

identificar o dispositivo, o *major number* permite agulhar uma chamada a um dispositivo ao *driver* correspondente. O mesmo *major number* não pode portanto, ser igual em dois *driver*.

A atribuição do *major number* pode ser feita de duas formas [28]:

1. Estática – é determinado antes da execução do sistema, quer por convenção do próprio Sistema Operativo (como é o caso do Linux onde por exemplo as consolas virtuais e as portas série são identificadas pelo *major number* de 4) ou como no caso do RTEMS, onde o *major number* corresponde ao índice numa tabela contendo os *drivers* usados.
2. Dinâmica – é atribuído em *runtime* através de uma chamada de sistema. Nem todos os sistemas operativos permitem esta funcionalidade, pois associa a necessidade de memória secundária, nem sempre presente em núcleos de sistema operativo.

Um outro modo de identificação em tempo de execução do driver que se pretende utilizar é conseguido através da atribuição de um nome que lhe é único. Por exemplo no RTEMS, a consola utilizada no VITRAL é identificada com o nome "/dev/console", que é utilizado pela biblioteca ANSI C para pesquisar o *driver* correspondente [17]. A atribuição do nome é normalmente realizada durante a fase de inicialização do sistema.

### 5.1.3 Acesso Simultâneo ao Gestor de Dispositivos

Em sistemas multiprocessador ou onde o escalonador seja preemptivo, o acesso simultâneo de duas tarefas em escrita no *driver* na mesma posição de memória pode levar a resultados indesejáveis caso não tenham sido tomadas precauções durante no seu desenvolvimento. Imagine-se um cenário em que duas tarefas desejam adicionar dados no fim de uma lista dados. Pode existir o caso em que ambos acedem ao mesmo tempo e portanto vêem o mesmo estado da lista. Assim, a última tarefa a ser executada consegue adicionar os seus dados, mas a primeira perde-os, pois o ponteiro é sobreposto. Neste caso, a solução habitual é a criação de uma zona de exclusão mútua, onde cada tarefa acede exclusivamente à memória e assim não existem conflitos. Em sistemas com apenas um processador ou não preemptivos, este caso não é relevante visto que cada processo acede ao *driver* exclusivamente. O *driver* tem portanto que ser desenvolvido tendo em conta o ambiente onde vai ser executado.

### 5.1.4 Classes de Gestores de Dispositivos

A variedade de dispositivos existentes nem sempre permite o consenso na atribuição de uma certa classe a um dispositivo específico, no entanto, como é necessário estabelecer uma interface bem definida entre o Sistema Operativo e o *driver*, encontram-se na literatura separados em três classes principais [5,27,28]:

1. Dispositivos de carácter – esta classe é caracterizada pelo seu fluxo sequencial de informação entre a aplicação e o dispositivo.
2. Dispositivos de bloco – possuem tal como os dispositivos de carácter, o acesso em *stream* de *bytes*, com a diferença no modo como os dados são tratados internamente. Os dados são acedidos tipicamente em blocos de um *kilobyte* ou outra potência de

dois, para tratamento mais eficiente do dispositivo. São utilizados para aceder a discos rígidos, *floppy disks*, etc.

3. Dispositivos de comunicação – transfere dados por meio de transacções com outras máquinas (inclusive a própria máquina, por meio de uma interface "*loopback*"). Este tipo de dispositivo é encarregue do envio e recepção de pacotes de dados, ao contrário das outras classes, que cujo acesso é em *stream* de *bytes*. A interface com o Sistema Operativo é diferente pois em vez de leitura e escrita possui funções relativas com a transmissão de pacotes.

### 5.1.5 RTEMS I/O Manager

Nesta secção é aprofundado o tratamento do gestor de entrada/saída mencionado no terceiro capítulo sobre o núcleo de sistema operativo RTEMS. Tal como foi apresentado, este gestor apresenta uma interface bem definida que estabelece a ligação entre a aplicação e o *driver* [17].

A interface do gestor do dispositivo com a aplicação realiza-se, através deste gestor de entradas/saídas, a partir das funções

- Abertura
- Fecho
- Leitura
- Escrita
- Controlo

Estas funções são gerais para qualquer dispositivo da classe de caracter. Existe ainda mais uma função, que é chamada pelo sistema operativo durante a inicialização do sistema: Inicialização. Esta função, como o nome indica, inicia a estrutura de dados que irá utilizar e estabelece a comunicação com o dispositivo que controla.

As funções que estabelecem a interface são utilizadas indirectamente pela aplicação. A aplicação apenas chama funções do gestor de entrada/saída, que recebem como argumento o identificador (*major* e *minor numbers*) do dispositivo e o comando.

Como mecanismo de identificação de um dispositivo, o RTEMS permite a atribuição de um nome para cada periférico, tal como o Linux. A pesquisa do nome, realizada através do gestor de entrada/saída, retorna o *major* e *minor numbers* que são usados para identificar o periférico durante as chamadas ao gestor de entrada/saída. Como já foi referido, a atribuição do major number é realizada de forma estática, onde corresponde ao índice numa tabela contendo os *drivers* usados.

## 5.2 Tratamento de Eventos externos

A ocorrência de eventos, como por exemplo, um pacote recebido por uma placa de rede, pode ser detectada de duas formas: geração de uma interrupção; teste cíclico (*polling)*. A primeira consiste na mudança do processamento do CPU para uma função especializada quando

ocorre um evento. A segunda consiste na leitura periódica do estado do dispositivo e se detectar que houve alterações, detecta que ocorreu um evento.

As interrupções constituem o método mais comum, desde que o dispositivo o suporte, pois apresenta uma eficácia e rapidez normalmente superiores à segunda, sendo somente executada no momento em que um evento é despoletado [5].

O método de *polling*, ao contrário das interrupções, é executado mesmo quando não é necessário, podendo demorar, no pior caso, um intervalo de tempo equivalente ao seu período para detectar que um evento ocorreu e, se o seu período for muito grande, pode mesmo perder alguns eventos [29].

### 5.2.1 RTEMS Interrupt Manager

Nesta secção é aprofundado o tratamento do gestor de interrupções mencionado no terceiro capítulo sobre o núcleo de sistema operativo RTEMS [17].

O gestor de interrupções do RTEMS, no início do processamento de uma interrupção, é responsável pela salvaguarda do contexto do *hardware* que poderá ser modificado pela rotina e que não foi automaticamente salvaguardado pelo processador. De seguida, invoca a ISR da aplicação. Após seu processamento, é completado o tratamento do RTEMS, que determina se uma tarefa de mais alta prioridade se tornou executável, caso a ISR da aplicação tenha realizado chamadas sistema.

Caso a aplicação opte por não utilizar o gestor de interrupções (por exemplo, para possuir uma menor latência de interrupções), necessita pelo menos de garantir que a salvaguarda do contexto do *hardware* é realizada, por forma a não provocar incoerências no contexto das tarefas que podem causar consequências imprevisíveis.

## 5.3 Características de Tempo Real

Para a integração do *driver* com um sistema de tempo real não é suficiente a sua ligação através da interface estabelecida. É imprescindível também que garanta as características de tempo real fornecidas pelo sistema operativo.

Como se viu para o caso do VITRAL, a escolha da prioridade alta de uma tarefa coordenadora de alguns eventos é necessária para não causar interferências potencialmente desastrosas na aplicação. Por outro lado, a diminuição da prioridade pode causar problemas de inversão de prioridade, levando ao bloqueamento de tarefas de alta prioridade por outras de média prioridade. A escolha dos mecanismos de sincronização e de herança de prioridade são fundamentais para garantir uma interferência nula ou bastante limitada do *driver* na aplicação.

Por outro lado, como se verificou na versão original do VITRAL, a operação de inibição das interrupções durante um intervalo de tempo necessita de ser muito bem controlado de forma a não causar a perda de eventos nem a consequente preempção de tarefas. No caso normal, a versão original apenas inibia as interrupções de num máximo 13us, mas em alguns casos especiais este tempo chegava aos 12 ms, o que aumenta em três ordens de grandeza a latência das interrupções no RTEMS. Estas operações necessitam de ser uma análise de pior caso para se garantir que não interferem com a aplicação de forma prejudicial.

Outra consequência do uso das interrupções provém de, em situações de sobrecarga de eventos, ocuparem o tempo do processador, levando a que as tarefas sejam executadas com uma grande latência. Caso este tempo adicional seja grande, o cumprimento das metas temporais pode ser comprometido. Tome-se o exemplo de uma interface Ethernet a 1 Gb/s. O ritmo máximo das interrupções é de 1 500 000 Hz. Supondo que cada interrupção demora 200 ns a ser processada (o que, na maior parte dos sistemas, é um valor optimista), apenas o processamento das interrupções ocupa 30% do tempo do processador. Este procedimento pode ser causado por um agente malicioso que transmita pacotes pela rede de tamanho mínimo para sobrecarregar os outros sistemas.

Como alternativa às interrupções, tem-se o *polling*, que embora seja lento (quando o período é alto) e ineficiente, produz garantias temporais, sendo possível determinar qual a percentagem máxima do processador que deve de utilizar e com isso basear o seu período [29].

## 5.4 Integração Modular de Dispositivos de Entrada/Saída

O tratamento de periféricos é extremamente complexo de uniformizar, devido à diversidade de dispositivos e às funcionalidades requeridas no seu tratamento. A colocação de um gestor de dispositivo (*device driver*) constitui o primeiro passo na construção de uma abordagem modular. A definição de uma interface entre o gestor de dispositivo e aplicação (fornecida através do gestor de entrada/saída do sistema operativo) uniformiza a construção de gestores para os vários sistemas. No entanto, ainda não se encontra definido como é que os componentes activos (ISRs, tarefas do gestor, tarefas da aplicação) e passivos (memória) da aplicação e gestor se interligam. Tome-se o exemplo de uma aplicação que deseja ser informada assincronamente sobre eventos provenientes de um dado dispositivo (como por exemplo, a descida ou subida de uma prensa). Este tratamento assíncrono (e outros) ainda não se encontra "standardizado".

A Figura 5.1 demonstra um exemplo do processamento assíncrono de eventos por parte da aplicação, através da recepção de interrupções directamente para o espaço da aplicação (esta Figura não contempla todas as possíveis transferências de informação entre os elementos). O gestor de interrupções (do sistema operativo), em vez de chamar apenas uma ISR, como é habitual, chamaria as ISRs do gestor e da aplicação. Outra solução consiste na colocação no gestor de interrupções de apenas a ISR do gestor, que posteriormente chamaria a ISR da aplicação.

Supondo que uma interrupção é despoletada sempre que surge um qualquer evento e que a tarefa da aplicação deseja ser notificada apenas quando ocorreram alguns eventos particulares (como uma determinada tecla). Consequentemente, a ISR da aplicação necessita de aceder ao Controlo E/S do dispositivo para verificar qual o evento que ocorreu. Esta ligação necessita ainda de ser abordada, de modo a criar uma interface uniforme e modular para este tipo de tratamento.

Caso a aplicação deseje um outro tratamento (síncrono), como a sincronização directa entre a ISR do gestor com as tarefas da aplicação, apenas necessita que a interface do sistema operativo a permita[1].

[1] No RTEMS, por exemplo, cada função do gestor recebe como argumento um ponteiro genérico, pelo que é possível realizar transferências de quaisquer elementos.

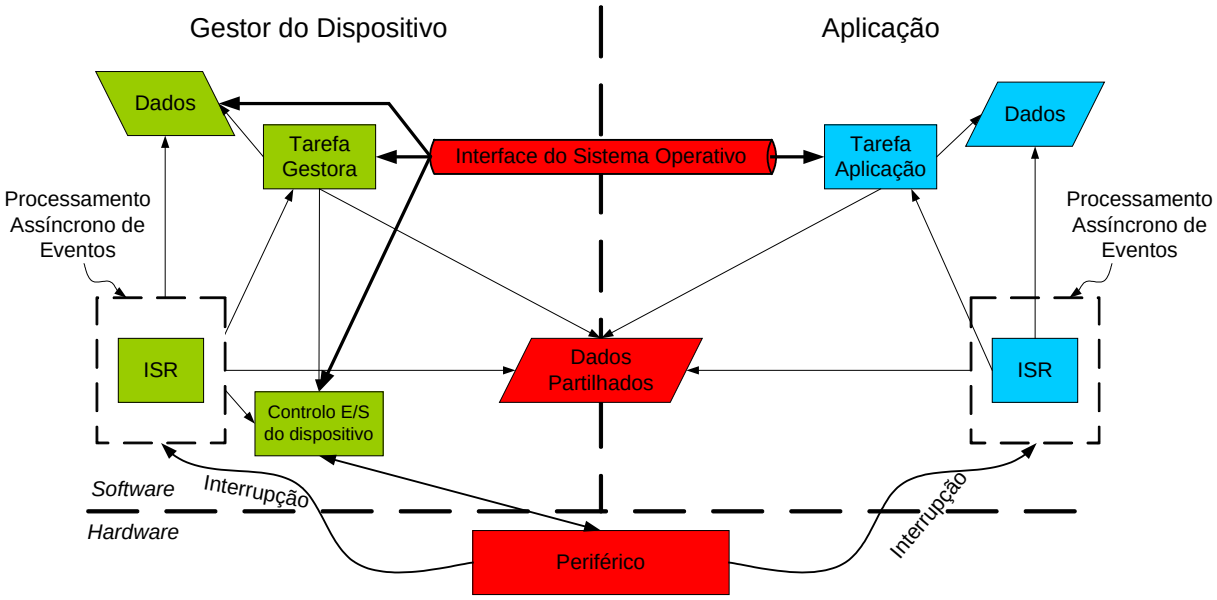


Figura 5.1 – Possível Integração Modular de dispositivos de Entrada/Saída

A Figura 5.1 introduz ainda um outro problema durante o acesso a dados partilhados entre a aplicação e a gestor do dispositivo. Os dados partilhados podem fornecer uma interface mais rápida e eficiente, mas o seu acesso em exclusividade perturba a separação entre gestor e aplicação, dado que são necessários meios, ou fornecidos pelo *hardware* ou pelo sistema operativo, que necessitam de ser comuns.

A integração de dispositivos inteligentes adiciona ainda mais problemas na sua interligação modular com o sistema operativo. Tome-se o exemplo de um braço *robot* com processador interno, capaz de executar um conjunto de acções paralelamente ao sistema computacional a que se encontra associado através de um meio de comunicação, e. g., porta série. A transferência do código/opções de configuração para o *robot* torna-se complexa caso se tome uma abordagem standard dos *device drivers*.

Complicando ainda mais o cenário e introduzindo acessos remotos (provenientes de outra máquina na rede) ao dispositivo, não deve caber à aplicação residente no nó onde se encontra o braço *robot* (por exemplo), a descodificação da mensagem e consequente tradução para o dispositivo. A sua integração deve contemplar o acesso remoto, juntando-se a um componente adicional que estabelece a ligação entre a infraestrutura de comunicação e o gestor do dispositivo. Como exemplo prático, tem-se o acesso a uma impressora remota. A sua implementação normal consiste na colocação de um servidor exterior ao gestor da impressora, o qual recebe pedidos de impressão, colocando-os numa fila de espera e implementando políticas próprias de gestão, seleccionando os pedidos de impressão de acordo com o critério de prioridades [5]. Como se verifica, existe neste exemplo uma camada intermédia entre a aplicação e o gestor da impressora, que consiste na tarefa servidora dos pedidos.

A continuação deste TFC num Mestrado Integrado abordará estes problemas segundo uma visão mais aprofundada e fundamentada, reagindo a exemplos práticos e propondo uma arquitectura que visa englobar todos os componentes activos e passivos, tanto do gestor do dispositivo como da aplicação, e associar uma camada intermédia que permite acessos remotos ao periférico.

## 5.5 Exemplo Prático – Prensa Electro-Pneumática

Para além da implementação do gestor de janelas VITRAL, também se efectuou a junção de uma placa controladora de uma prensa electro-pneumática [30] com o RTEMS – Figura 5.2.

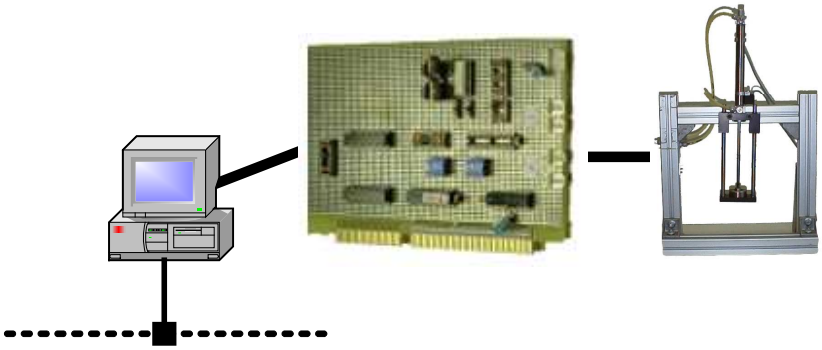

Figura 5.2 – Prensa Electro-Pneumática

Este fase do trabalho integra este dispositivo como um *device driver*, seguindo a mesma filosofia que o VITRAL e a recomendada neste capítulo.

Durante a inicialização do sistema é, conforme as características dos *drivers*, chamada a função de inicialização do *driver* da prensa. Esta função realiza operações de controlo com a placa que controla a prensa e associa-se ao nome "/dev/press".

Durante a execução da aplicação, as tarefas que desejem operar sobre a prensa devem chamar uma função do gestor de Entrada/Saída do RTEMS, que pesquisa o mesmo nome associado à prensa. É retornado o identificador do *driver* a utilizar (*major number*) que depois é utilizado para agulhar as operações de subida/descida da prensa para o driver correcto.

Dentro da classe possível de *drivers* permitida pelo RTEMS, considerou a mais adequada a classe de carácter, visto que as outras classes não correspondem à funcionalidade da prensa.

De entre as operações possíveis descritas na interface do gestor de Entrada/Saída do RTEMS[1], optou-se por implementação das operações de subida/descida da prensa através da função de controlo, dado que foi considerado inadequado a realização destas operações através das outras funções.

[1] Funções de abertura, fecho, leitura, escrita e controlo.

# Sumário

O objectivo deste capítulo consiste em fornecer um visão global da integração de dispositivos de entrada/saída em sistemas computacionais. Apresenta e descreve o gestor de dispositivo (*device driver*) como um elemento modular de integração dos dispositivos. Esta uniformidade tem uma grande vantagem dado que permite redireccionar as Entradas/Saídas através de mecanismos de comunicação.

Identifica vários factores a ter conta durante a construção de gestores de dispositivos

- Identificação unívoca do dispositivo, e. g., pesquisa pelo seu nome.
- Problemática de acesso simultâneo ao mesmo *device driver*, que pode causar incoerências na estrutura de dados/estado do dispositivo se não forem implementados mecanismos de sincronização
- Diferenciação normalizada entre os vários dispositivos, identificando uma interface própria para cada classe com o sistema computacional.
- Tratamento de eventos externos através de interrupções ou por teste cíclico (*polling)*.
- Preservação das características de tempo real, tendo em atenção os factores de
    - Sincronização entre elementos activos, que podem, se não forem tomadas as devidas precauções, criar problemas de inversão de prioridade (solucionadas pelos mecanismos de herança de prioridade) ou esperas prolongadas devido ao processamento de eventos.
    - Estabelecimento de regiões "pequenas"[1] de inibição das interrupções.
    - Impedir uma interferência temporal excessiva do tratamento das interrupções com as tarefas de tempo real.

Apresenta os gestores de Entrada/Saída e Interrupções do RTEMS para um enquadramento com os mecanismos descritos.

Aborda a integração modular e uniforme dos dispositivos de entrada/saída, tendo em conta funcionalidades acrescidas, como o tratamento assíncrono dos eventos por parte das tarefas da aplicação (em contraposição com as tarefas do gestor do dispositivo). Aborda também a necessidade de uma camada intermédia, que estabelece a ligação entre o gestor de dispositivos e o resto da aplicação, de forma a estender o uso do dispositivo a acessos remotos através da rede de comunicação.

Por último, apresenta a integração de um gestor de uma prensa electro-pneumática com o núcleo de sistema operativo RTEMS, especificando as opções tomadas e a interface estabelecida.

[1] Em comparação com as regiões que o próprio Sistema Operativo estabelece. O RTEMS possui, como pior tempo de inibição das interrupções, 13 μs, para um sistema Intel 80386 a 16 Mhz – secção 3.4.1.

# Capítulo 6

# Controlo de Sobrecargas de Interrupções

A geração de eventos é considerada assíncrona com o processamento das tarefas do processador pois, em geral, não se sabe *a priori* quando irão ocorrer. O seu processamento pode ser realizado, tal como já foi mencionado no quinto capítulo, por meio de interrupções ou através de *polling* do dispositivo. Enquanto que o primeiro é eficaz e rápido (executado somente e no momento em que é necessário), produz consequências indesejáveis para o resto do sistema pois provoca um aumento no tempo de execução das tarefas. O segundo meio, embora ineficiente[1] e lento (executado periodicamente), torna possível construir garantias temporais de modo a não prejudicar a aplicação.

O primeiro método é utilizado pela esmagadora maioria das aplicações, levando a que em muitos casos, os sistemas tornam-se extremamente frágeis em relação a *bursts* de eventos. Dado que o processamento de qualquer interrupção possui normalmente prioridade superior às tarefas em execução no CPU, é introduzido um certo grau de incerteza temporal (*jitter*) que interfere com a apresentação dos resultados. Em sistemas de tempo real, essa incerteza temporal necessita de ser limitada e conhecida de modo a não provocar o incumprimento das metas temporais. Existem normalmente no *hardware*, mecanismos que permitem estabelecer um limite mínimo de tempo entre duas interrupções de um mesmo dispositivo (Tabela 6.1), mas é muitas vezes demasiado pequeno para a aplicação em causa e, interligando vários dispositivos geradores de interrupções, só por si não chega pois podem ser geradas demasiadas interrupções numa janela temporal crítica.

Numa situação de sobrecarga de interrupções, para impedir a sua interferência com as restantes tarefas, é necessário inibi-las de modo a evitar que o tempo de execução das tarefas seja aumentado de forma ilimitada [31]. O tratamento de eventos tem que recorrer agora a um mecanismo que assegura que as metas são cumpridas, embora não tão eficiente como as interrupções.

| Dispositivo | Frequência máxima de interrupções (Hz) |
|---|---:|
| Teclado | 33 |
| Fio solto | 500 |
| *Bounce* de um interruptor | 1 300 |
| Porta série a 115 Kbps | 11 500 |
| Ethernet  a 10 Mbps | 14 880 |
| CAN bus a 1 Mbps | 15 000 |
| I2C bus | 50 000 |
| USB | 90 000 |
| Ethernet a 100 Mbps | 148 800 |
| Ethernet a 1 Gbps | 1 488 000 |

Tabela 6.1 – Ritmos de interrupções máximos de alguns dispositivos (extraído parcialmente de [31])

[1] Podendo levar a perda de eventos entre duas leituras consecutivas.

Para determinar se uma situação de sobrecarga de eventos está a decorrer, o sistema pode possuir uma tarefa especializada que, periodicamente, determina se uma interrupção foi gerada e, consoante o seu ritmo, determina se comprometem o cumprimento das metas temporais, decidindo se deve (ou não) inibi-las. Este método é ineficiente pois é executado mesmo que não tenha sido gerada uma interrupção e, se o seu período for muito alto, pode mesmo perder eventos e não contabilizar todas as interrupções ocorridas. Para além destes problemas, apenas o tempo de comutação entre tarefas pode ocupar a maior parte do processamento do CPU. Considerando que as interrupções surgem, por exemplo, com um intervalo mínimo de 1 ms, o tempo de comutação entre processos no RTEMS é de 51.3 μs e logo, um décimo do tempo de CPU (102,6 μs em 1 ms) estaria ocupado somente em comutações entre tarefas. Mais ainda, se uma sobrecarga de interrupções estiver em progresso, a tarefa demora mais tempo a ser executada e a mudança de mecanismo de detecção de eventos não é imediata.

A detecção de sobrecargas dentro das ISR (*Interrupt Service Routine*) é um método que apresenta várias vantagens. Não perde nenhuma interrupção e apenas é executado somente quando necessário, para além de que o tempo de salvaguarda e reposição do contexto numa interrupção é menor do que na comutação entre tarefas. A Figura 6.1 apresenta o esquema do processamento de interrupções com o mecanismo de detecção de sobrecarga adicionado. Verifica-se que este sistema aumenta o tempo de latência das interrupções pelo que o processo de decisão tem que obedecer a limites temporais muito restritivos. A secção 6.2 trata mais profundamente este método.

Após ter sido detectado uma situação de sobrecarga e as interrupções terem sido inibidas, para permitir uma degradação graciosa dos serviços (*graceful service degradation*) é necessário processar alguns eventos, mas de um modo mais controlado.

Adaptando o tempo em que as interrupções são permitidas, é possível processar as interrupções de modo eficiente e controlado. Este método necessita de uma tarefa que recebe informação temporal de quando deve desinibir as interrupções, enquanto que a sua inibição é realizada dentro da ISR.

Um outro método consiste no *polling* periódico do dispositivo. Uma tarefa executa-se com período e prioridade, definidos pelo programador, tais que não interfira com o resto do sistema, e verifica se algum evento foi despoletado pelo dispositivo desde a última leitura. Caso os eventos possuam de novo um ritmo normal, o sistema pode voltar ao modo de funcionamento inicial, baseado em interrupções.

## 6.1 Detecção de Sobrecargas

Sendo a detecção de eventos realizada durante o processamento de uma interrupção, o método mais simples de determinação de sobrecargas consiste em medir o tempo entre duas interrupções consecutivas e, caso seja menor que um nível definido *a priori*, a anomalia é detectada. Este algoritmo é muito frágil em relação a pequenos *bursts* de eventos, como é o caso do teclado ou de tráfego na rede de comunicação.

Outro algoritmo mais robusto consiste na determinação do valor instantâneo da taxa de interrupção e num estabelecimento de um limiar máximo que, caso seja ultrapassado, é detectada uma situação de sobrecarga. A secção 6.1.1 contém uma breve abordagem aos conceitos de

processamento de sinais discretos, apresentando dois tipos de sistemas que determinam o ritmo instantâneo das interrupções.

Na secção 6.1.2 é apresentado outro sistema que têm em conta as tarefas em execução. É portanto, mais robusto e menos intrusivo no resto do sistema, sendo, no entanto, mais difícil de implementar. A conjugação destas duas técnicas fornece um sistema ainda mais robusto.

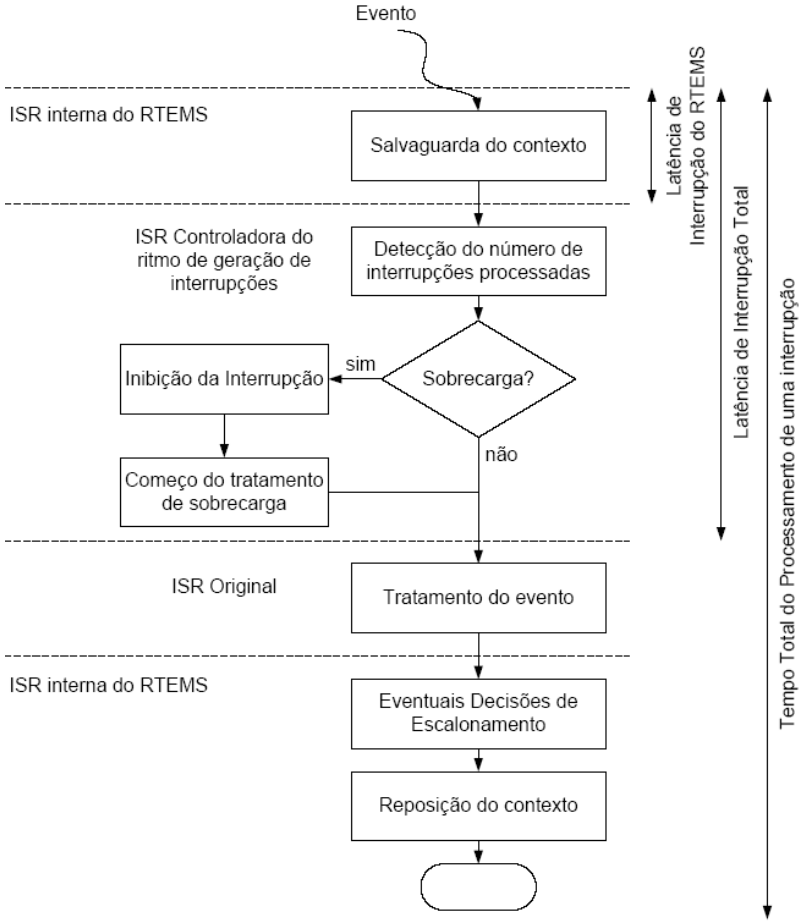


Figura 6.1 – Fluxograma do tratamento total de uma interrupção com o mecanismo de protecção

## 6.1.1 Detecção do ritmo das interrupções

Para aplicar o processamento de eventos discretos é necessário, primeiro que qualquer outro passo, discretizar o tempo [32]. Embora o ritmo do relógio interno do sistema possa ser considerado como discreto (resolução de 1 μs), numa escala macroscópica do ponto de vista do intervalo de geração de eventos, pode ser considerado como contínuo. Introduz-se assim, o conceito de discretização por *sample*/*hold*, que, estabelecido um período de amostragem ($T_{amostragem}$), verifica o estado do sistema (*sample*) e guarda-o até à próxima amostragem (*hold*). O tempo contínuo $t$ é transformado num tempo discreto $n$ através de $t = nT_{amostragem}$. Para apresentar a ocorrência de um evento em tempo discreto, é introduzido o sinal $x[n]$ (x: $\mathbb{Z} \rightarrow \{0,1\}$) definido da seguinte forma

$$x[n] = \begin{cases} 1, & \text{oc orrência de uma interrupção no instante n} \\ 0, & \text{caso contrário} \end{cases} \tag{6.1}$$

Considerando uma função que apresenta o número de eventos ocorridos $z[n]$ dada por

$$z[n] = z[n-1] + x[n] \tag{6.2}$$

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 a equação 6.1.2 numa forma não recursiva, tem-se

$$z[n] = \sum_{i=1}^{n} x[n] \tag{6.3}$$

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

$$y[n] = \frac{z[n]}{n} \tag{6.4}$$

que providencia o número médio de eventos no tempo n. Para uma forma recursiva de $y[n]$ pode-se encontrar, a partir de 6.2 e 6.4, a equação

$$y[n] = \frac{z[n]}{n} = \frac{z[n-1] + x[n]}{n} = \frac{(n-1)y[n-1] + x[n]}{n} = \left(1 - \frac{1}{n}\right) y[n-1] + \frac{1}{n} x[n] \tag{6.5}$$

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 ($D$). Este sistema é conhecido na literatura como um filtro FIR (*Finite Impulse Response*) onde $z[n]$ passa a ser dado por

$$z[n] = \sum_{i=n-D+1}^{n} x[n] \tag{6.6}$$

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

$$y[n] = \frac{z[n]}{D} = \frac{1}{D} \sum_{i=n-D+1}^{n} x[n] \tag{6.7}$$

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. O ritmo é dado por

$$y[n] = (1-\alpha)y[n-1] + \alpha x[n] \tag{6.8}$$

onde o parâmetro $\alpha$ encontra-se no intervalo ]0;1[ para que o sistema seja estável e $y[n]$ positivo. De novo, este filtro torna-se equivalente ao primeiro fazendo $\alpha$ variante com o tempo ($\alpha = 1/n$) por comparação com a equação 6.5. Dado que $\alpha \neq 0$, a pesagem 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). Com a gama de $\alpha$ considerada, apresenta-se uma simulação na Figura 6.2 com este filtro

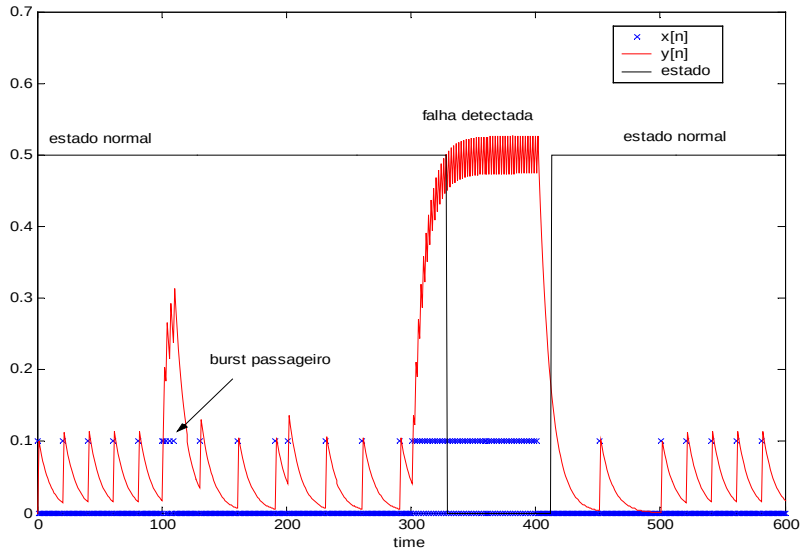


Figura 6.2 – Simulação de eventos e de detecção de falha com um filtro IIR de primeira ordem

Como se pode verificar, a saída do sistema segue o ritmo de geração dos eventos. A ocorrência de um burst temporário quando $n = 100$ demonstra a sua capacidade de suportar ritmos temporariamente muito grandes. Quando uma situação de sobrecarga ocorre (a partir de $n = 300$), $y[n]$ cresce até ao ritmo da interrupção (neste caso uma interrupção em cada dois instantes de tempo) e a sobrecarga é detectada. Neste caso estabeleceu-se um valor $M = 0.5$ para a taxa de interrupções máxima do sistema.

Nesta classe de sistemas apresentados, a actualização do estado interno é feita de uma forma tipicamente periódica. Na arquitectura apresentada, esta actualização é realizada de forma síncrona apenas com os eventos, dentro de uma ISR.

### 6.1.2 Alocação de tempo para processamento de interrupções

O modelo descrito na secção anterior consiste apenas em estabelecer um ritmo máximo das interrupções. Um mecanismo mais robusto incorpora também a informação de qual a tarefa a correr no sistema. Por exemplo, existem certas tarefas nas quais não se deseja um processamento elevado de eventos para não produzir um grande *jitter* havendo, por outro lado, tarefas não críticas às quais é permitido demorar mais tempo a ser executadas. A tarefa de *idle*, por exemplo, pode processar um número arbitrário de interrupções pois não interfere com nenhuma tarefa útil. É desejável que as tarefas de *hard real time* sejam susceptíveis de ser interrompidas por um número reduzido de interrupções, enquanto que as de *soft real time* podem processar um número maior e em *idle*, não é necessário estabelecer um limite.

Deste modo, define-se um intervalo de tempo $\Delta_i$ para o processamento de interrupções em cada tarefa de *hard real time* $J_i$. Esgotado $\Delta_i$, as interrupções são inibidas e quando outra tarefa ocupar o processador, é reposto outra vez o estado. Para as tarefas de *soft real time*, o intervalo de tempo pode ser superior, mas limitado para garantir que são processadas. O programador pode estabelecer um compromisso entre o processamento de eventos e a produção de resultados das tarefas em tempo útil. Caso as tarefas de *hard real time* necessitem de reagir rapidamente a eventos, todas as tarefas no sistema (incluindo as de baixa prioridade) têm que alocar $\Delta_i$ de forma a que possam apanhar eventos e as tarefas de alta prioridade não fiquem bloqueadas à espera das de baixa. Por último, quando nenhuma tarefa ocupar o processador, todas as interrupções são permitidas visto que não prejudicam nenhuma tarefa. Este mecanismo necessita de tantas variáveis como tarefas a ocupar o processador e um mecanismo que permita conhecer qual a tarefa que está a ocupar o processador.

## 6.2 Estabelecimento de Garantias em Sobrecargas

Sendo limitada a interferência das interrupções com as tarefas em execução, é necessário integrar o pior cenário nas condições do escalonador de modo a garantir que as metas são cumpridas. O terceiro capítulo apresentou condições pelas quais é garantido o cumprimento das metas temporais em sistemas com

- Tarefas periódicas
- Tarefas independentes – não comunicam entre si nem partilham recursos
- Escalonador preemptivo
- Metas temporais coincidentes com o período ($D_i = T_i$)
- Tempo máximo de execução conhecido ($c_i$)
- Tempo de
    - comutação entre tarefas nulo
    - envio de mensagens de CPU para CPU nulo

De acordo com as condições gerais de escalonamento e sob estas restrições, pode-se estabelecer que, na ausência de interrupções, tem-se

$$\sum_{i=1}^{N} \frac{c_i}{T_i} \leq f(N) \tag{6.9}$$

onde é introduzida a função *f*(*N*) apenas para aumentar a generalidade do tratamento[1]. É aqui apresentado que, aumentando artificialmente o tempo de execução máximo das tarefas, as condições dos escalonadores, equações 3.1 e 3.2, estabelecem as garantias suficientes para o cumprimento das metas temporais. Admitindo que o máximo tempo gasto em interrupções para cada tarefa é dado por $b_i$, a condição que contabiliza o tratamento de interrupções é

$$\sum_{i=1}^{N} \frac{c_i + b_i}{T_i} \leq f(N) \tag{6.10}$$

Repare-se que o método de alocação de tempo para processamento de interrupções (secção 6.2.2) produz logo $b_i = \Delta_i$, pelo que a condição é imediata. Para os dois métodos apresentados de determinação do ritmo das interrupções, encontra-se no Anexo C a apresentação do pior cenário[2] e a determinação analítica da interferência temporal com as tarefas.

Particularizando ainda mais para o método de alocação de tempo para processamento de interrupções, verifica-se que podem ser interligados com outros algoritmos, provenientes do âmbito da poupança de energia em sistemas embebidos de tempo real, que tomam em consideração tarefas que não ocupem o tempo máximo de execução [33]

- SD – Static ordering and Dynamic voltage assignment
- DD – Dynamic ordering and Dynamic voltage assignment

São algoritmos usados em ambientes dinâmicos, que têm como principal princípio a alocação de tempo extra para uma tarefa sabendo que outra terminou mais cedo. Caso a tarefa que liberte o processador demore menos do que $c_i$, a próxima tarefa pode demorar o tempo extra equivalente ao libertado. Neste caso, pode utilizar o tempo extra para o processamento de interrupções em vez de diminuir a velocidade do processador para diminuir o consumo de energia.

Para um sistema em que as cinco condições admitidas não se verifiquem é possível, de acordo com os mecanismos descritos no terceiro capítulo, estabelecer condições semelhantes.

## 6.3 Tratamento de situações de sobrecarga

Nos sistemas onde a inibição das interrupções é realizada com base no seu ritmo, após o sistema ter detectado uma sobrecarga, os eventos podem ser tratados de um modo mais

[1] Tem-se $f(N) = N\left(2^{1/N} - 1\right)$ para o escalonador RMS e $f(N) = 1$ para o EDF ou MLF

[2] Máximo número de interrupções num dado intervalo de tempo sem que a sobrecarga seja detectada.

controlado, sem comprometer as outras tarefas. A leitura periódica (*polling*) permite processar alguns eventos (note-se que se deseja perder propositadamente alguns eventos), de modo a permitir que a aplicação execute num modo com maior latência mas garantindo as metas temporais. A leitura por *polling* permite, para além de processar alguns eventos, determinar se a situação de sobrecarga continua presente e, caso não esteja, o sistema retorna ao funcionamento normal, desinibindo as interrupções. O método de determinação da presença de uma sobrecarga através da leitura periódica é semelhante ao por interrupções, onde $y[n]$ é actualizado da mesma forma mas com grande perda de eventos pelo que necessita de um limiar $m$ muito menor que $M$, abaixo do qual o sistema não se encontra em sobrecarga[1]. Um ciclo de histerese, com a definição dos limiares $M$ e $m$ deve ser implementado – Figura 6.3.

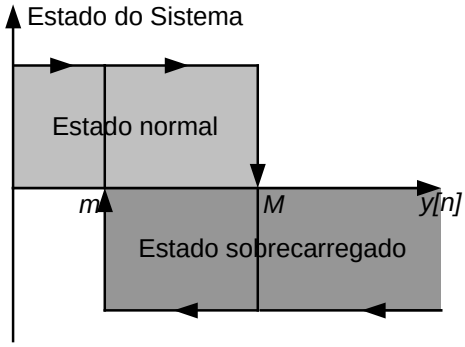


Figura 6.3 – Ciclo de histerese na determinação de situações entre sobrecarga e normal

Para o método da alocação do tempo para o processamento de interrupções, não existe tratamento especial para situações de sobrecarga. O sistema espera que outra tarefa ocupe o processador para receber mais eventos. Este algoritmo já permite o máximo número de interrupções sem interferir com o cumprimento das metas temporais. É, no entanto, frágil em relação a aplicações que desejam reagir rapidamente a certos eventos e que possuem tarefas extremamente demoradas. Caso uma dessas tarefas ocupe o processador e uma sobrecarga de eventos provoque a inibição das interrupções, um grande intervalo de tempo pode demorar até que o sistema receba de novo eventos. Nestas situações, uma interligação entre os dois métodos pode tornar o sistema mais robusto, permitindo apenas um certo ritmo máximo de interrupções, impedindo que tarefas longas esgotem logo no início do processamento todas as interrupções admitidas, mas tendo também em conta a tarefa que se encontra em processamento, que deseja (ou não, no caso da tarefa de *idle* ou outras que não são de tempo real) um limite máximo para o *jitter*.

## 6.4 Implementação

Nesta secção é discutido como implementar os algoritmos propostos de uma maneira eficiente dentro da ISR. O tempo do processamento das interrupções – Figura 6.1 – afecta o número de interrupções admitidas, pois as condições que estabelecem as garantias do

[1] $y[n]$ não é actualizado com a mesma frequência (existe perda de eventos) e estabiliza num valor mais baixo.

cumprimento das metas temporais (equação 6.10) definem o tempo máximo permitido para tratamento de interrupções ($b_i$). O aumento de tempo gasto no controlo do ritmo das interrupções no processamento de interrupções deve ser minimizado de forma a maximizar o número de interrupções processadas.

### 6.4.1 Implementação do método de detecção do ritmo das interrupções

A implementação dos sistemas que detectam o ritmo das interrupções necessita de um certo tratamento pois a actualização do sistema é feita assincronamente. No filtro IIR, sendo $y[n]$ actualizado dentro da ISR, o pseudocódigo encontra-se na Figura 6.4.

```
ISRControladora( ){
      /* Determinação do ritmo das interrupções */
      t = tempoActual();
      n = sampleAndHold(t);
      y[n+1] = α^(n -n_ant) y[n] + (1-α);
      n_ant = n;

      /* Teste se existe sobrecarga */
      if( y[n+1] > M ){
                        estado = sobrecarga;
                  inibeInterrupçao();
            comecaTratamentoSobrecarga();
      }
      ISROriginal();
}
```

Figura 6.4 – Pseudocódigo da ISR com actualização do estado do filtro e inibição das interrupções

Este pseudocódigo tira partido de $y[n]$ ser decrescente entre interrupções e consequentemente ultrapassa o limiar $M$ durante o tratamento de uma interrupção e não no tempo intermédio. Dado que $y[n]$ é exponencialmente decrescente entre duas interrupções, o factor $\alpha^{n-n\_ant}$ fornece a actualização de $y[n]$ entre o tempo $n$ e o instante da última interrupção

$$y[n] = \alpha y[n-1] = \alpha^2 y[n-2] = \ldots = \alpha^N y[n-N] \tag{6.11}$$

A determinação em tempo de execução de $\alpha^{n-n\_ant}$ pode ser bastante demorado se não forem tomadas as devidas precauções. Para impedir que o cálculo seja dependente do seu expoente, é construída uma tabela durante a inicialização do sistema de modo a que o índice corresponda ao seu valor[1].

Tanto neste filtro como para o FIR, outra alteração importante no cálculo da saída em relação ao modo convencional provêm do facto de não ser possível utilizar a unidade de vírgula flutuante

[1] O expoente é inteiro positivo e se for superior aos limites da tabela, pode-se assumir que $\alpha^{n-n\_ant}$ é aproximadamente nulo ($\alpha < 1$).

– FPU (*Float Point Unit*) – existente no sistema[1]. Um factor de escala foi adicionado nos cálculos de modo a permitir utilizar unidades inteiras com pouca perca de resolução.

Por último, a função tempoActual necessita, de acordo com o modelo do filtro discreto utilizado, de amostrar com um período inferior ao tempo mínimo entre duas interrupções consecutivas, de modo a não perder nenhuma. Como o RTEMS apenas suporta resoluções temporais na ordem dos *clock ticks*, uma função foi criada que lê a contagem interna do Timer0 (utilizado para o relógio de tempo real) para fornecer uma precisão de 1 μs – discutido na secção 3.4.1.

Para o filtro FIR a implementação do sistema assincronamente é mais complexa e requer mais tempo para ser calculada. Quando uma interrupção ocorre, é guardado o tempo da sua ocorrência numa tabela contendo as interrupções mais recentes – Figura 6.5.

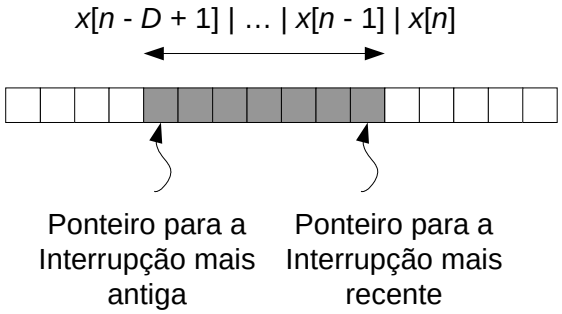


Figura 6.5 – Vector em Anel contendo a memória do filtro FIR

O filtro decide quais as interrupções 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 interrupção válida, avança o ponteiro. O somatório da equação 6.7 pode ser calculado pela diferença entre os dois ponteiros da Figura 6.5 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.

[1] O RTEMS não possui por omissão a salvaguarda do contexto da FPU durante uma ISR para não aumentar a latência das interrupções. Este comportamento é usual nos Sistemas Operativos de Tempo Real, embora seja possível à própria aplicação salvaguardar o estado e repô-lo no final da ISR.

### 6.4.2 Implementação do método de alocação de tempo para tratamento de interrupções

Este algoritmo baseia-se na informação de qual a tarefa em execução no processador. Para isso, é proposto a utilização dos mecanismos de "*Extension Manager*" providenciados pelo RTEMS [17]. Através de funções definidas pela aplicação, é possível interceptar as mudanças de contexto e criação de tarefas para actualizar variáveis. Assim, é possível conhecer qual a tarefa em execução com base numa variável que é modificada sempre que é realizado uma mudança de contexto entre tarefas. Com base nesta variável, é possível aceder a um vector contendo os tempos máximos e actuais de alocação para o processamento das interrupções ($\Delta_i$). Quando uma tarefa libertar o processador, é também realizado uma mudança de contexto, mas com base na informação dada por uma variável no TCB (*Task Control Block*) da tarefa, é possível saber se foi preemptada ou se libertou o processador porque acabou o seu processamento. Caso tenha terminado, coloca-se o tempo actual de alocação com um valor nulo. Dentro da ISR, é incrementado o tempo despendido no processamento de interrupções e realizada a comparação com o valor máximo permitido para esta tarefa. Caso tenha excedido, a interrupção é inibida e só voltará a ser admitida quando uma nova mudança de contexto a consentir. Este mecanismo, ao contrário dos outros apresentados anteriormente, aumenta o tempo de comutação entre tarefas (devido à actualização da variável que indica qual a tarefa em execução) mas é ligeiramente mais rápido dentro da ISR levando a uma menor latência das interrupções.

## 6.5 Resultados

Esta secção apresenta os resultados referentes à implementação do método que elimina a sobrecarga das interrupções através da determinação do seu ritmo e realizando a técnica de *polling* para o tratamento de eventos de uma forma controlada. O teclado foi o dispositivo testado, devido ao contacto prévio com o VITRAL. Este dispositivo alcança as 33 interrupções por segundo. Embora seja um valor extremamente baixo quando comparado com, por exemplo, interfaces de rede (Tabela 6.1), 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. A Figura 6.6 e 6.7 apresentam 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 tabela que calcula a função de $\alpha^{n-n\_ant}$ ser excedida (dimensão de 200 períodos de amostragem). Nestes casos, 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[1].

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 – Figura 6.7 b). Possui um mesmo comportamento que o IIR mas estabiliza mais rapidamente, especialmente quando as interrupções estão inibidas, devido a forma linear como varia, em vez da curva exponencial decrescente do IIR, que varia muito lentamente quando próxima de zero.

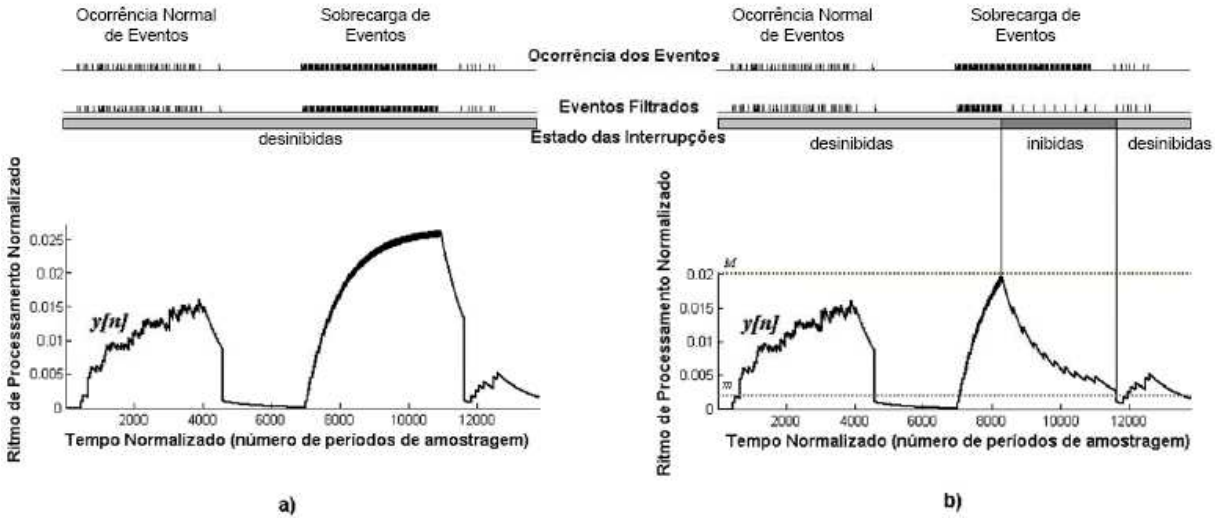


Figura 6.6 – Evolução de $y[n]$ no filtro IIR sem e com os mecanismos de protecção – a) e b)

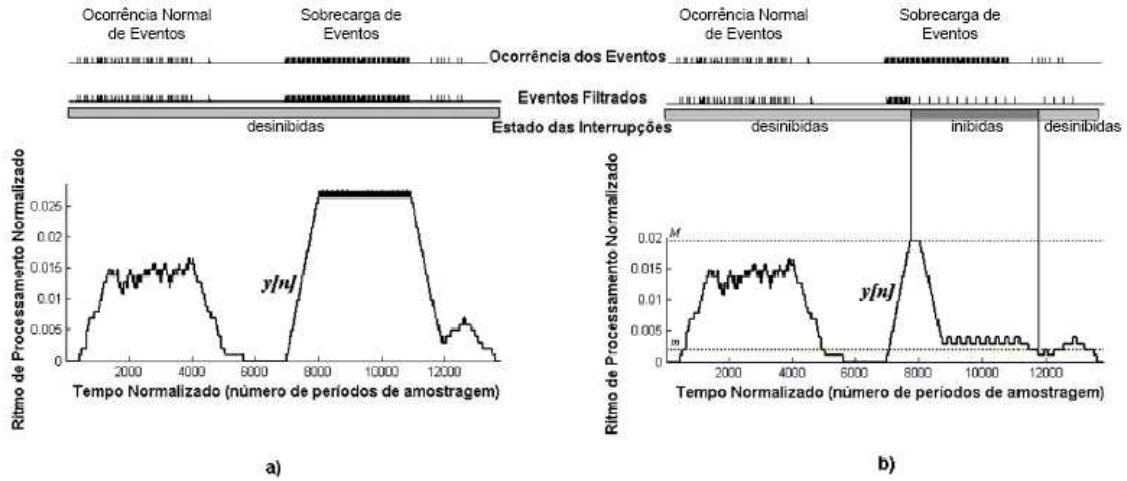


Figura 6.7 - Evolução de $y[n]$ no filtro FIR sem e com os mecanismos de protecção – a) e b)

[1] Fazendo a dimensão da tabela suficientemente alta é possível manter o pior caso calculado.

# Sumário

O tratamento de interrupções nas ISRs interfere com o processamento das tarefas, pois possui uma prioridade superior. Caso esta interferência não esteja limitada, pode provocar o não cumprimento de metas temporais em sistema de tempo real.

O objectivo deste capítulo é fornecer mecanismos que limitem a interferência do tratamento das interrupções limitando o número de interrupções processadas. Substituindo a ISR original da aplicação por uma que controla o número de interrupções processadas e que chama a ISR original, o sistema não interfere com as aplicações residentes e fornece um mecanismo eficiente de detecção de sobrecargas de interrupções.

São apresentados dois métodos para detectar sobrecargas de interrupções utilizando o mecanismo indicado

- Detecção do ritmo das interrupções
- Alocação de tempo para processamento de interrupções

O primeiro método detecta o ritmo a que surgem as interrupções através de filtros lineares e invariantes no tempo (SLIT). São apresentados dois sistemas: filtro de primeira ordem IIR *(Infinite Impulse Response*); filtro FIR (*Finite Impulse Response*) de dimensão variável. Estabelecido um ritmo máximo *M* a partir do qual não se aceitam interrupções, o controlo verifica se o ritmo actual é superior ao limiar e, em caso afirmativo, inibe as interrupções.

O segundo método aloca tempo para processamento das interrupções recorrendo às condições do escalonador, apresentadas no terceiro capítulo.

Os dois métodos são analisados matematicamente para fornecerem condições sobre o cumprimento das metas temporais. Enquanto que o segundo método fornece uma garantia quase imediata sobre o cumprimento das metas, o primeiro requer uma análise bastante mais precisa[1] que se encontra no Anexo C.

Em situações de sobrecarga, existem vários métodos para permitir uma degradação graciosa dos serviços (*graceful service degradation*). A mais simples e a implementada, consiste no teste cíclico do dispositivo por eventos. Este método é fácil de integrar nas condições do escalonador que garantem o cumprimento das metas temporais, embora menos eficaz. Em situações de sobrecarga, como é a situação actual, o *polling* permite perder alguns eventos, tal como se deseja para não sobrecarregar o sistema.

Aspectos relativos à implementação dos dois métodos são abordados. A abordagem típica do primeiro método, em que a discretização do tempo requer normalmente a actualização periódica do estado interno do filtro, é impraticável, pois requer demasiado tempo de computação e não segue a abordagem da intercepção da ISR original. Alterações na actualização do estado interno, tanto no filtro IIR como no FIR, foram realizadas para resultarem numa forma assíncrona. Estas modificações exploraram o sistema computacional e sistema operativo utilizados, com especial atenção ao seu tempo de execução, visto que a computação é realizada dentro de uma ISR. Concluiu-se que a actualização do filtro IIR tem complexidade $O(1)$, isto é, o tempo que demora

[1] Esta análise necessita de uma revisão mais profunda, pelo que nesta fase ainda não são fornecidas garantias absolutas da sua correcção. No entanto, é certo que condições matemáticas podem ser estabelecidas, tal como para o segundo método.

a calcular não é dependente de nenhuma variável, enquanto que para o filtro tem-se complexidade $O(log(D))$, em que $D$ é a dimensão do filtro.

O segundo filtro obriga ao uso da exploração dos mecanismos internos do sistema operativo. O RTEMS, por exemplo, oferece um mecanismo que permite chamar funções em períodos cruciais do escalonamento, como a mudança de contexto entre tarefas. Caso se explore este mecanismo, é possível saber qual a tarefa em execução e alterar as variáveis de acordo, para garantir a coerência das variáveis que são testadas e/ou modificadas durante as ISRs. Não utilizando este mecanismo, o programador necessita de alterar o próprio escalonador para a alteração das variáveis.

Por último, descrevem-se os resultados obtidos, tanto para o filtro IIR como para o filtro FIR, quando aplicados num contexto de interrupções geradas pelo teclado, ilustrativo da geração assíncrona de eventos. Conclui-se que os filtros detectam o ritmo de geração de interrupções e, caso ultrapasse o limiar estabelecido ($M$), o sistema inibe as interrupções. O tratamento é realizado através de *polling* que, enquanto detectar a presença contínua de eventos, não regressa ao normal. Quando o ritmo dos eventos baixar de um outro limiar menor ($m$), as interrupções são de novo desinibidas e o sistema regressa ao normal.

Capítulo 7

# Conclusões & Perspectivas Futuras

## Conclusões

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:

1. Definição e adaptação do ambiente de geração de aplicações para o sistema operativo RTEMS, tendo em visto a sua utilização específica em ambientes distribuídos envolvendo operações de entradas/saídas.
2. Implantação dos mecanismos necessários a um arranque rápido e flexível da aplicação através de mecanismos de Arranque Remoto.
3. Definição, concepção e integração de uma interface de gestão de janelas, denominada VITRAL, que visa a qualidade e organização da apresentação de dados na consola.
4. Definição, concepção e integração de uma interface modular e flexível para incorporação de gestores de entrada/saída que comunicam com o "mundo exterior" através de sensores e actuadores.
5. Controlo da pontualidade do sistema/aplicações em condições de sobrecarga de eventos.

No primeiro ponto, a instalação nativa do RTEMS foi enriquecida com um ambiente de desenvolvimento, separando o contributo dos vários programadores, de uma forma modular e que permite abstraí-lo de pormenores de implementação não relativos ao seu desenvolvimento.

A inicialização do sistema através de mecanismos de Arranque Remoto, descrito no segundo ponto, trata uma questão de engenharia, na qual foi tomada esta opção relativamente a outros métodos. A sua implementação fornece uma rapidez e flexibilidade de que de outra forma não são conseguidas.

O aumento da qualidade de apresentação através de uma organização e clareza dos dados foi conseguida através da implementação de uma interface de gestão de janelas, denominada VITRAL. Comparou-se a sua interferência com as características de tempo real do RTEMS e verificou-se que não as altera significativamente. Uma comparação com outros ambientes gráficos demonstrou a sua superioridade nos aspectos temporais, embora não permita tantas funcionalidades. Acrescentou-se ainda uma camada de adaptação a outros sistemas operativos de forma a aumentar a sua portabilidade.

Abordou-se a integração modular e uniforme dos dispositivos de entrada/saída, tendo em conta funcionalidades acrescidas, como o tratamento assíncrono dos eventos por parte das tarefas da aplicação (em contraposição com as tarefas do gestor do dispositivo). Abordou-se ainda a necessidade de uma camada intermédia, que estabelece a ligação entre o gestor de dispositivos e o resto da aplicação, de forma a estender o uso do dispositivo a acessos remotos através da rede de comunicação. Como exemplos, o gestor de janelas VITRAL e a interface a uma prensa electro-pneumática foram integrados de forma no núcleo de sistema operativo RTEMS.

Como exemplos, o gestor de janelas VITRAL e uma prensa electro-pneumática foram integrados de forma modular no núcleo de sistema operativo RTEMS.

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. Um trabalho recente, também na área da eliminação de sobrecargas de interrupções [31], foi desenvolvido em paralelo, revelando que as ideias propostas têm fundamento e que a sua análise matemática, que ainda não tinha sido realizada, produz uma mais-valia nas áreas de sistemas de tempo real.

# Perspectivas Futuras

Em cada área deste Trabalho Final de Curso existem novos mecanismos que podem ser integrados de forma a produzir um sistema mais robusto.

A arquitectura do ambiente de geração de aplicações pode incorporar outros sistemas para além do CVS, de forma a permitir maior funcionalidade. Uma interface gráfica introduz um ambiente mais apelativo para o programador, reduzindo o custo de desenvolvimento e manutenção da aplicação.

A utilização de novos protocolos de comunicação e/ou a integração de uma infra-estrutura de comunicação de tempo real (e. g. CAN) com o processo de arranque remoto permite um sistema mais robusto e/ou eficiente e com características mais apelativas.

O aumento de portabilidade do VITRAL para novos adaptadores vídeo e/ou plataformas computacionais é extremamente apelativo. A inclusão de novas funcionalidades de adaptadores recentes (principalmente para operações de *scroll*), pode introduzir um aumento de eficiência que, como se viu pelos tempos apresentados[1], é de significativa importância.

A integração da interface termios, presente em vários sistemas operativos (incluindo RTEMS), no VITRAL também é interessante, visto que permite que certas operações possam ser definidas em tempo de execução, como por exemplo, a escolha de imprimir para o ecrã a tecla premida, aumentado assim a funcionalidade e a compatibilidade com outros sistemas operativos.

A definição e implementação de uma arquitectura de uma agência distribuída que fornece um tratamento de dispositivos de entrada/saída, explorando conceitos do GEAR (*Generic-Events Architecture*) [34], será incluída na continuação académica deste Trabalho Final de Curso, o Mestrado Integrado, integrado no projecto DARIO (*Distributed Agency for Reliable Input/Output*), financiado pela FCT (Fundação para a Ciência e Tecnologia), através do programa POSC/EIA/56041/2004.

Como trabalho futuro na área do tratamento de sobrecargas de interrupções, tem-se prevista a implementação do método de alocação para tempo de processamento. Este método, em conjunto com uma abordagem mais dinâmica, suportada pelos algoritmos retirados da minimização do consumo de energia (SD – *Static ordering and Dynamic voltage assignment* ou DD – *Dynamic ordering and Dynamic voltage assignment*), produz uma versão alternativa ao da determinação do ritmo das interrupções. A aliança destes três métodos em conjunto, tal como descrito nas secções 6.2 e 6.3, fornece um sistema ainda mais robusto, tanto para processamento de eventos como o fornecimento de garantias temporais.

[1] Uma operação normal de escrita demora cerca de 11 μs enquanto que uma de *scroll* pode chegar aos 11 ms.

# Referencias


[1] Kopetz H. and Grunsteidl, G., "*TTP – A Protocol for Fault-Tolerant Real-Time Systems*", IEEE Computer, Vol. 27, No. 1, Jan. 1994.

[2] Rufino, J., Veríssimo, P. and Arroz G., "*Embedded Platforms for Distributed Real-Time Computing – Challenges and Results"*. J. Rufino, P Veríssimo, G. Arroz. Proceedings of the 2nd IEEE International Symposium on Object-Oriented Real-Time Distributed Computing. Saint Malo, France, 1999.

[3] Mota, p., *Design and Implementation of CAN Remote Bootstrap*, IST Graduation Project Report, Instituto Superior Técnico, Lisboa, Portugal, Jul 1997.

[4] Lorin, H. and Deitel, H. M., *Operating Systems*, Addison-Wesley Publishing Company Inc., Massachusetts, United States of America, 1981.

[5] Marques, J. A. And Guedes, P., *Fundamentos de Sistemas Operativos*, Editorial Presença, Lisboa, Portugal, 1990.

[6] Burns, A. and Wellings A., *Real-Time Systems and Programming Languages*, Addison-Wesley, Harlow, England, 2001.

[7] Veríssimo, P. and Koptez, H., *Design of Real-Time Systems*, In S. Mullender, Addison-Wesley, 1993.

[8] Beyhnam, M. H., "*Flexible Scheduling for Real Time Control Systems based on Jitter Margin"*, Mälardalens Högskola, Västerås, 2005.

[9] J. Lehoczky, J. and Sha, L. and Ding, Y., "*The rate monotonic scheduling algorithm: Exact characterization and average case behavior*", In *Proceedings IEEE Real-Time Systems Symposium*, Santa Monica, United States of America, Dec. 1989.

[10] Liu, C. L. and Layland J. W., "*Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment*", Journal of the ACM, Vol. 20, No. 1, 1973.

[11] Stewart, D. B. and Khosla, P. K., "*Real-Time Scheduling of Dynamically Reconfigurable Systems*", In Proceedings of the IEEE International Conference on Systems Engineering, Dayton, United States of America, Aug 1991.

[12] Levine D. L., Gill C. D. and Schmidt D. C., "*Dynamic Scheduling Strategies for Avionics Mission Computing*", In the 17th IEEE/AIAA Digital Avionics Systems Conference, Seattle, United States of America, Oct/Nov 1998.

[13] *Åkerholm, M. and Samuelsson, T., Design and Benchmarking of Real-Time Multiprocessor Operating System Kernels*, Master Thesis, *Mälardalen University, 2002.*

[14] Cornhill, D., Sha, L. and Lehoczky, J., and Rajkumar, R., and Tokuda, H., "*Limitations of Ada for real-time scheduling*", In Proceedings of the International Workshop on Real Time Ada Issues, 1987.

[15] Sha, L., Rajkumar, R. and Lehoczky, J. P., "Priority Inheritance Protocols: An Approach to Real-Time Synchronization", In IEEE Transactions on Computers, Vol. 39, No. 9, 1990.

[16] *Parks, T. and Lee, E. A., "Non-Preemptive Real-Time Scheduling of Dataflow Systems", In* IEEE International Conference on Acoustics, Speech, and Signal Processing, May 1995

[17] On-Line Applications Research Corporation (OAR), *RTEMS C User's Guide*, edition 4.6.2, for rtems 4.6.2 edition, August 2003.

[18] On-Line Applications Research Corporation (OAR), *RTEMS Intel i386 Applications Supplement*, edition 4.6.2, for rtems 4.6.2 edition, August 2003.

[19] Straumann, T. "*Open Source Real Time Operating Systems Overview*", In 8th International Conference on Accelerator & Large Experimental Physics Control Systems, San Jose, United States of America, 2001.

[20] http://that.gsfc.nasa.gov/osgroup/benchmarks.html

[21] Romano, P., *Design and development of a rtems board support package for the pc386 architecture*, Final graduation project, Instituto Superior Técnico, Lisboa, Portugal, Oct 1998.

[22] Messmer, H., *The Indispensable PC Hardware Book*, Addision-Wesley, Harlow, England, 1997.

[23] On-Line Applications Research Corporation (OAR), *RTEMS Development Environment Guide*, edition 4.6.2, for rtems 4.6.2 edition, August 2003.

[24] Beijing Feynman Software Technology, *MiniGUI Technology White Paper*. (www.minigui.com/whitepaper/ MiniGUITechWhitePaper-1.6E.pdf)

[25] http://www.microwindows.org

[26] Beijing Feynman Software Technology, *MiniGUI Programming Guide*. (http://www.minigui.com)

[27] Bovet, D. P. and Cesati, M., *Understanding the Linux Kernel*, O'Reilly, London, England, 2002.

[28] Rubini, A. and Corbet,. J., *Linux Device Drivers*, O'Reilly, London, England, 2001.

[29] Kopetz, H., *"Event-triggered versus time-triggered real-time systems"*, In Proceedings of the International Workshop of the 90s and Beyond, Springer-Verlag, London, 1991.

[30] Teles, F. e Fonseca, J., *Integração Hierárquica de Componentes e Serviços em Plataformas para Controlo Industrial Distribuído,* Trabalho Final de Curso, Instituto Superior Técnico, Lisboa, Portugal, Set 2000.

[31] Regehr, J. and Duongsaa, U., *"Preventing Interrupt Overload", In Proceedings of the ACM Conference on Languages*", *Compilers, and Tools for Embedded Systems*, Chicago, United States of America, June 2005.

[32] Oppenheim, A. V. and Schafer, R. W., *Discrete Time Signal Processing, Prentice-Hall International*, 2nd Edition.

[33] Okuma T. and Yasuura H. and Ishihara T., *"Software Energy Reduction Techniques for Variable-Voltage Processors", IEEE Design and Test of Computers*, May/Apr, 2001.

[34] Verríssimo, P., Kaiser, J. And Casimiro, A., *"An architecture to support interaction via generic events*", In Proceedings of the 24th IEEE Real-Time Systems Symposium *Work in Progress Sessions.*, Cancun, Mexico, Dec 2003.