Faculdade de Engenharia da Universidade do Porto

Licenciatura em Engenharia Informática e Computação

Suporte de Gestão de Memória em RTEMS na Critical Software

Relatório de Projecto do MIEIC 2007/08

André Neves

Orientador na FEUP: Miguel Pimenta Monteiro

Tutor na Critical Software: Pedro Luciano Braga

Fevereiro de 2008

Dedicado à comunidade de Software Livre,

pela qual tenho ganho mais e mais respeito à medida que os anos passam.

Resumo

RTEMS é um sistema operativo de resposta em tempo real usado no desenvolvimento de aplicações embutidas críticas, que acarretam perdas consideráveis (materiais ou mesmo humanas) em caso de falha.

RTEMS é usado para desenvolvimento de aplicações de diversos âmbitos, tanto do âmbito aeroespacial (caso das aplicações desenvolvidas na Critical Software) como em sectores tão diferentes como o da defesa, médico, das comunicações, científico, etc.

Dada a criticidade das aplicações desenvolvidas, estas requerem níveis elevados de confiança tanto no sistema como no determinismo do seu desempenho, pelo que uma parte considerável do esforço e responsabilidade envolvidos são gastos em revisões, testes e validações que visam assegurar a qualidade do produto.

Embora nem todas as tarefas de um projecto sejam críticas, um erro numa tarefa não-crítica pode propagar-se para o resto do sistema, porque o RTEMS não tem compartimentação de memória.

Este projecto tem por objectivo fazer uma prova de conceito da validade da implementação do suporte para MMU (unidade de gestão de memória) em RTEMS. O suporte para MMU permitiria criar protecção de memória ao compartimentá-la, evitando a propagação de erros. O resultado seria um sistema dotado de um maior grau de confiança e mais robusto permitindo que os recursos de validação da qualidade e funcionamento se foquem mais nas tarefas críticas a desenvolver.

O projecto iniciou-se com um estudo detalhado das tecnologias envolvidas, organizando uma base de conhecimento necessária tanto para o seu adequado desenvolvimento como para facilitar a eventual posterior familiarização de outras entidades com o projecto.

De seguida foi analisado o âmbito do problema e as limitações impostas, culminando na produção de um Documento de Requisitos que define as funcionalidades e restrições do sistema proposto.

Por fim desenvolveu-se um Documento de Arquitectura, que delineia e define os módulos da solução e os associa com requisitos, explicando-os em detalhe.

Estima-se que sejam necessário um total de 6 meses de trabalho de uma pessoa, para a implementação da arquitectura proposta.

Foi um trabalho decididamente interessante e que contribuiu para um grande alargamento de conhecimentos, dada a componente enorme de pesquisa e aprendizagem envolvida, que culminou num aprofundar do interesse pela área de sistemas embutidos.

Agradecimentos

Quero agradecer a Pedro Braga, por me ter apoiado, guiado, ajudado, ao longo do estágio na Critical Software. Obrigado à ilha de trabalho por toda a discussão que providenciaram e, claro, pela companhia. Obrigado também a Luís Henriques "pela perninha dada".

Obrigado ao professor António Miguel Pimenta Monteiro, por ter acedido a ser meu orientador de projecto.

Agradecimentos, claro, prestados ao prestável Secretariado de Curso: Lina e Mónica.

Que me perdoem aqueles que contribuiram para a existência e sucesso deste projecto, sem serem reconhecidos ou lembrados. Aprecio o vosso contributo, embora não o saiba quantificar nem atribuir aos efeitos que teve.

Por fim, agradecimentos à minha família por todo o apoio prestado durante os meus estudos, que culminam neste relatório que assinala o término do curso universitário. Sem eles nada de mim seria eu.

Índice de Conteúdos

1 Introdução

1.1 Apresentação da Instituição Onde Decorreu o Projecto – Critical Software

Fundada em 1998, a Critical Software é uma empresa focada no desenvolvimento de soluções de qualidade para mercados competitivos. Tem como *core-business* o desenvolvimento de soluções, serviços e tecnologias para os sistemas de informação críticos das empresas, respondendo às necessidades de clientes de mercados como o das telecomunicações, o sector público, a indústria, o sector aeroespacial e a defesa. Presta também serviços de consultoria e auditoria na área das Tecnologias da Informação.

Os projectos desenvolvidos pela empresa enquadram-se no âmbito de EAI (integração de aplicações corporativas), sistemas embutidos e de resposta em tempo real, confiabilidade, verificação e validação de código, redes e comunicações, bases de dados, HPC (computação de alto desempenho) e MES (sistemas de informação fabril).

1.2 O Projecto Suporte de Gestão de Memória na Instituição de Estágio Critical Software

RTEMS (Real Time Executive for Multiprocessor Systems) é um sistema operativo de resposta em tempo real usado no desenvolvimento de aplicações embutidas que são, tipicamente, sistemas críticos ou com componentes cujo desempenho é crítico.

MMU (unidade de gestão de memória) é um componente físico do processador que gere a virtualização da memória, traduzindo endereços virtuais em endereços físicos e lançando uma excepção se a tradução for inválida no contexto actual. O correcto funcionamento da MMU depende de uma estrutura em memória que guarda as traduções para cada contexto, estrutura esta normalmente gerida pelo Sistema Operativo e protegida de acessos por outrem pelas definições nela própria contidas.

O RTEMS não tem suporte para MMU, usando um modelo de *flat memory*. O acesso à memória física é feito directamente, sem tradução, e todos os endereços são acessíveis. Nos casos em que o sistema tem MMU, o RTEMS mantém-na desligada e os endereços virtuais são mapeados directamente para endereços físicos com o mesmo valor.

Os sistemas que geralmente são desenvolvidos em RTEMS requerem níveis elevados de confiança no sistema e no seu desempenho devido à sua criticidade (ver [01] Exemplos de aplicações desenvolvidas em RTEMS), pelo que uma parte considerável do esforço e responsabilidade envolvidos são gastos em revisões, testes e validações que visam assegurar a qualidade do produto. A adição de gestão de memória ao RTEMS (através da implementação de suporte para MMU) permitiria impedir as tarefas não-críticas de afectar o sistema, resultando num sistema mais robusto e possibilitando que os recursos de validação da qualidade e funcionamento se foquem mais nas tarefas críticas do sistema.

Com este projecto pretende-se fazer uma prova de conceito sobre a possibilidade de alterar o RTEMS para suportar uma unidade de gestão de memória no processador LEON2, por forma a possibilitar a referida protecção de memória, devendo-se no entanto ter uma abordagem estruturada que permita portá-lo para outros processadores e arquitecturas.

1.3 Glossário

Ao longo do relatório irão ser referidos diversos conceitos e acrónimos que podem ser desconhecidos a pessoas que não estejam familiarizadas com o projecto ou o seu âmbito. Aqueles com que iremos deparar com mais frequência definem-se de seguida:

- BSP – Board Support Package, código [do RTEMS (neste caso)] que é específico a uma arquitectura (SPARC, i386, PowerPC, etc);
- CVS – Concurrent Versions System, um sistema de controlo de versões;
- CMMI 4 – Capability Maturity Model Integration, é um modelo de referência de práticas que visam maturar os processos de uma empresa;
- IDE – ambiente de desenvolvimento integrado (Integrated Development Environment);
- LEON – família de processadores, de arquitectura SPARC, para os quais este projecto foi desenvolvido.
- MIEIC – Mestrado Integrado em Engenharia Informática e Computação;
- MMU – Memory Management Unit, componente do processador que gere a virtualização da memória e a sua tradução;
- RISC – Reduced Instruction Set Computer, processador cuja design é baseado num conjunto de instruções simples;
- RTEMS – Real Time Executive for Multiprocessor Systems, sistema operativo de resposta em tempo real usado em sistemas embutidos;
- Tarefa (task) – nome que é dado, em RTEMS, a uma entidade de execução independente (processo);
- Tarefa de sistema – tarefa que corre em modo privilegiado do processador;
- Tarefa de utilizador – tarefa que corre em modo protegido do processador;
- TLB – Translation Lookaside Buffer, memória associativa interna à MMU onde são guardadas as traduções de memória virtual para memória física usadas mais recentemente, funcionando como cache de traduções.

1.4 Organização e Temas Abordados no Presente Relatório

No presente capítulo, Introdução, descreveram-se a empresa e o contexto do projecto, apresentou-se um pequeno glossário e refere-se agora a organização e temas deste relatório.é apresentado o glossário, referida a organização do relatório e descritos a empresa e o contexto do estágio.

No segundo capítulo, Análise do Problema, o leitor é contextualizado com toda a envolvente do problema: o estado corrente do RTEMS, as vantages e desvantagens das alterações propostas, e os objectivos do projecto.

No terceiro capítulo, Revisão Tecnológica, são explicados os conceitos tecnológicos envolvidos: sistemas embutidos. sistemas de resposta em tempo real, memória virtual, memória segmentada, funcionamento da MMU em SPARC/LEON2.

No quarto capítulo, Revisão de Ferramentas, são analisadas as ferramentas usadas.

No quinto capítulo, Requisitos, são apresentados os requisitos definidos no âmbito do projecto.

No sexto capítulo, Casos de Uso, são apresentados os casos de uso reunidos durante o projecto para satisfação dos Requisitos.

No sétimo capítulo, Visão Geral de Arquitectura, é apresentada sucintamente a arquitectura proposta como solução.

No oitavo capítulo apresentam-se, com algum detalhe, os aspectos importantes da arquitectura proposta como solução.

No nono e último capítulo são avaliados resultados, apresentadas as conclusões e delineado o trabalho futuro.

2 Análise do Problema

2.1 Sistemas Típicos de RTEMS

RTEMS (*Real-Time Operating System for Multiprocessor Systems*) é um sistema operativo de tempo real desenvolvido para sistemas embutidos. Trata-se de um sistema Open Source cujas principais características são o suporte de vários standards (POSIX 1003.1b API, Classic API baseada em RTEID/ORKID, uITRON 3.0, etc), suporte para multi-processamento (homogéneo/heterogéneo), escalonamento preemptivo baseado em prioridades e em eventos, mecanismos de comunicação e sincronização entre tarefas, implementação de mecanismos de prevenção de *deadlocks* (priority inheritance, priority ceiling), suporte de diversas plataformas (SPARC, PowerPC, x86, etc), etc.

Devido à necessidade de resposta em tempo real e às limitações impostas pela capacidade do hardware alvo, o RTEMS é um sistema minimalista que se foca em determinismo e alta performance. (Como exemplo demonstrativo das limitações de hardware, note-se que o Atmel LEON2-FT, um dos processadores usado no sector aeroespacial, funciona a 100MHz.)

Os projectos desenvolvidos em RTEMS (ver [01] Exemplos de aplicações desenvolvidas em RTEMS) incluem tanto aplicações evidentemente críticas (cujo mau funcionamento acarretaria perdas de vidas humanas ou perdas materiais consideráveis, tais como aplicações médicas, aeroespaciais e de aviação, industriais, militares, etc) como aplicações não críticas (tais como aplicações científicas, robóticas, musicais e de som, industriais ou para comunicações).

O RTEMS não tem suporte para MMU, usando um modelo de *flat memory*. O acesso à memória física é feito directamente, sem tradução, e todos os endereços são acessíveis. Nos casos em que o sistema tem MMU, o RTEMS mantém-na desligada e os endereços virtuais são mapeados directamente para endereços físicos com o mesmo valor.

Os modos de execução do processador também não são utilizados, correndo este sempre em modo privilegiado – o que resulta na inexistência de restrição de execução de instruções privilegiadas.

2.2 Domínio do Projecto

A gestão de memória é normalmente descartada nos sistemas operativos em tempo real, em detrimento de um aumento do determinismo e da eficiência. Perde-se assim a capacidade de proteger a memória, mas ganha-se capacidade de processamento (que não é gasta no processo de gestão e protecção da memória) e previsibilidade.

Como normalmente os projectos de sistemas embutidos têm desenvolvimento de software monolítico, não é relevante haver gestão de memória, já que é conhecido o comportamento de todo o sistema.

No entanto, a protecção de memória revela-se um importante aliado no desenvolvimento de projectos de grande envergadura que envolvam variadas tarefas de diferentes níveis de prioridade: ao permitir a protecção de memória, ou seja, isolar a memória de uma tarefa ou processo relativamente a acessos executados por outros, permite uma maior concentração no desenvolvimento das tarefas mais importantes, sem medo de que as menos importantes possam influenciar o desempenho ou a execução das mais importantes.

2.3 Vantagens

Os sistemas desenvolvidos em RTEMS na Critical Software têm por alvo usos na indústria aeroespacial, tendo um nível de criticidade que requer um investimento considerável de recursos em revisões, testes e validações que visam assegurar a qualidade do produto.

A adição de protecção de memória permitiria impedir as tarefas não-críticas de afectar o resto do sistema, podendo então os recursos de testes, validação e verificação serem focados mais nas tarefas críticas. Tanto a protecção de memória como o novo foco dos recursos nas tarefas mais críticas aumentariam o desempenho e robustez do sistema.

2.4 Desvantagens

Ao aumentar o número de instruções usadas no manuseamento das tarefas, o sistema demorará mais tempo a mudar de tarefa. Tanto o tempo demorado a inicializar o sistema como alguns dos acessos à memória serão mais lentos, porque será preciso carregar a tradução de memória para o TLB da MMU (Translation Lookaside Buffer, explicado na secção Funcionamento da MMU em SPARC/LEON2).

Embora não sejam exageradamente grandes, estes dois pontos são relevantes – no âmbito de sistemas embutidos todos os aumentos de *overhead* ou baixas de determinismo contam.

2.5 Objectivos

O objectivo deste projecto é produzir uma prova do conceito, de forma a validá-lo. O conceito é a produção de uma extensão do RTEMS que permita usar a MMU de forma a fornecer protecção de memória entre tarefas e entre tarefas e o núcleo (*kernel*) do sistema operativo. Deve ser feito um estudo de todas as modificações funcionais que serão necessárias efectuar ao RTEMS e as implicações ao nível da interface de aplicações do sistema.

3 Revisão Tecnológica

Este projecto consiste em fazer uma prova de conceito da executabilidade da extensão de RTEMS para permitir a gestão de memória. Assim, as tecnologias estão definidas à partida, não havendo lugar a pesquisa comparativa.

O estudo focou-se, então, directamente nas tecnologias envolvidas, que são de seguida expostas.

3.1 Sistemas Embutidos (Embedded)

Um sistema embutido é um sistema computacional criado para executar tarefas dedicadas. As aplicações são geralmente compiladas num único objecto executável, que inclui o sistema operativo. O desenvolvimento dá-se, por norma, num sistema alternativo (sistema de desenvolvimento) devido às limitações de desempenho do sistema alvo, sendo o executável final depois transferido para o hardware alvo – onde será executado.

As aplicações desenvolvidas são, por norma, optimizadas por forma a aumentar o desempenho e a fiabilidade, o que se revela especialmente importante dadas as limitações técnicas, económicas e de desempenho dos sistemas (componentes de desempenho limitado e muito caros, e sistemas críticos).

3.2 Sistemas de Resposta em Tempo Real

Os sistemas de resposta em tempo real são sistemas cujo tempo de resposta a um estímulo deve respeitar limites temporais (*deadlines*). Estes limites de latência devem ser respeitados independentemente da situação de carga do sistema e são definidos, claro, dependendo do sistema/aplicação em questão.

Por forma a cumprir estes limites de latência, o determinismo e a fiabilidade do sistema são essenciais. A característica mais importante de um sistema torna-se não a sua performance média (ou máxima), mas a sua performance mínima – pois é esta que vai ter uma importância fulcral no desempenho adequado do sistema e no cumprimento das restrições.

As aplicações de um sistema de resposta em tempo real são, geralmente, “mission-critical”, isto é: a missão do sistema só é bem sucedida se nenhuma das aplicações falhar a sua execução ou o limite de latência.

Há dois sub-tipos de sistemas de resposta em tempo real:

- **Soft real time** – em que a falha em responder a um estímulo dentro dos limites de tempo definidos é tolerável, se bem que não desejável (exemplo: uma aplicação de apresentação de vídeo que falhe o processamento de um frame dentro do tempo limite);
- **Hard real time** – em que a falha em responder a um estímulo dentro dos limites de tempo definidos é incomportável, potencialmente resultando numa falha crítica do sistema (exemplo: uma aplicação que controle a reacção num reactor de fissão nuclear).

3.3 Memória Virtual

Virtualização é uma técnica que permite esconder (das aplicações) características físicas de recursos de computação, ao fornecer uma camada de abstracção.

Memória Virtual é uma técnica de virtualização de memória, em que os endereços de memória que são acedidos pelas aplicações são na realidade endereços não-físicos(virtuais), que são traduzidos (em tempo-real, aquando do acesso) para os reais endereços físicos. Isto permite ao sistema fornecer ou negar acesso a uma dada região de memória, fornecer às aplicações memória que parece contígua mas na realidade está fragmentada pelos recursos

de memória do sistema, e até aumentar o espaço de memória utilizável usando trocas, quando necessário, com memória secundária de maior capacidade (por exemplo: http://en.wikipedia.org/wiki/DOS/4GW).

Ao serem fornecidos vários espaços de endereçamento virtual providencia-se a possibilidade de ter mapeamentos de memória virtual para memória física diferentes para cada aplicação/tarefa/processo.

Os sistemas embutidos normalmente não usam memória virtual, por forma a aumentar a velocidade, consistência e determinismo da resposta.

3.3.1 Memória Paginada

No caso da memória paginada, o espaço de endereçamento virtual de uma aplicação, assim como o espaço da memória física, são divididos em páginas, blocos de endereços de memória contíguos e com fronteiras bem definidas. As páginas têm tamanho fixo, cabendo à MMU fazer o mapeamento entre páginas do espaço virtual e físico.

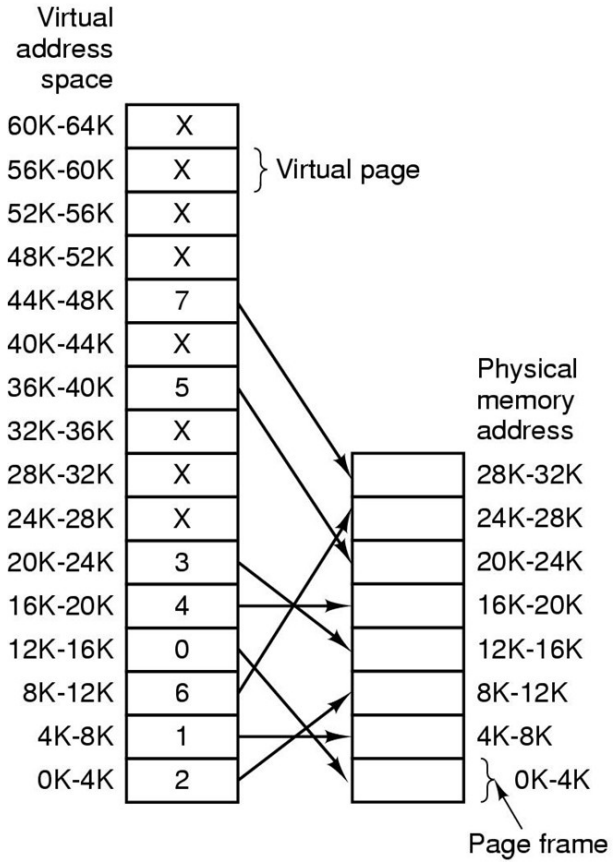

Figura 1: Memória Paginada

3.3.2 Memória Paginada Multi-Nível

A memória paginada multi-nível é uma especialização do caso da memória paginada.

Em vez da paginação se dar em blocos de tamanho fixo, a memória é dividida numa série de páginas que por sua vez podem ser novamente divididas, e assim sucessivamente até ao nível máximo de profundidade de segmentação (3 níveis, no caso da arquitectura SPARC).

Isto permite adequar a granularidade ao uso desejado do sistema, contribuindo para um uso mais racionado dos recursos disponíveis.

A explicação dada em 3.4.2 Page Table abaixo permite compreender este processo de segmentação como é feito na arquitectura SPARC..

3.3.3 Memória Segmentada

A memória segmentada é uma especialização alternativa da memória paginada.

Nestal há, em vez de vários níveis de granularidade, um tamanho variável para as páginas, sendo o resto da implementação semelhante à memória paginada com os endereços virtuais a serem divididos em segmento e *offset*.

3.4 Funcionamento da MMU em SPARC/LEON2

A MMU é um componente de hardware do processador. Apoiada numa Page Table que tem as traduções de memória, a MMU virtualiza os endereços físicos de memória orientando e gerindo os acessos à memória, sendo que a inicialização, protecção e gestão desta Page Table é da responsabilidade do sistema operativo.

A virtualização ou não de memória é transparente, desde que o sistema operativo assegure a definição de *handlers* para as excepções e a configuração da MMU e da Page Table.

Se a MMU estiver desligada, um acesso a um endereço de memória passa na mesma pela MMU, mas o endereço virtual não é traduzido – o endereço físico correspondente tem o mesmo valor que o endereço virtual.

3.4.1 Contextos

A MMU pode guardar traduções para vários espaços de endereçamento relativos a processos diferentes ao mesmo tempo. Cada um destes espaços é identificado por um número de contexto. A gestão dos contextos, bem como a alteração na MMU do registo que indica qual o contexto corrente, é da responsabilidade do sistema operativo.

A protecção de memória atingida pela diferente tradução de memória virtual é alcançada através do uso de contextos diferentes para cada tarefa.

Com o [número de] contexto e o Context Table Pointer (guardado num registo da MMU), a MMU obtém o endereço da Page Table correspondente ao espaço de endereçamento do contexto corrente.

3.4.2 Page Table

A MMU tem um registo onde guarda o endereço de memória (na RAM) onde está localizada a Context Table, a tabela que contém o Root Pointer para a Page Table de cada um dos contextos. A Context Table é indexada pelo número de contexto.

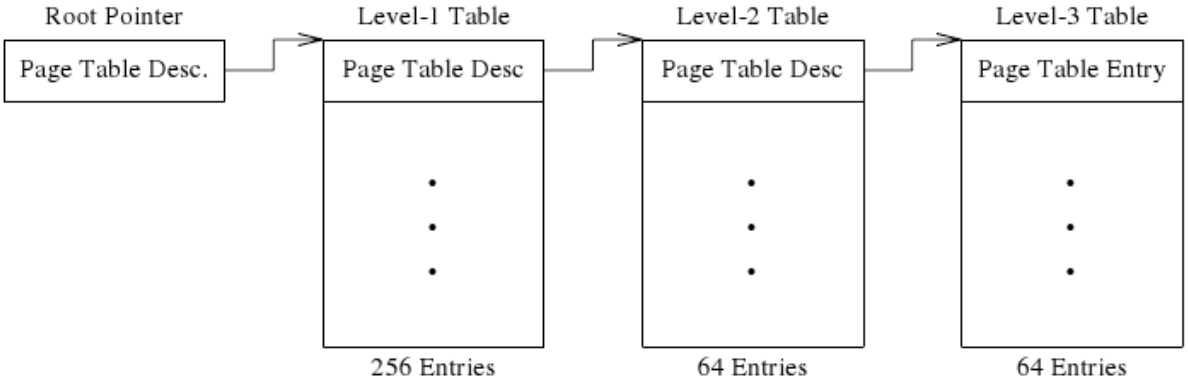

Figura 2: estrutura da Page Table (retirado de [05] Manual Sparc V8, pág. 243)

Cada Page Table é composta por (até) três níveis de profundidade de tabelas, que vão segmentando a memória sucessivamente, resultando numa estrutura semelhante a uma árvore binária cuja raiz tem 256 ramos, cujos nós de segundo nível têm folhas ou 64 ramos, e cujos nós de terceiro nível têm 64 folhas – sendo que cada folha irá traduzir uma área de memória (tanto maior quanto menor for o nível de profundidade).

A Page Table é baseada em dois tipos de entidades:

- PTD – Page Table Descriptor, que aponta para uma tabela do nível seguinte que compartimenta o segmento de memória corrente em sub-segmentos, tabela essa que vai ser indexada por uma parte do endereço virtual (ver Figura 4) e que traduz endereços com uma granularidade mais pequena;
- PTE – Page Table Entry, que guarda a informação para a tradução do segmento de memória a que corresponde.

Em cada uma das posições de uma tabela pode existir um PTE – que indica como mapear o segmento de memória corrente, ou um PTD – que aponta para uma nova tabela (que vai então sub-dividir este segmento em novos segmentos).

3.4.3 Tradução de endereços

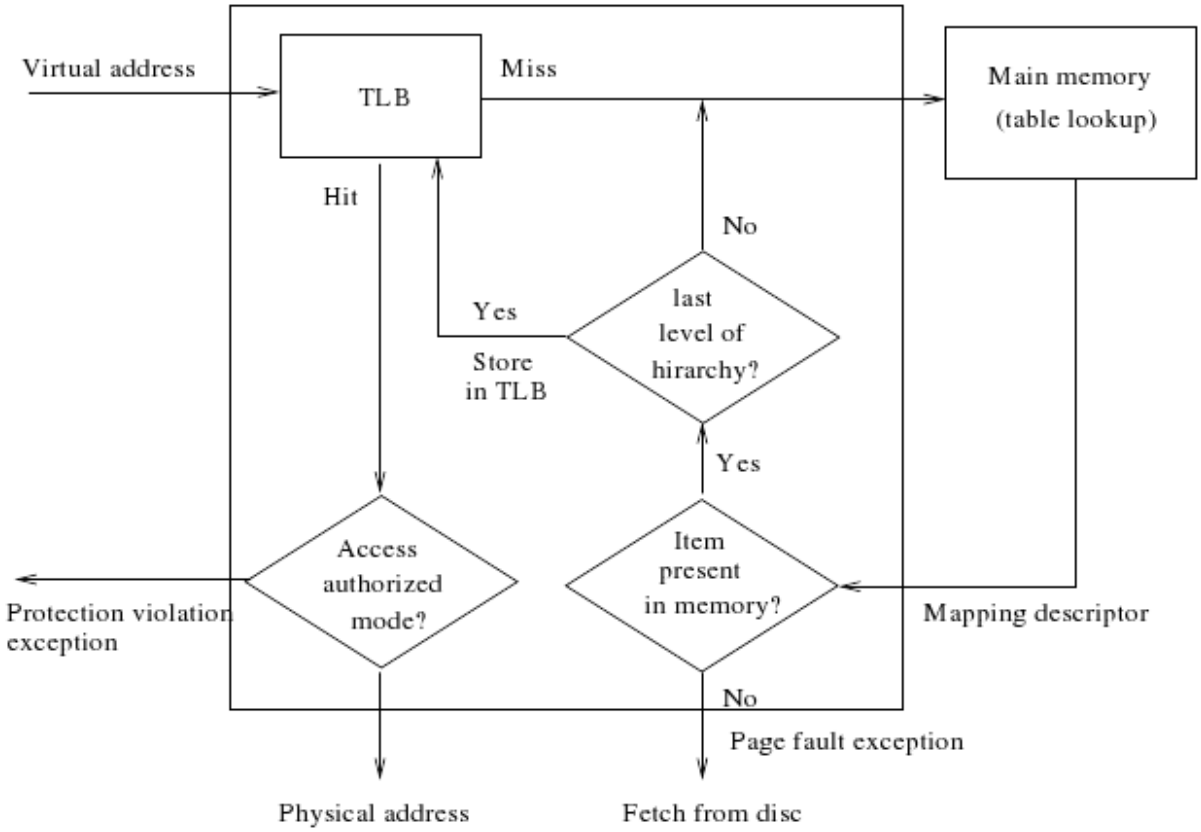

Figura 3: Esquema geral do processo abstracto de tradução de memória (retirado de [11]MMU.pdf, pág. 13)

O TLB (Translation Lookaside Buffer) é uma *cache* interna à MMU, onde a MMU guarda as traduções de memória usadas mais recentemente, para acesso rápido. O uso desta memória interna pela MMU agiliza a tradução de endereços de memória, tornando-a consideravelmente mais rápida.

Note-se, no entanto, que o determinismo requerido pelos sistemas embutidos causa a sobre-avaliação dos tempos de execução, já que devido à necessidade de determinismo estes cálculos têm de ser majorados.

Aquando de uma tradução de memória, a MMU procura efectuá-la com as entradas que tem guardadas no TLB. Se não conseguir, vai procurar a tradução da Page Table e guarda-a no TLB, traduzindo então o endereço (lançando uma excepção se a tradução for inválida).

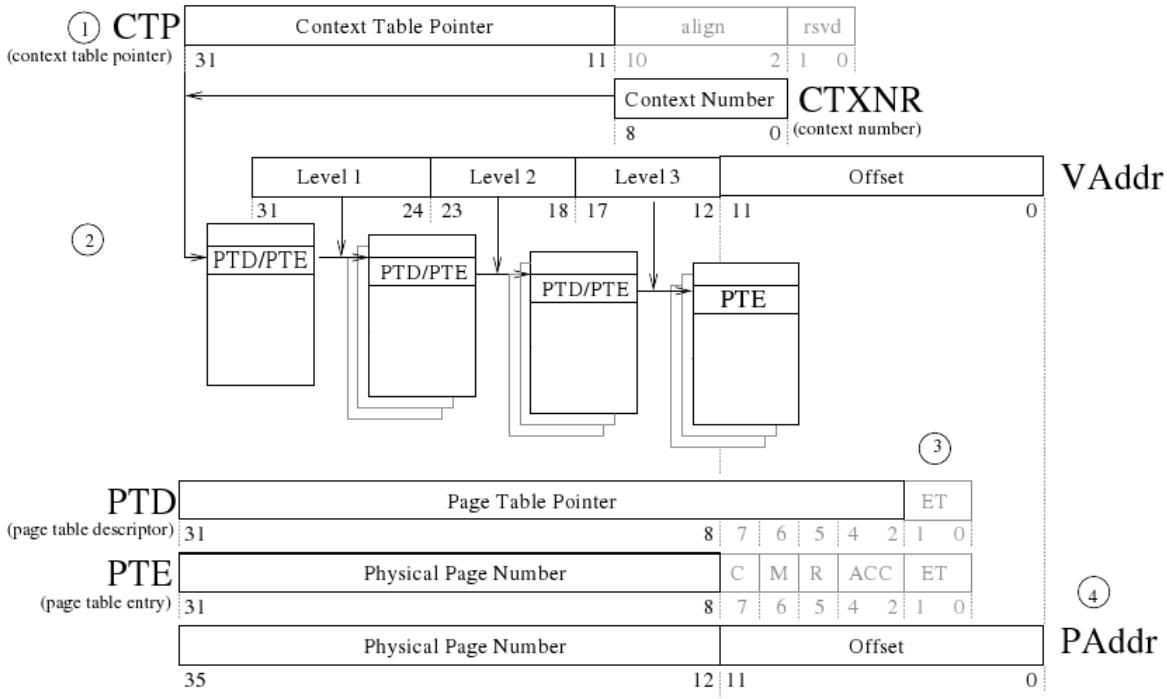

Figura 4: Esquema ao pormenor da tradução de um endereço de memória (retirado de [11]MMU.pdf, pág. 28)

Depois de obter do TLB a tradução, a MMU traduz o endereço.

Aquando da tradução de um endereço virtual, intervêm:

(1) O Context Table Pointer (CTP), registo da MMU;

1. O campo *rsvd* é reservado e deve estar a 0;
2. O campo *align* deve estar também a 0 por forma a alinhar o endereço físico da tabela de contextos;

(2) A Page Table, cujo endereço é obtido ao indexar pelo número de contexto (CTXNR, normalmente associado pelo sistema operativo à tarefa que está a correr) a Context Table apontada pelo Context Table Pointer;

1. É navegada indexando a tabela de primeiro nível da Page Table com o valor correspondente (*Level 1*) do *VAddr* (endereço virtual);
2. Se se encontrou um PTE está completa a pesquisa, senão repete-se na tabela de nível 2 que é apontada pelo PTD encontrado, e se necessário itera-se da mesma forma novo para o terceiro nível;
3. O *Offset* do *VAddr* não é usado nesta fase;

(3) PTDs e PTEs, dependendo do *Entry Type* (ET: 0 para inválido, 1 para PTD, 2 para PTE, 3 sendo um valor reservado que não deve ser usado);

1. PTDs contêm um apontador para a tabela de nível seguinte (que vai segmentar a zona de memória traduzida com granularidade ainda mais baixa) e o Entry Type (com o valor de 1, por ser um PTD);

2. PTEs contêm o Physical Page Number, as propriedades booleanas *Cacheable Modified* e *Referenced*, um valor de 3 bits que indica as permissões de acesso e o Entry Type (com o valor de 2, por ser um PTE);

ACC	Accesses Allowed	
	User access (ASI = 0x8 or ASI = 0xA)	Supervisor access (ASI = 0x9 or ASI = 0xB)
0	Read Only	Read Only
1	Read/Write	Read/Write
2	Read/Execute	Read/Execute
3	Read/Write/Execute	Read/Write/Execute
4	Execute Only	Execute Only
5	Read Only	Read/Write
6	No Access	Read/Execute
7	No Access	Read/Write/Execute

Figura 5: Tipos de Acesso permitidos (retirado de [11]MMU.pdf, pág. 28)

(4) O endereço físico final, obtido usando o Physical Page Number e o Offset;

1. Se o PTE for encontrado numa tabela de nível inferior a 3 (ou seja, se a memória virtual não tiver a granularidade máxima), fazem parte do *Offset* os índices não usados (*Level 3*, talvez *Level 2*, e possivelmente até mesmo *Level 1*), havendo na prática um *Offset* de tamanho alargado – este tamanho corresponde à granularidade.

4 Revisão de Ferramentas e Conhecimento Processual

Sendo objectivo deste estágio desenvolver uma extensão ao RTEMS, há que analisar e considerar tanto as ferramentas como o conhecimento processual (*know-how*) envolvidos.

4.1 Desenvolvimento da extensão

No desenvolvimento da extensão estão envolvidas diversas ferramentas:

- Eclipse – editor de código;
- Eclipse C/C++ Development Tooling (CDT) – extensão do Eclipse para desenvolvimento de código em C/C++;
- Eclipse C/C++ IDE plugin for LEON and ERC32 software development – extensão do CDT desenvolvida pela Gaisler Research;
- RTEMS (escrito em C e Assembly);
- ht://Dig – um sistema de indexação e pesquisa.

4.1.1 Eclipse

O Eclipse foi o editor escolhido por ser software livre e por existir uma extensão específica e indispensável para desenvolvimento de código para o processador LEON2. Em conjunto com a extensão CDT e a extensão desenvolvida pela Gaisler Research, afigura-se uma óptima ferramenta para o desenvolvimento da extensão do RTEMS.

Eclipse é um IDE (ambiente de desenvolvimento integrado) *open-source* cuja funcionalidade básica é o desenvolvimento em Java. As suas capacidades podem aumentar as suas capacidades instalando extensões, que podem ser criadas e usadas livremente.

A extensão CDT (ver exemplo na Figura 6) permite o desenvolvimento em C, permitindo o desenvolvimento da extensão tendo acesso a *auto-complete* inteligente, abertura imediata de declarações de funções/macros seleccionadas, *refactoring*, *highlighting*, etc.

A extensão ao CDT desenvolvida pela Gaisler Research (ver exemplo na), [07]Eclipse C/C++ IDE plugin for LEON and ERC32 software development, permite integrar no Eclipse/CDT a compilação (*cross-compiling*), o *debug* e a simulação (em TSIM) ou execução em hardware remoto (através do GRMON).

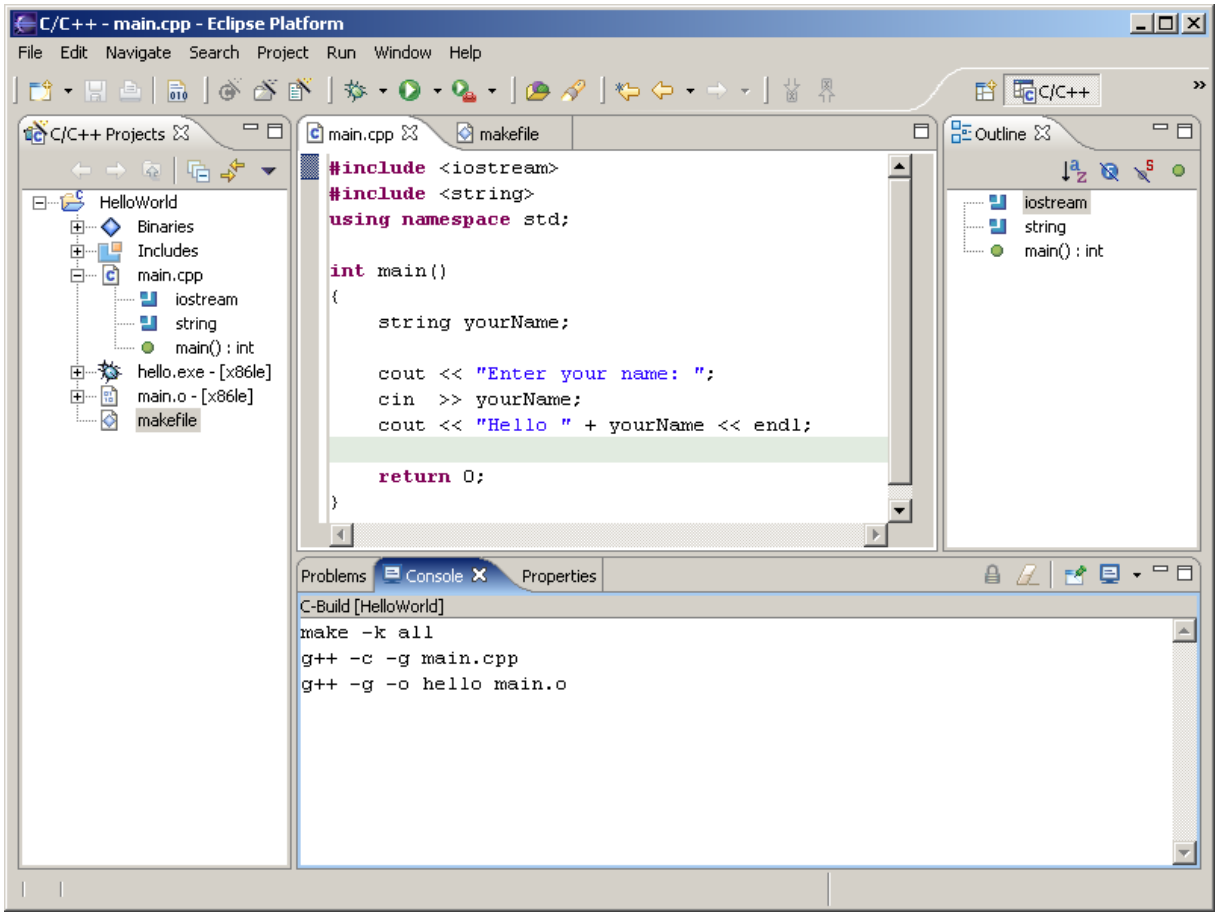

Figura 6: Eclipse com CDT, usando a perspectiva "C/C++"

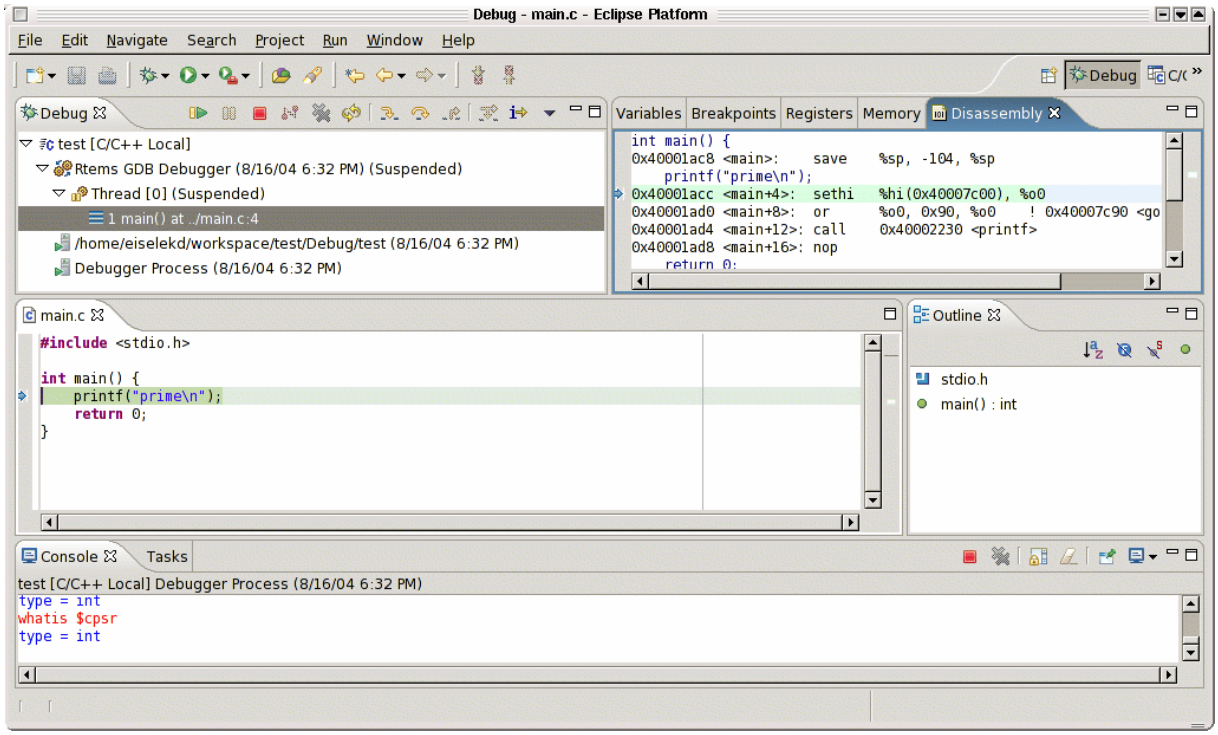

Figura 7: Vista de debug da extensão de CDT desenvolvida pela Gaisler Research

4.1.2 RTEMS

No âmbito do RTEMS foram analisados tanto as linguagens utilizadas como a estrutura de directórios.

As linguagens utilizadas são C e assembly. Como o alvo é o processador LEON2, o assembly é SPARC e não o assembly x86 mais conhecido. No entanto a ambientação ao fluxo de informação não é difícil, especialmente tendo como referência os documentos [03] SPARC Assembly Summary e [02] SPARC Assembler Language Programming in a Nutshell, e os apêndices A (Suggested Assembly Language Syntax) e B (Instruction Definitions) do documento [05] Manual Sparc V8.

Há ainda a referir a estrutura de directórios do RTEMS. A sua organização pode ser consultada no documento *develenv.pdf* presente na referência [04] Documentação do RTEMS. De salientar especialmente a separação entre bibliotecas, e a existência de um directório que está fragmentado em BSPs e, dentro de cada um destes, em modelo de CPUs.

4.1.3 ht://dig

O ht://Dig ([12] ht://Dig) é um sistema de indexação e pesquisa open source, desenvolvido para servir de motor de pesquisa de conteúdos em pequenos sites ou redes locais.

O seu uso para pesquisar informação na enorme estrutura de directórios do RTEMS revelou-se essencial, ao permitir localizar definições e usos de constantes, macros e funções.

4.2 Compilação

As ferramentas presentes no âmbito da compilação são:

- Compilador (gcc, como cross-compiler);
- Makefile;
- Linker sripts;

4.2.1 Cross-Compiler

Um cross-compiler é um compilador capaz de compilar código executável para uma plataforma diferente daquela em que o compilador é executado. Neste estágio é usado um cross-compiler que compila, em Linux x86 (arquitectura i386), objectos para serem executados no processador LEON2 (arquitectura SPARC).

A Gaisler Research fornece um cross-compiler para compilar aplicações RTEMS (escritas em C, com a possibilidade de complementação com *assembly*), documentado em [06] RTEMS Cross Compilation System (RCC). Este vem completamente preparado, sendo apenas necessário descompactá-lo para a localização */opt/rtems-4.6/* e pôr o directório */opt/rtems-4.6/bin* no PATH. [1]

A utilização deste cross-compiler é semelhante à do gcc "normal", com a diferença de que todos os utilitários são precedidos do sufixo "sparc-rtems-", evidenciando a arquitectura alvo para que compilam e o facto de compilarem a aplicação junto com o Sistema Operativo RTEMS, num único executável.

Uma ferramenta muito útil do cross-compiler é o utilitário *sparc-rtems-objdump*, que permite analisar um executável: ver os cabeçalhos, as etiquetas (labels), descompilá-lo, etc.

---

[1]Para isso, escrever numa consola o seguinte: *export PATH="/opt/rtems-4.6/bin":$PATH* . Para fazer a alteração permanente, adicionar uma linha com esse conteúdo no ficheiro *~/.bashrc*

4.2.2 Makefiles e Linker Scripts

Make é um utilitário para a construção automática de uma aplicação ou programa complexo, e Makefile é o ficheiro onde se encontram especificadas as operações a efectuar para a construção final. O *make* monitoriza quais os ficheiros que foram alterados desde a última vez que se fez a construção da aplicação, e ao ser chamado compila apenas esses ficheiros e outros ficheiros que dependam deles.

Os linker scripts (ficheiro *linkcmds* no RTEMS) são ficheiros com instruções de como o compilador deve organizar os conteúdos de um objecto, ao construí-lo. Estes ficheiros são indicados ao compilador por argumento.

Descrições e explicações das makefiles e linker scripts usadas em RTEMS podem ser encontradas no ficheiro *bsp_howto.pdf* presente na referência [04] Documentação do RTEMS, servindo como uma boa introdução.

Nota: para alterar o processo de "ligação" com o compilador da Gaisler, dever-se-á usar uma de duas opções: alterar a makefile do RTEMS, e depois recompilar o cross-compiler, ou então usar a flag "*-T <linker.lds>*" no *sparc-rtems-gcc* (em vez de compilar para objecto.o e depois ligar com *sparc-rtems-ld*, porque desta forma o compilador não junta o sistema operativo ao executável [e, então, o executável não funciona]).

4.3 Recompilação do compilador

Ao fazer alterações ao RTEMS é necessário recompilar o próprio cross-compiler, para que as bibliotecas deste passem a incluir as diferenças implementadas.

Para recompilar o cross-compiler:

- A pasta */opt/rtems-4.6/src/rtems* deve incluir directa ou indirectamente (através de uma ligação do sistema de ficheiros) o código-fonte do RTEMS;
- cd /opt/rtems-4.6/src/rtems
- ./bootstrap
- cd /opt/rtems-4.6/src/
- make install

O script "bootstrap" actualiza o processo de construção do sistema operativo após haver alterações ou adições/remoções de algum dos ficheiros ou pastas do código-fonte do sistema operativo. Isso preparará o sistema para ser instalado.

O comando "make all" instala o cross-compiler, substituindo os objectos de sistema pré-compilados que se encontram no crosscompiler, o que propagará as alterações feitas ao código-fonte para o cross-compiler e, consequentemente, para os novos objectos compilados.

4.4 Execução de aplicações para LEON2

Para correr estes executáveis é necessário possuir o hardware alvo ou ter acesso a um simulador. No âmbito deste projecto foi usado o TSIM 2.0.8, um simulador de LEON/ERC32 criado pela Gaisler Research. O TSIM pode ser corrido isoladamente, ou ligado à ferramenta de *debugging* gdb (GNU debugger).

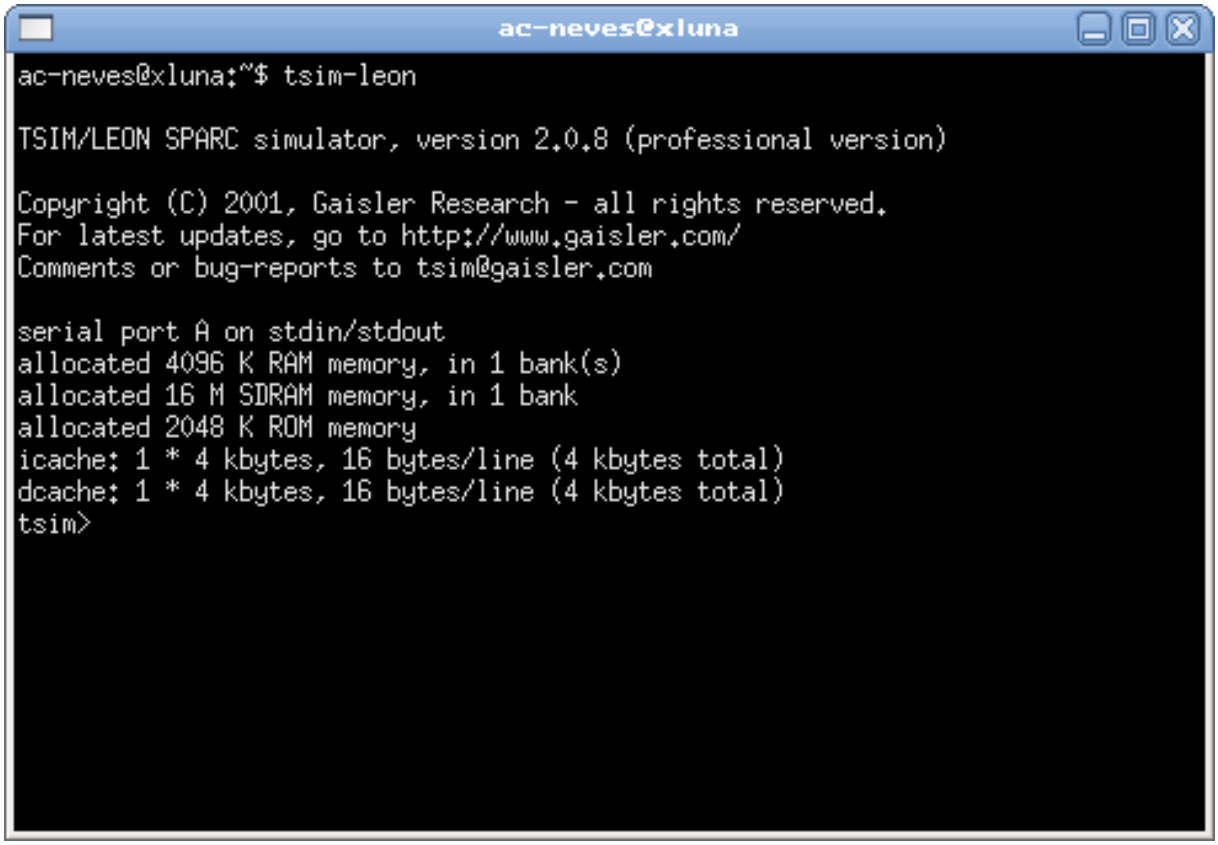

Figura 8: TSIM

Para correr o executável, estando o TSIM instalado, basta executar *tsim-leon <executavel>* , e depois invocar o comando *go*.

Embora o manual do TSIM seja a melhor forma de aprender a trabalhar com o emulador, apresentam-se aqui os comandos considerados mais úteis:

- **break** <addr> – adiciona um breakpoint em <addr> (monitoriza a execução da instrução nesse endereço de memória)
- **del** <num> – apaga o breakpoint <num>
- **break** – imprime todos os breakpoints
- **bt** – imprime backtrace
- **cont** [cnt/time] – continua a execução por [cnt] instruções ou [time] tempo
- **dis** [addr] [count] – disassemble [count] instruções, começando no endereço [addr]
- **go** [addr [cnt/time]] – inicia a execução em <addr>
- **help** – ajuda
- **hist** [trace_length] – activa/mostra o histórico de instruções executadas
- **load** <file_name> – carrega um ficheiro executável para a memória do simulador
- **mem** [addr] [count] – mostra a memória (física) a partir de [addr], por [count] bytes
- **vmem** [vaddr] [count] – mostra a memória (virtual) a partir de [vaddr], por [count] bytes
- **quit** – sair do simulador
- **reg** [reg] [val] – mostra/modifica os registos do processador (ou janelas, ex: 'reg w2')
- **reset** – faz reset ao simulador
- **run** [cnt/time] – faz reset e inicia a execução do endereço zero
- **shell** <cmd> – executa o comando de shell <cmd>
- **step** – executa a instrução seguinte
- **tra** [inst_count] – faz trace de [inst_count] instruções

- **watch** <addr> – adiciona um watchpoint no endereço <addr> (monitoriza o acesso a esse endereço de memória)
- **wmem** <addr> <val> – escreve a word <val> no endereço <addr>

Nota: o comando **hist**, chamado sem argumentos, apresenta o histórico das últimas n instruções executadas, enquanto que com um argumento selecciona o número de instruções a guardar no histórico. De notar que o histórico tem um valor por omissão de **0** instruções.

4.5 Sistema de controlo de versões

Para sistema de controlo de versões foi seleccionado o CVS, por ser o sistema de controlo de versões usado em toda a Critical Software, havendo um repositório unificado.

No entanto, dada o fraco desempenho do CVS quando tem de gerir um número elevado de ficheiros, no caso do código desenvolvido optou-se por uma abordagem diferente: é mantido controlo de versões localmente, usando Git, e é feita sincronização para o CVS sempre que se atinge um ponto estável de desenvolvimento. Sugere-se ainda submeter para o CVS apenas as diferenças do código-fonte do RTEMS, e não a estrutura de ficheiros completa.

Sugere-se que, para ambientação ao Git, se consulte a referência [09] Git Cheat Sheet. As principais características do CVS aparecem na referência [07] CVS Cheat Sheet.

4.6 Documentação

A documentação foi desenvolvida em OpenOffice.org, usando os *templates* existentes: *templates* da Critical Software, para os documentos de Requisitos e Arquitectura; *template* de relatório de projecto/dissertação do MIEIC, para o relatório de projecto.

5 Requisitos

Pretende-se implementar o uso da MMU em RTEMS, de forma a que haja protecção de memória.

Pretende-se alcançar com isso protecção de memória entre o kernel do sistema operativo e as tarefas do utilizador, protecção de memória entre as diversas tarefas do utilizador, possíveis zonas de memória partilhada, e um sistema mais resistente e seguro. Prevê-se também a definição de tarefas de sistema, que correm com acesso total ao kernel.

Os requisitos identificados do sistema agrupam-se em 9 grupos distintos:

1. Tarefas de sistema;
2. Protecção entre tarefas;
3. Gestão de falhas;
4. Configuração;
5. Activação e desactivação da extensão do RTEMS;
6. Consistência das estruturas de dados;
7. Partilha de memória;
8. Protecção dos programas;
9. Portabilidade.

Apresentam-se de seguida os requisitos recolhidos.

5.1 Tarefas de sistema

UR-0100 – Criar tarefas de sistema.
Deve ser possível criar tarefas de sistema, que corram em modo privilegiado do processador. (Com acesso directo ao kernel do sistema operativo, ou seja partilhando o seu espaço de endereçamento.)

Estas apresemtam-se em contraste com as tarefas de utilizador (todas as outras), que correm em modo protegido do processador e às quais, por isso, será negado acesso directo à generalidade dos registos e funcionalidades tanto do kernel do sistema operativo e das tarefas de sistema como das outras tarefas de utilizador.

5.2 Protecção entre tarefas

No âmbito da protecção de tarefas é importante atentar na existência de uma biblioteca do sistema, a quem cabe generalizar o acesso a funcionalidades cuja utilização incorrecta pode ter efeitos nefastos no sistema, ou que permitem acesso a zonas vulneráveis ou sensíveis do sistema. Ao fornecer apenas acesso indirecto a estas funcionalidades, pode-se assegurar que são utilizadas correctamente e de forma controlada, mantendo a estabilidade e a fiabilidade do sistema. (Esta biblioteca prepara o sistema para a mudança para o modo privilegiado do processador antes de fazer a chamada da função de sistema, e ao retornar para a tarefa de utilizador volta a restaurar o modo de utilizador.)

Assim, as tarefas de utilizador (que não têm acesso a acções e funcionalidades exclusivas do núcleo do sistema) podem ter acesso indirecto à zona do núcleo de sistema. Isto não quer dizer, no entanto, que possam alterar ou

utilizar arbitrariamente os recursos do sistema, continuando de facto impedidas de manipular elementos fora das zonas de memória em que se encontram isoladas – sendo esta a principal característica da protecção de memória entre tarefas.

UR-0200 – Proteger tarefas de sistema.	
Detalhes	Criado em 12-11-2007. Modificado em 12-11-2007.
As tarefas de sistema devem ser protegidas das tarefas de utilizador.	

UR-0210 – Código de kernel só deve poder correr em *Supervisor Mode*.
As instruções em memória pertencentes ao kernel só devem poder ser executadas no modo privilegiado do processador.

UR-0220 – Não executar código de kernel quando estiverem a ser executadas tarefas de utilizador.
Não deve ser possível executar código do kernel quando estiverem a ser executadas tarefas de utilizador. As tarefas poderão, no entanto, utilizar as funcionalidades fornecidas pela biblioteca de chamadas ao sistema.

UR-0230 – Protecção de dados e código de kernel.
Os dados e código de kernel não devem poder ser directamente modificados por tarefas de utilizador. (Tal mudança será possível se efectuada através do uso de uma função da biblioteca de sistema.)

UR-0240 – Usar funcionalidades do sistema a partir de *User Mode*.
Devem ser criados *wrappers*, de forma a possibilitar o uso de funcionalidades do sistema por parte das tarefas de utilizador. O conjunto destes *wrappers* constituirá a biblioteca de sistema acima referida, que fornece todas as directivas do RTEMS através do mecanismo de chamadas ao sistema por forma a poder ser usada na programação de tarefas de utilizador.

Nota: as funções da biblioteca de sistema não permitem circundar (nem directa nem indirectamente) os requesitos UR-0300 a UR-0350.

UR-0300 – Proteção de tarefas.
As tarefas devem ser protegidas de acessos indevidos por parte de outras tarefas.

UR-0310 – Só tarefas de sistema podem criar objectos de sistema.
Só tarefas de sistema devem poder criar objectos de sistema (tarefas, message queues, semáforos, etc).

UR-0320 – Só tarefas de sistema podem modificar prioridade de uma tarefa.
Só tarefas de sistema devem poder modificar a prioridade de uma tarefa (incluindo elas próprias).

UR-0330 – Só tarefas de sistema podem bloquear ou suspender uma tarefa.
Só tarefas de sistema devem poder bloquear ou suspender outra tarefa.

UR-0340 – Só tarefas de sistema podem adormecer outra tarefa.
Só tarefas de sistema devem poder adormecer outra tarefa.

UR-0350 – Só tarefas de sistema podem acordar outra tarefa.
Só tarefas de sistema devem poder acordar outra tarefa.

5.3 Gestão de falhas

UR-0400 – Gestão de falhas.
Deve haver gestão de falhas. (Devem ser criados vectores para tratamento das novas excepções que passem a existir no sistema.)

UR-0410 – Forçar reset aquando de erros internos.
O software deve considerar erros internos como fatais, e forçar um reset do sistema.

5.4 Configuração

UR-0500 – Escritas na configuração de memória devem ser feitas em *Supervisor Mode*.
As configurações de memória, que estão na própria memória, devem estar numa área protegida, em que a escrita só poderá ser efectuada em *Supervisor Mode*.

UR-0600 – Iniciar correctamente a MMU quando o sistema é iniciado.
A MMU deve ser correctamente iniciada quando o sistema é iniciado.

UR-0610 – Configurar MMU correctamente antes de iniciar qualquer tarefa.
Nenhuma tarefa pode correr antes da configuração correcta, e posterior activação, da MMU.

5.5 Activação e desactivação da extensão

UR-0700 – Definir a utilização ou não da extensão aquando da construção do sistema.
Deve ser possível definir a utilização ou não da extensão, na altura da construção do sistema.

5.6 Consistência das estruturas de dados

UR-0800 – Consistência das estruturas de dados.
Deve ser assegurada a consistência das estruturas de dados usadas. (PTEs, PTDs, e outras estruturas que sejam eventualmente utilizadas.)

5.7 Partilha de memória

UR-0900 – Partilha de memória
Deve ser permitido haver partilha de memória entre tarefas. Este requisito será subdividido e especificado nos requisitos seguintes (UR-0910 a UR-0922).

UR-0910 – Partilha de memória restrita (n tarefas).
Deve ser permitida partilha de memória restrita (só para n tarefas específicas) através da criação, via biblioteca de sistema, de zonas de memória que serão mapeadas apenas pelos contextos de memória das tarefas especificadas.

UR-0920 – Memória global.
Deve ser permitida memória global através da criação, via biblioteca de sistema, de zonas de memória que serão mapeadas em todos os contextos de memória.

UR-0921 – Acesso global à memória global.
Todas as tarefas devem poder aceder à memória global, pelo que todos os contextos de memória devem mapear todas as zonas de memória global.

UR-0922 – Edição da memória global apenas por tarefas de sistema.
Só tarefas de sistema devem poder alterar os conteúdos da memória global, sendo que todas as outras devem ter apenas permissão de leitura.

5.8 Protecção dos programas

UR-1000 – Tarefas de utilizador não devem poder escrever em partes de memória que contenham código executável.
Deve ser assegurado que os programas em memória não possam ser modificados pelas tarefas de utilizador, ou seja, zonas de memória que contenham código devem ser apenas de execução e/ou leitura.

5.9 Portabilidade

UR-1100 – Respeitar a estruturação modularizada do RTEMS.
Para que seja possível criar portabilidade para outros sistemas no futuro, a implementação deve ter em conta e respeitar a estrutura modularizada por arquitectura e BSP do RTEMS.

6 Casos de Uso

Os casos de uso reflectem a necessidade de permissão ou negação de algumas funcionalidades do sistema consoante os actores que as executam. Esta necessidade surge da hierarquização da execução no sistema em modo privilegiado e em modo protegido, já que algumas funcionalidades estarão restringidas no modo protegido por forma a implementar a compartimentação de memória no que toca às tarefas de utilizador.

Serão então agrupados os casos de uso consoante os actores que a eles tenham acesso, e descrito cada um em maior pormenor. Serão também, quando relevante, associados aos requisitos previamente delineados.

Atente-se no pormenor de que há requisitos que não estão associados a nenhum caso de uso. Isto acontece porque destes requisitos não resulta uma funcionalidade mas sim uma restrição.

6.1 Actores

Foram identificados dois actores no sistema:

1. Tarefa de Utilizador, uma tarefa que corre em modo protegido do processador e para quem a memória está compartimentada por forma a fornecer protecção de memória para si mas também para todas as outras tarefas;
2. Tarefa de Sistema – tarefa que corre em modo privilegiado do processador e que não sofre restrições devido à protecção de memória, tendo acesso indiscriminado à memória.

Note-se que o núcleo do sistema não é uma tarefa por si só, e depois de activar a primeira tarefa tem um papel meramente de suporte do sistema – por exemplo, gerindo o escalonamento, alocando memória dinamicamente quando pedido, etc.

6.2 Tarefa de Utilizador

Foram identificados os seguintes casos de uso referentes às tarefas de utilizador:

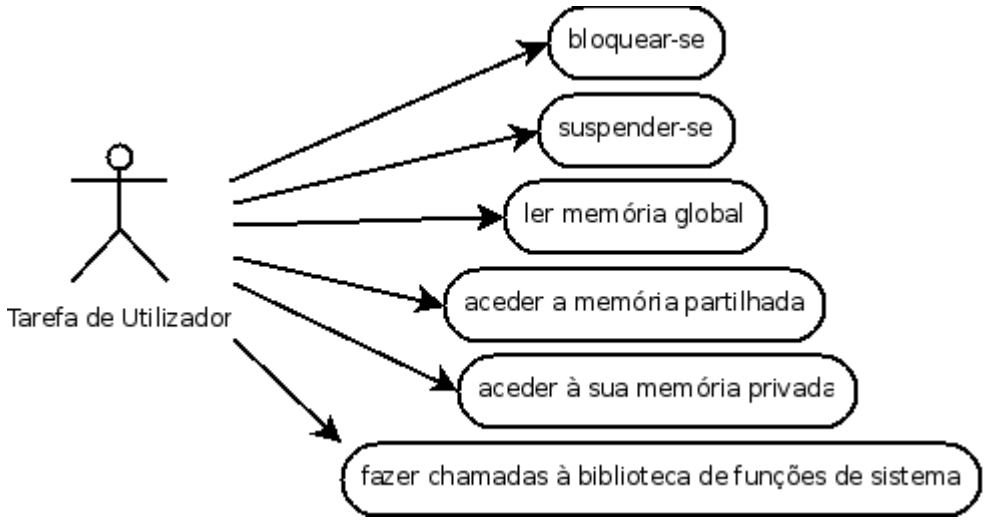

Figura 9: Casos de uso – tarefa de utilizador.

Estes casos de uso caracterizam-se pela sua natureza não-invasiva e pela não-interferência em áreas do sistema operativo ou de outras tarefas.

6.2.1 Bloquear-se

Deve ser permitido à tarefa bloquear a sua própria execução por um tempo definido, permitindo-lhe parar a execução e ceder o controlo do processador ao sistema por forma a que o escalonador o forneça a outra tarefa.

A tarefa continua bloqueada enquanto não passar o limite de tempo definido para o bloqueio.

Este caso de uso enquadra-se no requisito UR-0240 (Usar funcionalidades do sistema a partir de *User Mode"*).

6.2.2 Suspender-se

A tarefa deve poder suspender a sua própria execução, permitindo-lhe suspender a execução e ceder o controlo do processador ao sistema por forma a que o escalonador o forneça a outra tarefa.

A tarefa não executará enquanto não for reactivada por cancelamento do estado de suspensão e por término do tempo de qualquer bloqueio que haja.

Este caso de uso enquadra-se no requisito UR-0240 (usar funcionalidades do sistema a partir de *User Mode"*).

6.2.3 Ler Memória Global

O contexto de memória da tarefa deve permitir leitura de todas as regiões de memória global.

Este caso de uso enquadra-se no requisito UR-0921 (acesso global à memória global).

6.2.4 Aceder a Memória Partilhada

Se estiver registada como uma das tarefas que partilham essa região de memória partilhada, a tarefa deve poder aceder à região em questão, com acesso completo (leitura e escrita ou execução).

Este caso de uso enquadra-se no requisito UR-0900 (partilha de memória).

6.2.5 Aceder à Sua Memória Privada

A tarefa deve poder aceder à sua própria memória privada, por forma a possibilitar o seu correcto funcionamento.

Embora não se enquadre em nenhum requisito em especial, sem este caso de uso se verificar não é possível as tarefas executarem qualquer funcionalidade própria – o simples aceder a uma variável local seria impossível.

6.2.6 Fazer Chamadas à Biblioteca de Funções do Sistema

Todas as tarefas devem poder fazer chamadas à biblioteca de funções do sistema, por forma a não invalidar a funcionalidade da biblioteca: fornecer a toda e qualquer tarefa acesso indirecto a funcionalidades seleccionadas do sistema.

Este caso de uso enquadra-se no requisito UR-0240 (usar funcionalidades do sistema a partir de *User Mode*).

6.3 Tarefa de Sistema

Foram identificados os seguintes casos de uso relativos às tarefas de sistema:

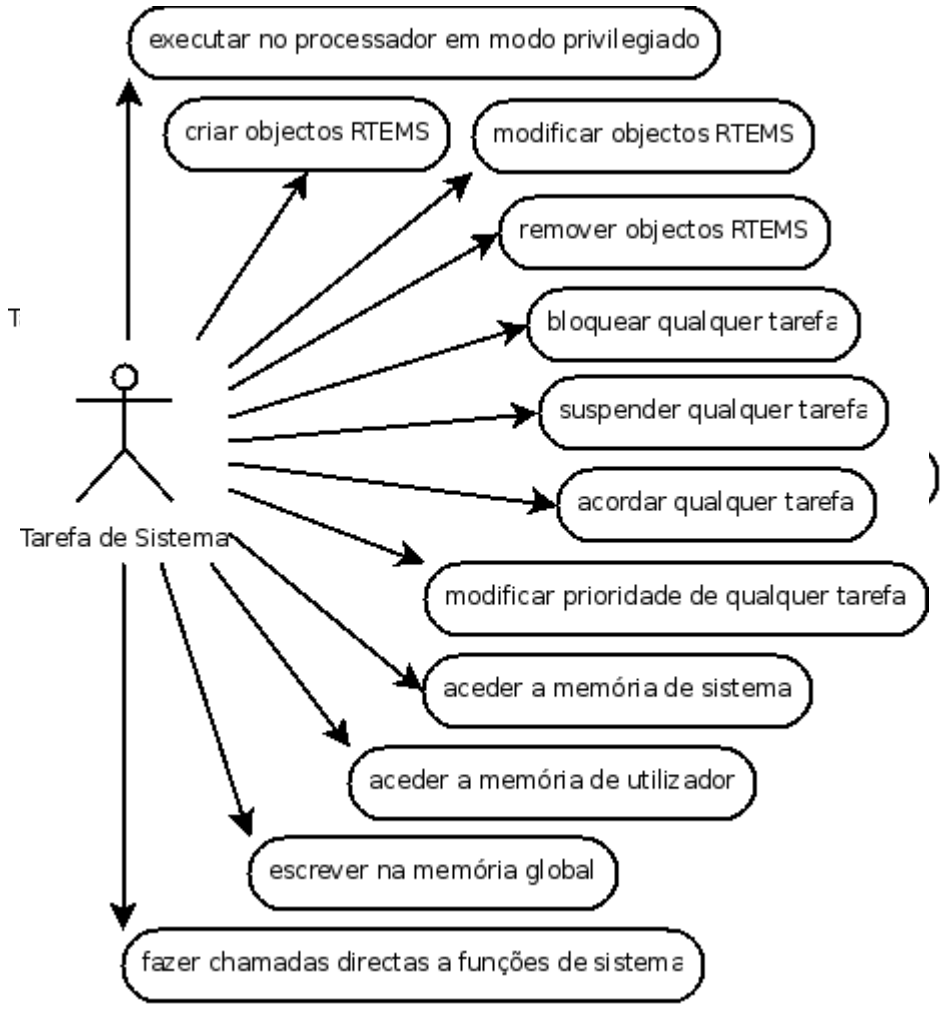

Figura 11: Casos de uso – tarefa de sistema.

Note-se que uma tarefa de sistema extende uma tarefa de utilizador, tendo por isso também todos os casos de uso das tarefas de utilizador, que não serão aqui repetidos.

Uma tarefa de utilizador, no entanto, não tem os casos de uso que são aqui descritos.

6.3.1 Executar no Processador em Modo Privilegiado

As tarefas de sistema deverão executar no processador em Modo Privilegiado, por forma a poderem executar instruções privilegiadas. (Ex: *store alternate* [STA %i1 %i2], instrução que guarda um valor num registo que se encontra num espaço de endereçamento alternativo.)

Este caso de uso enquadra-se no seguinte grupo de requisitos:

- UR-0310 – Só tarefas de sistema podem criar objectos de sistema;
- UR-0320 – Só tarefas de sistema podem modificar prioridade de uma tarefa;
- UR-0330 – Só tarefas de sistema podem bloquear ou suspender uma tarefa;
- UR-0340 – Só tarefas de sistema podem adormecer outra tarefa;
- UR-0350 – Só tarefas de sistema podem acordar outra tarefa;

- UR-0922 – Edição da memória global apenas por tarefas de sistema (um modo de assegurar este requisito pode ser, por exemplo, configurar os contextos de memória de tal forma que só se possa escrever na memória global em modo privilegiado do processador).

6.3.2 Criar objectos RTEMS

Dado que estas têm acesso completo ao sistema deve ser permitido às tarefas de sistema criar, manipular e remover objectos RTEMS.

Estes objectos incluem, por exemplo, os objectos que definem e configuram as tarefas.

Este caso de uso enquadra-se no requisito UR-0310 (*só* tarefas de sistema podem criar objectos de sistema).

6.3.3 Modificar Objectos RTEMS

Como dito em 6.3.2, deve ser permitido às tarefas de sistema manipular objectos RTEMS.

Este caso de uso enquadra-se no seguinte grupo de requisitos, que referem alterações de propriedades dos objectos de sistema que contêm as definições de uma tarefa:

- UR-0320 – Só tarefas de sistema podem modificar prioridade de uma tarefa;
- UR-0330 – Só tarefas de sistema podem bloquear ou suspender uma tarefa;
- UR-0340 – Só tarefas de sistema podem adormecer outra tarefa;
- UR-0350 – Só tarefas de sistema podem acordar outra tarefa.

6.3.4 Remover Objectos RTEMS

Como dito em 6.3.2, deve ser permitido às tarefas de sistema remover objectos RTEMS.

Embora não tenha sido definido nenhum requisito específico para a remoção dos objectos, a remoção é um caso-limite da edição, pelo que se enquadra no grupo de requisitos acima expostos.

6.3.5 Bloquear Qualquer Tarefa

Decorrente das permissões de edição das propriedades de objectos de sistema, deve ser permitido às tarefas de sistema:

- Bloquear qualquer tarefa;
- Suspender qualquer tarefa;
- Acordar qualquer tarefa;
- Modificar Prioridade de qualquer tarefa.

Este caso de uso enquadra-se no requisito UR-0330 (só tarefas de sistema podem bloquear ou suspender uma tarefa).

6.3.6 Suspender Qualquer Tarefa

Como exposto em 6.3.5, deve ser permitido às tarefas de sistema suspender qualquer tarefa.

Este caso de uso enquadra-se no mesmo requisito que o caso de uso acima, o requisito UR-0330 (só tarefas de sistema podem bloquear ou suspender uma tarefa).

6.3.7 Acordar Qualquer Tarefa

Como exposto em 6.3.5, deve ser permitido às tarefas de sistema acordar qualquer tarefa.

Este caso de uso enquadra-se no requisito UR-0350 (só tarefas de sistema podem acordar outra tarefa).

6.3.8 Modificar Prioridade de Qualquer Tarefa

Como exposto em 6.3.5, deve ser permitido às tarefas de sistema modificar a prioridade de qualquer tarefa.

Este caso de uso enquadra-se no requisito UR-0320 (só tarefas de sistema podem modificar prioridade de uma tarefa).

6.3.9 Aceder a Memória de Sistema

Dado que não têm restrições de protecção de memória, deve ser permitido às tarefas de sistema aceder indiscriminadamente à memória de sistema.

Este caso de uso não é associável directamente a nenhum requisito, embora a sua existência seja um postulado derivado da diferenciação entre tarefa de sistema e tarefa de utilizador, no âmbito deste projecto.

6.3.10 Aceder a Memória de Utilizador

Pela mesma razão que no ponto anterior, deve ser permitido às tarefas de sistema aceder indiscriminadamente à memória de utilizador.

A associação a um ou mais requisitos segue a lógica do caso de uso anterior: as tarefas de utilizador podem aceder a toda a memória devido ao facto assente que não têm restrições de protecção de memória, devido À sua definição.

6.3.11 Escrever na Memória Global

Deve ser permitido às tarefas de sistema escrever na memória global, já que estas têm acesso indiscriminado à memória.

Este caso de uso enquadra-se no requisito UR-0922 (edição da memória global apenas por tarefas de sistema).

6.3.12 Fazer Chamadas Directas a Funções de Sistema

Por fim, ainda do facto das tarefas de sistema não sofrerem restrições de protecção de memória sucede que as tarefas de sistema devem ser capazes de fazer chamadas directas a funções de sistema.

Este caso de uso enquadra-se no requisito UR-0922 (edição da memória global apenas por tarefas de sistema).

7 Visão Geral da Arquitectura

7.1 Visão de arquitectura

O principal objectivo do projecto é fornecer um mecanismo que permita proteger de eventuais danos as tarefas vitais e do kernel, danos esses provocados inadvertidamente por código menos confiável presente nas tarefas menos críticas.

As tarefas vitais referidas foram definidas como "tarefas de sistema", e correm em modo privilegiado do processador. São tarefas que são consideradas responsáveis, e que serão tipicamente testadas e validadas de forma muito extensa.

A única forma viável e prática de um sistema operativo fornecer mecanismos de protecção é usando os próprios mecanismos do Hardware pensados para tal fim – nomeadamente a *Memory Management Unit* do processador. Sendo esta utilização da camada inferior de Hardware (que fornece serviços específicos) a esperada, esta decisão não carece de justificação. (Uma justificação seria necessária em caso contrário, ou seja, se se usasse uma forma alternativa ou menos ortodoxa de abordar a questão da protecção.)

Os dois mecanismos básicos usados na arquitectura do sistema são o modo protegido do processador e a memória virtual. O modo protegido permite ao Sistema Operativo impedir a execução de instruções privilegiadas (tais como alterar registos de estado/configuração do processador), provando-se ser assim a forma ideal de proteger o Sistema Operativo. A gestão de memória virtual permite ao Sistema Operativo proteger dados e instruções localizadas em determinadas áreas contra a escrita, leitura e/ou execução (leitura de instruções). A determinação das áreas a proteger e como as proteger deverá ser feita pelo Sistema Operativo, tendo em conta a tarefa/aplicação responsável pelo conteúdo que esteja em cada uma dessas áreas. Só deverá ser possível executar esta configuração de memória em modo privilegiado do processador.

Criar-se-á uma biblioteca de chamadas ao sistema, que deverá ser acessível globalmente, dado que as funcionalidades fornecidas pelo kernel estarão protegidas e, portanto, as tarefas de utilizador estão impedidas de lhes aceder.

A conjugação destes três mecanismos, quando usada convenientemente, garante a protecção efectiva do conteúdo de memória sem no entanto reduzir a funcionalidade e utilidade do sistema. Esta garantia consegue ser provada e testada.

7.2 Visão geral do software e sistema

Os **módulos** principais necessários são então os que se listam, ilustrados na Figura 12:

- Gestão da memória virtual;
- Protecção do kernel;
- Gestão das (novas) propriedades das tarefas;
- Confiabilidade do sistema.

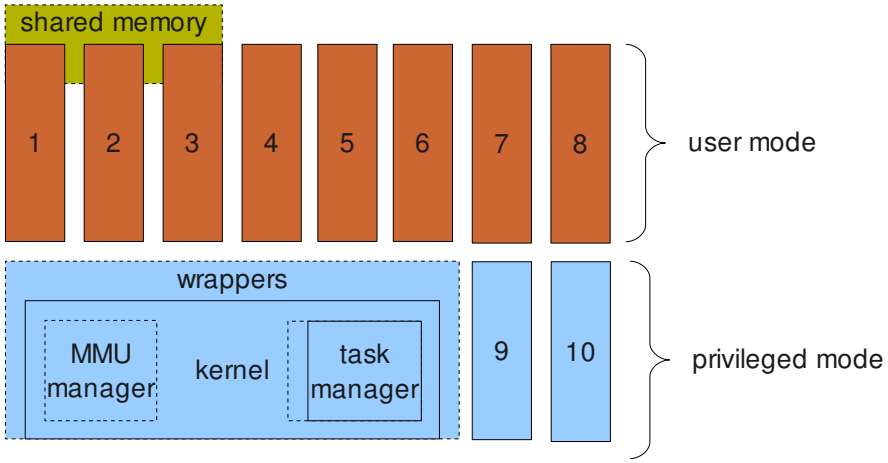

Figura 12: Arquitectura alterada, com kernel e tarefas de utilizador e de sistema.

Gestão da memória virtual

Este módulo deve assegurar a correcta inicialização da MMU e da(s) Page Table(s), e a sua alteração quando houver mudanças relevantes nas tarefas – criação ou destruição de tarefas, alteração dos atributos de tarefas, alteração da situação da memória partilhada, alteração na memória alocada por ou para uma tarefa, ou mudança da tarefa que está a executar.

Os requisitos associados a este módulo são os seguintes:

- UR-0300 – Protecção de tarefas.
- UR-0500 – Escritas na configuração de memória devem ser feitas em *Supervisor Mode*.
- UR-0600 – Iniciar correctamente a MMU quando o sistema é iniciado.
- UR-0610 – Configurar MMU correctamente antes de iniciar qualquer tarefa.
- UR-0900 – Partilha de memória
- UR-0910 – Partilha de memória restrita (n tarefas).
- UR-0920 – Memória global.
- UR-0921 – Acesso global à memória global.
- UR-0922 – Edição da memória global apenas por tarefas de sistema.

Protecção do kernel

Este módulo deve assegurar que o processador só corre código do kernel em modo privilegiado, que a memória do kernel está protegida das tarefas de utilizador, e que o processador passa para modo protegido quando o controlo é passado para uma tarefa de utilizador (incluindo ao retornar de uma chamada ao sistema).

O início do sistema (*boot*) não precisa de ser alterado, exceptuando o caso da inicialização da MMU e das Page Tables referido atrás.

Para alterar o modo de execução ao passar o controlo para uma tarefa de utilizador basta alterar o modo de execução imediatamente antes do salto para o código da tarefa.

Os requisitos associados a este módulo são os seguintes:

- UR-0200 – Proteger tarefas de sistema.
- UR-0210 – Kernel só deve poder correr em Supervisor Mode.
- UR-0220 – Não executar código de kernel quando estiverem a ser executadas tarefas de utilizador.
- UR-0230 – Protecção de dados e código de kernel.
- UR-0240 – Usar funcionalidades do sistema a partir de *User Mode*.

Gestão das propriedades das tarefas

Este módulo deve acrescentar ao RTEMS a gestão das novas propriedades das tarefas. Deve, por exemplo, assegurar que o modo de execução e o número de contexto de memória virtual são actualizados, aquando da mudança de tarefa no escalonador.

Os requisitos associados a este módulo são os seguintes:

- UR-0100 – Criar tarefas de sistema.
- UR-0310 – Só tarefas de sistema podem criar objectos de sistema.
- UR-0320 – Só tarefas de sistema podem modificar prioridade de uma tarefa.
- UR-0330 – Só tarefas de sistema podem bloquear ou suspender uma tarefa.
- UR-0340 – Só tarefas de sistema podem adormecer outra tarefa.
- UR-0350 – Só tarefas de sistema podem acordar outra tarefa.

Confiabilidade do sistema

Deve-se tentar manter a capacidade de resposta do sistema, e assegurar que o sistema trate os novos erros.

Os requisitos associados a este módulo são os seguintes:

- UR-0400 – Gestão de falhas.
- UR-0410 – Forçar reset aquando de erros internos.
- UR-0700 – Definir a utilização ou não da extensão aquando da construção do sistema.
- UR-0800 – Consistência das estruturas de dados.
- UR-1000 – Tarefas de utilizador não devem poder escrever em partes de memória que contenham código executável.
- UR-1100 – Respeitar a estruturação modularizada do RTEMS.

8 Arquitectura Proposta

As alterações conceptuais propostas para o RTEMS são a diferenciação entre tarefas de sistema e tarefas de utilizador (que correm, respectivamente, em modo privilegiado e em modo protegido) e a protecção de memória (impedindo as tarefas de utilizador de acederem indiscriminadamente à memória).

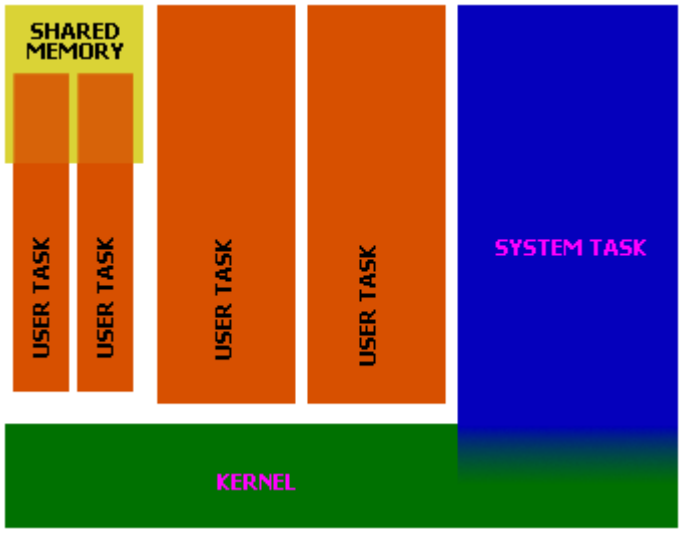

Figura 13: Partição Espacial dos Elementos do Sistema

Para implementá-lo é necessário alterar o RTEMS das seguintes formas:

- Alterar o sistema de gestão de tarefas por forma a permitir a existência de tarefas de utilizador e de sistema;
- Adicionar ao kernel um gestor [dos mapeamentos] de memória virtual, bem como alterar a sequência de inicialização do sistema por forma a inicializar a MMU;
- Adicionar *wrappers* às funções de sistema, por forma a que as tarefas de utilizador possam utilizá-las chamando código em modo supervisor (do kernel) a partir do modo utilizador.

É também preciso compilar o objecto final de forma a que as variadas zonas de memória estejam alinhadas pelos limites necessários ao seu mapeamento para memória virtual.

8.1 Gestão da memória virtual

A gestão da memória virtual tem duas secções:

1. Inicializar a MMU;
2. Gerir a Page Table, assegurando que a Page Table contém as traduções apropriadas e que é alterada aquando de mudanças no sistema.

Propõe-se que a Page Table seja implementada da seguinte forma:

- Um contexto geral mapearia toda a memória com acesso completo (RX para instruções, RW para dados) tanto de utilizador como de sistema, com a granularidade necessária, que seria alterado quando houvessem mudanças na distribuição de memória no sistema;
- Todas as tarefas de sistema usariam ou esse contexto ou um contexto que mapeasse a memória directamente (sem tradução, sendo os endereços virtuais iguais aos endereços físicos). Justifica-se este

acesso indiscriminado com a motivação-base deste projecto, que é isolar as tarefas não-críticas por forma a permitir focar o esforço nas tarefas realmente relevantes, as que determinam o sucesso (ou insucesso) da missão. Assim, a protecção é necessária para proteger as tarefas críticas de acessos indevidos por parte das tarefas "menores" e não vice-versa, pelo que não se justifica limitar as tarefas de sistema;

- Cada tarefa de utilizador teria um contexto próprio;
- Os contextos de utilizador re-usariam partes do contexto geral para mapear as zonas às quais tivessem permissões (procurando assim evitar replicação desnecessária de informação das Page Tables e poupar espaço) e PTEs que permitissem acesso somente em modo privilegiado para as zonas onde não devessem ter acesso, por forma a definir completamente os mapeamentos de memória para esse contexto;
- Se os contextos estiverem construídos de tal forma que o sistema tenha sempre acesso de execução (ou escrita, se for uma zona de dados) e leitura, independentemente do contexto que for e das permissões dadas à tarefa, não é necessário tratar o kernel como uma tarefa, simplificando-se assim a implementação (a tarefa de assegurar que o sistema corra em modo privilegiado adequadamente é da responsabilidade dos *wrappers*, logicamente);
- Obviamente, não deve ser permitido a nenhuma tarefa de utilizador acesso às zonas de memória onde se encontrem o núcleo do sistema, as Page Tables, os endereços de componentes físicos, e outros itens sensíveis.

8.1.1 Inicializador da MMU

Deve existir um conjunto de instruções que permitam inicializar a MMU (UR-0600). Este deve ser chamado aquando do início do sistema, e antes de ser lançada qualquer tarefa (UR-0610).

A melhor solução parece ser incluir estas instruções na sequência de inicialização do sistema – presente no ficheiro rtems/c/src/lib/libbsp/sparc/shared/start.S. Uma abordagem alternativa seria defini-las como uma macro, que seria utilizada no início da tarefa *Init*. Dado o facto da extensão ser activada ou não aquando do re-compilar do RTEMS, e assim ser incluído o activar da MMU sempre que relevante, torna-se mais simples a primeira solução por incluir a inicialização automaticamente.

A inicialização da MMU inclui reservar espaço suficiente, em memória, para a Page Table (alinhado a 0x100). Deve também inicializar a Page Table, se esta não tiver sido inicializada previamente.

Este módulo deve correr em *Supervisor Mode* (UR-0500).

8.1.2 Gestor da Page Table

O funcionamento deste terá de ser, de alguma forma, coordenado com o gestor de tarefas e com o gestor de (alocação de) memória dinâmica. Isto porque, aquando de alterações ao sistema por parte de um destes dois componentes, o gestor da Page Table deve adaptá-la (a Page Table) às novas condições causadas por essas alterações ao sistema.

Este módulo deve correr em *Supervisor Mode* (UR-0500).

O gestor da Page Table deve permitir às tarefas do kernel e às tarefas de sistema (definidas pelo utilizador mas que correm em modo privilegiado) acesso normal e completo ao sistema.

Cada tarefa de utilizador (corre em modo protegido) deve ter acesso de escrita só às suas áreas de memória (UR-0300) (e às áreas de memória global e partilhada).

Deve ser criada a figura da memória partilhada (UR-0900, UR-0910). Para este efeito, deve ser criado um gestor, no kernel, que atribua acesso (completo) a uma região de memória a uma dada tarefa. Esta zona de memória torna-se então, na prática, partilhada. Este gestor não poderia ser chamado ou executado a partir de tarefas de utilizador (ou seja, em modo protegido).

Deve ser criada a figura da memória global (UR-0900, UR-0920). Qualquer tarefa pode aceder a esta memória (UR-0921). Apenas tarefas de sistema podem alterar o conteúdo desta memória (UR-0922).

8.2 Protecção do kernel

A protecção do kernel tem duas características distintas:

3. Protecção do sistema contra adulteramento por parte das tarefas de utilizador;
4. *Wrappers* que permitem manter a acessibilidade a funcionalidades do sistema por parte de tarefas de utilizador.

8.2.1 Protecção contra adulteramento

O sistema deve ser protegido de adulteramento por parte de tarefas de utilizador. (As tarefas de sistema são responsáveis pelo seu próprio comportamento.)

Assim, os espaços de memória das tarefas de sistema só devem poder ser alterados por tarefas de sistema (UR-0200, UR-0230).

As tarefas de sistema (incluindo o kernel) só devem correr no modo privilegiado do processador (UR-0210).

Todos os componentes da extensão fazem parte do código de kernel. Assim, excepto se definido de outro modo, todo o código contido neles deverá ser corrido em modo privilegiado do processador.

8.2.2 Wrappers

O código de sistema não deve ser executado no âmbito de tarefas de utilizador (UR-0220).

Por forma a proporcionar o uso das funcionalidades do sistema às tarefas de utilizador devem ser criados *wrappers* para as chamadas às funções do sistema (UR-0240).

8.3 Gestão das propriedades das tarefas

5. As novas propriedades das tarefas devem ser geridas correctamente.

É necessário poder criar tarefas de sistema, que corram em modo privilegiado do processador (UR-0100). Sugere-se que, por omissão, as tarefas sejam criadas como tarefas de utilizador.

Só as tarefas de sistema podem:

- Criar objectos de sistema (UR-0310), incluindo tarefas;
- Modificar a prioridade de uma tarefa (UR-0320);
- Bloquear ou suspender uma tarefa (UR-0330);
- Adormecer outra tarefa (UR-0340);
- Acordar outra tarefa (UR-0350);

8.4 Confiabilidade do sistema

No que toca à confiabilidade do sistema, as características da extensão foram divididas em:

6. Segurança e robustez;
7. Compatibilidade.

8.4.1 Segurança e Robustez

O sistema deve gerir os novos tipos de falhas lançadas (UR-0400). No caso destas serem devido a erros internos, deve ser forçado o *reset* do sistema (UR-0410).

Deve haver especial preocupação com a consistência das estruturas de dados criadas ou alteradas pela extensão (UR-0800) – deve-se procurar assegurar que não estão mal configuradas, para que delas não derive nenhum erro. No caso mais relevante da Page Table, devem ser definidas funções e macros por forma a manipulá-la da forma mais estruturada possível.

Tarefas de utilizador não devem poder escrever em partes de memória que contenham código executável (UR-1000), por forma a não comprometerem o desempenho do sistema.

8.4.2 Compatibilidade

As alterações que sejam feitas ao RTEMS devem respeitar a estruturação modularizada do RTEMS (UR-1100), separando os componentes e funcionalidades desenvolvidos separados em *architecture-dependent* e *CPU-dependent*.

As alterações devem também ser feitas de tal modo que seja possível compilar o sistema com e sem a extensão (UR-0700). Assim, todas as alterações devem ser encapsuladas:

```
#ifdef MMU
      <novo código>
#else
      <código antigo>
#endif
```

9 Conclusões e trabalho futuro

9.1 Avaliação de Resultados e Estado Actual do Projecto

O projecto provou-se ser mais longo e complexo do que à partida era esperado, dada a complexidade inesperada do problema. Foi preciso absorver conhecimento sobre muitos temas novos e foi também preciso refazer diversas vezes a abordagem e os contornos definidos para o problema por forma a adaptá-los à realidade encontrada, resultando em alterações profundas ao longo do projecto.

O Documento de Requisitos foi produzido e refinado, não se prevendo futuras alterações relevantes a este.

O documento de Especificação da Arquitectura foi produzido, contudo prevê-se uma adição no que toca à utilização da extensão: será preciso agrupar e alinhar correctamente na memória os conteúdos do objecto compilado, por forma a viabilizar a segmentação de memória.

Embora se tenha experimentado o sistema de implementação, não se chegou ao ponto de produzir um documento de Design Detalhado nem de implementar um módulo representativo, que servisse de prova de conceito. Foram feitas experiências de alteração ao sistema, com bons resultados: é conhecido o sistema, as áreas a alterar e como propagar as alterações.

O projecto de estágio foi bem sucedido tanto nos âmbitos estudantil e empresarial como na fronteira entre os dois. Foi um projecto estimulante e uma oportunidade única de aprendizagem, que não foi desperdiçada.

9.2 Concretização e Previsões para Trabalho Futuro

Considera-se a implementação da extensão viável, e estima-se necessário o trabalho de uma pessoa a 6 meses para o levar a bom porto – assumindo que já existem, pelo menos, bases sobre a área.

No que toca às previsões de trabalho futuro, não se prevê nenhum excepto a conclusão do trabalho efectuado durante o estágio, resultando na implementação apropriada da extensão. A um dado momento considerou-se a hipótese do uso de Swap/Cache, mas esta foi descartada já que implicaria um aumento considerável mas pouco frequente da latência máxima, e a perda de determinismo consequente. Foi então considerada inadequada para sistemas de resposta em tempo real.

9.3 Considerações finais relativas ao estágio

O projecto enquanto último elemento de preparação para o mercado de trabalho revelou-se infinitamente precioso. Embora a formação curricular já tivesse focado, de uma forma ou de outra, todos os pontos deste projecto com os quais aprendi, o projecto veio culminar esta aprendizagem, dando o último passo para a minha preparação no mercado de trabalho.

Tudo no projecto se pareceu conjugar para fazer deste último semestre de curso a cereja no topo do bolo: a interacção com o meio empresarial, a familiarização com uma empresa cujos níveis de qualidade de processos e procedimentos alcançam o CMMI 4, a participação por 6 meses num projecto tão interessante, importante e, sobretudo, desafiador como um projecto de Investigação e Desenvolvimento no âmbito dos sistemas de resposta em tempo real... não podia ter desejado melhor.

Durante o projecto revi velhos conhecimentos como C e assembly, familiarizei-me com novos conceitos como sistemas de resposta em tempo real, segmentação e protecção de memória, abstracção de recursos físicos e lógicos, linker scripts e simulação/debugging.

Foi um projecto proveitoso e no qual aumentei seriamente tanto os *soft-skills* (responsabilidade, organização, sociabilidade, etc) como capacidades técnicas, sendo as minhas capacidades reconhecidas pela avaliação interna de desempenho da Critical Software, na qual obtive uma classificação de superior (4 em 5).

Referências e Bibliografia

Todas as páginas-web aqui referenciadas foram consultadas pela última vez em Fevereiro de 2008.

- [01] Exemplos de aplicações desenvolvidas em RTEMS – http://www.rtems.com/wiki/index.php/RTEMSApplications
- [02] SPARC Assembler Language Programming in a Nutshell – http://www2.ics.hawaii.edu/~chin/331/SPARCnut.pdf
- [03] SPARC Assembly Summary – http://www.cs.princeton.edu/courses/archive/spring02/cs217/precepts/sparcassem.pdf
- [04] Documentação do RTEMS – http://www.rtems.com/onlinedocs/releases/rtemsdocs-4.6.6/share/rtems/pdf/
- [05] Manual Sparc V8 – http://www.sparc.org/standards/V8.pdf
- LEON/ERC32 [06] RTEMS Cross Compilation System (RCC) – http://www.gaisler.com/cms/index.php?option=com_content&task=view&id=150&Itemid=31
- [07]Eclipse C/C++ IDE plugin for LEON and ERC32 software development – http://www.gaisler.com/cms/index.php?option=com_content&task=view&id=292&Itemid=31
- [07] CVS Cheat Sheet – http://www.slac.stanford.edu/grp/cd/soft/cvs/cvs_cheatsheet.html
- [08] Git – http://git.or.cz/
- [09] Git Cheat Sheet – http://zrusin.blogspot.com/2007/09/git-cheat-sheet.html
- [10] Design of a Memory Management Unit for System-on-a-Chip Platform "LEON", by Konrad Eisele – http://www.iti.uni-stuttgart.de/LeonMMU/ (com relatório de tese no ficheiro [11]MMU.pdf)
- [12] ht://Dig – http://www.htdig.org/ e http://en.wikipedia.org/wiki/Htdig