UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática

APLICAÇÃO DE SISTEMAS OPERATIVOS DE TEMPO REAL NO SISTEMA GALILEO

Ivo Daniel Oliveira Guimarães

Mestrado em Engenharia Informática

2008

UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática

APLICAÇÃO DE SISTEMAS OPERATIVOS DE TEMPO REAL NO SISTEMA GALILEO

Ivo Daniel Oliveira Guimarães

PROJECTO

Projecto orientado pelo Prof. Dra. Antónia Lopes
e co-orientado por Dr. Carlos Santos

Mestrado em Engenharia Informática

2008

Abstract

Real-time operating systems have been vastly used in the development of embedded and/or critical systems. However, these operating systems present reliability and security problems at implementation level, providing services and functions that may compromise the application and, in some situations, the integrity of the systems themselves. This could have catastrophic consequences in the case of critical applications.

Throughout this document, real-time operating systems are analyzed, with focus on important specifications such as POSIX and ARINC 653, in order to understand what can be offered to develop real-time critical/embedded systems.

The document also presents a solution that was developed to protect the application layer from a real-time operating system in the presence of unprocessed errors, offering portability to the application layer w.r.t. the operatyng system.

The work presented was developed the course of *Project in Informatics Engineering*, Masters in Informatics Engineering, Faculty of Sciences, University of Lisbon. The work was carried out in Skysoft, in the context of a Galileo System project, the OSPF RTMC.

KEYWORDS: Galileo system, RTOS, POSIX, ARINC 653

Resumo

Os sistemas operativos de tempo real têm sido imensamente utilizados no desenvolvimento de aplicações embebidas e/ou críticas. No entanto, todos apresentam problemas de fiabilidade e segurança na sua implementação ao disponibilizarem serviços e funcionalidades que ameaçam a integridade da aplicação e por vezes do próprio sistema operativo.

Ao longo do documento serão analisados vários aspectos dos sistemas operativos de tempo real, passando por importantes especificações como o POSIX e ARINC 653. O objectivo da análise é perceber o que podem oferecer a quem desenvolve sistemas embebidos e/ou críticos de tempo real. Esta análise servirá de base de conhecimento à configuração de um sistema operativo de tempo real, explorando várias alternativas de configuração.

Neste documento será ainda apresentada uma solução concretizada para oferecer portabilidade, segurança e fiabilidade à camada de aplicação em relação ao sistema operativo.

Este trabalho foi realizado no âmbito da disciplina do Projecto em Engenharia Informática do Mestrado em Engenharia Informática da Faculdade de Ciências da Universidade de Lisboa. O projecto OSPF RTMC insere-se num dos projectos do Sistema Galileo, realizado na Skysoft.

PALAVRAS-CHAVE: Sistema Galileo, RTOS, POSIX, ARINC 653

Índice

Lista de figuras

Lista de tabelas

1. Introdução

O objectivo deste documento é apresentar o trabalho realizado no âmbito da disciplina do projecto de Engenharia Informática do Mestrado em Engenharia Informática num projecto do sistema Galileo.

1.1. Enquadramento

O trabalho que se apresenta foi desenvolvido no contexto de um projecto da Skysoft do sistema Galileo, o OSPF RTMC — *Orbit determination and time Synchronization Processing Facility, Real Time Monitoring and Control*.

O sistema Galileo é um projecto da ESA (*European Space Agency*) que visa providenciar um sistema de posicionamento e de tempo global, semelhante ao sistema GPS (*Global Positioning System*). No sistema Galileo, o OSPF é o elemento responsável pela determinação das órbitas de uma constelação de satélites e pela sincronização temporal de todo do sistema Galileo.

O projecto foi dividido em duas partes. A primeira parte consistiu na configuração do sistema operativo de tempo-real que executará o OSPF. A segunda parte consistiu na concretização de uma camada entre o sistema operativo e o OSPF, com o objectivo de fornecer ao OSPF portabilidade, assim como maior fiabilidade e segurança.

1.2. Skysoft

A Skysoft é uma empresa tecnológica com quase uma década de experiência nos domínios da Aeronáutica, Espacial e Telemática. O objectivo principal da empresa consiste em fornecer sistemas de software de alta fiabilidade para os mercados referidos, assim como manter o compromisso tecnológico em investigação e desenvolvimento, a fim de identificar oportunidades para produtos de elevada complexidade.

No âmbito do projecto OSPF RTMC a Skysoft conta actualmente com uma equipa formada por:

- Um gestor de projecto;

- Um gestor técnico (Carlos Santos – o co-orientador deste trabalho);

- Um líder de desenvolvimento;

- Um líder AIV (*Assembly Integration and Verification*);

- Uma equipa de dois engenheiros AIV;

- Uma equipa de cinco engenheiros de software (na qual o autor do trabalho esteve integrado);

1.3. Organização do documento

Este documento foi organizado de forma a facilitar a sua leitura. No início do documento serão fornecidos os dados necessários ao enquadramento do leitor com o projecto do MEI, na zona central serão abordados os conceitos e conhecimentos tangentes ao projecto, e no final serão então explicados os detalhes do projecto.

Mais precisamente, o documento está organizado da seguinte forma:

- ☐ Capítulo 2, constituído pela introdução ao sistema Galileo
  - Introdução
  - Serviços do sistema Galileo
  - Arquitectura geral
  - Standards usados no âmbito do projecto
  - Projecto OSPF RTMC
- ☐ Capítulo 3, constituído pela introdução aos sistemas operativos de tempo real, orientados às necessidades do OSPF
  - Introdução
  - Características de sistemas operativos de tempo real
  - Introdução ao POSIX (*Portable Operating System Interface*)
  - Introdução à especificação ARINC 653 (*Avionics Application Standard Software Interface*)
  - Exemplos de sistemas operativos de tempo real
- ☐ Capítulo 4, constituído pela explicação e exploração do sistema operativo de tempo real LynxOS-178b
  - Introdução, características e objectivos
  - Arquitectura
  - Desenvolvimento versus a produção no LynxOS-178B
  - Configuração necessária ao sistema operativo
  - Construção e versatilidade no OSPF
- ☐ Capítulo 5, apresenta a camada RTOS_Rtos
  - Introdução e objectivos
  - Arquitectura da camada RTOS_Rtos
  - Detalhes
  - Testes da camada RTOS_Rtos
- ☐ Capítulo 6, apresenta a distribuição e organização das diferentes tarefas ao longo do tempo

- ☐ Conclusão

- ☐ Bibliografia e anexos

1.4. Acrónimos e definições

Acrónimo	Significado
ANSI	American National Standards Institute
APEX	Application/Executive
API	Application Programming Interface
ARINC 653	Avionics Application Standard Software Interface

ASN.1	Abstract Syntax Notation One
BSP	Board Support Package
COTS	Commercial Off The Shelf
CSP	CPU Support Package
CVS	Concurrent Version System
FCS	Fixed Cycle Scheduling
GACF	Ground Assets Control Facility
GCC	Ground Control Center
GLONASS	Global Orbiting Navigation Satallite System
GPS	Global Positioning System
GSS	Galileo Sensor Station
GSWS	Galileo Software Standard
HM	Health Monitoring
IEEE	Institute of Electrical and Electronics Engineers
IMA	Integrated Modular Avionics
IPC	Interprocess communication
IPF	Integrity Processing Facility
ISO	International Organization for Standardization
KDI	Kernel Downloadable Image
MDDN	Mission Data Dissemination Network
MEI	Mestrado em Engenharia Informática
MGF	Message Generation Facility
MMU	Memory Management Unit
MUCF	Galileo Monitoring and Uplink Control Facility
Mutex	Mutual exclusion
OS	Open Service
OSPF	Orbit determination and time Synchronization Processing Facility
PDL	Pseudo-code Design Language
PEI	Projecto de Engenharia Informática
POSIX	Portable Operating System Interface
PRS	Public Regulated Service
PTF	Precision Timing Facility
RTCA	Radio Technical Commission for Aeronautic
RTEMS	Real-Time Executive for Multiprocessor System
RTMC	Real Time Monitoring and Control
RTOS	Real-Time Operating System
SCS	Software Coding Standards
SDK	Software Development Kit
SISA	Signal-in-Space accuracy
SNMP	Simple Network Management Protocol

SOL	Safety-of-Life Service
SUP	Software Unit Test Plan
SW-DAL	Sofware Development Assurance Levels
ULS	Uplink Station
UPS	Uninterruptible Power Supply
VCT	Virtual Machine Configuration Table
VM	Virtual Machine

Tabela 1-1: Lista de acrónimos

2. Sistema Galileo

2.1. Introdução

O sistema Galileo, nome pelo qual é conhecido o *European Global Navigation Satellite System,* é um sistema de posicionamento e navegação global que irá providenciar aos seus utilizadores serviços de posicionamento geográfico à escala global. Este será o primeiro sistema de posicionamento e navegação desenvolvido especificamente para o uso civil, disponibilizando vários tipos de serviços para uso global.

2.2. O porquê do sistema Galileo

O primeiro argumento poderá ser o político, uma vez que é do interesse europeu ter independência e soberania sobre um serviço de localização geográfica inovador, fiável e seguro, surgindo como um complemento e/ou alternativa ao actual sistema existente e amplamente utilizado, o GPS.

Ao nível social, o sistema Galileo irá surgir como uma vantagem à população em geral, visto apresentar serviços livres de melhor qualidade, aumentando assim a fiabilidade e segurança na utilização dos serviços prestados.

Do ponto de vista tecnológico, servirá para impulsionar o desenvolvimento e a inovação de novas tecnologias em território europeu, melhorando as disciplinas envolvidas.

O factor económico é o no entanto o mais importante, pois impulsionará o surgimento de novas empresas, novos empregos, novos cursos, maior concorrência em serviços baseados em localização, tudo isto, antes e após a conclusão do projecto [4].

2.3. Arquitectura

O sistema Galileo é constituído por duas partes principais: o segmento espacial e o segmento terrestre.

O segmento espacial será constituído por uma constelação de satélites em torno do planeta terra. O segmento terrestre será constituído por toda a infra-estrutura terrestre para computação dos dados recebidos e comunicação com a constelação. A Figura 2-1 ilustra com mais detalhe a organização do sistema Galileo.

O segmento espacial vai ser composto por 30 satélites, dos quais 27 serão operacionais e 3 serão usados caso uma falha ocorra. Desta forma espera-se garantir a qualidade de serviço à escala global, robustez da constelação quanto a disponibilidade de serviço e fiabilidade em caso de falha.

O segmento terrestre do Galileo tem a função de controlar toda a constelação, medindo e monitorizando os equipamentos para determinação de parâmetros orbitais (parâmetros de *Kepler* e efemérides [19]), bem como efectuar a sincronização temporal. Este segmento é ainda responsável por gerar dados de navegação a enviar aos satélites. Este segmento será constituído por:

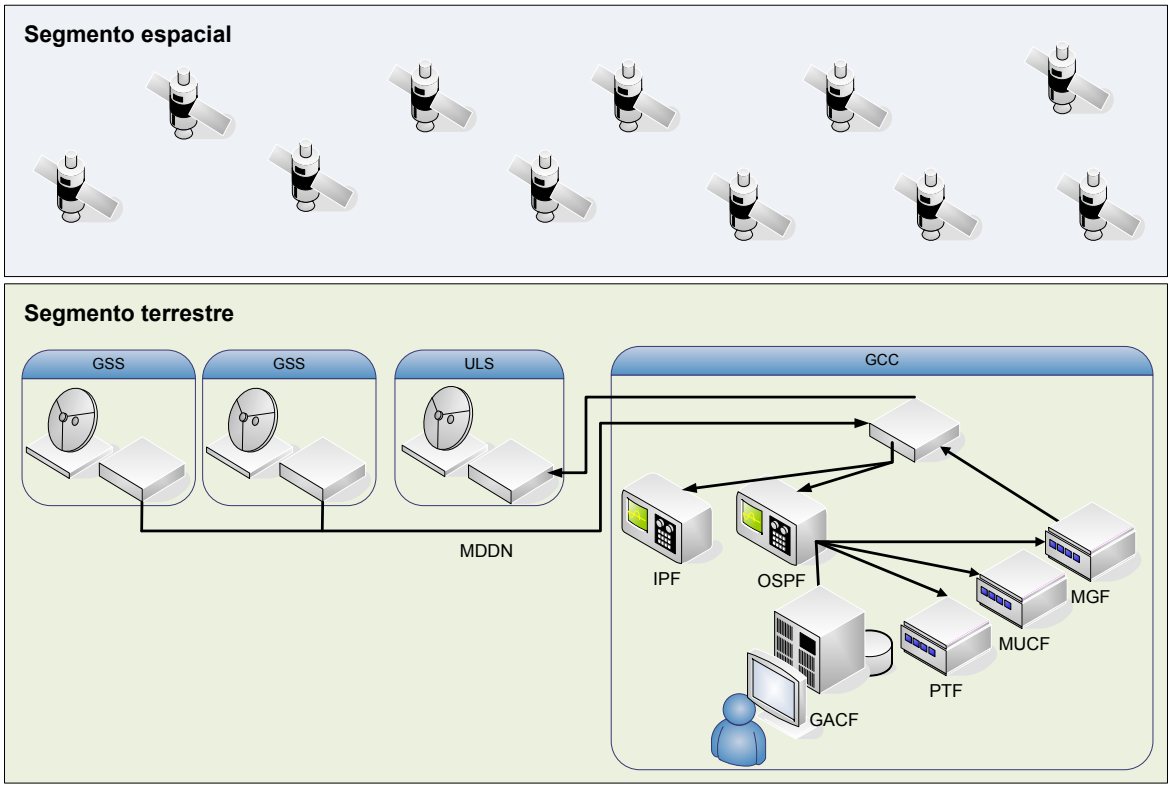

Figura 2-1: Arquitectura geral do sistema Galileo

- Uma rede de comunicações global dedicada chamada MDDN (*Mission Data Dissemination Network*) irá conectar todas as estações terrestres e instalações. Esta rede terá a função de monitorização e aquisição contínua de informação relevante das diferentes entidades;

- 2 GCC (*Ground Control Center*) cuja função será controlar a constelação, bem como a sincronização dos relógios atómicos dos satélites e computar os dados recebidos e os dados de controlo a enviar para a constelação de satélites;

- 50 GSS (*Galileo Sensor Station*) geograficamente dispersos ao longo do planeta terra formando uma rede para recolher os dados da constelação e monitorizar a qualidade do sinal de navegação. Os dados recolhidos são então retransmitidos através da rede de comunicação Galileo para os dois GCC [18];

- 9 ULS (*Uplink Station*) usadas para enviar à constelação informação sobre navegação e integridade computadas pelos GCC.

O elemento central do segmento terrestre do sistema Galileo é o GCC. Este é constituído por um conjunto de elementos que vão processar a informação recebida e a informação a ser enviada pelo GCC. Estes elementos são:

- O OSPF (*Orbit determination and time Synchronization Processing Facility*), que é responsável pela determinação de parâmetros de navegação, ou seja, computação

de efemérides, previsão dos relógios dos satélites e determinação da precisão do sinal espacial ao qual se dá o nome de SISA (*Signal-in-Space accuracy*);

- O IPF (*Integrity Processing Facility*), que é responsável pela determinação da integridade do sinal em tempo-real de cada satélite;

- O MGF (*Message Generation Facility*), que tem a função de efectuar a multiplexagem de todas as mensagens de navegação e integridade geradas no contexto do GCC ou das mensagens recebidas por entidades externas num único fluxo de dados a ser enviado para cada ULS, que por sua vez será reencaminhado para a constelação de satélites;

- O PTF (*Precision Timing Facility*), que é responsável por computar o GST (*Galileo System Time*), providenciando uma referência temporal fiável e estável para coordenar todo o sistema;

- O GACF (*Ground Assets Control Facility*), que é responsável por monitorizar e controlar todos os elementos do segmento terrestre em tempo-real, fornecendo uma interface ao operador deste elemento.

- O MUCF (*Galileo Monitoring and Uplink Control Facility*), que é responsável por monitorizar a constelação de satélites e os dados enviados para esta.

2.4. Serviços Galileo

O sistema Galileo envolve uma constelação de satélites Galileo onde cada um emitirá sinais temporais de navegação com sinais de dados de navegação. Estas transmissões irão não só conter informações essenciais à navegação, como também sinais de integridade que fornecem um reforço à precisão do serviço de posicionamento global.

As diferentes categorias de serviços previstas para o sistema Galileo estão qualificadas em termos de precisão, qualidade de serviço, integridade e outros parâmetros. Essas categorias de serviços encontram-se divididas de acordo com os seus requisitos, níveis de desempenho e aspectos de segurança. Deste modo, temos:

- O serviço OS (*Open Service*), definido para aplicações de mercado em massa, que fornecerá sinais temporais e de posicionamento sem custo. O OS será acessível a qualquer utilizador equipado com um receptor, sem necessidade de autorização. Neste serviço não será oferecida informação sobre a qualidade do sinal, podendo esta ser estimada pelo receptor do utilizador.
- O serviço SoL (*Safety-of-life Service*), usado na maioria das aplicações na área dos transportes, onde vidas podem ficar em risco se o desempenho do sistema de navegação perder qualidade e não houver uma resposta em tempo real. O SoL irá fornecer a mesma precisão no posicionamento que o OS. A principal diferença é o alto nível de integridade do sinal à escala global para as aplicações onde a segurança é crítica, como a navegação marítima, aérea e ferroviária.
- O serviço CS (*Commercial Service*), orientado a aplicações que necessitam de uma qualidade de serviço superior à oferecida pelo OS. Serviços típicos de valor acrescentado incluem a transmissão de dados de alta velocidade e serviços temporais precisos para posicionamento de alta precisão.

- O serviço PRS (*Public Related Service*), usado por grupos governamentais como polícia, alfândega, etc. O Galileo é um sistema civil, logo irá conter um serviço robusto e de acesso controlado para aplicações governamentais. Instituições civis irão controlar o acesso ao PRS. Um dos requisitos deste serviço é de que funcione em qualquer circunstância, mesmo em períodos de grande congestionamento como em situações de crise.
- O serviço SAR (*Search and Rescue Service*) é a contribuição da Europa para o esforço cooperativo internacional na busca e salvamentos humanitários. Este serviço irá permitir importantes melhoramentos no sistema existente, incluindo a recepção de mensagens de emergência de qualquer parte do planeta em tempo real e localização precisa dos alertas. O sistema Galileo irá ainda introduzir novas funções às de busca e salvamento, como a conexão bidireccional, facilitando assim as operações de salvamento, eliminando deste modo o número de falsos alarmes.

2.5. Standards de desenvolvimento

Dada a importância, complexidade e o número de participantes no sistema Galileo, é necessário estabelecer regras para que todos os participantes sigam procedimentos que assegurem a qualidade dos projectos. O GSWS (*Galileo Software Standard*) é o documento que especifica requisitos e procedimentos a serem seguidos nas várias engenharias de software, nomeadamente qualidade do produto e a gestão da configuração. O GSWS é aplicado a todos os produtos de software do projecto Galileo.

O desenvolvimento de software para projectos Galileo tem de ser necessariamente rigoroso, levando a que a quantidade de requisitos seja extensa. O SW-DAL (*Software Development Assurance Levels*) é uma especificação que define o nível de garantias a que o software deve obedecer. Estas especificações definem metodologias e regras a seguir durante todo o processo de desenvolvimento. A Tabela 2-1 apresenta os diferentes níveis associados ao desenvolvimento de software em projectos Galileo. O nível da certificação requerida corresponde assim às consequências de uma potencial falha de software.

SW-DAL	Definição
Nível A	Software onde um comportamento anómalo poderá causar ou contribuir para uma falha resultando num evento **catastrófico**.
Nível B	Software onde um comportamento anómalo poderá causar ou contribuir para uma falha resultando num evento **crítico**.
Nível C	Software onde um comportamento anómalo poderá causar ou contribuir para uma falha resultando num evento **importante**.
Nível D	Software onde um comportamento anómalo poderá causar ou contribuir para uma falha resultando num evento **sem importância**.
Nível E	Software onde um comportamento anómalo poderá causar ou contribuir para uma falha resultando num evento **negligenciável**.

Tabela 2-1: Definição de SW-DAL

A definição do SW-DAL é baseada no RCTA DO-178B (*Software Considerations in Airborne Systems and Equipment Certification*) [8] que define as orientações para o

desenvolvimento de software. Este foi publicado pela RTCA (*Radio Technical Commission for Aeronautics*). O standard DO-178B é centrado principalmente no processo de desenvolvimento. Como resultado a certificação para DO-178B implica a entrega de múltiplos documentos e registos de suporte. A quantidade de documentos e esforço necessário para a certificação DO-178B está directamente relacionada com o nível de certificação que é requerido [9].

De acordo com o nível de certificação requerida, existem geralmente cinco aspectos distintos de produção de documentos e registos. Estes aspectos dividem-se em planeamento, desenvolvimento, verificação, gestão de configuração e garantia de qualidade [10], sendo cada, uma disciplina em si. Os tipos de documentos e registos que usualmente são necessários produzir para o mais baixo destes níveis, o nível E, encontra-se representada na Tabela 2-2.

Acrónimo	Titulo	Fase	Tipo
PSAC	Plan for Software Aspects of Certification	Planeamento	Documentos
SDP	Software Development Plan		Documentos
SVP	Software Verification Plan		Documentos
SCMP	Software Configuration Management Plan		Documentos
SQAP	Software Quality Assurance Plan		Documentos
SRS	Software Requirements Standards		Documentos
SDS	Software Design Standards		Documentos
SCS	Software Coding Standards		Documentos
SRD	Software Requirements Data	Desenvolvimento	Documentos
SDD	Software Design Description		Documentos
	Source Code		
	Executable Object Code		
SVCP	Software Verification Cases and Procedures	Verificação	Documentos
SVR	Software Verification Results		Registos
SVCP	Software Verification Cases and Procedures	Gestão de configuração	Documentos
SVR	Software Verification Results		Registos
SQAR	Software Quality Assurance Records	Garantia de qualidade	Registos
SAS	Software Accomplishment Summary		Documento

Tabela 2-2: Documentos necessários para cada fase

2.6. Projecto OSPF RTMC

O OSPF é dividido em duas componentes, a AF (*Algorithmic Facility*) e o RTMC (*Real Time Monitoring and Control*). A componente AF é responsável pelo cálculo algorítmico do OSPF, enquanto o RTMC é responsável por actuar como uma ponte entre o hardware e o AF.

O projecto OSPF RTMC (*Orbit determination and time Synchronization Processing Facility, Real Time Monitoring and Control*) é um projecto do sistema Galileo pertencente à componente de algoritmos de navegação. A função do OSPF é disponibilizar informação de navegação aos utilizadores do sistema Galileo. Para tal, o OSPF irá computar um conjunto de determinações de órbitas para cada satélite Galileo, disponibilizando assim os serviços OS (*Open Service*), SOL (*Safety-of-life Service*) e PRS (*Public Regulated Service*).

O resultado da computação do OSPF consiste num conjunto preciso de efemérides, qualidade de serviço nas métricas, modelos dos temporais (relógios), ajustes na propagação do sinal pela ionosfera e ajustes no cálculo de atrasos na difusão para grupos. A computação dos dados de navegação é efectuada pelo OSPF baseada na recepção de informação dos satélites, em bruto, recebida a uma frequência de 1Hz. Esta informação é enviada pelos satélites e recebida por um conjunto de 50 GSS espalhados pelo planeta terra que a reenviam para o GCC, onde se encontra o OSPF.

Assim, e em detalhe, o OSPF deverá processar a informação necessária dentro do prazo estabelecido, sendo este processamento caracterizado por:

- Aquisição da informação proveniente dos GSS's;

- Pré-processamento e filtragem da informação proveniente dos GSS's;

- Estimativas dos parâmetros de navegação;

- Computação da qualidade de serviço;

- Monitorização e controle interno do OSPF;

- Cálculo dos desvios dos relógios espaciais em relação à referência terrestre;

- Cálculo instantâneo da posição de cada satélite da constelação em relação à referência terrestre;

- Computação precisa das efemérides de todos os satélites da constelação;

- Cálculo de modelos paramétricos de modo a prever desvios nos relógios dos satélites;

- Cálculo da precisão do sinal ao longo do espaço percorrido.

Esta secção tem como objectivo descrever o projecto OSPF ao nível de *standards* aplicados, organização e estrutura do projecto, processos de implementação e tecnologias usadas.

2.6.1. Standards aplicados ao OSPF RTMC

O OSPF, sendo um projecto Galileo, tem de se reger pelos standards de projectos Galileo (ver secção 2.5). Na sua estrutura este projecto é constituído por vários componentes, como será explicado mais adiante. A menor unidade de cada um destes componentes é o módulo, e a este é atribuído um SW-DAL. O SW-DAL atribuído reflecte a exigência de "esforço" necessário na produção de um determinado módulo. Por exemplo, o módulo RTOS_Rtos é DAL-C, ou seja, este módulo contém um conjunto de requisitos que têm de ser garantidos, como por exemplo, a impossibilidade de usar alocação de memória dinamicamente. No caso de um módulo DAL-E, como o XML_Parser, a mesma reserva não se aplica, e é possível usar memória dinâmica.

Os requisitos a seguir durante a todo o projecto encontram-se especificados no documento GSWS (Galileo Software Standards). Este documento especifica para cada requisito, qual o SW-DAL que se aplica.

O projecto OSPF, para além dos standards dos projectos Galileo, tem de respeitar uma série de metodologias de gestão e codificação ditadas pelos nossos "*Contractors*". No que diz respeito ao processo de codificação, as especificações são ditadas pelo documento SCS (*Software Coding Standards*), documento criado pelo nosso "*Contractor*", a GMV neste caso. O SCS especifica normas e requisitos para codificação, como por exemplo:

- Nomenclatura de tipos, variáveis, funções, métodos, etc;

- Número máximo de linhas de código por ficheiros;

- Número máximo de profundidade numa função;

- Complexidade do código.

2.6.2. Processo de desenvolvimento

O projecto seguiu a metodologia de desenvolvimento de software em cascata. O trabalho reportado neste relatório foi desenvolvido maioritariamente durante a fase de implementação. O desenvolvimento do código fonte, os testes unitário e as verificações fazem obrigatoriamente parte desta fase, como é ilustrado na Figura 2-2.

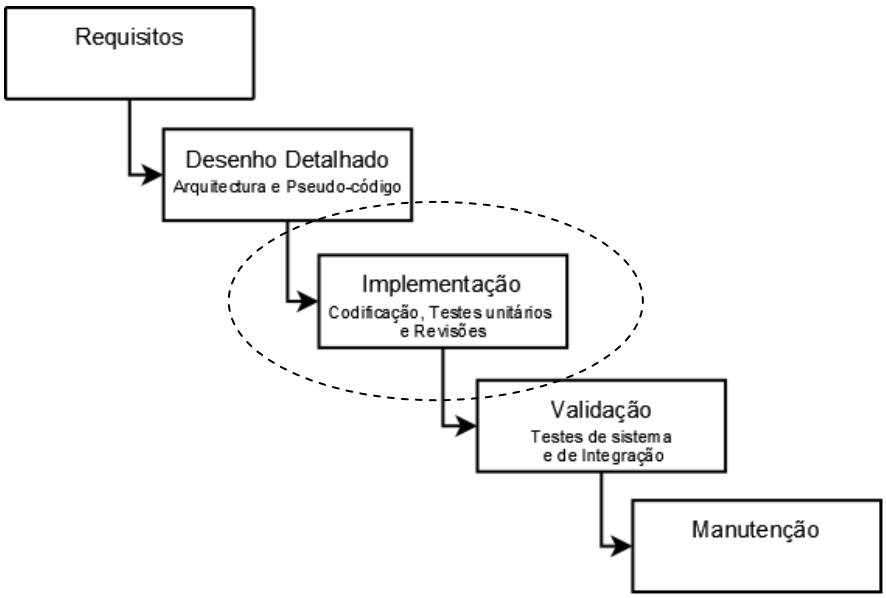

Figura 2-2: Modelo de Desenvolvimento de software em cascata

Durante a fase de Implementação o funcionamento normal do processo de codificação é desenvolvido segundo as fases indicadas na Figura 2-3.

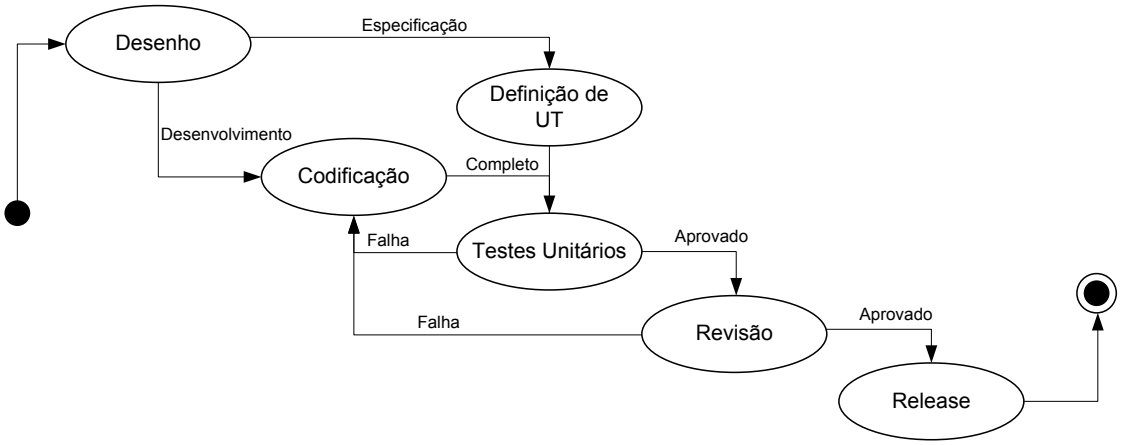

Figura 2-3: Principais fases da Implementação

Quando se dá o início da fase de codificação, em paralelo começam a ser definidos os SUP's (*Software Unit Test Plan*) a partir da fase de desenho detalhado anteriormente efectuado. O desenho detalhado define a forma como os testes unitários serão implementados. Quando a fase de codificação termina dá-se o início da implementação dos testes unitários e caso se encontrem falhas, volta-se novamente à etapa de codificação. Se a etapa de testes unitário não acusar falhas, pode assim então passar-se à etapa de revisão, onde se revê a etapa de codificação e de implementação de testes. A etapa de revisão segue assim a mesma filosofia da etapa de implementação de testes, onde caso se encontrem problemas, volta-se à etapa de codificação.

2.6.3. Estrutura, organização e tecnologias

O processo de desenvolvimento OSPF RTMC está estruturado em duas componentes, a componente de software e a componente de hardware. A componente de software é constituída por:

- Código de desenvolvimento do OSPF RTMC;
- SDK da LynuxWorks [5], o sistema operativo de tempo-real escolhido para executar o elemento OSPF, que contém o compilador, bibliotecas, serviços e o código do *kernel* do LynxOS-178b para a arquitectura de hardware específica;
- Bibliotecas externas, como ASN.1 [11], libxml2 [12] e Net-SNMP [13], necessárias ao funcionamento, codificação e testes do OSPF RTMC.

A componente de hardware é constituída por:

- Máquinas pessoais usadas no desenvolvimento do projecto, tipicamente PC's com Windows XP/Vista;
- Servidor do projecto com Windows XP;
- Bastidor do OSPF, que será a máquina final onde será executado o OSPF.

A Figura 2-4 ilustra mais em especifico qual o principal software e hardware usado no desenvolvimento do OSPF RTMC, dividindo essas duas componentes por clientes/utilizadores, servidor e bastidor do OSPF.

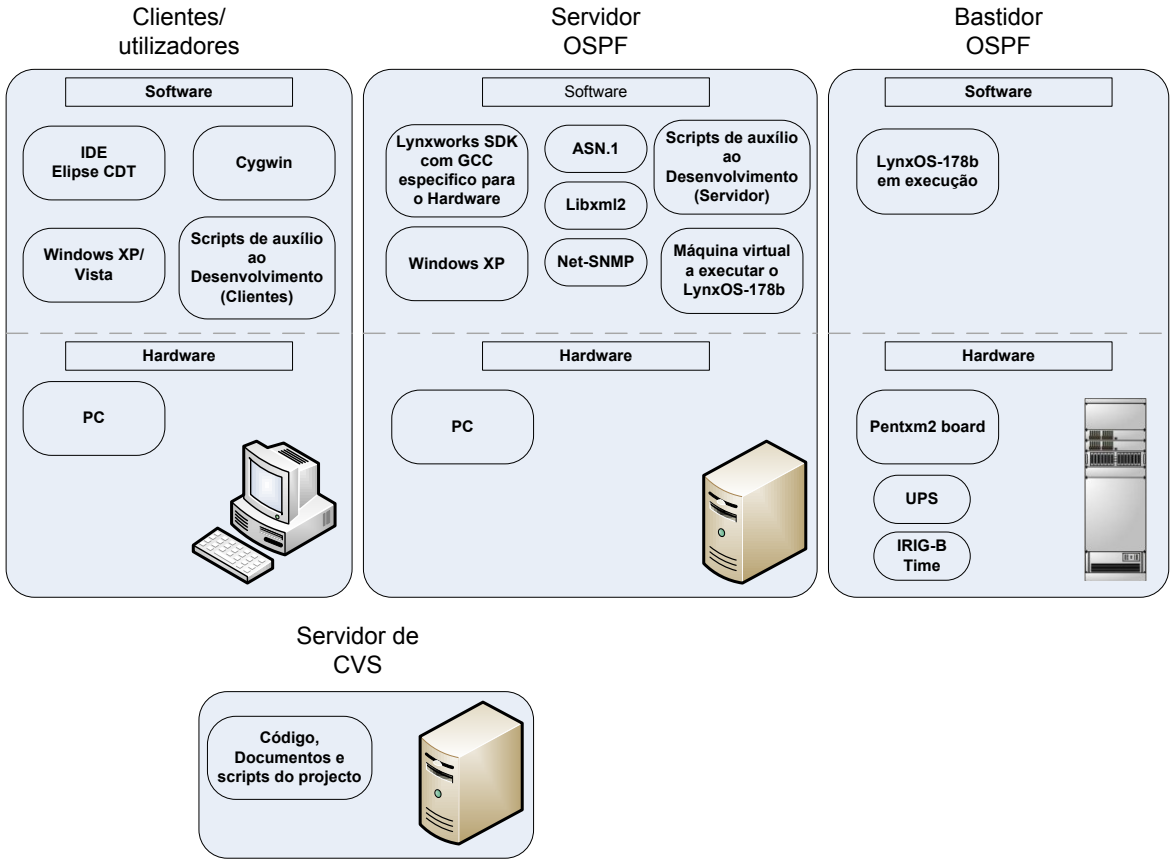

Figura 2-4: Estrutura do software e hardware usado

Durante o desenvolvimento a linguagem de programação usada para codificação do OSPF foi o ANSI C [20] e como IDE foi usado o Eclipse CDT [21] e/ou Notepad++ [22]. Este desenvolvimento era efectuado do lado do cliente e provisoriamente compilado também deste lado. O compilador de C usado para este propósito foi o do Cygwin [23], que é um software livre que emula o ambiente Linux em Windows. Este software foi usado para possibilitar a codificação e compilação local para cada cliente, eliminando desde início erros simples antes de passar a uma compilação final no compilador da LynuxWorks.

Visto ser necessário compilar e ligar o código com um compilador remoto, localizado no servidor, foi construído um conjunto de scripts que possibilitavam a operação de compilação e execução do código no bastidor do OSPF ou na máquina virtual. A linguagem de programação usada na implementação destes scripts foi o Python [24], que é extremamente portável e de simples codificação, ideal para o pretendido. Deste modo foi possível agilizar e centralizar o processo de codificação. Assim foi construída uma aplicação cliente-servidor que permitia a compilação remota e ver os resultados. A aplicação cliente residia em cada PC local, estando a aplicação servidora no servidor do OSPF. Deste modo foi possível centralizar todo o desenvolvimento e apenas utilizar uma licença do software da LynuxWorks.

O servidor, representado na Figura 2-4, é principalmente constituído pelo SDK da LynuxWorks que contém o compilador específico, pelas bibliotecas de ASN.1 (*Abstract Syntax Notation One*) [25], Libxml2 [26] e Net-SNMP (*Simple Network Management Protocol*) [27], pelos scripts para auxiliar a codificação e por uma máquina virtual que executa o LynxOS-178b.

O bastidor do OSPF apenas chegou à Skysoft a meio da fase de implementação. Dado este facto, uma máquina virtual (Microsoft Virtual PC) foi inicialmente e provisoriamente usada para executar o RTOS LynxOS-178b e testar e executar o código desenvolvido, o que foi essencial à fase de desenvolvimento de código e de implementação de testes. Para executar o LynxOS-178b na máquina virtual, foi utilizado um protocolo de *boot* via ethernet (Protocolo PXE - DHCP e TFTP). Este protocolo permitia enviar uma imagem do RTOS para ser executado em outra máquina. Quando o bastidor chegou, a máquina virtual foi então substituída pelo bastidor, convergindo assim o projecto para a arquitectura de desenvolvimento e testes finais.

O código do OSPF encontra-se armazenado em CVS (*Concurrent Version System*) e está organizado de forma modular em subsistema, pacotes e módulos. Esta organização permite agregar os serviços comuns em áreas de fácil e intuitivo acesso. Os subsistemas são: Controlo de dados; Serviços do elemento; Controlo do elemento; Controlo de interfaces; Monitorização do elemento.

A Figura 2-5 ilustra um exemplo da organização do OSPF que segue a estrutura modular acima indicada. Desta forma, é possível manter todo o código com as alterações efectuadas e a efectuar sem influenciar significativamente outros módulos não directamente envolvidos, facilitando o processo de organização e manutenção, aumentando a funcionalidade do projecto.

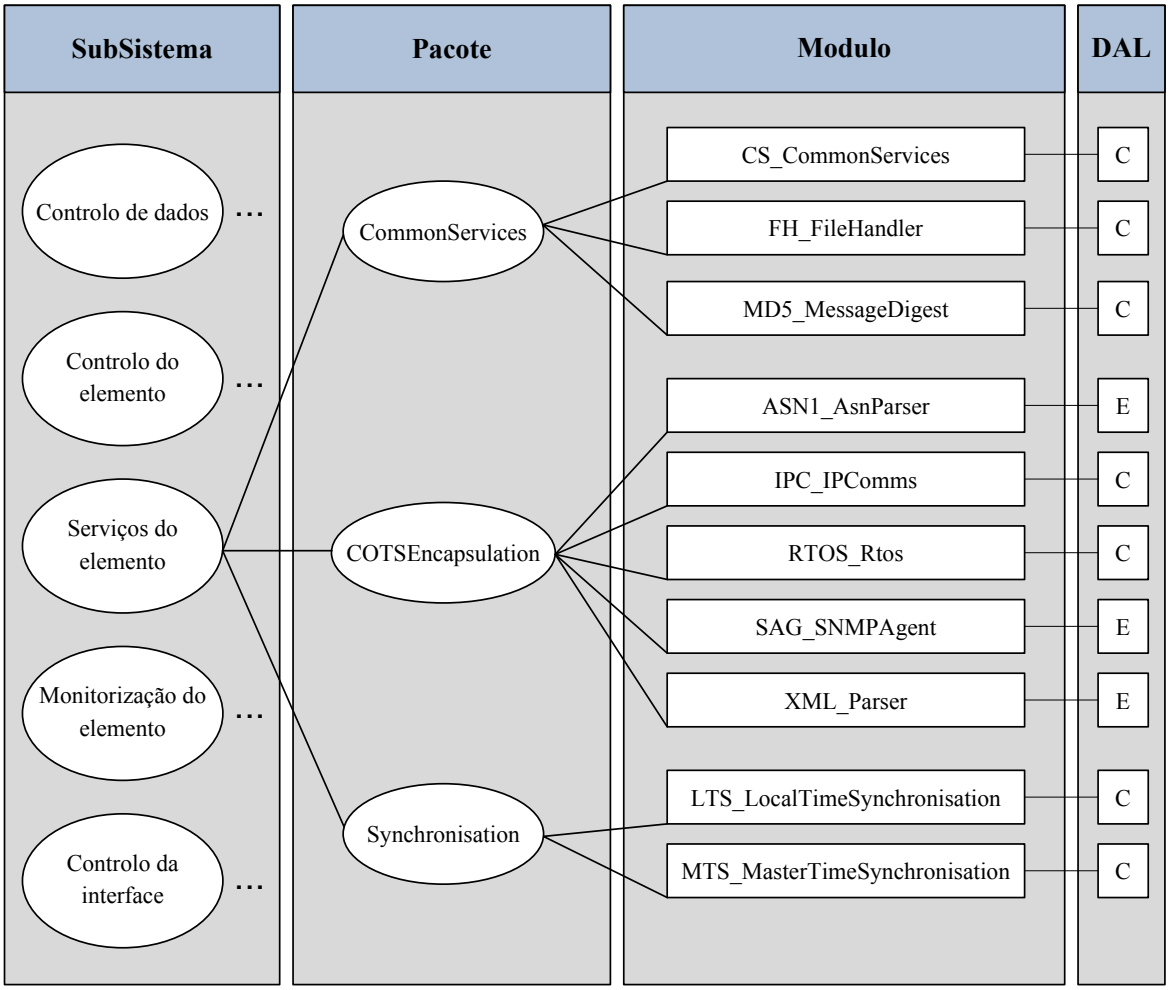

Figura 2-5: Exemplo da estrutura do código do OSPF

Os scripts de auxílio à codificação acima referidos, a máquina virtual com o LynxOS-178b e todo o processo *boot* via DHCP e TFTP foram processos construídos por mim.

2.6.4. Processo de codificação

Depois de concluída a fase de desenho detalhado do OSPF, foi necessário criar toda a infra-estrutura para suportar o desenvolvimento do sistema operativo LynxOS-178b e a fase de produção de código. Esta necessidade é causada pela obrigação de compilar o código para a arquitectura do LynxOS-178b e para uma plataforma de hardware específica (pentxm2), onde o OSPF irá ser executado em fase de produção. Essa infra-estrutura encontra-se representada na Figura 2-6.

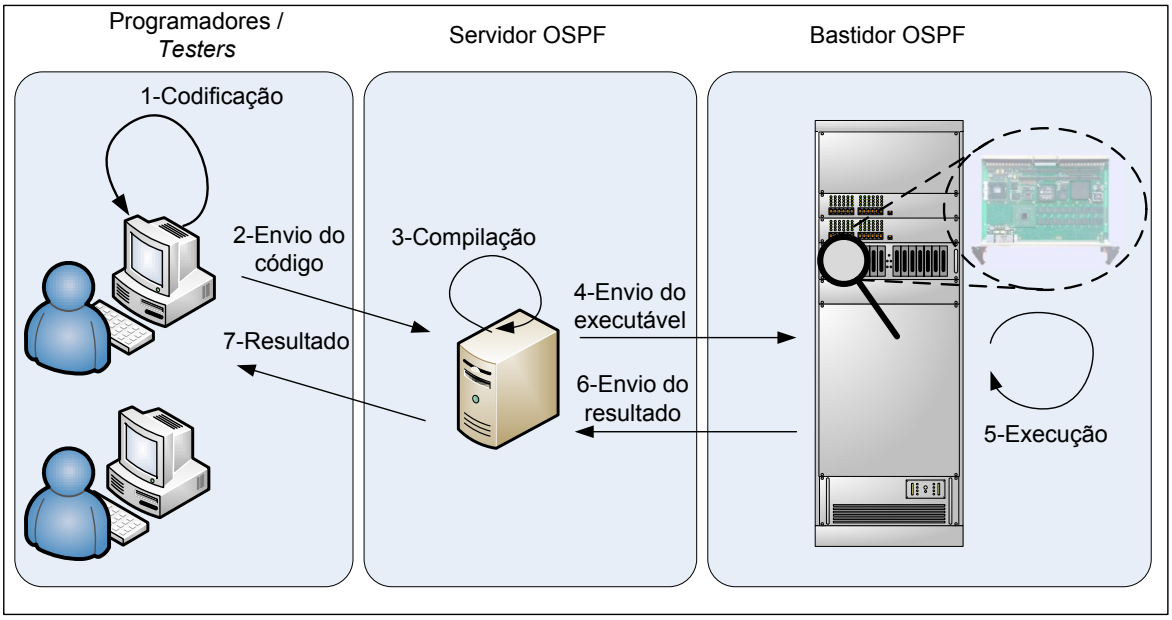

Figura 2-6: Desenvolvimento *Cross-platform*

A Figura 2-6 ilustra como na prática o processo de codificação, incluindo implementação de testes unitários, decorre. No processo de codificação, o código é enviado do cliente para o servidor, compilado no servidor e é devolvido ao cliente. Caso seja necessário, o código pode ser executado na máquina de *target*, inserida no bastidor. Na implementação dos testes unitários, o funcionamento é idêntico ao descrito anteriormente, com a diferença de que o código é sempre executado na máquina de *target*, para que se conheça de facto o comportamento de uma dada função a ser testada.

No entanto, este processo é mais complexo e demorado do que a compilação local, apesar de estar optimizado. Dado este facto, o código é compilado e executado também localmente por cada programador/*tester* para agilizar todo o processo de codificação.

3. Sistemas operativos de tempo real

3.1. Introdução

Com a proliferação do uso de sistemas informáticos na sociedade actual, as aplicações desenvolvidas com requisitos estritos de tempo real são cada vez mais correntes. Estas aplicações variam muito em relação à complexidade e às necessidades de garantia nas restrições temporais. As aplicações variam desde aplicações muito simples como controladores embutidos em unidades domésticas, até aplicações complexas como sistemas industriais de alta precisão, sistemas de controlo de tráfego aéreo ou sistemas de localização geográfica.

Um RTOS (*Real-Time Operating System*) é um sistema operativo que fornece às aplicações mecanismos e características essenciais à concretização de sistemas de tempo real. Um sistema operativo de tempo real com uma aplicação que não é de tempo real, não formam um sistema de tempo real.

Ao longo dos anos mais recentes têm sido desenvolvidos vários sistemas operativos de tempo real, procurando colmatar e aperfeiçoar as falhas dos anteriores sistemas e procurando também acompanhar a evolução constante do hardware. A evolução impõe novas dificuldades, novos desafios, novas metas. Para os ultrapassar, os S.O.'s têm de evoluir em paralelo de forma a responder às dificuldades em tempo útil.

Estes S.O's são normalmente aplicados a sistemas embebidos de tempo real nomeadamente em sistemas em que é necessário assegurar fiabilidade e capacidade na recuperação de falhas.

Este capítulo faz uma breve introdução às características gerais de sistemas operativos de tempo real, passando ainda por dois importantes *standards* ligados à construção de sistemas de tempo-real estrito. Por fim apresentam-se três sistemas operativos de tempo real muito utilizados em sistemas embebidos.

3.2. Características de um RTOS

Um sistema operativo de tempo real é tipicamente um sistema multitarefa direccionado para aplicações de tempo real. Neste, os algoritmos de escalonamento de tarefas adquirem uma grande importância pois irão ditar o cumprimento de prazos de computação, algo que é necessário alcançar para um comportamento determinista no sistema final.

Os sistemas de tempo real operam em ambientes de computação e memória limitados. Os seus requisitos incluem a obrigação de providenciar serviços onde o prazo é limitado e conhecido. Os limites de memória, velocidade e tempo de execução ditam assim as bases de requisitos de um RTOS.

3.2.1. *Kernel*

O *kernel* é o principal elemento de um S.O. Este tem a responsabilidade de providenciar a maior parte dos serviços básicos à camada de aplicação. O *kernel* de um RTOS

disponibiliza uma camada de abstracção à camada de aplicação para que os detalhes mais complexos de hardware e software sejam transparentes. Desta forma, a camada de aplicação, pode assim adquirir um maior grau de independência em relação ao hardware ficando a responsabilidade de interface do lado do RTOS. A Figura 3-1 ilustra a organização das camadas que fazem parte de um sistema embebido de software.

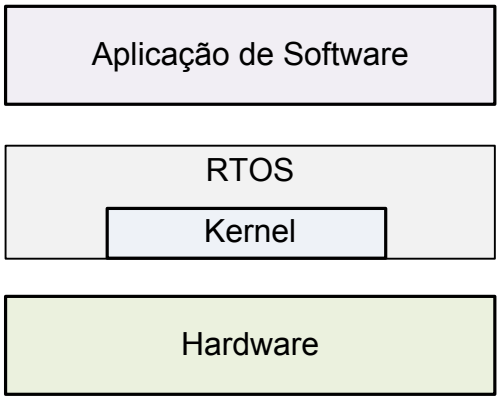

Figura 3-1: Camadas envolvidas num RTOS

3.2.2. Serviços básicos de um *Kernel*

A mais básica categoria de serviços disponibilizada pelo *kernel* é a de gestão de tarefas. É esta categoria de serviços que possibilita que haja um conjunto vasto de aplicações distintas entre si, com objectivos e propósitos diferentes, a serem executadas provavelmente com prazos diferentes. Os serviços incluídos nesta categoria permitem lançar tarefas e, atribuir e modificar as suas propriedades. O principal serviço desta categoria é o escalonamento de tarefas. Este controla a execução das diferentes tarefas das diferentes aplicações, ditando o seu comportamento ao longo do tempo.

A segunda categoria de serviços disponibilizados pelo *kernel* é a comunicação e sincronização entre tarefas. Estes serviços fazem com que seja possível a passagem de informação entre tarefas sem que esta seja corrompida. Possibilitam ainda que tarefas se coordenem para que o seu funcionamento seja cooperativo obtendo uma computação mais eficiente. Se estes serviços não estivessem presentes num RTOS, as tarefas poderiam sofrer corrupções nas suas comunicações e causar conflitos entre tarefas, levando assim a possíveis falhas em todo o sistema.

Dada a importância que os requisitos temporais têm para um sistema embebido crítico, é necessário que serviços temporais estejam disponíveis. Serviços que nos informem sobre características temporais das tarefas, como atrasos, deverão fazer parte de um *kernel*.

A alocação de memória dinamicamente é geralmente uma das categorias de serviços de um *kernel*. Estes serviços oferecem assim às diferentes tarefas capacidade de requisitar memória ao S.O. dinamicamente. No entanto existem casos em que este serviço pode não existir por não ser necessário.

Adicionalmente o *kernel* de um RTOS oferece frequentemente um conjunto de serviços de acessos a dispositivos de entrada e saída. Estes serviços, se disponíveis, providenciam uma camada que organiza e permite aceder a inúmeros dispositivos de hardware que compõem os sistemas embebidos, acesso tal que é efectuado através de *device drivers*.

As categorias de serviços descritas acima encontram-se representadas na Figura 3-2.

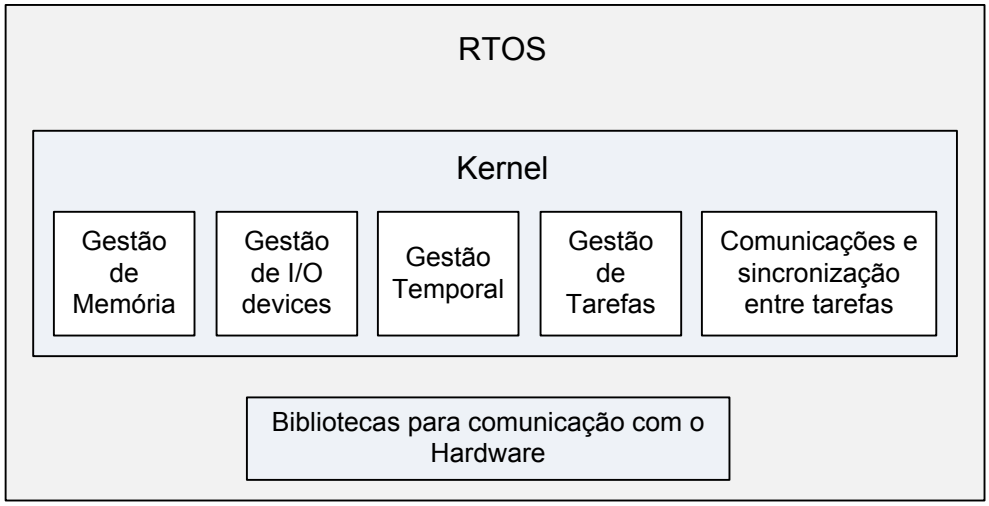

Figura 3-2: Serviços básicos disponibilizados Kernel

3.2.3. Escalonamento de tarefas

O "escalonador" é uma das principais componentes de qualquer S.O. Este têm a responsabilidade de gerir uma lista de tarefas, atribuindo tempo e espaço de computação às diferentes tarefas a executar.

Normalmente, uma tarefa pode estar num de três estados:

Running;

Ready;

Blocked.

As tarefas encontram-se por norma bloqueadas visto que cada CPU apenas pode conter uma tarefa a ser executada de cada vez. O escalonamento traduz-se assim na forma de efectuar a computação das várias tarefas de uma forma eficiente (óptima). A definição de eficiente e óptima varia de sistema para sistema, dependendo dos seus requisitos. Dado este facto, é comum que cada sistema implemente o seu algoritmo de escalonamento específico, que poderá ser apropriado para determinadas tarefas enquanto outros algoritmos são melhores para outras aplicações. Muitas vezes, a escolha de um RTOS adequado passa por escolher o esquema de escalonamento desejado.

São os algoritmos de escalonamento que vão ditar se as diferentes tarefas cumprem os seus requisitos temporais. Estes podem ser classificados estáticos ou dinâmicos, *preemptivos*[1] ou não *preemptivos*, *off-line ou on-line*.

Os algoritmos de escalonamento estático são utilizados em situações onde o trabalho a ser realizado e os requisitos temporais são conhecidos previamente.

---

[1] *Preemptivo* – Tradução livre do termo inglês *preemptive*. Adoptado neste documento que significa que kernel tem controlo sobre o tempo de execução de cada processo podendo atribuir o tempo de execução a outro processo.

Os algoritmos de escalonamento dinâmicos não atribuem prioridades fixas às tarefas. As decisões de escalonamento são tomadas em tempo de execução e as prioridades das tarefas podem mudar com a evolução do sistema. Os critérios para essas decisões variam de algoritmo para algoritmo.

Os algoritmos são ditos *preemptivos* quando em qualquer momento uma tarefa de maior prioridade pode interromper uma tarefa que esteja a executar. Quando tal não é possível, o algoritmo diz-se *não preemptivos*.

Os algoritmos de escalonamento são classificados de *off-line* quando produzem o seu escalonamento em tempo de projecto e de *on-line* quando o escalonamento é efectuado em tempo e execução.

Seja qual for o tipo de escalonamento, o objectivo permanece, as diversas tarefas devem ter um comportamento determinista, ou seja, as restrições temporais têm que ser cumpridas, algo necessário a um sistema de tempo real.

3.2.4. Sistemas baseados em processos e em *threads*

Num sistema baseado em processos ou *threads*, e subentenda-se processo e tarefa como representando o mesmo, os processos protegem e isolam a memória de outros processos, sendo estes dinamicamente alocados em memória. Um processo pode ser dinamicamente carregado para a memória a partir de um executável a ele ligado, localizado no disco rígido ou em memória RAM. Um processo pode ainda requisitar memória dinamicamente visto não haver um limite estabelecido para a execução deste. O único limite é sem dúvida o limite físico.

As aplicações num contexto geral podem ser caracterizadas em dois tipos: sequenciais e concorrentes. A maioria das aplicações desenvolvidas apenas é executada por uma tarefa, sendo portanto chamadas de sequenciais. Este tipo aplicações tem um comportamento linear, onde somente existe um fluxo de controlo durante a sua execução. Uma aplicação concorrente por outro lado é executada simultaneamente por diversas tarefas que cooperam entre si.

Uma forma de programar sistemas concorrentes é utilizar a abstracção de *thread*. As *threads* são processos leves, no sentido em que os seus únicos atributos são os associados ao seu contexto de execução. Os restantes atributos da *thread* são herdados do processo onde é executada. Todas as *threads* de um processo partilham um mesmo espaço de endereçamento, a MMU *(Memory Management Unit*), que não é afectada pela mudança de contexto entre *threads*. Desta forma, a mudança de contexto entre duas *threads* de um processo é mais rápida do que a mudança de contexto entre dois processos.

Normalmente, as aplicações de tempo real são compostas por vários processos e/ou *threads* concorrentes. No entanto as aplicações concorrentes que usam *threads* aumentam a sua eficiência visto não necessitar de mudar de contexto por partilharem a memória do processo que as criou. Pelo contrário, as aplicações que usam vários processos concorrentes podem ver a sua eficiência a decrescer dada a necessidade de permanentes mudanças de contexto.

Dada a utilização e importância do conceito de processo e *threads*, o seu suporte por parte do RTOS surge como um requisito básico. O RTOS deve disponibilizar chamadas de sistema para criar, destruir, suspender, retomar e controlar o escalonamento de processos e *threads*.

3.2.5. Comunicação entre processos

Os sistemas baseados em múltiplos processos podem conter um conjunto de processos em execução concorrente. Estes serão independentes se não comunicarem ou se não interagirem, ou por outro lado, podem ser cooperativos, quando a sua computação afecta a execução de outros processos. Deste modo, processos que partilham informação são processos cooperativos.

Para alcançar esta cooperação é necessária a existência de comunicação entre processos (IPC – *Interprocess communication*). Existem fundamentalmente dois modelos de comunicação entre processos: memória partilhada e troca de mensagens [16].

- No modelo de memória partilhada existe uma zona de memória que é partilhada pelos processos cooperativos. A zona de memória partilhada reside no espaço de endereçamento do processo que a cria. Outros processos que necessitem de comunicar usando a memória partilhada, têm de adicioná-la ao seu espaço de endereçamento. Esta é uma excepção quanto à impossibilidade de um processo poder aceder a um espaço de memória de outro processo. Desta forma os processos podem trocar informação lendo e escrevendo para a memória partilhada. No entanto o S.O. não providencia mecanismos de serialização das acções sobre a memória partilhada, podendo haver dois ou mais processos a escrever ao mesmo tempo, tornando a informação incoerente. Estes mecanismos tipicamente têm de ser implementados ao nível da aplicação.
- No modelo de troca de mensagens a comunicação é efectuada por envio/recepção de mensagens entre os processos cooperativos. Este mecanismo fornece aos processos uma forma de comunicação e de sincronismo das suas acções sem a necessidade de partilhar um mesmo espaço de endereçamento de memória. Este mecanismo é ainda útil em ambientes distribuídos, onde os processos podem residir em diferentes computadores ligados em rede. Este modelo contém basicamente duas funções, a de enviar e a de receber.

A partir destes dois modelos, foram construídos vários serviços de comunicação e sincronização entre processos. Estes serviços permitem gerir o acesso concorrente dos processos a recursos do RTOS de forma controlada. Desta forma os processos não necessitam de ficar bloqueados à espera que o recurso seja libertado. Os serviços são:

- Serviço de sinais - serviço de envio de notificações assíncronas para um processo de modo notificá-lo de que ocorreu um dado evento. Quando um evento é enviado para um processo, o RTOS interrompe a normal execução do processo, que no entanto, só pode ser interrompido durante uma instrução não atómica.
- Serviço de *sockets* - permite a comunicação entre dois processos localizados no mesmo ou em diferentes *hosts*.
- Serviço de semáforos - método clássico para restringir o acesso a recursos partilhados como, memória partilhada num ambiente multi-aplicacional. Os semáforos podem conter diferentes variantes, como por exemplo, semáforo contador ou semáforo binário, que é conhecido por Mutex (*Mutual exclusion*).
- Serviço de filas de mensagens – serviço que providência um protocolo assíncrono de comunicação. Neste, os processos que enviam e recebem mensagens não necessitam de estar sincronizados. A mensagem é guardada até o receptor da mensagem receber a mensagem.

3.2.6. Particionamento ou máquinas virtuais

Existem RTOS's que suportam partições ou máquinas virtuais (representam o mesmo conceito). Nesta situação, o RTOS tem a capacidade de dividir a memória e o tempo de processamento em partições estaticamente alocadas e definidas de forma fixa. O objectivo é dividir o tempo de processador e memória pelas diversas partições de modo a que o comportamento global do sistema se assemelhe a subsistemas completamente isolados. As partições devem ser definidas em tempo de configuração e após esse período, cada partição estará limitada apenas à memória e tempo de processamento fixado. Não deverá ser possível em tempo de execução proceder a qualquer alteração nestes parâmetros.

O ambiente dentro de cada partição deve ser semelhante ao ambiente de um RTOS sem partições, e caso este suporte múltiplas tarefas e múltiplas *threads* então também as partições o deverão suportar [17].

3.3. POSIX

3.3.1. Introdução

O POSIX (*Portable Operating System Interface*) é uma família de standards desenhados para garantir a portabilidade do código fonte das aplicações através do hardware e de sistemas operativos. O POSIX foi desenvolvido pelo IEEE (*Institute of Electrical and Electronics Engineers*) e é reconhecido pelo ISO (*International Organization for Standardization*) e pelo ANSI (*American National Standards Institute*).

O POSIX define as interfaces do S.O. mas não a sua implementação e assim sendo, é possível definir o POSIX como uma API (*Application Programming Interface*), onde se definem conceitos que formam por si só uma plataforma de programação.

Grande parte dos S.O. implementa as chamadas de sistema através de interrupções de software. O que o POSIX faz é padronizar a sintaxe das funções de bibliotecas que por sua vez executam as interrupções de software que são as verdadeiras interfaces do *kernel*. Esta necessidade existe porque a forma de gerar interrupções depende do processador em questão, sendo a padronização destas interrupções complexa de alcançar. Dado este facto, é mais adquado padronizar as interrupções ao nível das bibliotecas de métodos e funções, oferecendo uma maior abstracção.

A linguagem de programação originalmente usada na definição do POSIX foi o C. No entanto, os serviços do POSIX são definidos em linguagem natural o que torna mais simples a sua compreensão pelos utilizadores. Posteriormente surgiram outras implementações baseadas noutras linguagens de programação. As linguagens C++, Java, Python, etc., são exemplos de linguagens de programação que disponibilizam bibliotecas que suportam este standard, que por vezes, mais não faziam que encapsular o que originalmente foi implementado na linguagem C.

O standard POSIX tem vindo a evoluir e é agora constituído por diversas componentes. A sua primeira versão apenas definia uma API para chamadas de sistemas gerais. Posteriormente, novos componentes foram adicionados procurando padronizar serviços que começavam a ser regularmente usados em diversas aplicações. Foi o caso de serviços relacionados com sistemas de tempo real e sistemas embebidos.

Um dos principais objectivos da criação deste standard foi o de facilitar a portabilidade de aplicações entre sistemas operativos que implementem este standard. Em casos em que se constrói um aplicação usando uma API proprietária, essa aplicação fica geralmente presa às primitivas e parâmetros desse S.O., e para mudar de sistema implica um esforço imenso e constantes incompatibilidades. Outra das vantagens que apresenta é permitir que se definam guias para quem desenvolve hardware para que assim se evolua e se optimize os dispositivos, aumentando o desempenho e fiabilidade. Este standard é assim de grande importância para aplicações comerciais e/ou governamentais

A família de standards POSIX é extensa e aplicada aos mais diversos campos da computação. Neste relatório, apenas nos restringiremos às famílias que se enquadrem no âmbito deste.

3.3.2. A família POSIX

A documentação do POSIX está dividida em diversos documentos que normalmente representam áreas da computação distintas. Nesta secção serão apresentados três standards relevantes no contexto de sistemas de tempo real.

O POSIX.1 (IEEE Std 1003.1-1990) descreve interfaces e funcionalidade para serviços básicos do sistema operativo, abstraindo os detalhes de implementações do programador/utilizador do sistema operativo. Isto possibilita que os diversos sistemas operativos disponibilizem pelo menos um mesmo conjunto de serviços independente da sua implementação e arquitectura mas semelhantes na interface e funcionalidade que apresentam. O POSIX.1 inclui definições de interfaces para gestão de tarefas, dispositivos, sistemas de ficheiros, comunicação básica entre tarefas. Esses serviços encontram listados na Tabela 3-1.

[*POSIX.1] define uma interface standard para o sistema operativo e um ambiente para suportar a portabilidade ao nível código fonte. Deve ser usado por quem desenvolve aplicações ou por quem implementa os sistemas. [IEEE 1990]*

Process Creation and Control	Serviços que permitem a criação e controlo de processos.
Signals	Serviços que permitem a comunicação assíncrona entre processos.
Floating Point Exceptions	Mecanismos que permitem a detecção de erros relacionados com *Floating Point Exceptions*.
Segmentation Violations	Mecanismos que permitem a detecção de erros relacionados com *Segmentation Violations*.
Illegal Instructions	Mecanismos que permitem a detecção de erros relacionados com instruções ilegais.
Bus Errors	Mecanismos que permitem a detecção de erros relacionados

	com acessos ilegais a recursos.
Timers	Serviços temporais com diferentes resoluções.
File and Directory Operations	Serviços para manipulação sobre o sistema de ficheiros e directórios.
Pipes	Mecanismos para efectuar a comunicação entre processos.
C Library (Standard C)	Implementação do standard utilizando a linguagem de programação C.
I/O Port Interface and Control	Interface geral que providencia uma comunicação

Tabela 3-1: POSIX.1

Posteriormente foram adicionadas duas extensões ao POSIX.1, o POSIX.1b (POSIX tempo real, 1003.1b) e o POSIX.1c (POSIX *threads,* 1003.1c). Estas duas extensões são de extrema importância, abundantemente usadas em sistemas de tempo real e sistemas embebidos. A extensão de tempo real inclui especificações para escalonamento baseado em prioridades, sinais de tempo real, relógios e temporizadores, semáforos, envio de mensagens, memória partilhada, I/O assíncrono e síncrono e memória partilhada. A extensão de *threads* inclui especificações para criação, controlo e destruição de *threads*, escalonamento de *threads*, sincronismo de *threads* e tratamento de sinais. Este último standard possibilita a existência de múltiplas *threads* no mesmo espaço de endereçamento, algo extremamente importante ao desenvolvimento de aplicações de tempo real complexas.

A Tabela 3-2 e Tabela 3-3 listam as especificações de funcionalidades para cada uma dos standards POSIX.1b e POSIX.1c.

Priority Scheduling	Possibilidade de atribuir prioridades a processos.
Real-Time Signals	Suporte a comunicação assíncrona de tempo-real através de sinais.
Clocks and Timers	Suporte a temporizadores de tempo-real.
Semaphores	Suporte a semáforos de tempo-real.
Message Passing	Suporte a envio de mensagens entre processos.
Shared Memory	Suporte a partilha de memória entre processos.
Asynch and Synch I/O	Suporte a comunicações assíncronas e síncronas de entrada e saída.
Memory Locking	Mecanismos para prevenir atrasos de paginação por parte de processos de tempo real.

Tabela 3-2: POSIX.1b

Thread Creation, Control, and Cleanup	Serviços que permitem a criação, controlo e destruição de threads.
Thread Scheduling	Serviços que permitem a especificação do tipo de escalonamento aplicado às threads.
Thread Synchronization	Serviços que permitem a sincronização entre threads.
Signal Handling	Serviços que permitem a recepção de sinais.

Tabela 3-3: POSIX.1c

3.3.3. **Escalonamento**

Como já foi anteriormente referido, um dos aspectos centrais em sistemas de tempo real é o escalonamento. Dado este facto, o POSIX suporta diferentes tipos e políticas de escalonamento, sendo possível definir prioridades para as diferentes tarefas. As prioridades podem ser definidas em tempo de execução, estando definido que no mínimo deverão ser suportados 32 níveis de prioridades, não estando definido no entanto, um número exacto e assim sendo, as implementações podem diferir. Juntamente com as prioridades, as políticas de escalonamento definem como são escalonadas as *threads* com a mesma prioridade. Para tal, o POSIX disponibiliza três políticas:

- SCHED_FIFO (*First In-First Out scheduling*), que é preemptivo baseado em prioridades estáticas;
- SCHED_RR (*Round Robin scheduling*), semelhante ao SCHED_FIFO com a diferença de que cada tarefa apenas executa um tempo máximo especificado por *quantum*;
- SCHED_OTHER (*Default Linux time-sharing scheduling*), apenas pode ser usado para prioridade estática a 0 (zero), oferecendo o escalonamento standard de Linux de partilha temporal;

De início as *threads* são criadas para competir com todas as outras *threads* do sistema pelo tempo de processador. No entanto, a competição pode também dar-se entre processos, sendo então também necessário especificar políticas para divisão do tempo de processador entre diversas tarefas.

3.4. ARINC 653

3.4.1. Introdução

O ARINC 653 (*Avionics Application Standard Software Interface*) é uma especificação de software para particionamento no espaço e no tempo (ver secção 3.2.6 sobre particionamento em RTOS). Este standard define uma API que segue a arquitectura IMA (*Integrated Modular Avionics*). Esta especificação faz parte da série de standards ARINC

600 (*Digital Aircraft and Flight Simulators*). O ARINC 653 é utilizado em sistemas seguros e de missão crítica, em particular na indústria da aviação.

O ARINC 653 define uma APEX (*Application/Executive*) para particionamento no tempo e no espaço para que múltiplas aplicações possam partilhar um único processador e memória através de partições ou VM (*Virtual Machines*). Cada partição num sistema ARINC 653 é uma aplicação separada que usa um espaço de memória e processamento dedicado a essa partição. A interface APEX pode ser do ponto de vista da aplicação de software como uma especificação de uma linguagem de alto nível e do ponto de vista do S.O. como uma definição de parâmetros e mecanismos de entrada.

A APEX tem como requisito funcional o de garantir que, caso uma falha ocorra numa dada partição, essa falha não influencia outras partições. No entanto deve oferecer ainda um conjunto de serviços básicos, sendo estes:

- Gestão de partições
- Gestão de processos
- Gestão temporal
- Comunicações inter-partição
- Comunicações intra-partição
- *Health Monitoring*

Nesta especificação estão também definidos requisitos e listas de serviços que permitem que a aplicação controle o escalonamento, comunicação e informação sobre o seu estado de processamento. No entanto não é do intuíto desta especificação definir detalhes de implementação de software ou de hardware [18].

3.4.2. Arquitectura do ARINC 653

A arquitectura do standard ARINC 653 é ilustrada pela Figura 3-3. A partir desta figura podemos observar que a arquitectura ARINC 653 contém uma camada de aplicação onde cada aplicação é executada num contexto confinado, o qual se define de partição. A camada de aplicação pode ainda incluir um conjunto de partições de sistema para gerir interacções com hardware específico, como é o caso dos acessos a uma mesma placa de Ethernet por parte de múltiplas partições. Este suporte é da responsabilidade da camada software do *kernel*. Esta camada deve comportar interfaces para o hardware e controladores dos dispositivos para que assim sejam acessíveis por camadas superiores.

As aplicações executadas em partições consistem geralmente em um ou mais processos que apenas podem usar serviços disponibilizados pela interface APEX. No entanto, uma partição de sistema pode usar funções específicas fornecidas pela camada de software do *kernel* e as fornecidas pela APEX.

O ambiente de execução fornecido pelo *kernel* do S.O. deve providenciar um conjunto relevante de serviços como escalonamento e gestão de processos, gestão de relógios e temporizadores, comunicação e sincronização inter-processo.

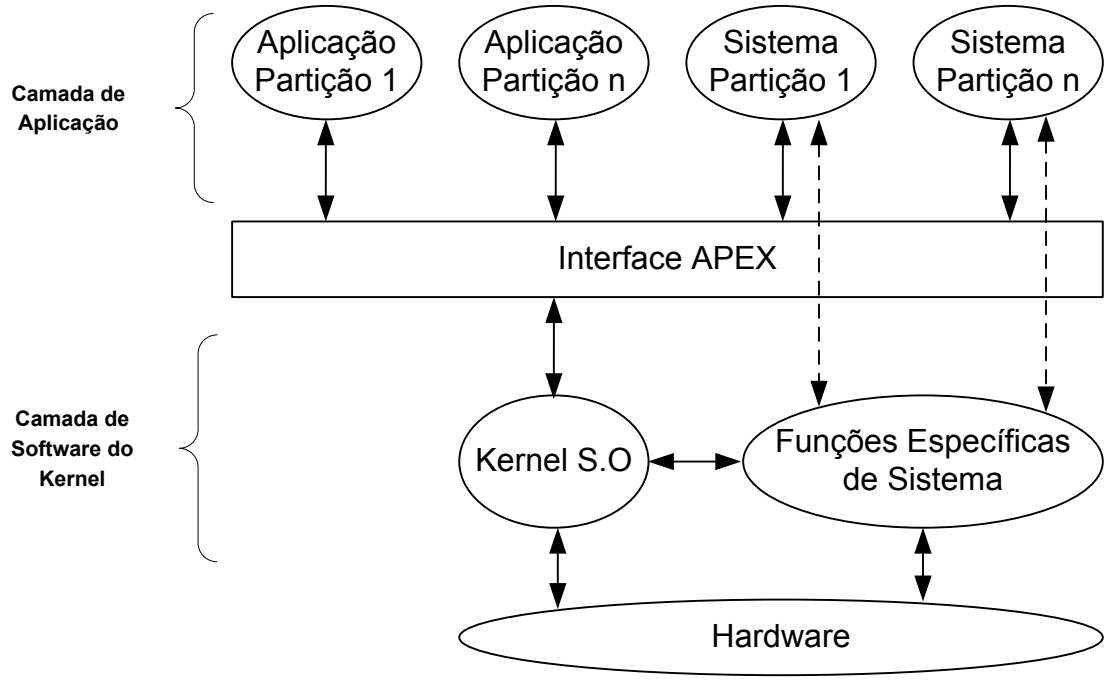

Figura 3-3: Relações entre os componentes do ARINC 653

3.4.3. Serviços ARINC 653

O standard específica um conjunto de serviços que possibilitam a gestão e interacção com o exterior e interior das partições. Um S.O compatível com ARINC 653 deverá implementar pelo menos esse conjunto de serviços.

3.4.3.1. Gestão de partições

A ideia central à filosofia do ARINC 653 é o conceito de particionamento, onde as funções do *kernel* são particionadas de acordo com o espaço (memória) e tempo (tempo de processador) correspondentes.

O particionamento espacial garante que não é possível a uma aplicação aceder ao espaço de memória (código e dados) de uma outra aplicação a ser executada numa outra partição. O particionamento temporal garante que as tarefas numa partição não influenciam o cumprimento temporal de outras tarefas de outras partições e isto é conseguido por um escalonamento FCS (*Fixed Cycle Scheduling*), que é um escalonamento definido em tempo de configuração das diferentes partições. Desta o escalonamento é determinista visto que cada partição tem um tempo fixo de computação, libertando os recursos do sistema após esse tempo.

Assim as partições ARINC podem suportar diferentes aplicações de diferentes níveis de criticalidade e caso uma partição falhe, essa falha não influencia o funcionamento de outras partições.

3.4.3.2. Gestão de processos

De acordo com as características associadas às partições, estas podem, independentemente, conter processos concorrentes. Os processos definidos em partições ARINC são processos

ARINC, obedecendo portanto a definições especiais. Estes processos são constituídos por atributos especiais que têm de ser conhecidos ao S.O.

Os recursos usados na gestão dos processos são indicados estaticamente em tempo de configuração. Quando a partição é inicializada, os processos são criados e os recursos alocados. No entanto é de ter em conta que cada processo é apenas criado uma e só uma vez no tempo de vida da partição que o hospeda, e posteriormente é possível que passe por quatro dos seguintes estados:

- *Dormant,* estado em que o processo deixa de receber recursos do sistema. O processo pode alcançar este estado antes ou após a sua execução;

- *Ready*, estado em que o processo está pronto para ser escalonado.

- *Running*, estado em que o processo corrente está em execução;

- *Waiting*, estado em que o processo fica impedido de receber recursos até um evento particular ocorrer. Este evento pode causado por um atraso, um semáforo que bloqueia um recurso, a espera de uma mensagem, etc.

Cada processo contém uma prioridade e essa prioridade é usada no escalonamento ao nível do S.O. O tipo de escalonamento usado na gestão dos processos é baseado em prioridades e é preemptivo. Assim o S.O. gere a execução do ambiente da partição, decidindo qual o processo a executar num dado momento, baseando-se na sua prioridade. Caso a prioridade dos processos seja idêntica o S.O. selecciona o mais antigo.

O número de processos existentes numa dada partição é estático. Não é possível lançar processos dinamicamente. Estes deverão ser definidos em tempo de configuração da partição e existirão sempre num dos estados acima indicados até ao término do sistema.

3.4.3.3. Gestão temporal

O S.O. deve fornecer tempo de processamento para o escalonamento das diversas partições, prazos, periodicidades tendo em conta atrasos no escalonamento de processos e *timeouts* para comunicação intra-partição e inter-partição. A gestão do tempo tem de ser eficiente para suportar todas estas unidades temporais. Para tal o S.O. providencia serviços temporais de forma a alcançar os objectivos de um sistema de tempo real.

3.4.3.4. Comunicações inter-partição

O S.O. deve oferecer um conjunto de serviços responsáveis por efectuar a comunicação entre processos hospedados em partições diferentes. Toda a comunicação é efectuada através de "mensagens". Uma "mensagem" é definida como um bloco de dados contínuo de tamanho finito. Esta é enviada de uma origem para um ou mais destinos. O destino, no entanto não é um processo, mas sim a partição.

Deverão pelo menos existir dois tipos de serviços de comunicação inter-partição, o *SAMPLING PORT* e o QUEUING PORT.

3.4.3.5. Comunicações intra-partição

O S.O. deve oferecer um conjunto de serviços responsáveis por efectuar a comunicação entre processos hospedados na mesma partição. Os mecanismos de comunicação são *buffers*, *blackboards*, semáforos e eventos. Os serviços de *buffers* e *blackboards* são

providenciados pela comunicação e sincronização inter-processos típica dos S.O. Os semáforos e eventos são providenciados pelo sincronismo inter-processos.

3.4.3.6. *Health monitoring*

O HM (*Heath Monitor)* é uma funcionalidade tradicional de RTOS's críticos. É responsável por monitorizar e reportar faltas e falhas de hardware, S.O. e aplicações. O HM ajuda a isolar faltas e a prevenir a propagação de falhas. O HM é uma aplicação multi-*thread*, onde cada *thread* é desenhada para monitorizar e/ou reportar o estado de determinados componentes do sistema, como a RAM, processador, disco, etc.

3.5. RTOS frequentemente usados

Existe uma grande variedade de RTOS para os mais diversos tipos de aplicações, sendo uma maioria destes orientados às necessidades de sistemas embebidos. No entanto nem todos poderiam ser usados no âmbito do Projecto OSPF dada as suas necessidades de certificação para garantir fiabilidade e segurança durante o desenvolvimento e a fase de produção. A escolha do RTOS tem um impacto directo no tempo e custo de desenvolvimento de aplicações de tempo real, logo a sua escolha deve ter em conta este factor.

3.5.1. LynxOS

O LynxOS é um sistema operativo de tempo real comercializado pela LynuxWorks [28]. Com uma arquitectura baseada em Unix, implementa o standard de POSIX sendo assim compatível com o Linux. O LynxOS é usado maioritariamente em sistemas embebidos e aplicado a ramos como os da aviação, aeroespacial, militar, indústria de controlo de processos e telecomunicações.

Os componentes do LynxOS estão desenhados para cumprir os requisitos de um sistema de absoluto determinismo (*hard real-time perfomance*), o que significa que o sistema tem um período de resposta conhecido e garantido mesmo em situações de extrema exigência.

De acordo com a sua versão este pode implementar as especificações de POSIX e ARINC 653, como é o caso do LynxOS-178b (ver secção 4 abaixo).

O LynxOS suporta três de políticas de escalonamentos, duas baseadas na especificação POSIX e uma outra política de escalonamento proprietária da LynxOS chamada "*Priority based quantum*":

- SCHED_FIFO (*First-In, First-Out*) – Ver secção 3.3.3;
- SCHED_RR (*Round Robin*) – Ver secção 3.3.3;
- SCHED_OTHER (*Priority based quantum*) – Semelhante à politica de escalonamento *Round Robin*, mas com *quantum* variável.

Informação adicional sobre este sistema operativo encontra-se disponível em http://www.mnis.fr/ocera_support/rtos/c3426.html.

3.5.2. VxWorks

O VxWorks é um sistema operativo de tempo real comercializado pela *Wind River Systems*. Este RTOS é desenhado para sistemas embebidos oferecendo portanto a possibilidade de a compilação ser efectuada numa máquina diferente da máquina onde irá executar, a este processo dá-se o nome de *cross-compiling*.

O VxWorks caracterizado por conter um *kernel* multi-tarefas com escalonamento preemptivo e *round-robin* com rápida resposta a interrupções de sistema. A protecção de memória para isolar as aplicações executadas pelo *kernel*, os mecanismos para comunicações entre processos e de sincronização de *threads,* são algumas das suas características principais.

De acordo com a sua versão este pode implementar as especificações de POSIX e ARINC 653, como é o caso do VxWorks 653. Este pode ainda implementar sistemas de ficheiros e protocolos de rede.

O VxWorks 653 (*Wind River Paltaform for Safety Critical* ARINC 653) é actualmente usado em projectos na área da aviação como "US Air Force test aircraft X47B – Pegasus".

Informação adicional sobre este sistema operativo encontra-se disponível em http://www.mnis.fr/ocera_support/rtos/c3281.html.

3.5.3. RTEMS

O RTEMS (*Real-Time Executive for Multiprocessor System*) é um sistema operativo de tempo real, *open-source* e livre, tendo sido desenhado para sistemas embebidos críticos e aplicações militares, pois providencia uma ambiente de grande desempenho.

O RTEMS foi concebido para suportar várias API's abertas e standards de interfaces, incluindo POSIX. Este sistema não providencia gestão de memória ou de processos. Assim sendo, apenas suporta um processo e um ambiente de multi-*threads*.

3.6. Conclusão

Existe uma grande variedade de sistemas operativos de tempo real com os mais diversos propósitos. No entanto, estes têm em comum uma característica que os distingue dos restantes S.O, pois facilitam a construção de aplicações de tempo real.

Dada a grande variedade e propósitos dos RTOS's, estes tipicamente implementam serviços e especificações próprias, dificultando a portabilidade entre sistemas e aumentando a curva de aprendizagem face a um determinado RTOS, projudicando o desenvolvimento profissional de sistemas de tempo real.

O aparecimento de standards como o POSIX visa beneficiar o desenvolvimento de aplicações onde a portabilidade e compatibilidade são apresentados como requisitos. Deste modo os benefícios apresentam-se como uma forma de redução de custos na aquisição/produção de hardware e software. Quando uma aplicação é implementada para um S.O. POSIX-*conformant*, o custo de adicionar novas funcionalidades e portar o código fonte para outros sistemas POSIX é reduzido. Deste modo a portabilidade do código fonte é facilitada entre sistemas operativos POSIX. No entanto, é necessário ter alguma

precaução na utilização e codificação para um S.O. que se diz POSIX-*conformant*, pois este pode não implementar o mesmo conjunto de funções POSIX que outro S.O. O problema adquire maior relevo quando o fornecedor de um determinado S.O., não especifica qual o conjunto de serviços POSIX implementado, qual o perfil especificado, obrigando o programador/utilizador a efectuar um estudo detalhado do sistema. Uma das vantagens de utilizar S.O. certificados está relacionada como facto de serem obrigados a publicar a lista de funções/conceitos implementados, bem como os desvios e extensões que foram implementados.

O standard ARINC 653 por outro lado tem uma aplicabilidade muito própria, sendo este orientado a sistemas de tempo real críticos para a área da indústria da aviação, defesa, etc. O ARINC 653 especifica que as aplicações estarão isoladas. Logo um S.O. que implemente a especificação ARINC 653 possibilita que se executem aplicações com diferentes níveis de certificação DO-178B na mesma máquina, usando diferentes partições. O ARINC 653 detalha ainda uma serie de serviços de forma a abstrair a toda a gestão e comunicação envolvida no funcionamento das partições e processos. Actualmente, as implementações existentes da especificação do ARINC 653 são comerciais e apresentam-se como soluções extremamente caras no mercado da aeronáutica.

O RTOS escolhido para executar o OSPF foi o LynxOS-178b (ver capítulo 4), por ser um RTOS que implementa os dois standards referidos. Essa escolha não foi no entanto, da responsabilidade da Skysoft, sendo que a responsabilidade da Skysoft apenas incidia em configurar de modo optimizado o RTOS para a arquitectura de hardware necessária ao OSPF.

A ideia inicial para a concepção do OSPF RTMC foi baseada num RTOS que providenciasse os dois standards juntos na sua execução, i.e. um sistema que providenciasse a totalidade dos serviços ARINC 653 suportando a execução de aplicações POSIX na mesma partição. Como este princípio se revelou impossível optou-se por usar partições POSIX recorrendo a serviços ARINC 653.

4. Aplicação do LynxOS-178b ao OSPF

4.1. Introdução

O LynxOS-178b é um sistema operativo de tempo real fiável baseado no LynxOS, que tem as características essenciais à execução de aplicações de software críticas e fiáveis, sendo estas aplicadas em áreas como a aviação, defesa, medicina, indústria aeroespacial, etc.

O LynxOS-178b foi o RTOS escolhido para a arquitectura de hardware Pentxm2 [29], para executar o elemento OSPF

A escolha de um RTOS para uma determinada aplicação e hardware exige que se investigue qual a configuração do RTOS mais apropriada, i.e. qual a configuração que permite optimizar a utilização de recursos de hardware disponíveis e ao mesmo tempo, satisfazer os requisitos da aplicação. Como tal, a decisão de usar o do LynxOS-178b para executar o OSPF exige que se encontre a forma mais apropriada de o conigurar.

Um dos objectivos do projecto que este documento descreve era exactamente explorar o LynxOS-178b no sentido de perceber qual a configuração que devia ser escolhida para a aplicação OSPF. Este trabalho exigiu perceber o enquadramento, as implicações e dependências de diferentes parâmetros de configurações suportadas pelo LynxOS-178b. Para tal foram testadas várias configurações e, para cada configuração, foi efectuado um conjunto de testes, para melhor definir um cenário/configuração aplicável ao OSPF, bem como perceber as implicações das diferentes configurações. Este processo possibilitou um conhecimento profundo do sistema operativo e detecção de problemas neste, que eram reportados à LynuxWorks. Deste modo, foram efectuadas melhorias ao próprio sistema operativo de modo a garantir o normal funcionamento do OSPF.

4.2. Características

O LynxOS-178b e as ferramentas de desenvolvimento oferecem artefactos que facilitam a construção de sistemas que necessitam de níveis até ao nível A do standard RTCA DO-178B (ver secção 2.5).

O código fonte e bibliotecas do LynxOS-178b, o compilador de C específico para a arquitectura de hardware e restantes ferramentas (como ferramentas para *debugging, bash,* etc.) constituem o SDK (*Software Development Kit*) da LynuxWorks. Neste SDK, é possível configurar e compilar o próprio RTOS, tal como as aplicações que vão ser executadas no RTOS, como o OSPF.

O LynxOS-178b, como sistema operativo, é similar ao Unix na organização e ambiente de trabalho. No que diz respeito ao ambiente de desenvolvimento e configuração este pode ser requerido para diversos sistemas operativos baseados em Linux ou Windows. No caso específico do projecto OSPF foi requerido para Windows.

O LynxOS-178b é o único sistema operativo de tempo real rígido DO-178b certificado para o nível A. Este é baseado em standards abertos e é desenhado especificamente para ambientes de extrema exigência computacional, disponibilizando assim a possibilidade de existência de múltiplas *threads* e múltiplos processos em sistemas seguros e críticos.

Do ponto de vista do sistema operativo, o LynxOS-178b implementa o conceito de partição (ver secção 3.2.6), no qual os recursos de hardware são logicamente divididos no tempo e no espaço de forma a estabelecer um isolamento dos serviços e aplicações entre partições. Cada partição é chamada de VM (*Virtual Machine*). Neste domínio, os processos que são executados numa dada partição estão isolados em relação às outras partições e apenas conseguirão interagir com eventos recebidos pela sua partição. A excepção à regra é a VM0, que tem privilégios especiais no LynxOS-178b. Estes privilégios são semelhantes aos de *root* num sistema Unix.

A conformidade com as interfaces do POSIX confere a este sistema uma extensa aplicabilidade em sistemas embebidos de tempo real. Para alcançar este propósito o LynxOS-178b oferece a conformidade com POSIX.1, POSIX.1b (extensão de tempo-real) e POSIX.1c (extensão de *threads*) (ver secção 3.3 sobre o POSIX).

O LynxOS-178b implementa um algoritmo de escalonamento baseado em *time-slices* que fornece a cada partição um tempo de execução fixo, para que o sistema possa ser seguramente determinista. O sistema permite que aplicações de diferentes níveis de criticalidade sejam executadas em partições isoladas, no mesmo recurso de hardware. Neste sistema, cada tarefa é executada de forma protegida contendo o seu próprio espaço de computação para não comprometer a fiabilidade e segurança de cada partição/VM (*Virtual Machine*).

A conformidade do LynxOS-178b com o standard ARINC 653-1 (ver secção 3.4) garante a portabilidade da aplicação, reutilização de software e interoperabilidade entre sistemas embebidos.

No entanto, este RTOS não suporta em simultâneo ARINC 653 e POSIX, portanto, uma das arquitecturas teve de ser adoptada. Para o OSPF foi adoptada a arquitectura POSIX, visto todo o projecto seguir este standard. De qualquer modo, claramente são implementados conceitos comuns em ambas as arquitecturas.

Assim, este sistema disponibiliza os seguintes serviços:

- Serviços para a arquitectura POSIX:
  - Escalonamento;
  - *Message queues*, *pipes* e *sockets;*
  - Gestão de memória;
  - Manipulação de excepções;
  - Gestão de memória partilhada;
  - Gestão de processos e *threads;*
  - Sinais POSIX de tempo-real;
  - Semáforos;
  - Mutex;
  - Gestão de particionamento;
  - Comunicação inter-partições
  - *Health Monitoring*;
- Serviços para a arquitectura ARINC 653:
  - Gestão de partições;
  - Gestão de processos;
  - Gestão temporal;

- Comunicação inter-partições;
  - *Sampling Port Services, Queuing Port Services*;
- Comunicação intra-partições
  - *Buffer Services, Blackboard Services, Semaphore Services, Event Services*;
- *Health Monitoring*;

O LynxOS-178b disponibiliza ainda serviços de rede como o TCP/IP para permitir o desenvolvimento e *debugging*. O TCP/IP e o NFS incluem os serviços ping, rcp, rlogin, route, cliente NFS, ftpd, irshd, rlogind, rshd, telnetd, etc.

4.3. Arquitectura

A Figura 4-1 ilustra a arquitectura do LynxOS-178b, mostrando a arquitectura usada para suportar múltiplas aplicações de tempo-real de diferentes níveis de criticalidade com execuções concorrentes no mesmo processador.

A aplicação que é executada numa determinada partição é verificada para o nível de criticalidade adequado para a função pretendida.

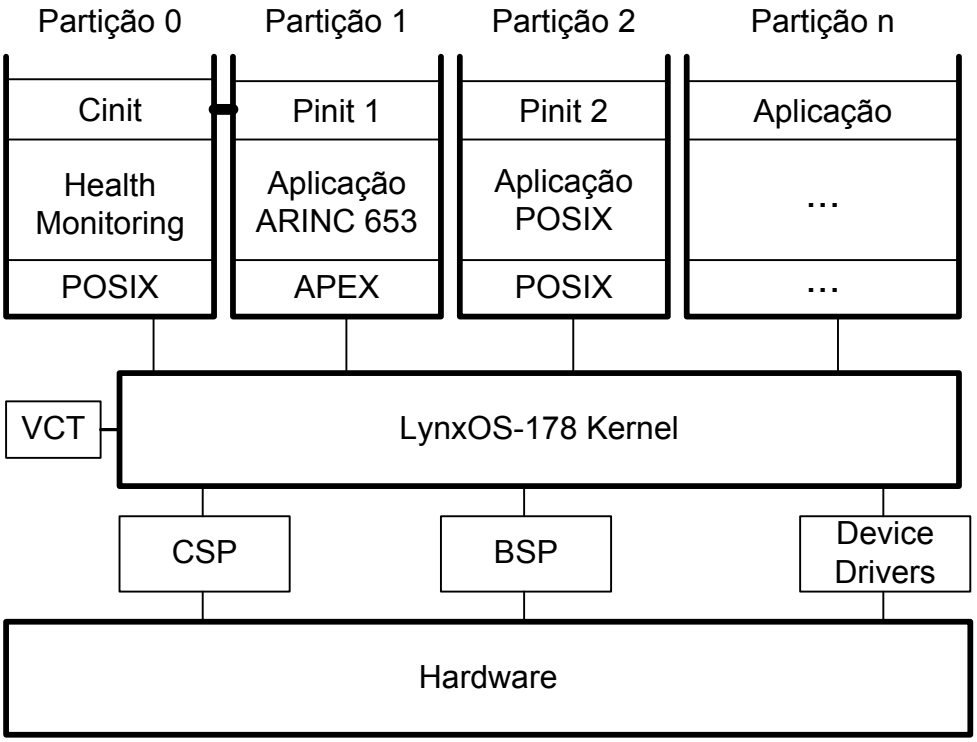

Figura 4-1: Arquitectura do LynxOS-178b

O LynxOS-178b é composto por quatro componentes principais, o CSP (*CPU Support Package*), BSP (*Board Support Package*), *Device Drivers* e o *kernel*. O CSP é a componente do LynxOS-178b que contem todas as funções específicas da família de processadores, incluindo MMU (*Memory Management Unit*), *floating point* e *handlers* para excepções do processador. O BSP contém a implementação de código de suporte

específico ao hardware para o S.O., principalmente com a *motherboard* e *buses* de ligação a dispositivos externos com PCI, ISAS, USB, etc. O *kernel* (ver secção 3.2.1) estabelece a ligação e interfaces entre a camada de aplicação e o hardware.

O VCT (*Virtual Machine Configuration Table*) contém a informação de configuração para criar as partições no LynxOS-178b. Este contém ainda perfis de configuração para cada aplicação de software de cada partição. Esta informação é usada para reservar recursos do sistema para a aplicação, definindo uma configuração válida, fornecida pelo utilizador.

O Cinit (Common Initialization) é o primeiro processo POSIX a ser executado pelo *kernel* do LynxOS-178b. O Cinit lê o VCT (*Virtual Machine Configuration Table*) e cria as partições com as especificações definidas neste ficheiro. As principais responsabilidades do Cinit são as seguintes:

-Validar e ler o ficheiro VCT;

-Inicializar as variáveis de ambiente;

-Inicializar os sistemas de ficheiros;

-Inicializar o escalonador.

Após terem sido criadas as partições pelo Cinit, este transforma-se num processo Pinit (*Partition initialisation*) para cada partição. O Pinit é o primeiro processo de cada partição a ser executado e tem a responsabilidade de completar o processo de inicialização do ambiente e de lançar a aplicação de software em cada partição.

4.4. Desenvolvimento vs Produção

O LynxOS-178b SDK é constituído por dois modos de configuração, o modo de desenvolvimento e o modo de produção. O modo de desenvolvimento é usado durante a fase de desenvolvimento de software aplicacional, no qual se realizam testes de hardware, testes de software, testes de desempenho e *debugging* aplicacional. O modo de desenvolvimento é na realidade o mais usado durante a fase de implementação de código. É neste modo que se descobrem particularidades no funcionamento do LynxOS-178b, nomeadamente:

-Configuração do hardware;

-Configurações de software;

-Configurações de rede;

-Testes de hardware, como leitura de e escrita de dados em UPS, Leds, *Health monitoring*, hardware *watchdog timers*;

-Testes de desempenho da comunicação de rede, como TCP, UDP, FTP, TFTP.

O modo de produção é um subconjunto certificado do modo de desenvolvimento. Este modo deve ser visto como a configuração do sistema operativo para a plataforma final,

onde apenas existirão os serviços necessários ao funcionamento da aplicação de software desenvolvida, neste caso o OSPF.

Por exemplo, é útil que em modo de desenvolvimento exista uma forma de interagir directamente com o sistema operativo, logo, é disponibilizada uma *bash*. Em produção, como não existe essa necessidade e nem sempre é possível interagir visto se tratar de um sistema embebido, tal serviço não é necessário.

A arquitectura de compilação existente e usada em ambos os modos é a de desenvolvimento *cross-compiling* onde a compilação e ligação é efectuada para outra arquitectura de hardware, ou seja, o código é compilado e ligado numa máquina, mas é executado noutra.

4.5. Configuração

Como já foi anteriormente referido, o LynxOS-178b é um RTOS extremamente modular e configurável, tanto em modo de desenvolvimento como em modo de produção. Na configuração é possível definir detalhes como o número de partições, tempo de computação para as partições, serviços disponíveis, características do sistema de ficheiros, etc. Estas configurações são efectuadas em ficheiros que fazem parte do SDK do LynxOS-178b e que vão ditar a forma como se processa a compilação do RTOS.

O VCT (*Virtual Machine Configuration*) é o ficheiro de configuração das máquinas virtuais. Este contém a informação para criar e configurar as VM's/Partições no LynxOS-178b, tal como as configurações particulares para cada aplicação a ser executada numa dada partição. Esta informação é usada para reservar e preparar os recursos do sistema (ver exemplo do ficheiro VCT no 0). Neste ficheiro podemos especificar:

- O número de partições e tipo de partições, se partições POSIX ou partições ARINC 653;

- Comunicações ARINC 653 entre partições;

- Número de sistemas de ficheiros e a localização onde são montados (RAM, disco ou flash);

- Para cada partição, o tempo de processador e a quantidade de memória;

- O número limite de processos e *threads* POSIX, tal como o número limite de processos ARINC 653;

- O executável a arrancar quando o sistema ficar operacional;

- A RAM disponível no sistema;

- Número de processos;

- Quantidade de memória reservada aos processos;

- Tipo de Escalonamento.

O ficheiro lynxos-178.spec especifica a estrutura e organização do sistema de ficheiros do LynxOS-178b. Neste ficheiro estão definidos todos os directórios e ficheiros com as permissões associadas que farão parte do RTOS. Se for necessário adicionar um binário para ser executado posteriormente no RTOS, esse binário terá de ser especificado neste ficheiro com as permissões correctas (ver exemplo do ficheiro lynxos-178.spec no 0).

No ficheiro config.tbl são definidos todos os *devices drivers* a incluir no RTOS. Estes podem ser diversos e variam desde a configuração para rede como a placa de rede até discos, memória flash, etc. (ver exemplo do ficheiro config.tbl no 0).

4.5.1. Configurações aplicadas ao OSPF

Ao longo do projecto foram exploradas imensas possibilidades de configuração de forma as conhecer as potencialidades, problemas e particularidades do RTOS.

No entanto o OSPF tinha requisitos específicos quanto ao tipo de configurações requeridas para o LynxOS-178b.

4.5.1.1. Requisitos de configuração do OSPF

O OSPF contém um conjunto de processos que serão executados segundo diferentes configurações do RTOS para as diferentes cartas de processamento do bastidor OSPF. Essas configurações podem ser divididas em 4 tipos:

- Configuração A: LynxOS-178b com 1 partição, certificação DAL-C
  - Principais serviços: *Heath Monitoring;* servidor de TFTP.
- Configuração B: LynxOS-178b com 1 partição, certificação DAL-C
  - Principais serviços: *Heath Monitoring;* TFTP.
- Configuração C: LynxOS-178b com 2 partições
  - Partição 1, certificação DAL-C
    - Principais serviços: *Heath Monitoring.*
  - Partição 2, certificação DAL-E
    - Principais serviços: servidor de TFTP, agent SNMP.
- Configuração D: LynxOS-178b com 2 partições
  - Partição 1, certificação DAL-C
    - Principais serviços: *Heath Monitoring.*
  - Partição 2, certificação DAL-E
    - Principais serviços: servidor de TFTP.

A Figura 4-2 representa as várias cartas de processamento pelas quais o Bastidor OSPF é constituído. Num total de 11 cartas de processamento, ou seja, 11 configurações necessárias.

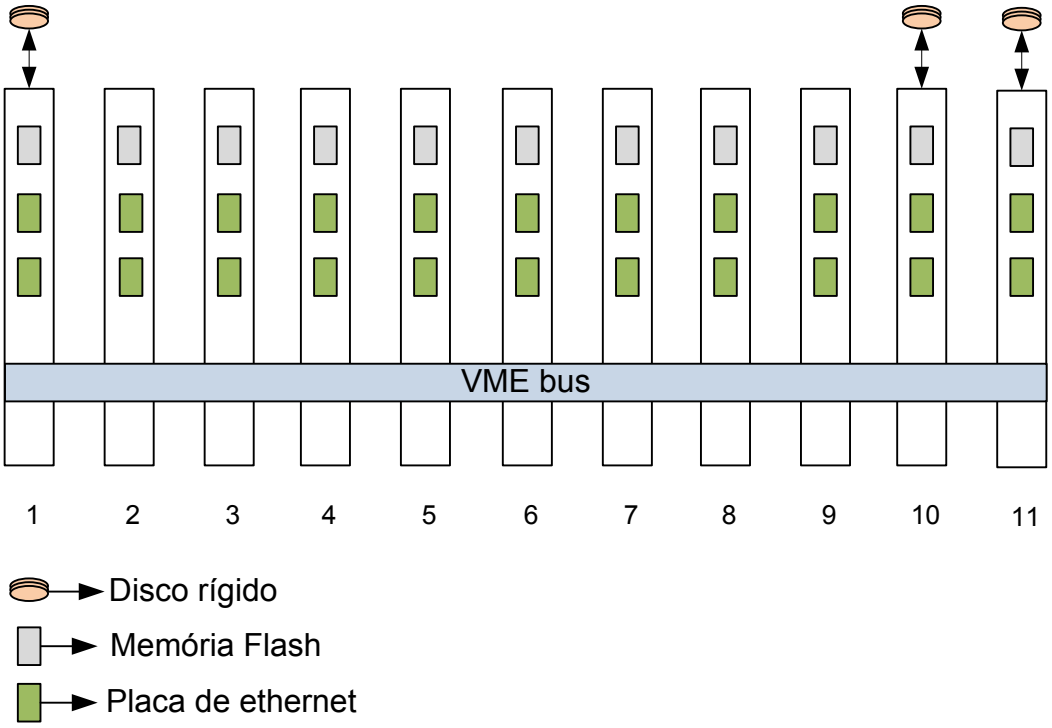

Figura 4-2: Cartas de processamento do Bastidor OSPF

A configuração "A" foi aplicada à carta de processamento número 1. Nesta carta de processamento foi necessário configurar os acessos ao disco rígido, memória flash e placas de ethernet. A configuração "B" foi aplicada às cartas de processamento número 2 até ao número 9. Nestas cartas de processamento não existiam discos rígidos, no entanto, foi necessário configurar os acessos à memória flash e placas de ethernet. A configuração "C" foi aplicada à carta de processamento número 10. A configuração nesta carta de processamento ao nível de hardware parece semelhante à carta número 1, mas a necessidade de particionamento ARINC 653 (ver secção 3.4.3.1 sobre particionamento ARINC 653) implica a divisão de recursos. A configuração "D" foi aplicada à carta de processamento 11. No entanto, tem uma configuração semelhante à configuração "C", mas com diferenças ao nível se serviços disponibilizados.

4.5.1.2. Estratégia

A estratégia seguida consistiu na construção, em primeira instância, de um KDI simples para auxílio à equipa de desenvolvimento de código, onde se podiam executar protótipos de forma a obter resultados semelhantes aos corridos numa arquitectura final. Este KDI foi executado numa máquina virtual do Windows XP [30]. Este abordagem permitiu, enquanto não recebíamos o bastidor do OSF, testar o sistema operativo e a execução de aplicações nesta máquina virtual (ver secção 2.6.3).

Posteriormente, com a chegada do bastidor do OSPF, foi necessário construir KDI's para as diversas cartas de processamento. A estratégia seguida para a configuração das diferentes cartas de processamento foi simples. As primeiras configurações foram orientadas às configurações "A" e "B", visto se tratarem das mais simples por não envolverem particionamento de recursos. Depois destas configurações concluídas

seguiram-se as configurações "C" e "D", que mais complexas que as anteriores por necessitarem de particionamento de recursos.

4.6. Construção e execução do KDI

Após a configuração do LynxOS-178b é possível compilar o *kernel* com as novas configurações definidas na secção acima. Desta compilação surge um KDI (*Kernel Downloadable Image*). Este é estaticamente ligado com o CSP, BSP e com os *devices drivers* estáticos (ver Figura 4-1) para criar o RTOS LynxOS-178b.

O KDI é assim uma imagem que contém o LynxOS-178b e o sistema de ficheiros, que pode ser recebida por exemplo, através de rede, por um *host*. O sistema de ficheiros é baseado em Unix com uma API POSIX, e deve conter no mínimo o software necessário para o Cinit completar a inicialização das diversas tarefas. Também contém os pontos de montagem dos sistemas de ficheiros especificados para as outras VM's, que se encontram definidos no ficheiro VCT. Todos os sistemas de ficheiros definidos no VCT são montados em RAM, com permissões de leitura, à excepção do directório "/tmp".

Depois de construir o KDI é necessário transferir este para a plataforma de hardware que irá correr o OSPF. A próxima secção exemplifica algumas estratégias para executar o KDI.

4.6.1. Estratégia para executar o KDI

O RTOS LynxOS-178b vai ser executado numa placa PENTXM2, onde os binários do OSPF estarão em memória para posteriormente serem também executados. No entanto em fase de desenvolvimento era necessária uma arquitectura que possibilitasse a construção do LynxOS-178b com diferentes configurações para este ser enviado para a placa sendo posteriormente executado e testado no hardware final.

4.6.1.1. Arquitectura

O bastidor é composto 11 placas PENTXM2, onde serão executados os vários processos do OSPF. O bastidor é ainda composto por uma UPS (Uninterruptible *Power Supply*) e uma placa IRIG-B para sincronização temporal de alta precisão e 2 *switchs*.

A arquitectura do servidor que contém os KDI's do LynxOS-178B construídos e do Bastidor do OSPF encontra-se representada na Figura 4-3.

O SDK da LynuxWorks está instalado no servidor, e neste são configuradas as imagens do RTOS que será executado na placa PENTXM2.

Com a necessidade de transferir a KDI do RTOS a executar para uma das 12 placas, são utilizados 2 serviços, um servidor de DHCP e um servidor de TFTP, estabelecendo o protocolo PXE. O protocolo PXE é usado para efectuar *boot* do RTOS remotamente. Com este protocolo é possível então o envio e a execução de uma imagem do RTOS para diferentes placas. Para tal, apenas é necessário configurar o servidor de DHCP, o servidor de TFTP e o *bootloader* das placas para arrancar por rede.

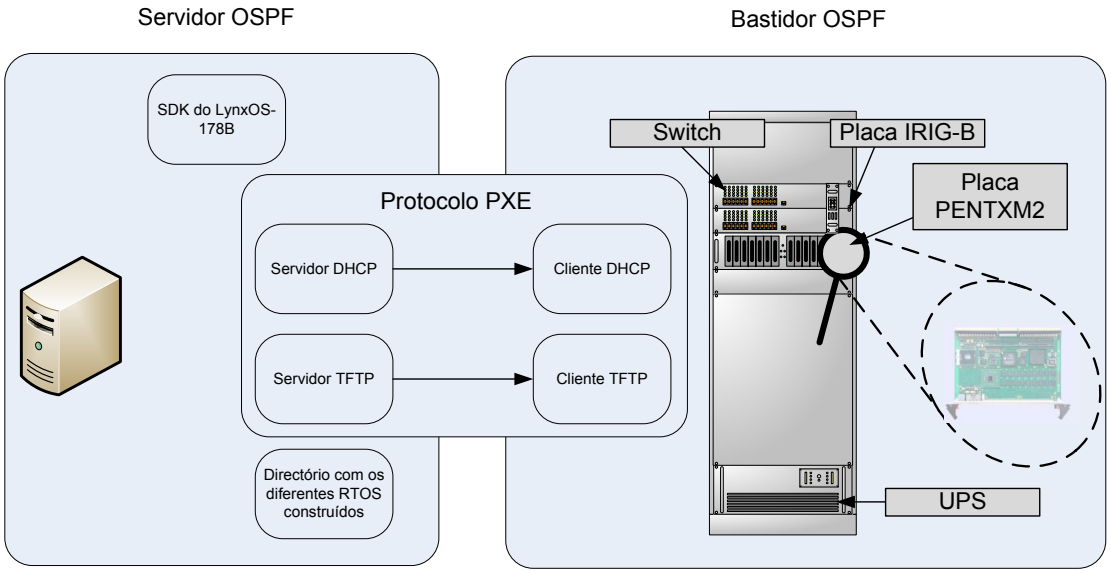

Figura 4-3: Arquitectura do Servidor OSPF e Bastidor OSPF

4.7. Testes de Hardware e de RTOS

Ao recebermos o hardware final do OSPF, o bastidor representado na Figura 4-3, foi necessário verificar como se comportava o LynxOS-178b com a arquitectura de hardware final. Para verificar o comportamento do LynxOS-178b foram implementados um conjunto de testes de forma a averiguar o funcionamento e desempenho de componentes importantes ao OSPF. Os testes efectuados permitiram reportar problemas existentes ao nível do RTOS possibilitando correcção destes. Com estes testes, surgiram novos BSP que funcionavam basicamente correcções/adaptações do RTOS ao hardware.

Hardware

Os testes de hardware consistiam em testar o acesso e escrita/leitura a um determinado componente de hardware. Estes testes permitiam descobrir certas discrepâncias entre a documentação fornecida/especificada e o real resultado do teste para posteriormente se proceder à correcção. Os testes foram efectuados aos seguintes componentes:

- *Health Monitor*
- Leds do bastidor
- UPS
- Placa IRIG-B
- Hardware *watchdog*

LynxOS-178b

Os testes do RTOS tinham o propósito de despistar falhas no LynxOS-178b que pudessem influenciar o funcionamento do OSPF e de testar o desempenho de diferentes componentes de hardware. Assim foram testados:

- Hardware - Tempos de acesso para escrita e leitura:

    - Disco rígido
    - Memória flash
- Protocolos de redes – Tempos de transmissão pela rede:
    - TCP
    - UDP
    - FTP
    - TFTP
    - UDP Multicast
    - IGMP
- RTOS – funcionamento de funções de POSIX:
    - Controlo de processos
    - Execução de aplicações
    - Controlo de threads
    - Controlo do tempo de sistema
    - Escalonamento do RTOS
- Bibliotecas externas – desempenho e funcionamento:
    - SNMP
    - ASN.1
    - XML

Estes testes possibilitaram detectar problemas de desempenho e erros associados aos diversos componentes e serviços, que se não fossem detectados, poderiam provocar falhas na execução do OSPF.

A cada novo BSP (nova versão), todos os testes eram corridos, de forma a despistar a ocorrência de novos problemas como para observar se os problemas antigos tinham sido realmente corrigidos.

4.7.1. Resultados dos testes

Para alguns dos testes acima indicados, apenas interessava saber se os serviços disponibilizados funcionavam ou não. No entanto, uma grande variedade dos testes efectuados servia para constatar desempenhos de determinados serviços em diferentes arquitecturas do RTOS. Alguns desses desempenhos encontram-se no Anexo VI.

4.8. Conclusão

Este RTOS está em constante evolução, dada a especificidade em relação ao *hardware* onde irá executar. Cada evolução constitui um novo BSP e/ou Controladores de dispositivos, e quando tal acontece é necessário verificar, através de testes de hardware e de RTOS, se realmente foram despistadas inconsistências.

Como se demonstrou, o LynxOS-178b é um sistema operativo altamente configurável, oferecendo um conjunto de mecanismos externos que possibilitam essa configuração.

No entanto, a ligação entre o LynxOS-178b e o Bastidor do OSPF nem sempre foi a melhor. Com a exploração e utilização do RTOS apareceram imensos problemas ao nível da implementação do próprio RTOS, que por vezes, se revelavam complicados de

reproduzir de forma padronizada para que fosse possível reportar eficazmente. Apenas como exemplo, de início surgiu um problema relacionado com o espaço ocupado em disco rígido. O problema estava relacionado com o facto de nos ser dada a indicação de que o disco rígido estava cheio quando por vezes apenas estavam ocupados 2% da totalidade do disco rígido, sendo portanto impossível a escrita neste dispositivo. A suspeita no momento seria de que o RTOS possivelmente não estaria a gerir correctamente a tabela de ficheiros.

A conformidade do LynxOS-178b com as especificações POSIX e ARINC 653 oferecem no entanto, a tão esperada portabilidade, segurança e redução custos aos sistemas de tempo real crítico, o que actualmente é essencial a aplicações comerciais e governamentais.

5. Camada RTOS_Rtos

5.1. Introdução

O RTOS_Rtos é um dos módulos do projecto OSPF pertencente ao pacote COTSEncapsulation do subsistema serviços do elemento (ver secção 2.6.3). Este módulo é responsável pelo encapsulamento de parte das funções de sistema de forma a disponibilizar serviços mais abstractos à camada de aplicação. O RTOS_Rtos é um módulo DAL-C, tendo portanto de obedecer a todos os requisitos deste nível (ver secção 2.5).

5.2. Objectivo

O RTOS_Rtos é um módulo desenvolvido para ser usado como uma camada a entre aplicação e o sistema operativo de tempo real. Esta camada funciona como uma interface, que disponibiliza à camada de aplicação funções especiais necessárias ao normal funcionamento do sistema. Para tal, a camada do RTOS, implementa uma maior abstracção em relação às funções de sistema, para que a sua utilização seja simplificada e segura ao nível da aplicação. A Figura 5-1 ilustra a constituição em camadas do sistema.

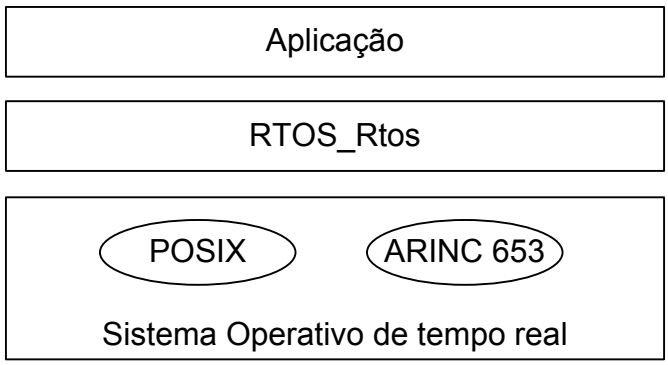

Figura 5-1: Camadas envolventes do RTOS_Rtos

A camada RTOS_Rtos utiliza as funções POSIX disponibilizadas pelo sistema operativo de tempo real. A essas funções correspondem maioritariamente:

- Funções POSIX.1, relativas à criação e controlo de processos, bibliotecas standard de C;

- Funções POSIX.1b, relativas à gestão de prioridades de escalonamento, relógios e temporizadores, memória partilhada e memória protegida;

- Funções POSIX.1c, relativas à criação, controlo e escalonamento de *threads;*

A implementação desta camada deve oferecer à camada de aplicação independência e portabilidade em relação ao sistema operativo. Desta forma, espera-se que caso haja uma modificação na escolha ou implementação do sistema operativo, que as modificações a realizar se restrinjam maioritariamente à camada do RTOS e não à camada de aplicação. Esta camada deve também providenciar uma detecção e tratamento de erros das chamadas POSIX. Estas chamadas normalmente não fornecem à camada de aplicação informação

sobre o seu estado. Por exemplo, a função "memcpy" recebe três parâmetros, sendo dois deles apontadores. Se um desses apontadores for igual a NULL, a função não fornece informação nenhuma caso ocorra um erro, e neste caso, ocorrerá um *segmentation fault*. Dado este facto é importante que a camada RTOS_Rtos proteja a aplicação do S.O. da ocorrência de erros não tratados pelo S.O.

O desenvolvimento desta camada visam providenciar em relação ao S.O:

- Bibliotecas de software simplificadas;

- Código documentado;

- A redução de dependências exteriores;

- O encapsulamento de colecções de funções de design fraco em funções com um bom design, adaptadas e optimizadas às necessidades concretas da aplicação.

5.3. Arquitectura

O RTOS_Rtos providencia à aplicação um conjunto de funções que encapsulam parte das funcionalidades disponibilizadas pelo POSIX.1, POSIX.1b e POSIX.1c, construindo serviços a disponibilizar à camada de aplicação. Espera-se que o RTOS_Rtos garanta a protecção contra a passagem errónea de informação, a detecção/tratamento de erros provenientes das funções de sistema de forma consistente e que simplifique a codificação ao nível da camada de aplicação. A Figura 5-2 ilustra a organização do módulo RTOS e a relação entre as diversas entidades.

A camada RTOS_Rtos faz uso da camada do sistema operativo de tempo real e disponibiliza as suas funcionalidades à camada de aplicação, servindo assim de interface entre o sistema operativo de tempo real e a aplicação de software. Esta é constituída por cinco componentes: POSIX.1, POSIX.1b e POSIX.1c (ver secção 3.3), que contêm as chamadas de funções do S.O., a componente do RTOS_Rtos que usa apenas as anteriores funções encapsuladas de forma a construir serviços robustos úteis à aplicação, a componente de dados privados a esta camada, que é constituída por uma lista que contém a informação necessária sobre o estado geral desta camada, como *threads* activas, estado, etc.

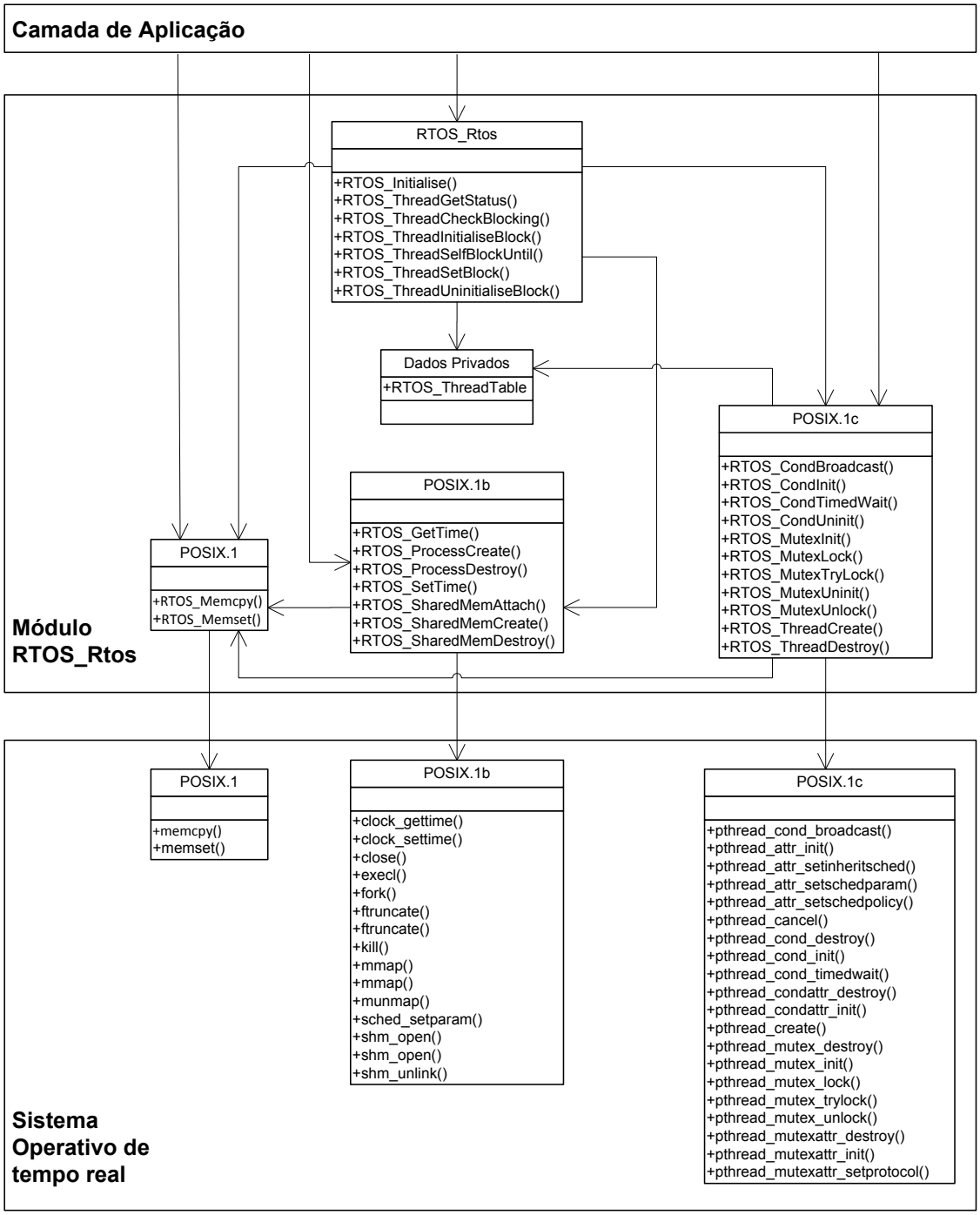

Figura 5-2: Arquitectura do módulo RTOS

5.4. Detalhes da camada RTOS_Rtos

A camada RTOS_Rtos é constituída pelas funções indicadas na Tabela 5-1 e por uma lista de RTMC_MAX_THREADS (valor inteiro configurável) elementos, correspondentes aos

"Dados Privados" ilustrados na Figura 5-2 da secção anterior. Os "Dados Privados" contêm a informação sobre o estado de cada *thread* POSIX passível de ser criada pela aplicação. Cada elemento da lista de "Dados Privados" tem os seguintes atributos:

- *Handler*;
- *Mutex*;
- *Condition*;
- *BlockControl*;
- *InUse*.

O *Handler* é o atributo que vai conter o identificador do S.O. de uma dada *thread.* O *Mutex* irá conter apontador para o *mutex* de uma dada *thread*. O atributo *Condition* indicará se a *thread* deve ou não bloquear. O *BlockControl* é o atributo que indica se uma *thread* está marcada ou não para bloquear. O *InUse* indica apenas se uma dada *thread* foi lançada ou não.

Estes atributos são importantes para proteger os recursos partilhados do sistema, auxiliando a implementação de exclusão mútua, sincronização e controlo de *threads*.

Função	**Descrição**
RTOS_CondBroadcast	Faz com que uma *thread* difunda uma determinada condição variável, libertando ou bloqueando *Mutex's*.
RTOS_CondInit	Inicializa uma condição variável específica.
RTOS_CondTimedWait	Faz com que uma *thread* espere até uma duração máxima em relação a uma condição variável específica.
RTOS_CondUninit	Destrói uma condição variável específica.
RTOS_GetTime	Esta operação lê e retorna o tempo do relógio do sistema.
RTOS_Initialise	Inicializa todas as variáveis privadas necessárias ao módulo.
RTOS_Memcpy	Copia um certo número de bytes de uma zona de memória para outra zona.
RTOS_Memset	Preenche uma zona de memória com um valor especificado.
RTOS_MutexInit	Inicialização de um *mutex*.
RTOS_MutexLock	Bloqueia um *mutex*.
RTOS_MutexTryLock	Bloqueia um *mutex* caso não esteja bloqueado. Caso contrário retorna de imediato sem bloquear.
RTOS_MutexUninit	Desactiva um *mutex*.
RTOS_MutexUnlock	Desbloqueia um *mutex*.
RTOS_ProcessCreate	Esta função cria um novo processo.
RTOS_ProcessDestroy	Mata um processo.
RTOS_SetTime	Esta operação insere a hora local no relógio do sistema.

RTOS_SharedMemAttach	Atribui a uma área de memória partilhada. Esta função ”abre” o espaço de memória partilhada sem o criar. Se a área de memória partilhada não existir a função irá indicar que falhou
RTOS_SharedMemCreate	Cria uma área de memória partilhada. Se a memória partilhada já foi criada então a função retorna a informação de insucesso.
RTOS_SharedMemDestroy	Destrói uma área de memória partilhada. É da responsabilidade da função garantir que todos os processos libertaram a memória partilhada antes de chamar esta função.
RTOS_ThreadCheckBlocking	Esta função bloqueia uma *thread* num ponto específico do ciclo, isto se a *thread* estiver marcada para bloqueio.
RTOS_ThreadCreate	Cria uma *thread* Posix.
RTOS_ThreadDestroy	Destrói uma *thread* Posix.
RTOS_ThreadGetStatus	Retorna o estado actual de uma *thread*, se activa ou não. Esta função deve ser utilizada para determinar se uma *thread* acabou a sua computação com sucesso.
RTOS_ThreadInitialiseBlock	Esta função inicializa o mecanismo de bloqueio para uma determinada *thread*.
RTOS_ThreadSelfBlockUntil	Esta função auto suspende a *thread* durante um período de tempo especificado.
RTOS_ThreadSetBlock	Esta função é chamada apenas a partir da *thread* principal do sistema. Esta função gere as várias *threads* indicando se uma dada *thread* deve bloquear / desbloquear.
RTOS_ThreadUninitialiseBlock	Esta função desactiva mecanismo de bloqueamento para uma determinada *thread*.

Tabela 5-1: Lista de funções do módulo RTOS

Cada função do módulo RTOS_Rtos garante o tratamento e detecção de erros dentro do seu contexto. Todos os parâmetros são validados, tanto os de entrada como os de saída. Cada função retorna a indicação de “sucesso” caso a função tenha computado o que devia ter sido computado e “insucesso” caso tenha ocorrido um erro. O código encontra-se protegido contra a propagação de erros, ou seja, caso um erro ocorra num dado ponto da computação o restante código não é executado, prevenindo assim a ocorrência de falhas no sistema.

No 0 encontra-se o conteúdo e organização de uma função, que neste caso é a função RTOS_ProcessCreate apenas a título de exemplo. Como se pode observar, a função é composta pelo seu desenho detalhado e pelo código da função que representa esse desenho. O desenho detalhado é representado por PDL (*Pseudo-code Design Language*), o pseudo-código, que na fase de implementação, testes e verificação, auxiliou na produção e organização de código.

5.5. Testes da camada RTOS

Ainda durante a fase de Implementação (ver secção 2.6.2) começaram a ser definidos os SUP's, baseado no design/PDL das funções, que mais não são do que a especificação dos teste funcionais para um dado módulo, que neste caso é RTOS_Rtos. Em paralelo decorre a fase de codificação, também baseada no design/PDL, que é executada por uma equipa diferente da equipa que especifica os SUP's. As tabelas do Anexo II são um exemplo da especificação dos testes que foram especificados e posteriormente implementados para testar apenas uma função. Cada tabela apenas representa um caso de teste, especificado apenas com acesso ao design da função, representando um teste independente para a função.

O teste a uma dada função é basicamente uma função de nível superior que irá chamar a função que se pretende testar, controlando os seus parâmetros de entrada e saída, as suas chamadas a funções através de *wrappers* ou *stubs*, o seu retorno, as suas variáveis externas, e controlando por vezes também as suas variáveis internas.

5.6. Conclusão

As metodologias de design e codificação facilitaram o processo de concretização da camada RTOS_Rtos. A definição do PDL para as várias funções ajudou a perceber o que era essencial à camada de aplicação, ao processo de codificação e à definição dos testes unitários. Os SCS, de início não foram naturalmente aceites dada a sua complexidade e esforço envolvido na codificação. No entanto, posteriormente foi notória a sua vantagem, pois ajudaram a manter o código legível ao longo de toda a fase de codificação, o que ajudava na manutenção do código.

A camada RTOS_Rtos oferece assim à camada de aplicação uma maior portabilidade, uma maior fiabilidade e segurança face ao oferecido pelo POSIX. Para tal, a prevenção e detecção de erros foram requisitos obrigatórios ao nível desta camada.

Um dos objectivos desta camada era o de oferecer independência à aplicação face às funções disponibilizadas ou não pelo RTOS. Assim foi possivel avançar com a codificação e testes da aplicação independentemente dos detalhes de implementação da camada RTOS_Rtos. A única informação necessária a aplicação resumia-se assim à assinatura das várias funções do RTOS_Rtos.

6. Calendarização

Na Figura 6-1 é apresentado o diagrama de *Gantt* da calendarização das várias tarefas realizadas por mim ao longo do projecto. Assim encontram-se descriminadas as tarefas relacionando-as com a data e tempo de execução tal como as suas precedências.

- T1 - Integração no OSPF RTMC
    - T2 - O início do projecto foi marcado pela apresentação da constituição da equipa (que no total perfazia quatro elementos) e pela leitura da documentação pertinente ao projecto. Esta documentação está relacionada com a secção 2.5 e secção 2.6.1.
    - T3 - Ainda durante o processo de integração no projecto OSPF RTMC foi necessário preparar todo o ambiente de desenvolvimento. A preparação do ambiente consistiu na criação de um repositório de CVS para controlo de versões, apenas acessível aos membros do projecto.
    - T4 - O projecto OSFP RTMC atravessava neste período uma necessidade de adaptação aos novos padrões de desenvolvimento impostos pelos nossos clientes. Deste modo foi necessário adaptar todo o design e código existente às novas imposições. Assim foram construídos um conjunto de *scripts* para modificar todo o código de modo a que ficasse de acordo com o esperado.
    - T5 - Com a necessidade de uma máquina distinta para receber o SDK do LynxOS-178b foi reservada uma máquina para o projecto. Assim esta máquina foi configurada como servidor do projecto, centralizando aplicações importantes e de uso comum ao elementos do projecto. Este servidor era ainda constituído ainda por uma máquina virtual que continha uma imagem do LynxOS-178b em execução. Esta máquina virtual foi usada como solução à inexistência do hardware final, possibilitando avanços no desenvolvimento de código e exploração do RTOS. Ver secção 2.6.3.
    - T6 – Com o servidor configurado era útil a existência de *scripts* que possibilitassem a compilação remota com o compilador do LynxOS-178b. Este processo deveria ser transparente ao programador. Ver secção 2.6.4.
- T7 – Análise sobre sistemas POSIX. Ver secção 3.3.
- T8 – Análise sobre sistemas ARINC 653. Ver secção 3.4.
- T9 – Análise de sistemas de bloqueio e escalonamento de processos e *threads* de tempo-real. Ver secção 3.2.
- T10 - Desenvolvimento de um módulo de encapsulamento do sistema operativo de tempo real. Ver capítulo 5.
    - T11 - Codificação da camada RTOS_Rtos
    - T12 - Testes preliminares da camada RTOS_Rtos
        - T13 - Controlo de processos
        - T14- Execução de aplicações
        - T15 -Controlo de threads
        - T16 - Controlo do tempo de sistema
- T17 - LynxOS-178b. Ver capítulo 4.
    - T18 - Configuração do LynxOS-178b 2.2.0
    - T19 - Configuração do LynxOS-178b 2.3.0
    - T20 - Concretização de testes de hardware e software
        - T21 - testes de desempenho e funcionamento às bibliotecas externas
            - T22 - ASN1

        - T23- LibXML
        - T24- SNMP
    - T25- Testes de desempenho
        - T26 - Disco rígido & Memória flash
        - T27 - TCP
        - T28 - UDP
        - T29 - FTP
        - T30 - TFTP
        - T31 - UDP Multicast
    - T32 - Testes de Hardware
        - T33 - Controlo de *Leds*
        - T34 - Software *Watchdog*
- T35 - Elaboração da Dissertação

O trabalho realizado por mim decorreu de acordo com o esperado. A ordem das tarefas facilitou a sua concretização pois os conhecimentos adquiridos nas tarefas anteriores eram importantes às tarefas seguistes. No entanto, houve tarefas que implicaram atrasos na concretização deste projecto, nomeadamente, o início da tarefa T16, que se deu mais tarde do que o esperado, visto o atraso da recepção do hardware do nosso cliente (bastidor OSPF). De qualquer forma, o meu objectivo foi o de realizar este trabalho retirando o máximo de conhecimento ao meu alcance, não considerando portanto prejudiciais deste ponto de vista.

Figura 6-1: Diagrama de *Gantt*

7. Conclusão

Neste documento descreveu-se um sistema que irá ter um enorme impacto na vida dos europeus. O sistema Galileo já hoje proporciona e proporcionará no futuro o impulsionamento de novas tecnologias, novas empresas e novas oportunidades. O projecto OSPF foi exactamente uma destas oportunidades para a Skysoft.

Ao longo deste documento foi abordada uma questão essencial ao funcionamento do OSPF, o sistema operativo de tempo real que o vai executar. Assim descreveram-se parte das características gerais de sistemas operativos de tempo real passando por importantes especificações como o POSIX e o ARINC 653, especificações que tinham de ser implementadas pelo RTOS que servisse o OSPF. De seguida apresentou-se então o sistema operativo de tempo real que executará o OSPF, o LynxOS-178b, explorando e identificando as necessidades de configuração. No final apresentou-se a camada RTOS_Rtos, que foi concebida de forma a ser uma salvaguarda em relação a uma eventual mudança de sistema operativo e a uma garantia de que a detecção, tratamento e prevenção de erros seria efectuada ao nível dessa camada, evitando que estes se propagassem para a camada de aplicação.

Um dos objectivos deste documento era o de explicar o funcionamento geral do OSPF, passando por standards, processos de codificação e de desenvolvimento, fornecendo deste modo o enquadramento necessário para entender a camada RTOS_Rtos e a componente do sistema operativo de tempo real com o restante projecto.

A execução deste projecto foi uma etapa fundamental na minha vida académica, pessoal e profissional uma vez que me fez ingressar no mercado de trabalho, dando-me a oportunidade de integrar uma equipa profissional. Por outro lado, foi uma oportunidade para aprender e utilizar novas técnicas e tecnologias aplicadas no mundo profissional das organizações, dotando-me assim de um conjunto de novas capacidades.

Como trabalho futuro, identificou-se que seria interessante investigar possíveis soluções de implementação de sistemas operativos de tempo real. Mais em concreto, seria interessante investigar a concretização de um RTOS que providenciasse o standard POSIX implementado por cima do standard ARINC 653. O objectivo seria providenciar todas as vantagens do desenvolvimento sobre POSIX aproveitando a certificação dos serviços ARINC 653. Poderia assim existir um perfil POSIX que fizesse a ligação com a maioria dos serviços ARINC 653.

Outra ideia interessante ao nível empresarial poderia ser a de estabelecer uma *framework* de desenvolvimento de software de tempo real, estabelecendo-se assim uma camada de *middleware* com o software até ao momento desenvolvido, que poderia ser utilizada em todos os projectos relacionados, poupando assim nos custos e esforço envolvidos.

Referências bibliográficas

[1]-ESA Sistema Galileo, Dezembro 2007, http://www.esa.int/esaNA/galileo.html

[2]-Serviços no sistema Galileo, Agosto 2007,
http://www.esa.int/esaNA/SEMTHVXEM4E_galileo_0.html

[3]-Detalhes dos serviços no sistema Galileo, Agosto 2007,
http://www.esa.int/esaNA/SEM5K8W797E_galileo_2.html

[4]-Politicas no sistema Galileo, Setembro 2000,
http://www.esgt.cnam.fr/sites/CNIG/PDF/galileo.pdf

[5]-Lynuxworks, LynxOS-178B operating system, Novembro 2007,
http://www.lynuxworks.com/rtos/rtos-178.php

[6]-LynxOS Network guide, Novembro 2007,
http://www.lynuxworks.com/support/lynxos/docs/0402-02_los4_networking_guide.pdf

[7]-POSIX Family, Março 2008, http://stephesblog.blogs.com/papers/acm-posix.pdf

[8]-Software Development under DO-178B, Janeiro 2002,
http://www.opengroup.org/rtforum/jan2002/slides/safety-critical/chilenski.pdf

[9]-DO-178B from LynuxWorks, Março 2008,
http://www.lynuxworks.com/solutions/milaero/do-178b.php3

[10]-DO-178B from FAA, Junho 2003,
http://www.airweb.faa.gov/Regulatory_and_Guidance_Library/rgOrders.nsf/0/640711b7b75dd3d486256d3c006f034f/$FILE/Order8110.49.pdf

[11]-Produtos Marben - ASN.1, Maio 2008, http://www.marben-products.com/asn.1/overview.html

[12]-Biblioteca de XML - libxml2, Maio 2008, http://xmlsoft.org/

[13]-Net-SNMP - protocolo SNMP, Março 2007, http://www.net-snmp.org/

[14]-Galileo - The European Programme for Global Navigation Services, Janeiro 2005, http://www.galileoic.org/la/files/Galileo%20Brochure%20EN.pdf

[15]-Operating System Concepts - John Wiley & Sons, Dezembro 2004, Operating System Concepts 7 th Edition

[16]-Partitioning Operating Systems Versus Process-based Operating Systems, Maio 2008, http://www.lynuxworks.com/products/whitepapers/partition.php

[17]-ARINC 653 (ARINC 653-1), Maio 2008, http://www.lynuxworks.com/solutions/milaero/arinc-653.php

[18]-Galileo Ground Mission Segment high level requirements document, Julho 2008, http://estext231.estec.esa.int/Galileo-FOC/Appendix-2-GMS/Appendix%202-A%20Ground%20Mission%20Segment%20High%20Level%20Requirements.pdf

[19]-GEOMETRIC RECTIFICATION OF EUROPEAN HISTORICAL ARCHIVES OF LANDSAT 1-3 MSS IMAGERY, Março 2008, http://www.spacemetric.com/files/references/mss.pdf

[20]-ANSI C, 1997, http://www.acm.uiuc.edu/webmonkeys/book/c_guide/

[21]-Eclipse CDT IDE software, Maio 2008, http://www.eclipse.org/cdt/

[22]-Notepad ++ IDE software, Maio 2008, http://notepad-plus.sourceforge.net/uk/about.php

[23]-Cygwin - ambiente linux para windows, Maio 2008, http://www.cygwin.com/

[24]-Python Programming Language, Maio 2008, http://www.python.org/

[25]-ASN.1 introduction, Julho 2005, http://asn1.elibel.tm.fr/introduction/index.htm

[26]-XML C parser, Maio 2008, http://xmlsoft.org/index.html

[27]-Simple Network Management Protocol (SNMP), Março 2007, http://www.net-snmp.org/

[28]-lynuxworks, Março 2008, http://www.lynuxworks.com/

[29]-Thales computer - Pentxm2 board, Maio 2008,
http://www.thalescomputers.com/pentxm2.asp

[30]-Microsoft Virtual PC, Julho 2008,
http://www.microsoft.com/windows/products/winfamily/virtualpc/overview.mspx

Anexos

Anexo I.

Implementação da função RTOS_ProcessCreate, que é responsável pela criação de processos ao nível do OSPF.

```
/*****************************************************************************
*
*  Function    :  RTOS_ProcessCreate
*
*  Description  :  Create a Posix Process.
*
*****************************************************************************
*
*  PDL:
*
*   CALL fork // to create a new process and return the oprocessid
*
*   IF this is the child process THEN
*
*     change thread a priority level
*
*     CALL exec // iprocessfilename
*
*   END IF //  this is the child process
*
*****************************************************************************/

RTMC_Status_e RTOS_ProcessCreate(

  const RTMC_Filename_sch iProcessFilename,/* I - Process executable */

                                    /* filename */
  const RTOS_ProcessPriority_si iPriority, /* I - Thread Priority Level */
  RTOS_ProcessId_t * const oProcessId     /* O - Process Id            */
  ) {

 /* Local variable declarations */
  RTMC_Status_e e_FunctionStatus = RTMC_FAILURE;    /* Function Return Status*/
  RTMC_Bool_e e_ErrorDetected = RTMC_FALSE;         /* Control variable for
                                     * error detection       */
  RTMC_Int32_si si_RTOSFunctionCallResult = 0;      /* Control variable for
                                     * RTOS error detection  */

  RTOS_ProcessId_t t_RTOSForkResult = 0;        /* Fork return PID       */
  struct sched_param st_RTOS_SchedParam; /* scheduling paramtrs required by  */
                                /* sched_setparam                 */

  /* Source Code Block */

  /* Parameters verification - oProcessId */
  if ((NULL == oProcessId) ||
     (NULL == iProcessFilename)) {
     e_ErrorDetected = RTMC_TRUE;
     (void) LL_LogEvent(RTMC_EVENT_SWPNULL ,__FUNCTION__,__LINE__,
          "RTOS: Received a null pointer as output parameter");

  }

  /* Verification of the ipriority value */
  if (RTMC_FALSE == e_ErrorDetected) {
    if (iPriority < 0) {
      e_ErrorDetected = RTMC_TRUE;
      (void) LL_LogEvent(RTMC_EVENT_SWPINVALID,__FUNCTION__,__LINE__,
               "RTOS: Received a negative priority value");
    }
  }


  /* CALL fork // to create a new process and return the oprocessid */
```

```
  if (e_ErrorDetected == RTMC_FALSE) {
    t_RTOSForkResult = fork();
    if(RTOS_FORKERROR_SI == t_RTOSForkResult) {
      e_ErrorDetected = RTMC_TRUE;
      (void) LL_LogEvent(RTMC_EVENT_PCREATEFAILED ,__FUNCTION__,__LINE__,
              "RTOS: The current process could not fork");
     }
     else {
      (*oProcessId) = t_RTOSForkResult;
     }
  }

  if (e_ErrorDetected == RTMC_FALSE) {
    /* IF this is the child process THEN  */
    if ( RTOS_CHILD_PROCESS_SI == t_RTOSForkResult)
    {
      /* change thread a priority level */
      st_RTOS_SchedParam.sched_priority = iPriority;
      si_RTOSFunctionCallResult =
                    sched_setparam(getpid(), &st_RTOS_SchedParam);
      if(0 != si_RTOSFunctionCallResult) {
        e_ErrorDetected = RTMC_TRUE;
        (void) LL_LogEvent(RTMC_EVENT_CMDRTOS ,__FUNCTION__,__LINE__,
              "RTOS: An error occurred in RTOS sched_setparam");
       }

      /* CALL exec // iprocessfilename */
      if (e_ErrorDetected == RTMC_FALSE) {
        si_RTOSFunctionCallResult = execl(iProcessFilename, NULL);

        /* if execl returns _there is an error_ */
        e_ErrorDetected = RTMC_TRUE;
        (void) LL_LogEvent(RTMC_EVENT_EXECLFAILED,__FUNCTION__,__LINE__,
            "RTOS: An error occurred in RTOS exec, since it returned");
      }

    }  /* END IF //  this is the child process  */
  }

  if(e_ErrorDetected == RTMC_FALSE) {
    /* Successful exit */
    e_FunctionStatus = RTMC_SUCCESS;
  }

  return ( e_FunctionStatus );
}
```

Anexo II.

Tabelas que definem os testes a implementar para testar uma dada função.

Test Case ID	**RTMC-RTOS-UT-014-TC-0010**
Test Objective	Create a Posix Process successfully
Test Inputs	N/A
Test Outputs	oProcessId = 45641
Test Stubs	fork = 45641 execl = 654
Test Checks	ReturnValue == RTMC_SUCCESS Function shall return defined output values
Notes	Functions provided from OS libraries are considered as wrappers which the truth function can be executed and the returned values can be reset to the desired. Output variables must be initialised differently than expected test outputs

Test Case ID	**RTMC-RTOS-UT-014-TC-0020**
Test Objective	Enter an valid iPriority (minimum allowable)
Test Inputs	iProcessId = minimum allowable - 1
Test Outputs	N/A
Test Stubs	fork = 45641 execl = 654
Test Checks	ReturnValue == RTMC_SUCCESS Function shall return defined output values
Notes	Functions provided from OS libraries are considered as wrappers which the truth function can be executed and the returned values can be reset to the desired. Output variables must be initialised differently than expected test outputs

Test Case ID	**RTMC-RTOS-UT-014-TC-0030**
Test Objective	Enter an valid iPriority (minimum allowable + 1)
Test Inputs	iProcessId = minimum allowable
Test Outputs	N/A
Test Stubs	fork = 45641 execl = 654
Test Checks	ReturnValue == RTMC_SUCCESS Function shall return defined output values

Notes	Functions provided from OS libraries are considered as wrappers which the truth function can be executed and the returned values can be reset to the desired. Output variables must be initialised differently than expected test outputs

Test Case ID	**RTMC-RTOS-UT-014-TC-0040**
Test Objective	Enter a valid iPriority (maximum allowable -1)
Test Inputs	iProcessId = maximum allowable - 1
Test Outputs	N/A
Test Stubs	fork = 45641 execl = 654
Test Checks	ReturnValue == RTMC_SUCCESS Function shall return defined output values
Notes	Functions provided from OS libraries are considered as wrappers which the truth function can be executed and the returned values can be reset to the desired. Output variables must be initialised differently than expected test outputs

Test Case ID	**RTMC-RTOS-UT-014-TC-0050**
Test Objective	Enter a valid iPriority (maximum allowable)
Test Inputs	iProcessId = maximum allowable
Test Outputs	N/A
Test Stubs	fork = 45641 execl = 654
Test Checks	ReturnValue == RTMC_SUCCESS Function shall return defined output values
Notes	Functions provided from OS libraries are considered as wrappers which the truth function can be executed and the returned values can be reset to the desired. Output variables must be initialised differently than expected test outputs

Test Case ID	**RTMC-RTOS-UT-014-TC-0060**
Test Objective	Enter an invalid oProcessId pointer and verify if the function checks that parameter is incorrect
Test Inputs	oProcessId = NULL
Test Outputs	N/A
Test Stubs	fork = 45641 execl = 654
Test Checks	ReturnValue == RTMC_FAILURE
Notes	Functions provided from OS libraries are considered as wrappers which the truth function can be executed and the returned values can be reset to the desired

Test Case ID	RTMC-RTOS-UT-014-TC-0070
Test Objective	If the fork function call fails then this function must detect it and return failure
Test Inputs	N/A
Test Outputs	N/A
Test Stubs	fork = -1 execl = 654
Test Checks	ReturnValue == RTMC_FAILURE
Notes	Functions provided from OS libraries are considered as wrappers which the truth function can be executed and the returned values can be reset to the desired

Test Case ID	RTMC-RTOS-UT-014-TC-0080
Test Objective	If the execl function call fails then this function must detect it and return failure
Test Inputs	N/A
Test Outputs	N/A
Test Stubs	fork = 45641 execl = -1
Test Checks	ReturnValue == RTMC_FAILURE
Notes	Functions provided from OS libraries are considered as wrappers which the truth function can be executed and the returned values can be reset to the desired

Anexo III.

Exemplo de um ficheiro VCT. Este ficheiro é usado para definir características do RTOS LynxOS-178b.

```
////////////////////////////////////////
// Example VCT file
////////////////////////////////////////
// Cyclical Redundancy Check (CRC) value for the VCT.
// If this field is empty, then CRC checking of VCT is disabled.
VctCrc=;
// The part number of the VCT. The value shall be a 12 to 20
// character ASCII string that contains the part number.
VctCpn=104-1234-002;
// Version of the VCT. It allows the software to be backwards
// compatible when the format of the VCT changes.
// The value shall be a number between 0 and 65535.
VctVersion=230;
// The number of virtual machines in the VM table.
// The range is between 1 and 15, inclusive.
NumOfVms=2;
// The number of dynamic device drivers and devices in the DDD table.
// The range shall be between 0 and 47, inclusive. A maximum of 32
// device driver drivers, 16 block devices, and 48 character devices
// can be installed in this section.
NumOfDdds=0;
// The number of File Systems in the FS table.
// The range shall be between 0 and 15, inclusive.
NumOfFs=0;
// The action to perform on module fatal errors. The actions are
// either reset or halt after N errors from power up (cold and warm).
// Set to 0 if the system is to be reset on every occurrence of
// a module fatal error.
// Set to N, where N is greater than 0 and less than 255, if the
// system is to be halted after N module fatal errors.
ActionOnModuleErr=3;
// Hardware strap information
// This feature is not supported.
SideStrapToProcess=All;
ActionOnInvalidSideStrap=Normal;
ActionOnInvalidSlotStrap=Normal;
ActionOnInvalidProcessorStrap=Normal;
// On power up, four possible start conditions can occur; factory,
// field, cold, and warm. The first three conditions result in the
// ColdStartSchedule being executed. The last condition results in
// the WarmStartSchedule being executed.
// The cold start schedule for the system. The value specifies the
// number of system ticks (e.g. 1 millisecond) each VM executes
// during a major (repeating) frame.
// For example, a value of "0[5] 1[30] 0[10] 1[5]" defines a major
// of 50 system ticks where VM0 runs for 5 ticks, followed by VM 1
// for 30 ticks, followed by VM0 for 10 ticks, followed by VM 2 for
// 5 ticks.
ColdStartSchedule=0[40] 1[10];
// The warm start schedule for the system.
// After the major frame is completed, the schedule is repeated until
// the WarmStartDuration is reached. At this point, the schedule
// switches to the RunTimeSchedule.
WarmStartSchedule=0[40] 1[10];
// The run-time schedule for the system.
RunTimeSchedule=0[40] 1[10];
// The length of time the cold start schedule executes before
// switching over to the run-time schedule.
ColdStartDuration=15000000; //15 seconds
// The length of time the warm start schedule executes before
// switching over to the run-time schedule.
WarmStartDuration=1000000; //1 second
// The length of time the system executes in IBIT mode before
// resetting to Normal mode.
IbitDuration=900000000;
// TBD
// This feature is not supported.
NetworkInterface=local;
// TBD
// This feature is not supported.
LoadShedState=Enable;
LoadShedMode=Static;
////////////////////////////////////////
// Virtual Machines
////////////////////////////////////////
<VM0>
// The supplementary group membership, if any, of the virtual machine.
GroupIds=2 1 0;
// The name of the virtual machine. It is used to logically identify
// the virtual machine.
LogicalName=System;
```

```
// The CommandLine is used to start the master process of the VM.
// It consists of a sequence of words separated by space characters.
// All words are passed to the master program in the argv[] array.
// The argc parameter is calculated and set appropriately.
CommandLine=/bin/runshell /dev/com1 /bin/bash;
// Environment variables that are VM-specific.
EnvironmentVars=VAR1=var1
VAR2=var2;
// The total amount of system RAM allocated to this VM. This
// includes memory used by the VM's application(s) (text, data, stack,
// heap segments, etc.) and internal kernel resources required to
// support a VM.
SysRamMemLim=20971520;
// The directory path and file name of the Standard In character
// device node. During initialization, cinit opens this device node
// and re-directs Standard In to this device.
// The device may reference a static or dynamic device driver.
StdInNodeFname=/dev/com1;
// The directory path and file name of the Standard Out character
// device node. During initialization, cinit opens this device node
// and re-directs Standard Out to this device.
// The device may reference a static or dynamic device driver.
StdOutNodeFname=/dev/com1;
// The directory path and file name of the Standard Error character
// device node. During initialization, cinit opens this device node
// and re-directs Standard Error to this device.
// The device may reference a static or dynamic device driver.
StdErrNodeFname=/dev/com1;
// The path name of the VM's working directory (i.e.home directory).
WorkingDir=/;
// The path name of a directory in the root file system where the VM's
// RAM file system will be mounted. The directory path is the absolute
// path or an empty string if there is no RAM file system for this VM.
// For VMs with RAM file systems, the RamFsMount should be set to /tmp.
RamFsMount=/tmp;
// The size of the VM's RAM file system. The value of RamFsLim is in
// bytes ranging from 0 to SystemRamMemLim. The RamFsLim shall be on
// a PPC page boundary (e.g. 4096 byte increments).
// If a RAM File system is not needed then the value shall be set to 0.
RamFsLim=3145728;
// The maximum number of Inodes allowed in the file system. One Inode
// is required for every file and directory.
RamFsNumOfInodes=200;
// The action to perform on VM fatal errors. The actions are either
// reset or halt after N errors from the current power cycle (cold
// and warm).
ActionOnVmErr=0;
// How the system software will handle a SIGILL exception when it
// occurs. This processor exception occurs when the processor
// executes an illegal instruction.
// The value of ActionOnSigillExcp is either "Default" or "Override".
// Default forces a VM fatal error and Override allows the VM to
// handle the exception.
// Should be set to "Default" for all VMs except for the first VM.
// Set to "Override" for the first VM (system partition, UserId=0).
ActionOnSigillExcp=Override;
// How the system software will handle a SIGFPE exception when it
// occurs. This processor exception occurs on floating point
// errors.
ActionOnSigfpeExcp=Override;
// How the system software will handle a SIGSEGV exception when it
// occurs. This processor exception occurs when Memory Management
// Unit (MMU) detects a segment violation.
ActionOnSigsegvExcp=Override;
// How the system software will handle a SIGBUS exception when it
// occurs. This processor exception occurs when an unauthorized
// access to memory occurs.
ActionOnSigbusExcp=Override;
// TBD
// These features are not supported.
PersStorOnLocalLim=1024;
PersStorOnPmcLim=0;
RamHoldUpStorLim=0;
// NB: The following per-VM resource limits are ignored for VM0.
// VM0 resource limits are specified in the uparam.h header file.
// The maximum number of processes on the system allocated to this VM.
// Any attempts by the VM to exceed this allocation will be rejected.
// The value of NumOfProcessesLim shall be between 2 and 64, inclusive.
NumOfProcessesLim=15;
// The maximum number of threads on the system allocated to this VM.
// Any attempts by the VM to exceed this allocation will be rejected.
// The value NumOfThreadsLim of shall be between 2 and 256, inclusive.
// This value must be equal to or greater than NumOfProcessesLim
// because a process always has a main thread.
NumOfThreadsLim=30;
// The maximum number of POSIX timers, available through
```

```
// timer_create(), on the system allocated to this VM. Any attempts
// by the VM to exceed this allocation will be rejected. The value
// of NumOfTimerLim shall be between 1 and 0x7fff, inclusive.
NumOfTimersLim=4;
// The size of the file system cache located in system Ram. One
// cache area is used for all file systems accessible by this VM.
// The sizing of the cache will directly affect the performance
// of the file systems since all read and write data goes through
// the cache.
FsCacheLim=4194304;
// The file cache attribute for the VM file system cache.
// The FsCacheAttr shall be set to WriteThrough or WriteBack.
// For write-through, writes to a file system will be written to
// cache and to the physical media during the write() system call.
// For write-back, writes to a file system will be written to the
// file cache.
FsCacheAttr=WriteThrough;
// TBD
FsNumOfInodesLim=150;
FsNumOfRecordLocksLim=60;
// The maximum number of file descriptors which can be in use at
// one time.
// The value of NumOfOpenFdsPerVmLim shall be between 3 and 32767,
// inclusive. A minimum of 3 is required for stdin, stderr, and
// stdout.
NumOfOpenFdsPerVmLim=150;
// The number of different named message queues a VM can open.
NumOfMsgQueuesLim=10;
// The maximum number of pipes which can be opened. There are named
// and unnamed pipes. Pipes consume other resources as well. An open
// to a named pipe consumes one file descriptor while an open of an
// unnamed pipe consumes two file descriptors.
NumOfPipesLim=80;
// TBD
NumOfSignalsLim=30;
// The maximum number of shared memory objects available. Shared
// memory objects can be used directly through the shm_open() POSIX
// call. They are also used indirectly by Posix message queues so
// add one for each message queue created. Shared memory objects
// also use a file descriptor.
NumOfSharedMemObjsLim=10;
// The maximum number of Posix named and unnamed semaphores (POSIX
// calls sem_open() and sem_init()).
NumOfSemaphoresLim=30;
</VM0>
<VM1>
GroupIds=;
LogicalName=Virtual Machine 1;
CommandLine=/bin/runshell /dev/com2 /bin/bash;
EnvironmentVars=;
StdInNodeFname=/dev/com2;
StdOutNodeFname=/dev/com2;
StdErrNodeFname=/dev/com2;
WorkingDir=/;
ActionOnVmErr=3;
ActionOnSigillExcp=Default;
ActionOnSigfpeExcp=Default;
ActionOnSigsegvExcp=Default;
ActionOnSigbusExcp=Default;
RamFsLim=1048576;
RamFsNumOfInodes=200;
RamFsMount=/tmp;
PersStorOnLocalLim=0;
PersStorOnPmcLim=0;
RamHoldUpStorLim=0;
SysRamMemLim=20971520;
NumOfOpenFdsPerVmLim=150;
FsNumOfInodesLim=150;
FsCacheLim=4194304;
FsNumOfRecordLocksLim=60;
FsCacheAttr=WriteThrough;
NumOfMsgQueuesLim=10;
NumOfPipesLim=80;
NumOfSignalsLim=30;
NumOfSharedMemObjsLim=10;
NumOfSemaphoresLim=30;
NumOfTimersLim=4;
NumOfProcessesLim=10;
NumOfThreadsLim=30;
</VM1>
//////////////////////////////////////////
// Start of dynamic devices and drivers
//////////////////////////////////////////
<DDD0>
// The type of the dynamic device driver, character or block (used
// by drinstall() and mknod()). The value of Type shall be either:
```

```
// "c" or "b".
Type=c;
// The driver id for statically installed drivers (used by
// devinstall()). The value is required to install dynamic devices
// for drivers that were installed statically by the BSP.
DriverId=;
// The full directory path and file name of the driver object module
// (used by drinstall()). The path name shall be the absolute path
// and file name or an empty string if installing only a device.
ObjectFname=/usr/bin/llscsi.dldd;
// The full directory path and file name of the device info (driver's
// configuration file, used by devinstall()). The path name shall be
// the absolute path and file name of the file.
InfoFname=/usr/etc/llscsi.info;
// The number of minor devices supported by this driver (used by
// mknod()). The value of NumOfMinorDevs shall be from 0 to 255,
// inclusive. A value of 0 or 1 will create one minor device.
NumOfMinorDevs=0;
// The absolute directory and base filename of the character device
// node or nodes, as appropriate. If the NumberOfMinorDevices is
// equal to 0 then the BaseCharNodeFname will be directly used. If
// the NumberOfMinorDevices is greater than 0 then a number will be
// appended to base name starting with 0. The number will be
// incremented by one for each minor device. The BaseCharNodeFname
// shall be the absolute path and base filename of the character
// device node or nodes.
// The absolute path for all dynamic device nodes shall be /dev/ddev/.
// In LynxOS-178, all device drivers must have a character device node,
// even block device devices.
BaseCharNodeFname=/dev/ddev/llscsi;
// Same for the block device nodes.
BaseBlockNodeFname=;
// The UserId (Uid) of the owner of the device node.
// The value of OwnerId shall be from 0 to NumOfVms-1, inclusive.
OwnerId=0;
// The GroupId (Gid) of the owner of the device node.
// The value of GroupId shall be from 0 to 32765, inclusive.
GroupId=0;
// The Unix file permission to be assigned to the device node file.
Permissions=0600;
</DDD0>
<DDD1>
Type=b;
DriverId=;
ObjectFname=/usr/bin/hdscsi.dldd;
InfoFname=/usr/etc/hdscsi.info;
NumOfMinorDevs=67;
BaseBlockNodeFname=/dev/ddev/sd2m;
BaseCharNodeFname=/dev/ddev/rsd2m;
OwnerId=0;
GroupId=0;
Permissions=0600;
</DDD1>
//////////////////////////////////////////
// Start of filesystems
//////////////////////////////////////////
<FS0>
// The path name of a directory in the root file system where the
// file system will be mounted. The directory path shall be an
// absolute path.
// All configurable file systems shall be mounted off the /mnt
// directory branch.
Mount=/mnt/a;
// The directory path and file name of the block device node that
// is used to access the file system. The character device may
// reference a static or dynamic device driver. When referencing
// a dynamic driver, the NodeFname will equal the BaseCharNodeFname
// data field of an entry in the Dynamic Device Driver table.
// The NodeFname shall be the absolute path and filename of the file
// system device node.
NodeFname=/dev/sdncr.0e;
// The maximum number of Inodes allowed in the file system. One
// Inode is required for every file and directory.
// If the value is Null then 1 inode in every 16 data blocks shall
// be reseved for inodes.
NumOfInodes=;
// Command line parameters that are passed to the ffsck utility
// when the file system is verified or checked.
// Warning: cinit expects to find the ffsck utility in /usr/bin.
FfsckArgs=;
// Command line parameters that are passed to the mkffs utility
// when the file system is created.
// If ffsck fails to find a filesystem on a device, mkffs utility
// will be called to create a new filesystem on that device.
// Warning: cinit expects to find the mkffs utility in /usr/bin.
MkffsArgs=;
```

```
// The UserId (Uid) of the owner of the File System mount point.
OwnerId=0;
// The GroupId (Gid) of the owner of the File System mount point.
GroupId=0;
// The Unix file permission to be assigned to the File System mount
// point. This value also describes the filesystem access mode.
// Filesystems with read-write access mode are only mounted by the VM
// identified by the OwnerId. Read-only filesystems are mounted by
// all VM's.
Permissions=0777;
// How the file system integrity checking is accomplished. The value
// of the IntegrityAttr shall be set to "Hardware", "Software",
// or "" (null string).
IntegrityAttr=;
// TBD
// This feature is not supported.
DataloadHdrPath=;
</FS0>
<FS1>
Mount=/mnt/b;
NodeFname=/dev/sdncr.0c;
MkffsArgs=;
FfsckArgs=;
OwnerId=0;
GroupId=0;Permissions=0777;
IntegrityAttr=;
DataloadHdrPath=;
</FS1>
```

Anexo IV.

Exemplo do ficheiro lynxos-178.spec que permite configurar a organização e composição do sistema de ficheiros do LynxOS-178b.

```
directory=/etc/VM0 owner=0 group=0 mode=drwxr-xr-x
file=lcsd.conf source=$(ENV_PREFIX)/etc/VM0/lcsd.conf owner=0 group=0 mode=-rw-r--r--
directory=/etc/VM1 owner=0 group=0 mode=drwxr-xr-x
file=lcsd.conf source=$(ENV_PREFIX)/etc/VM1/lcsd.conf owner=0 group=0 mode=-rw-r--r--
directory=/etc/VM2 owner=0 group=0 mode=drwxr-xr-x
file=lcsd.conf source=$(ENV_PREFIX)/etc/VM2/lcsd.conf owner=0 group=0 mode=-rw-r--r--
file=tftpd source=$(ENV_PREFIX)/net/tftpd owner=0 group=0 mode=-rxr-xr-x
file=ftpd source=$(ENV_PREFIX)/net/ftpd owner=0 group=0 mode=-rxr-xr-x
file=inetd source=$(ENV_PREFIX)/net/inetd owner=0 group=0 mode=-rsr-sr-x
file=arp source=$(ENV_PREFIX)/bin/arp owner=0 group=0 mode=-r-xr-xr-x
file=ifconfig source=$(ENV_PREFIX)/bin/ifconfig owner=0 group=0 mode=-r-xr-xr-x
file=ping source=$(ENV_PREFIX)/bin/ping owner=0 group=0 mode=-rxr-xr-x
file=route source=$(ENV_PREFIX)/bin/route owner=0 group=0 mode=-rsr-sr-x
file=ftp source=$(ENV_PREFIX)/bin/ftp owner=0 group=0 mode=-r-sr-sr-x
file=tftp source=$(ENV_PREFIX)/bin/tftp owner=0 group=0 mode=-rsr-sr-x
file=tcpdump source=$(ENV_PREFIX)/bin/tcpdump owner=0 group=0 mode=-r-sr-sr-x

…
```

Anexo V.

Exemplo de um ficheiro config.tbl onde se especificam quais os controladores que queremos no LynxOS-178b.

```
# ARINC 653 driver
I:arinc653.cfg

# serial port driver
I:rs232.cfg

#POSIX terminal driver
I:com.cfg

# Serial ATA (SATA) hard drive devices
I:sata.cfg

# IDE Flash drive and disk devices
I:ide.cfg

#TCP/IP Support
I:hbtcpip.cfg
I:gt.cfg

# Enabling/Disabling NFS Support
#I:nullnfs.cfg
I:nfs.cfg
```

Anexo VI.

Exemplos de testes efectuados para testar a funcinalidade e percormance de serviços.

Description	Performance (Kb/s) when relavante	Status
Disk Read in a Single Partition using different file sizes 1Mb up to 2000Mb	53906,09	Passed
Disk Write in a Single Partition using different file sizes 1Mb up to 2000Mb	50210,85	Passed
Disk Read in a Multi-Partition using different file sizes 1Mb up to 2000Mb	33196,1	Passed
Disk Write in a Multi-Partition using different file sizes 1Mb up to 2000Mb	8484,31	Passed
Flash Read in a Single Partition using different file sizes 1Mb up to 2000Mb	1735,92	Passed
Flash Write in a Single Partition using different file sizes 1Mb up to 2000Mb	206,51	Passed
Flash Read in a Multi-Partition using different file sizes 1Mb up to 2000Mb	933,18	Passed
Flash Write in a Multi-Partition using different file sizes 1Mb up to 2000Mb	356,68	Passed
SNMP Subagent with a simple MIB, Set and get values		Failed
SNMP Subagent with a simple MIB, Add rows to tables		Failed
ASN.1 Encode a medium complexity message	16329,29	Passed
ASN.1 Decode a medium complexity message	744035,75	Passed
XML Decode a medium complexity message		Passed

Description	Performance (Kb/s) when relavante	Status
TCP-BSD Single Partition Test, using different packages sizes from 1Kb, up to 64Kb	68310,75	Passed
UDP-BSD Single Partition Test, using different packages sizes from 1Kb, up to 64Kb	113644,52	Passed
FTP-BSD Single Partition Test, using large files from 1 Mbyte to 250 Mbytes	PUT – 4612,45/GET – 7920,20	Passed
TFTP-BSD Single Partition Test, using large files from 1 Mbyte to 250 Mbytes	PUT – 2438,1/GET – 2048,00	Passed
IGMP-BSD Single Partition Test	N/A	Passed
ICMP-BSD Single Partition Test	n/a	Passed
UDP Multicast Single Partition, using different packages sizes from 1Kb, up to 64Kb	N/A	Passed