**UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS**

RODRIGO WEISSMANN BORGES

**Aplicabilidade de Sistemas Operacionais de Tempo
Real (RTOS) para sistemas embarcados de baixo custo
e pequeno porte**

São Carlos
2011

RODRIGO WEISSMANN BORGES

Aplicabilidade de Sistemas Operacionais de Tempo Real (RTOS) para sistemas embarcados de baixo custo e pequeno porte

Dissertação de Mestrado apresentada à Escola de Engenharia de São Carlos da Universidade de São Paulo, como parte dos requisitos para obtenção do título de Mestre em Ciências, Programa de Engenharia Elétrica.

Área de Concentração:
Processamento de Sinais e
Instrumentação

Orientador: Prof. Dr.
Evandro L. L. Rodrigues

São Carlos

2011

Trata-se da versão corrigida da dissertação. A versão original se encontra disponível na EESC/USP que aloja o Programa de Pós-Graduação de Engenharia Elétrica.

AUTORIZO A REPRODUÇÃO E DIVULGAÇÃO TOTAL OU PARCIAL DESTE TRABALHO, POR QUALQUER MEIO CONVENCIONAL OU ELETRÔNICO, PARA FINS DE ESTUDO E PESQUISA, DESDE QUE CITADA A FONTE.

Ficha catalográfica preparada pela Seção de Tratamento
da Informação do Serviço de Biblioteca – EESC/USP

B732a	<p>Borges, Rodrigo Weissmann.</p> <p>Aplicabilidade de sistemas operacionais de tempo real (RTOS) para sistemas embarcados de baixo custo e pequeno porte / Rodrigo Weissman Borges ; orientador Evandro L. L. Rodrigues. São Carlos, 2011.</p> <p>Dissertação (Mestrado - Programa de Pós-Graduação em Engenharia Elétrica e Área de Concentração em Processamento de Sinais e Instrumentação) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2011.</p> <p>1. Sistemas operacionais. 2. RTOS. 3. Tempo real. 3. Sistemas embarcados. I. Título.</p>
-------	---

FOLHA DE JULGAMENTO

Candidato: Engenheiro **RODRIGO WEISSMANN BORGES**

Título da tese: **Aplicabilidade de sistemas operacionais de tempo real (RTOS) para sistemas embarcados de baixo custo e pequeno porte.**

Data da defesa: 05/07/2011:

Comissão Julgadora:

Resultado:

Prof. Associado **Evandro Luís Linhari Rodrigues (Orientador)**
(Escola de Engenharia de São Carlos/EESC)

APROVADO

Prof. Dr. **Marcelo Andrade da Costa Vieira**
(Escola de Engenharia de São Carlos/EESC)

APROVADO

Prof. Dr. **Jorge Luiz e Silva**
(Instituto de Ciências Matemáticas e de Computação/ICMC/USP)

APROVADO

Coordenador do Programa de Pós-Graduação em Engenharia Elétrica:
Prof. Titular **Denis Vinicius Coury**

Presidente da Comissão de Pós-Graduação:
Prof. Associado **Paulo Cesar Lima Segantine**

DEDICATÓRIA

À minha querida esposa Thais, por todo Amor, Carinho, Compreensão e Força...

Aos meus pais Vilson e Carmem, por todo Amor e pelos ensinamentos todos os dias...

AGRADECIMENTOS

Em primeiro lugar a Deus por sempre iluminar a minha jornada e jamais deixar-me desanimar.

À minha esposa Thais por todo amor e carinho dedicados a mim diariamente, pela compreensão nessa longa jornada, pela força nas horas de dificuldade e por alegrar todos os dias com seu apuradíssimo senso de humor.

A meus pais Vilson e Carmem por todo amor incondicional a mim dedicado e por me ensinarem a importância de lutar pelos sonhos.

A minha irmã Renata e ao meu cunhado Marcos pelo exemplo, apoio e dicas durante o desenvolvimento do trabalho.

Aos meus sobrinhos Matheus e Camila, por em sua graça pueril alegrarem meus dias.

À minha avó Sebastiana pelos deliciosos almoços nos “dias de Mestrado”.

Aos meus sogros Ângela e Nilson e ao meu cunhado Renato por todo apoio e compreensão durante esse período.

À Whirlpool S. A. nas figuras de Claudio Navarro, Marco Tanganeli, Raimundo Rengel e Gerson Eguni pelas horas concedidas para dedicação ao Mestrado e pela valorização do esforço. Em especial gostaria de agradecer ao Claudio Navarro pelas revisões e discussões científicas que muito agregaram a esse trabalho.

Ao professor Evandro por toda orientação e apoio neste trabalho, além da imensa compreensão com relação à dupla jornada (Emprego + Mestrado), ajuda e dedicação.

Aos companheiros do “time de Software” – Fabio, Diego, Celso, Drausio, Nacer, Felipe, Ricardo e André – pelo apoio no desenvolvimento da pesquisa e pelas produtivas discussões.

A Diego Sueiro do Portal Embarcados pela ajuda na disseminação da enquete sobre desenvolvimento de embarcados no Brasil.

A todos que colaboraram com a disseminação da enquete sobre desenvolvimento de embarcados no Brasil.

A todos os amigos que possa ter esquecido, mas que sou grato pela compreensão, apoio, sugestões e críticas.

RESUMO

BORGES, R.W. – Aplicabilidade de Sistemas Operacionais de Tempo Real (RTOS) para sistemas embarcados de pequeno porte e baixo custo - Julho/2008;

Sistemas embarcados ganham cada vez mais espaço devido ao aumento da demanda por novas funções em equipamentos, às normas regulatórias e às novas necessidades dos consumidores e do mercado. Esse aumento nos requisitos aumenta o tamanho e a complexidade dos softwares embarcados cuja importância cresce significativamente. Sistemas Operacionais de tempo real constituem uma ferramenta poderosa para gerenciar a complexidade, facilitar o reuso e aumentar a portabilidade do software e também reduzir o *time-to-market*. Este trabalho visa avaliar a aplicabilidade de sistemas operacionais de tempo real em sistemas embarcados de baixo custo que utilizam microprocessadores pequenos (8 e 16 bits), avaliando suas características e propondo as melhores alternativas para desenvolvimento de software embarcado. Para o atendimento desta proposta, foi realizado o levantamento de características sobre o desenvolvimento brasileiro de sistemas embarcados, uma análise das características de sistemas de pequeno porte, uma discussão da viabilidade do uso de RTOS e um estudo de caso comparando arquiteturas de software embarcado. Os resultados principais mostram que arquiteturas simplificadas como a *Superloop* apresentam vantagem sobre os sistemas operacionais devido ao baixo consumo de memória e processamento. Os sistemas operacionais, apesar de propiciarem desenvolvimentos de códigos modulares bem como facilitar o gerenciamento de tempo, são de difícil implementação em microcontroladores pequenos, devido ao seu elevado consumo de memória e processamento. O uso de sistemas operacionais é viável para sistemas de pequeno porte com no mínimo 4 Kbytes de memória RAM e processos com limite de tempo máximo para execução (*deadlines*) superiores a 1 ms, condições essas que evitam a sobrecarga do microcontrolador. Neste trabalho também é mostrado um retrato do desenvolvimento de embarcados no Brasil.

Palavras chaves: RTOS, Sistemas Operacionais, Tempo-real, Sistemas Embarcados.

ABSTRACT

BORGES, R.W. – Real-time Operating System Aplicability for small and low cost embedded systems - July/2008; Embedded Systems, more and more are gaining importance, due to the increase of features requested on equipments, the regulatory standards and the costumers and market requirements. This increment on requirements increases the software size and complexity, which importance significantly grows. Real-time Operating Systems represents a powerful tool to manage the complexity, help the software reuse and improve portability of the software and also reduce the time-to-market. This work aims to analyze the real-time operating systems, verifying their application on low cost embedded systems using small microcontrollers (8 and 16-bit), evaluating their characteristics and propose the best architectures for software development. To attend this proposal, it was performed a survey of Brazilian embedded system development, evaluates the low cost embedded system characteristics, discusses the viability of RTOS usage and performs a comparative study of embedded software architectures. Results show that simplified architectures like the *Superloop* presents vantages over the operating systems due to their low memory and processing consumption. The operating system, besides helping on time management and code modularity, is difficult to implement in small microcontrollers, due to the high memory and processing consumption. The operating systems are more applicable to small embedded systems with at minimum 4 Kbytes of RAM memory and process with maximum execution time (deadlines) over 1 ms, conditions that do not causes microcontroller overload. In This work is also presented an overview of Brazilian embedded system development.

Keywords: RTOS, Operating Systems, Real-Time, Embedded Systems.

LISTA DE FIGURAS

Figura 1 – Segmentação do mercado de microcontroladores - Dados de 2004. Fonte: (Shandle 2004).	17
Figura 2 – Principais etapas do processo de desenvolvimento de sistemas embarcados. Fonte: (Wolf 2005).	20
Figura 3 – Balanceamento dos requisitos no desenvolvimento de embarcados. Fonte: (Ganssle 1999).	21
Figura 4 - Reuso de código nos projetos de software embarcado. Dados de 2006. Fonte: (CMP United Business Media 2006)	25
Figura 5 - Arquitetura Foreground/Background. Fonte: (Labrosse 2002).	27
Figura 6 - Composição do RTOS e abstração entre Hardware e Software. Fonte: (Cadeño e Laplante 2007).	30
Figura 7 – Tipo de OS utilizado no projeto atual. Fonte: (CMP United Business Media 2006).	32
Figura 8 - Tipo de OS previsto para o próximo projeto. Fonte: (CMP United Business Media 2006)	33
Figura 9 – <i>Kernel</i> Não-preemptivo. Fonte: (Labrosse 2002).	34
Figura 10 - <i>Kernel</i> preemptivo. Fonte: (Labrosse 2002).	35
Figura 11 – Estado das Tarefas e transição entre eles. Fonte: (Li 2003).	39
Figura 12 – Máquina de Estados do Semáforo Binário. Fonte: (Li 2003).	40
Figura 13 – Máquina de Estados do Semáforo Contador. Fonte: (Li 2003).	41
Figura 14 – Máquina de Estado do Semáforo de Exclusão Mútua (Mutex). Fonte: (Li 2003).	41
Figura 15 – Diagrama de blocos do sistema embarcado utilizado na análise comparativa entre as arquiteturas Superloop e RTOS.	53
Figura 16 – Encapsulamento de 44 pinos do Microcontrolador MCS908AW32 (Freescale 2006).	53
Figura 17 – Diagrama de blocos da estrutura interna do Microcontrolador MCS908AW32 (Freescale 2006)	54
Figura 18 – Superloop com Slots e temporização do background	56
Figura 19 - Sistema de Testes para o Estudo de Caso comparativo entre as arquiteturas RTOS e <i>Superloop</i> .	60
Figura 20 – Cargos ocupados pelos desenvolvedores participantes da pesquisa - distribuição percentual.	64
Figura 21 – Perfil de Idade dos participantes da pesquisa. Distribuição percentual em faixas Etárias.	65
Figura 22 – Formação Superior dos desenvolvedores participantes da pesquisa – distribuição percentual	65
Figura 23 - Formação Complementar dos desenvolvedores participantes da pesquisa – distribuição percentual	66
Figura 24 – Experiência dos participantes no desenvolvimento de embarcados. Distribuição percentual em intervalos de tempo referentes aos anos de experiência	67
Figura 25 – Quantidade de projetos simultâneos desenvolvidos pelos participantes – distribuição percentual	67
Figura 26 – Duração dos projetos de embarcados desenvolvidos pelos participantes. Distribuição percentual em intervalos de tempo referentes à duração	69

Figura 27 – Desafios enfrentados pelos participantes no desenvolvimento dos projetos embarcados. Distribuição percentual das respostas nos itens citados.	70
Figura 28 – Arquitetura de microprocessadores utilizados nos projetos de sistemas embarcados pelos participantes – distribuição percentual das respostas.....	71
Figura 29 – Fatores determinantes na seleção dos microprocessadores. Distribuição percentual das respostas entre os itens citados.	73
Figura 30 – Principais linguagens de programação utilizadas no desenvolvimento de software embarcado. Distribuição percentual de respostas entre as linguagens citadas.	74
Figura 31 – Taxa de reuso de código no desenvolvimento de software embarcado. Distribuição percentual de respostas em níveis de reuso.	75
Figura 32 – Uso de sistemas operacionais nos projetos atuais de software embarcado. Distribuição percentual das respostas.	76
Figura 33 – Distribuição percentual do uso de sistemas operacionais entre as arquiteturas de microprocessadores.	76
Figura 34 – Principais fatores que motivam a utilização de sistemas operacionais no desenvolvimento de software embarcado. Distribuição percentual das respostas entre os itens citados.	77
Figura 35 - Principais fatores que motivam a não utilização de sistemas operacionais no desenvolvimento de software embarcado. Distribuição percentual das respostas entre os itens citados.	78
Figura 36 – Utilização de Sistemas Operacionais nos projetos de sistemas embarcados atuais versus pretensão de uso nos projetos futuros. Distribuição percentual das respostas.	79
Figura 37 – Tipos de sistemas operacionais preferidos pelos desenvolvedores brasileiros. Distribuição percentual das respostas.	80
Figura 38 - Fatores determinantes na seleção dos OSs embarcados. Distribuição percentual das respostas entre os itens citados.	81
Figura 39 – Sistemas Operacionais utilizados nos projetos de sistemas embarcados. Distribuição percentual das respostas	82
Figura 40 – Intenção de uso de sistemas operacionais de tempo real nos projetos de software embarcado. Distribuição percentual de respostas.....	82
Figura 41 – Requisitos temporais necessários ao RTOS embarcado segundo os participantes. Distribuição temporal das respostas.	83
Figura 42 - Requisitos temporais necessários ao RTOS embarcado segundo os participantes que consideram a sua utilização. Distribuição temporal das respostas.	84
Figura 43 – Lacuna no aprendizado necessário ao desenvolvimento de embarcados. Ambos os currículos de ciências da computação e engenharia elétrica não apresentam todos os conceitos necessários ao desenvolvimento de embarcados. Fonte: (Barr 2009).....	85
Figura 44 - Características necessárias às tarefas do RTOS embarcado segundo os participantes. Distribuição temporal das respostas.	86
Figura 45 - Características necessárias às tarefas do RTOS embarcado segundo os participantes que consideram a sua utilização. Distribuição temporal das respostas.	86
Figura 46 – Mapa de memória do Software implementado utilizando a arquitetura Superloop. Identifica os segmentos de memória ROM e RAM utilizados.....	87
Figura 47 - Mapa de memória do software implementado utilizando a arquitetura <i>Superloop</i> . Identifica os segmentos de memória ROM e RAM utilizados.....	89
Figura 48 – Exemplo de cálculo da pilha utilizada por uma tarefa. Soma das variáveis locais criadas e das chamadas locais encadeadas.	92

Figura 49 – Mapa de memória da aplicação implementada utilizando o sistema operacional de tempo-real MicroC-OS II.	94
Figura I - 1 - Fluxo das perguntas da enquete Desenvolvimento de Embarcados no Brasil	126

LISTA DE TABELAS

Tabela 1 – Exemplos de sistemas embarcados e seus mercados. Fonte: (Noergaard 2005)	15
Tabela 2 - Exemplos de arquiteturas de processadores embarcados para diferentes números de bits. Fonte: (Noergaard 2005)	17
Tabela 3 – Linguagens de programação utilizada no desenvolvimento de projetos embarcados. Dados de 2006. Fonte: (CMP United Business Media 2006)	25
Tabela 4 – Tarefas do sistema com seus deadlines e período.	55
Tabela 5 – Divisão das tarefas entre os níveis da arquitetura <i>Superloop</i>	58
Tabela 6 – Correlação entre tarefas do <i>Superloop</i> e tarefas do sistema operacional de tempo-real	59
Tabela 7 – Tipo de projeto embarcado desenvolvidos pelos participantes – distribuição percentual	68
Tabela 8 – Andamento dos projetos de sistemas embarcados com base no cronograma inicial. Distribuição percentual em faixas de variações temporais do cronograma.	70
Tabela 9 – Principais fabricantes dos microprocessadores utilizados nos projetos de sistemas embarcados. Distribuição percentual das respostas	72
Tabela 10 – Medidas Temporais da Arquitetura <i>Superloop</i> usadas na avaliação do atendimento de <i>deadlines</i> e do consumo de processamento do sistema.	88
Tabela 11 - Medidas temporais da arquitetura <i>Superloop</i> usadas na avaliação do teste exploratório de adição de funcionalidades e substituição de microcontrolador.	89
Tabela 12 – Prioridades atribuídas às tarefas pelos algoritmos <i>Deadline Monotonic</i> (DM) e <i>Rate Monotonic</i> (RM).....	91
Tabela 13 – Tarefas do sistema após otimização: funcionalidades, restrições temporais e prioridades.....	93
Tabela 14 – Tempos máximos de execução observados para as tarefas de maior prioridade do sistema.....	95
Tabela 15 – Tempo de execução da interrupção de tempo e das tarefas do sistema operacional.	95
Tabela 16 – Vantagens e desvantagens dos diferentes sistemas operacionais: <i>open-source</i> , comercial e proprietário. Classificação em três níveis: Ponto Forte ●, Intermediário ◐ e Ponto Fraco○	107

LISTA DE SIGLAS

APEX	<i>Avionics Application Software Standard Interface</i>
API	<i>Application Program Interface</i>
ARINC	<i>Aeronautical Radio Incorporated</i>
CAD	<i>Computer Aided Design</i>
CPU	<i>Central Processing Unit</i>
DM	<i>Deadline-Monotonic</i>
EDF	<i>Early Deadline First</i>
FIFO	<i>First-in-first-out</i>
GPOS	<i>General-purpose Operating System</i>
ID	Identificação
IEEE	<i>Institute of Electrical and Electronics Engineers</i>
ISR	<i>Interrupt Service Routine</i>
LED	<i>Light-emitting Diode</i>
LIFO	<i>Last-in-First-Out</i>
LST	<i>Least Slack Time First</i>
OS	<i>Operating System</i>
OSEK/VDX	Sistemas e interfaces abertas para eletrônica automotiva
PC	<i>Personal Computer</i>
POSIX	<i>Portable Operating System Interface</i>
RAM	<i>Random Access Memory</i>
RM	<i>Rate-Monotonic</i>
ROM	<i>Read-only memory</i>
RTOS	<i>Real-Time Operating System</i>
SCB	<i>Semaphore Control Block</i>
TCB	<i>Task Control Block</i>

SUMÁRIO

Capítulo 1	1
Introdução	1
1. Considerações iniciais.....	1
2. Motivação.....	2
3. Objetivos	3
4. Organização do Trabalho	4
Capítulo 2	5
Sistemas de Tempo Real	5
1. Introdução	5
2. Conceitos de Sistemas de Tempo Real.....	5
2.1. Processo (<i>Job</i>)	5
2.1.1. Parâmetros Temporais.....	6
2.1.2. Parâmetros de Interconexão	7
2.1.3. Parâmetros Funcionais	7
2.2. Tarefas.....	8
2.3. Recursos	8
3. Classificação de Sistemas de Tempo Real	8
4. Classificação de Tarefas.....	9
5. Escalonamento de Tarefas.....	10
Capítulo 3	13
Sistemas Embarcados	13
1. Introdução	13
2. Microprocessador Embarcado.....	16
3. Desenvolvimento de sistemas embarcados	18
Capítulo 4	23
Software Embarcado	23
1. Introdução	23
2. Arquitetura <i>Pooled Loop</i>	25
3. Arquitetura <i>Foreground/Background</i>	26
4. Sistemas Operacionais.....	28
4.1. Introdução	28
4.2. <i>Real-Time Operating Systems</i> - RTOS.....	29

4.3.	<i>Kernel</i>	33
4.3.1.	Escalonador	36
4.3.1.1.	<i>Round-robin e Weighted Round Robin</i>	36
4.3.1.2.	Algoritmos orientados à prioridade	37
4.3.2.	Objetos	38
4.3.2.1.	Tarefas	38
4.3.2.2.	Semáforos	40
4.3.2.3.	Fila de Mensagens	42
4.3.2.4.	<i>Pipes</i>	42
4.3.2.5.	<i>Registrador de Eventos</i>	43
4.3.3.	<i>Serviços</i>	43
4.4.	Arquiteturas de desenvolvimento de sistemas operacionais	43
4.4.1.	OS monolítico	44
4.4.2.	OS em camadas	44
4.4.3.	<i>Microkernel</i>	45
4.5.	Padrões	46
4.5.1.	RT- POSIX	46
4.5.2.	OSEK	47
4.5.3.	APEX	47
4.5.4.	μ ITRON	48
Capítulo 5		51
Metodologia		51
Capítulo 6		63
Resultados e Discussão		63
1.	Introdução	63
2.	Desenvolvimento de Embarcados no Brasil	63
2.1.	Perfil dos Desenvolvedores	63
2.2.	Características dos projetos embarcados	68
2.3.	Microcontrolador embarcado	71
2.4.	Características do software embarcado	73
2.5.	Sistemas operacionais embarcados	75
2.6.	Real-time operational system (RTOS)	82
3.	Estudo de Caso: <i>Superloop</i> versus RTOS	87
3.1.	<i>Superloop</i>	87
3.2.	Sistema Operacional de Tempo Real.	90
4.	Sistemas embarcados de pequeno porte	97

4.1.	Arquitetura de 8-bits x Arquitetura de 32-bits	97
4.2.	Assembly x Linguagem C	98
4.3.	Superloops x Sistemas Operacionais de Tempo-Real	100
4.4.	Sistemas Operacionais Preemptivos x Cooperativos	104
4.5.	RTOS Comercial x Proprietário	105
Capítulo 7		109
Conclusões		109
1.	Considerações Finais	109
1.1.	Desenvolvimento de Embarcados no Brasil	109
1.2.	<i>Superloop</i> versus RTOS	110
1.3.	Sistemas Embarcados de pequeno porte	111
2.	Conclusões	113
3.	Contribuições	115
4.	Trabalhos Futuros	115
Referências		117
Apêndice I		121
Enquete Desenvolvimento de embarcados no Brasil		121
1.	Perguntas:	121
2.	Fluxo da Pesquisa	126

Capítulo 1

Introdução

1. Considerações iniciais

Muitos cientistas e pensadores respeitados acreditam que a casa do futuro será repleta de dispositivos inteligentes, evoluções de nossos atuais equipamentos eletroeletrônicos. Prevêem também a disseminação de aparelhos para comunicação, acesso a informações e organização do nosso dia-a-dia, os quais estarão em toda parte, embutidos em nossas roupas e móveis. Na era da conectividade, refrigeradores não apenas irão armazenar e conservar alimentos, mas também monitorar seu frescor e prover seus dados nutricionais. Esse futuro descrito no artigo “*This new House*” (Murphy 2005), da revista *Fortune*, não parece tão distante, considerando a mudança que a eletrônica vem causando em nossas vidas.

Os sistemas eletrônicos estão hoje presentes em diversas áreas, da automação industrial à automação residencial, dos instrumentos médicos aos automóveis, dos celulares às máquinas de lavar. É difícil identificar hoje atividades de nossa vida diária em que os sistemas embarcados não estejam presentes.

Os sistemas embarcados são compostos de hardware, software e possíveis componentes mecânicos adicionais, e em geral, executam funções dedicadas e compartilham propriedades semelhantes. Recursos limitados (restrições de memória, peso, espaço e energia), requisitos de tempo real, concorrência de tarefas, alta complexidade e compartilhamento de recursos são alguns exemplos dessas propriedades.

Esses sistemas ganham cada vez mais espaço, devido ao aumento da demanda por novas funções em equipamentos, às novas necessidades dos consumidores, à concorrência do mercado, aos requisitos de segurança. Também ganham evidência devido às normas regulatórias que restringem emissões de poluentes, consumo de energia e água e definem os requisitos de operação do equipamento.

Esse crescimento nos requisitos dos equipamentos eletroeletrônicos aumenta a complexidade e o tamanho dos softwares embarcados cuja importância vem crescendo significativamente, incorporando funcionalidades do hardware, implementando algoritmos complexos de controle e permitindo inovações em produtos.

Sistemas Operacionais de tempo real constituem uma ferramenta poderosa para gerenciar a complexidade, facilitar o reuso e aumentar a portabilidade do software e também reduzir o *time-to-market*.

Esses sistemas operacionais também trazem outras vantagens ao desenvolvimento, facilitando a comunicação de dados, o gerenciamento de multitarefas, de recursos e de tempo. Contudo, também apresentam desvantagens consumindo grandes recursos de processamento e memória.

Atualmente, diferentes Sistemas Operacionais de Tempo Real (RTOS) comerciais, *open-source*, de pesquisa e proprietários desenvolvidos internamente por empresas, estão disponíveis no mercado.

RTOS são amplamente difundidos e utilizados em sistemas de grande porte que fazem uso de arquiteturas de microprocessadores de 32-bits e 64-bits. Contudo, grande parte dos sistemas embarcados atuais é desenvolvida utilizando microcontroladores de pequeno porte e baixo custo – 8-bits. (Bannatyne 2009). Pesquisas estimam em US\$ 5 bilhões o mercado de microcontroladores de 8-bits, com um crescimento de 48% entre 2007 e 2011. Isso se deve a grande pressão por custos existente na indústria, principalmente nos mercados emergentes (Sperling 2010).

2. Motivação

Sistemas embarcados de pequeno porte utilizam arquiteturas simplificadas de software, de fácil implementação e que apresentam baixo consumo de memória e de processamento.

No entanto, a exigência de novas funcionalidades vem aumentando significativamente a complexidade do software. Essa exigência crescente de gerenciamento de grande número de tarefas traz dificuldades à utilização dessas arquiteturas.

Sistemas operacionais de tempo real surgem como uma alternativa para esses desenvolvimentos. Contudo, seu grande consumo de memória e processamento traz desafios ao seu uso em microprocessadores de pequeno porte.

Além disso, não obstante à importância desse mercado de microprocessadores, ele é praticamente esquecido pelas empresas fabricantes de RTOS, que focam suas iniciativas para microcontroladores de 32-bits.

Contudo, o avanço da tecnologia dos microcontroladores de 8-bits vem barateando dispositivos com melhor desempenho, maiores recursos de memória e periféricos.

Outra questão a ser pontuada aqui e que também serve como motivação para esse trabalho, é que os desenvolvedores de software embarcado utilizam cada vez mais os conceitos de engenharia de software, de sistemas operacionais, etc., buscando elevar a qualidade do processo todo.

3. Objetivos

Pertence ao escopo deste trabalho, analisar a viabilidade da aplicação de sistemas operacionais de tempo real em sistemas embarcados de baixo custo que utilizam microcontroladores pequenos (8-bits), avaliando suas características e propondo as melhores alternativas para desenvolvimento de software embarcado.

Os objetivos principais deste trabalho são:

- Apresentar as características de sistemas operacionais de tempo real e sua operação.
- Apresentar as características do desenvolvimento de sistemas embarcados.
- Analisar e discutir a viabilidade do uso de sistemas operacionais em sistemas embarcados de pequeno porte (8 e 16 bits).
- Comparar os sistemas operacionais de tempo real com as demais arquiteturas de softwares embarcados, propondo as melhores alternativas para desenvolvimento de sistemas embarcados de pequeno porte.

4. Organização do Trabalho

Este trabalho está dividido em sete capítulos, incluindo este capítulo inicial que introduz o tema, a motivação e objetivos do trabalho.

Os capítulos 2, 3 e 4 apresentam a fundamentação teórica do trabalho. O capítulo 2 apresenta os conceitos de sistemas de tempo real (Processos, Tarefas, Recursos e Escalonamento) e a classificação desses sistemas. O capítulo 3 apresenta e discute conceitos de sistemas embarcados, características do microprocessador embarcado e do desenvolvimento de sistemas embarcados. O capítulo 4 apresenta as arquiteturas de software embarcado e os conceitos de sistemas operacionais de tempo-real.

O capítulo 5 apresenta a metodologia utilizada para atender os objetivos.

O capítulo 6 mostra os resultados relativos à pesquisa sobre características do desenvolvimento de embarcados no Brasil, discute e apresenta os resultados de um estudo de caso comparando arquiteturas de software embarcado e analisa as características dos sistemas embarcados de pequeno porte, características do software embarcado e o uso de sistemas operacionais embarcados.

O Capítulo 7 apresenta as conclusões, contribuições e as propostas de trabalhos futuros.

Capítulo 2

Sistemas de Tempo Real

1. Introdução

Sistemas de tempo real são sistemas capazes de processar dados e gerar resposta dentro de restrições temporais, antes de um tempo máximo pré-determinado (Aroca 2008).

Esses sistemas muitas vezes são erroneamente confundidos com sistemas velozes, pois, como definido por Timmeman e Perneel (Timmerman e Perneel 2001), em sistemas de tempo real, o importante é a previsibilidade e não a velocidade média do sistema.

Ramamritham e Stankovic (Ramamritham e Stankovic 1994) definem sistemas de tempo real como aqueles que não dependem apenas de um correto resultado lógico da computação, mas também do instante em que estes são produzidos.

Sistemas de tempo real vêm sendo amplamente usados em diferentes aplicações de grande importância econômica e social, como processamento de sinais, sistemas embarcados, aviónica, controle de tráfego aéreo, instrumentação biomédica e robótica (Liu 2000) (Nery 2009).

2. Conceitos de Sistemas de Tempo Real

2.1. Processo (*Job*)

O processo é a menor unidade de trabalho agendada e executada pelo sistema. Os processos são caracterizados por parâmetros temporais, funcionais, parâmetros de interconexão e de recursos.

2.1.1. Parâmetros Temporais

Os parâmetros temporais definem as restrições de tempo do processo. Os principais parâmetros são:

- Tempo de início: corresponde ao instante em que um processo requisitado inicia a execução.
- Tempo de término: corresponde ao instante em que um processo conclui a sua execução.
- Tempo de execução: corresponde ao tempo gasto para executar o processo, sem processos concorrentes e com todos os recursos disponíveis. O tempo de execução pode variar por diferentes razões (saltos condicionais, dados de entrada, *pipeline* etc.), porém, através de análises, é possível determinar os valores máximo e mínimo para um determinado processo (Liu 2000).
- Tempo de resposta: corresponde ao intervalo de tempo entre o instante em que um processo é requisitado ao sistema e o instante em que conclui sua execução.
- Tempo de chegada: corresponde ao instante em que o escalonador toma conhecimento da ativação da nova requisição.
- Tempo de liberação (*Release Time*): corresponde ao momento em que o processo se torna disponível para execução.
- *Deadline*: limite de tempo para execução do processo. Pode ser classificado em:
 - *Deadline* Relativo: máximo tempo de resposta permitido ao processo.
 - *Deadline* Absoluto: é igual ao tempo de liberação acrescido do *Deadline* Relativo.

2.1.2. Parâmetros de Interconexão

Os parâmetros de interconexão definem as dependências dos processos em relação aos demais processos.

Os processos podem apresentar dependência com relação aos demais processos como condições de precedência, caso existam restrições de ordem de execução. Caso os processos não apresentem ordem de execução, eles são descritos como independentes (Liu 2000).

Em muitos sistemas os processos comunicam-se através de dados compartilhados, sendo interessante sincronizar produtores e consumidores.

2.1.3. Parâmetros Funcionais

Os parâmetros funcionais definem as propriedades intrínsecas dos processos, dentre os quais se destacam a prioridade e a habilidade de preempção.

A criticidade ou prioridade define a importância de um processo, sendo normalmente representada como um número positivo que indica o quão crítico um processo é em relação aos outros. Quanto mais crítico maior sua importância.

A habilidade de preempção consiste em suspender a execução para que outros processos sejam executados. Um processo é dito preemptível se sua execução pode ser suspensa a qualquer momento para permitir a execução de outros processos. É dito não-preemptível: se sua execução não deve ser interrompida até sua conclusão. Essa restrição pode ocorrer devido à necessidade da execução ser reiniciada em casos de interrupção.

2.2.Tarefas

As tarefas consistem em conjuntos de processos que juntos provêem uma funcionalidade ao sistema.

2.3.Recursos

Os recursos de um sistema de tempo-real podem ser divididos em dois grupos: processadores e recursos.

Processadores, também chamados de servidores, são responsáveis pela execução dos processos. Todo processo necessita de um ou mais processadores para ser executado e garantir seu progresso.

Os processadores podem ser divididos por tipos, dependendo da aplicação. Eles são considerados do mesmo tipo se suas funcionalidades são idênticas e intercambiáveis.

Por recurso definem-se os recursos passivos do sistema. Esses recursos não interferem na velocidade de execução do sistema, porém necessitam estar disponíveis para garantir que um processo seja executado.

3. Classificação de Sistemas de Tempo Real

As tarefas são classificadas segundo suas restrições de tempo (*deadline*) em *hard* ou *soft real-time*.

Liu (Liu 2000) descreve as seguintes definições para os conceitos de *hard* e *soft real-time*, baseado em diferentes critérios:

- Baseada na criticidade da funcionalidade: um *hard deadline* é aquele que, caso não seja atendido, pode apresentar consequências desastrosas. Um *soft deadline* é aquele em que resultados tardios não são desejados, porém não

causam efeitos desastrosos, apenas deterioram o sistema. Esta definição recai sobre a subjetividade de mensurar o efeito de uma resposta atrasada.

- Baseada na usabilidade de resultados tardios: um *hard deadline* é aquele cuja usabilidade do resultado decai abruptamente com o atraso no atendimento do *deadline*. Um *soft deadline* é aquele cuja usabilidade do resultado decai lentamente conforme o atraso no atendimento do *deadline*. Essa é uma medida quantitativa, porém a dificuldade em determinar funções entre atraso no atendimento do *deadline* e a usabilidade do resultado tornam sua aplicação complexa.
- Baseada na probabilidade de atender os *deadlines*: um *hard deadline* é aquele que sempre deve ser atendido. Um *soft deadline* é aquele que pode esporadicamente não ser atendido dentro de certa probabilidade. Esta definição ignora completamente o resultado de um não atendimento do *deadline*.

Liu (Liu 2000) também propõe uma definição baseada na garantia da qualidade de serviço temporal do processo. Baseado neste critério, um *hard deadline* é aquele cujo usuário necessita da validação de que o sistema sempre irá atendê-lo, enquanto um *soft deadline* é aquele que exige apenas uma demonstração de que o sistema vai atender o *deadline* com certa probabilidade. A validação é feita através de exaustivos testes e simulações.

Uma aplicação (tarefa) com restrições de tempo *hard* é chamada uma aplicação *hard real-time*. Um sistema que contém em sua maioria aplicações *hard real-time* é chamado de sistema *hard real-time*.

Alguns autores definem também o conceito de sistema *firm real-time*. Aroca (Aroca 2008) o define como um sistema intermediário entre o *soft real-time* e *hard real-time*, no qual o acúmulo de atrasos nas tarefas *soft real-time* pode transformá-lo em um sistema *hard real-time*.

4. Classificação de Tarefas

As tarefas são modeladas segundo sua periodicidade e restrições de tempo.

Liu (Liu 2000) propõe três modelos de tarefas:

- Tarefas periódicas: compostas por processos executados em intervalos regulares ou semi-regulares para prover uma funcionalidade do sistema.
- Tarefas esporádicas: compostas por processos com restrições *hard real-time* liberados aleatoriamente para execução.
- Tarefas aperiódicas: compostas por processos com restrições *soft real-time* liberados aleatoriamente para execução.

Os processos que compõem as tarefas apresentam o mesmo comportamento estatístico e requisitos de tempo.

5. Escalonamento de Tarefas

Em sistemas de tempo real, os processos são escalonados e têm recursos alocados de acordo com algoritmos de escalonamento e protocolos de controle de acesso a recursos (Liu 2000).

O escalonador, componente responsável pela implementação dos algoritmos de escalonamento, é responsável por decidir, em determinado instante, a tarefa a ser executada. Apresenta grande relevância em sistemas de tempo real, pois a tarefa escalonada deve cumprir as restrições temporais (Liu 2000) (Nery 2009).

Segundo Ramamritham e Stankovic (Ramamritham e Stankovic 1994), escalonamento é o tópico de sistemas de tempo real mais amplamente pesquisado, devido à crença de que o problema básico de sistemas de tempo real é garantir que as restrições de tempo das tarefas sejam atendidas.

Liu (Liu 2000) descreve três abordagens comumente utilizadas para escalonamento em tempo real:

- Abordagem dirigida por tempo (*clock-driven ou time-driven*):
Nesta abordagem, decisões relativas ao escalonamento são realizadas em instantes específicos de tempo. Tipicamente em sistemas que usam essa abordagem, todos os parâmetros *hard real-time* são fixos e conhecidos. Nesses sistemas, geralmente uma escala é computada *off-line* e armazenada para utilização durante o tempo de execução.

- Abordagem de compartilhamento de tempo (*round-robin*):

A abordagem *round-robin* é comumente utilizada para escalonar aplicações de tempo compartilhado, sendo o algoritmo também conhecido como compartilhamento de processador. Nesta abordagem, cada processo recebe uma fatia de tempo (unidade básica de tempo) igual para execução. O período equivalente a todas as fatias dos processos, prontos para execução, é denominado *round*.

O algoritmo *weighted Round-Robin*, segue o mesmo princípio do algoritmo tradicional. Contudo, o processador não é dividido igualmente entre os processos. Processos diferentes recebem diferentes pesos (*weights*). Nesse caso, o *round* é igual à soma dos pesos. O ajuste dos pesos permite acelerar ou desacelerar o progresso dos processos.

- Abordagem dirigida por prioridade (*priority-driven*)

Os algoritmos dirigidos por prioridade compõem uma grande classe de algoritmos de escalonamento que maximizam a utilização dos recursos. Nestes algoritmos, os processos com maior prioridade são escalonados e executados nos recursos disponíveis.

As prioridades usualmente são atribuídas baseando-se nos parâmetros temporais como tempo de chegada, como no caso dos algoritmos FIFO (*First-in-First-out*) e LIFO (*Last-in-First-Out*), *deadline* como nos algoritmos EDF (*Earliest-deadline-first*) e DM (*deadline monotonic*), ou no período como no algoritmo RM (*Rate-Monotonic*).

Os algoritmos de escalonamento podem ser classificados quanto às suas características:

- Quanto à análise de escalonabilidade
 - Estáticos: nestes algoritmos a análise de escalonabilidade é feita de maneira estática. Assumem o conhecimento prévio das características relevantes de todas as requisições (Nery 2009) (Ramamritham e Stankovic 1994).
 - Dinâmicos: nestes algoritmos a análise de viabilidade é testada dinamicamente. São projetados para trabalhar com requisições imprevisíveis quanto ao tempo de chegada e, possivelmente com tempo de execução desconhecido (Nery 2009) (Ramamritham e Stankovic 1994).

Os sistemas dinâmicos podem, em média, apresentar melhor resposta do que sistemas estáticos. No entanto, nos piores casos seu desempenho de tempo real é mais pobre. Além disso, a validação de sistemas estáticos apresenta técnicas e resultados mais confiáveis, sendo mais utilizados em sistemas *hard real-time* (Liu 2000).

- Quanto à Preempção:
 - Preemptivos: permitem que uma tarefa seja temporariamente suspensa para a execução de outra com maior prioridade (Nery 2009).
 - Não-preemptivos: impedem que a execução de uma tarefa seja interrompida até sua conclusão (Nery 2009).
- Quanto à execução
 - *Off-line*: nestes algoritmos o escalonamento é pré-computado antes da execução do sistema. O cálculo é baseado no conhecimento dos tempos de liberação e dos requisitos de recursos e processadores em todos os instantes. Esses algoritmos apresentam comportamento temporal determinístico e tipicamente obtém uma utilização dos recursos próxima da máxima. No entanto, apresentam a desvantagem de serem inflexíveis (Liu 2000).
 - *On-line*: nestes algoritmos o escalonamento é realizado em tempo de execução e a decisão de escalonar uma tarefa é feita sem conhecimento das tarefas futuras. O escalonamento *On-line* acomoda variações dinâmicas nas demandas do usuário e na disponibilidade de recursos, porém, não aperfeiçoam a utilização dos recursos (Liu 2000).
- Quanto ao Sincronismo
 - Orientado a tempo: as decisões sobre qual requisição executar são tomadas de acordo com instantes previamente estabelecidos (Nery 2009).
 - Orientado a evento: as decisões de escalonamento são baseadas em eventos (*event-driven*) (Nery 2009).

O produto do escalonador é a escala, que consiste no agendamento de todos os processos de um sistema em um ou mais processadores.

Capítulo 3

Sistemas Embarcados

1. Introdução

Sistemas embarcados consistem em uma combinação de Hardware, Software e possíveis componentes adicionais mecânicos, desenvolvidos para a execução de uma função dedicada (Li 2003) (Stallings 2008). Esses sistemas computacionais aplicados diferem de sistemas computacionais de propósito geral, como computadores pessoais (PC - *Personal Computer*) ou supercomputadores, apresentando maiores limitações de funcionalidades de hardware e de software (Noergaard 2005).

O adjetivo embarcado reflete o fato de esses sistemas serem usualmente parte integrante de um sistema maior. No entanto, apesar de muitos sistemas embarcados poderem coexistir em um sistema, eles podem, por si só, representar o sistema completo e operarem individualmente (Li 2003).

O uso de sistemas embarcados vem aumentando drasticamente em nosso dia-a-dia. De forma não imaginada nas décadas passadas, sistemas embarcados estão transformando o modo de vida, trabalho e diversão das pessoas (Buttazzo 2006) (Li 2003).

Atualmente, é difícil encontrar na vida diária segmentos que não envolvam sistemas embarcados de alguma forma. Eles estão espalhados em diferentes áreas como indústria automotiva, eletrônica de consumo, aviação, controle industrial, instrumentos médicos e dispositivos de rede (*hubs, gateways, roteadores* etc.) (Li 2003) (Noergaard 2005) (Stallings 2008). A Tabela 1 mostra exemplos de aplicações de sistemas embarcados nos diferentes mercados.

Segundo Buttazzo (Buttazzo 2006) e Berger (Berger 2002), a maioria dos sistemas embarcados divide propriedades importantes:

- Recursos limitados: muitos sistemas embarcados são desenvolvidos sobre restrições de espaço, peso e energia, impostos pela aplicação.

- Sensíveis a custo: frequentemente apresentam também restrições de custo devido à produção em massa e grande competição industrial. Conseqüentemente, aplicações embarcadas tipicamente operam em pequenas unidades de processamento com grande limitação de memória e potencial computacional, sendo que para obtenção de custos efetivos é mandatório o uso altamente eficiente dos recursos computacionais.
- Limitações de tempo real: a maioria dos dispositivos embarcados interage com o ambiente e deve reagir a eventos externos e executar atividades computacionais dentro de restrições precisas de tempo, sendo necessária a previsibilidade e garantia *off-line* dos requisitos de desempenho.
- Comportamento dinâmico: consistem de dezenas ou centenas de tarefas concorrentes que interagem entre si para o uso de recursos compartilhados.
- Diferentes processadores: sistemas embarcados são suportados por uma grande quantidade de processadores e arquiteturas de processadores.

Friedrich (Friedrich 2009) define onze qualidades que definem requisitos não funcionais utilizados para julgar a operação dos sistemas embarcados. São elas:

1. Recursos limitados de computação: os recursos computacionais devem ser utilizados de maneira eficiente.
2. Requisitos de tempo real: sistemas embarcados interagem com o ambiente, necessitando reagir corretamente dentro de requisitos estritos de tempo.
3. Portabilidade: diferentes tipos de CPU's (Unidade de processamento central), periféricos e memórias podem ser usados em sistemas embarcados.
4. Alta confiabilidade: sistemas embarcados são usados remotamente e em aplicações críticas, o que torna a correção de falhas problemática, extremamente cara e até mesmo impossível de correção.
5. Robustez e estabilidade do sistema: o sistema deve operar fora de condições nominais e evitar interrupções na operação.
6. Tratamento de falhas: os sistemas devem identificar e tratar erros e garantir tolerância a falhas da aplicação.
7. Operação segura: os sistemas devem prevenir ferimentos, perdas de vida e danos a propriedades e ao ambiente.
8. Segurança de informações: os sistemas devem evitar que informações internas sejam usadas ou alteradas por usuários não autorizados.

9. Privacidade: os sistemas devem possuir a habilidade de isolar e revelar seletivamente informações.
10. Escalabilidade: o sistema deve ser capaz de gerenciar o aumento na carga de trabalho e possibilitar a expansão.
11. Atualização: os sistemas devem permitir que a especificação seja aprimorada, adicionando ou substituindo componentes.

Tabela 1 – Exemplos de sistemas embarcados e seus mercados. Fonte: (Noergaard 2005)

Mercado	Dispositivo Embarcado
Automotivo	Sistema de Ignição
	Controle de Motor
	Freios (ABS)
Eletrônica de Consumo	Televisores Analógicos e Digitais
	DVDs, Vídeo cassetes
	<i>Personal Data Assistants</i> (PDAs)
	Eletrodomésticos (Refrigeradores, Microondas, Torradeiras)
	Brinquedos, Jogos
	Telefones, <i>Pagers</i> e Celulares
	Câmeras
	<i>Global Positioning Systems</i> (GPS)
Controle Industrial	Robótica e Controle de Sistemas (Manufatura)
Medicina	Bombas de Infusão
	Próteses
	Equipamento de Diálise
	Monitores Cardíacos
Redes de Comunicação	Roteadores
	<i>Hubs</i>
	<i>Gateways</i>
Automação de Escritórios	Equipamentos de Fax
	Copiadoras
	Impressoras
	Monitores Cardíacos
	<i>Scanners</i>

Diferentes sistemas embarcados apresentam diferentes graus de exigência para esses requisitos. Os requisitos podem ser mandatórios, desejáveis ou opcionais (Friedrich 2009).

Sistemas embarcados em geral apresentam maiores requisitos de qualidade e confiabilidade, comparados a outros sistemas computacionais (Noergaard 2005).

2. Microprocessador Embarcado

O processador é a principal unidade funcional de um sistema embarcado, sendo responsável, primeiramente, por processar instruções e dados. Dispositivos eletrônicos contêm no mínimo um processador mestre, que atua como unidade de controle central, e podem possuir outros processadores adicionais que são controlados ou operam conjuntamente com o processador central (Noergaard 2005).

A maioria dos sistemas embarcados utiliza microcontroladores ao invés de microprocessadores. Os microcontroladores usualmente contêm a maioria dos periféricos e da memória do sistema integrados num mesmo *chip*, enquanto os microprocessadores praticamente não possuem periféricos (Berger 2002) (Noergaard 2005).

Sistemas computacionais, como computadores pessoais, são desenvolvidos utilizando processadores de propósito geral, também conhecidos como universais. Esses processadores apresentam *design* complexo provendo diferentes características e uma vasta gama de funcionalidades, adequando-se a uma grande variedade de aplicações. Essa complexidade reflete em altos custos de fabricação (Li 2003).

Sistemas embarcados representam plataformas geralmente desenvolvidas para tarefas específicas. Esta especificação faz com os sistemas desenvolvidos possam ser altamente otimizados uma vez que os limites de tarefas a serem executadas é bem definido (Berger 2002).

Assim, sistemas embarcados vêm crescentemente sendo desenvolvidos utilizando processadores embarcados ao invés processadores universais. Esses processadores possuem propósitos específicos, sendo desenvolvidos para classes especiais de aplicação. São desenvolvidos com foco em tamanho, desempenho, consumo de energia ou custo (Berger 2002).

Atualmente existem centenas de microcontroladores embarcados disponíveis, e não apenas um deles domina o desenvolvimento de sistemas embarcados (Berger 2002). Segundo dados de 2000 da Intel, 98% dos processadores fabricados no mundo são utilizados em sistemas automotivos, robótica e outros sistemas embarcados (Barbiero e Hexsel 2006).

O mercado de microcontroladores embarcados cresce significativamente puxado principalmente pela indústria automobilística. Entre os principais fabricantes estão

Renesas, Freescale, NEC, Infineon, Fujitsu e Microchip, que combinadas possuem 60% do mercado (Mokhoff 2010).

Processadores embarcados comumente utilizados suportam processamento em 4, 8, 16, 32 e 64 bits. Alguns processadores podem tratar grandes quantidades de dados e acessar grandes espaços de memória em apenas uma instrução, como arquiteturas de 128 bits. No entanto, são raramente usados no desenvolvimento de sistemas embarcados (Noergaard 2005). A Tabela 2 mostra alguns exemplos de modelos de processadores embarcados.

Tabela 2 - Exemplos de arquiteturas de processadores embarcados para diferentes números de bits.
Fonte: (Noergaard 2005)

Número de Bits	Modelos
4	Intel 4004
8	Mitsubishi M37273, 8051, 68HCS08, STM8
16	ST ST10, TI MSP430, Intel 8086/286
32	68K, PowerPC, ARM, x86 (386+), MIPS 32

A Figura 1, utilizando dados de 2004, mostra a segmentação do mercado de microcontroladores segundo o volume de vendas. Apesar do melhor desempenho das plataformas de 16 e 32 bits, os micros de 8 bits continuam a responder por uma porcentagem substancial do volume de vendas de microcontroladores no mundo (Shandle 2004).

Figura 1 – Segmentação do mercado de microcontroladores - Dados de 2004. Fonte: (Shandle 2004).

De acordo com Sperling (Sperling 2010), o número total de microcontroladores de 8-bits fornecidos ou com fornecimento previsto continua crescendo. Segundo Sperling (Sperling 2010), a ST Microelectronics estima o mercado de 8-bits em 5 bilhões de dólares, com um crescimento de 48% no número de unidades fornecidas entre 2007 e 2011. A Freescale estima o mercado entre US\$4,5 bilhões e US\$4,7 bilhões com

crescimento relativamente estável até 2011. E a Atmel estima o mercado de 8 e 16 bits combinados em torno de US\$11 bilhões.

Esses números justificam os novos lançamentos de famílias de microprocessadores de 8-bits pelos grandes fabricantes mundiais (Sperling 2010).

Segundo Bannatyne (Bannatyne 2009), há uma razão simples para o mercado de microcontroladores de 8-bits continuar crescendo: o custo.

Bannatyne (Bannatyne 2009) afirma: “Se há possibilidade de solucionar um problema utilizando 8-bits, esta solução sempre será mais barata do que resolver o mesmo problema usando 32-bits”.

Esta visão é a mesma de Shandle (Shandle 2004), que descreve o baixo custo dos *chips* e o desenvolvimento com poucos gastos de software como principais razões para manutenção do mercado de 8-bits.

Microcontroladores de 8-bits vêm ganhando mercado das aplicações de 4-bits como controles remotos de TVs e áudio, brinquedos e jogos. Crescem também na indústria automotiva com os sistemas distribuídos, na instrumentação médica, na linha branca e em novas iniciativas para controle eficiente de energia (Shandle 2004) (Sperling 2010).

O Japão continua sendo o maior mercado para 8-bits, porém, há grande crescimento de vendas nos mercados emergentes, como China e Índia (Shandle 2004).

Atualmente, os fabricantes de microcontroladores apresentam famílias de 8-bits com memórias *flash* que variam de 1 a 128 kbytes e memórias RAM que chegam a 8 Kbytes, além de grande variedade de periféricos e alta velocidade de processamento. De acordo com Bannatyne (Bannatyne 2009), não é exagero dizer que há uma sobreposição entre os microcontroladores mais especificados de 8-bits e os microcontroladores padrões de 32-bits.

3. Desenvolvimento de sistemas embarcados

Sistemas embarcados, assim como outros sistemas computacionais, consistem de software, hardware e do ambiente onde estão inseridos. Entretanto, sistemas embarcados apresentam restrições físicas de computação, fazendo com que a separação entre software e o componente físico (plataforma e ambiente), comumente usada na ciência da computação, não funcione para esses sistemas (Henzinger e Sifakis 2006).

Henzinger e Sifakis (Henzinger e Sifakis 2006) afirmam que o desenvolvimento de sistemas embarcados requer uma abordagem holística que integre, de maneira consistente, paradigmas do desenvolvimento de hardware, do desenvolvimento de software e da teoria de controle.

Segundo Noergaard (Noergaard 2005), existem diferentes modelos para descrever o ciclo de desenvolvimento de sistemas embarcados, a maioria destes baseados em um ou numa combinação dos seguintes modelos:

- Modelo *big-bang*: não há essencialmente nenhum plano ou processo antes ou durante o desenvolvimento do sistema.
- Modelo *code-and-fix*: os requisitos de produto estão definidos, porém não há um processo formal antes do início do desenvolvimento.
- Modelo cascata: há um processo para desenvolvimento do sistema em etapas, no qual os resultados de uma etapa são requisitos para a próxima etapa.
- Modelo em espiral: há um processo para desenvolvimento do sistema em etapas e, após o fim de várias etapas, *feedbacks* são obtidos e incorporados no processo.

As principais etapas do desenvolvimento de um sistema embarcado consistem em: definição dos requisitos, especificação, arquitetura, componentes e integração do sistema (Wolf 2005).

Nesta abordagem *top-down*, iniciada com a mais abstrata descrição do sistema e concluída com detalhes concretos, o desenvolvimento começa com os requisitos, sendo posteriormente criada uma descrição detalhada dos desejos por meio da especificação. Os detalhes internos do sistema tomam forma quando a arquitetura é desenvolvida, que confere estrutura quanto a componentes, ao sistema. Definidos os componentes, estes podem ser desenvolvidos incluindo os módulos de software e hardware. Depois de finalizados os componentes, o sistema é construído (Wolf 2005).

Alternativamente a abordagem *top-down*, há a abordagem *bottom-up*, iniciada com os componentes para construção do sistema. Esta abordagem é utilizada quando há poucas garantias dos estágios posteriores, como por exemplo, quanto tempo irá demorar o desenvolvimento uma determinada função (Wolf 2005).

A Figura 2 mostra as principais etapas do processo de desenvolvimento de sistemas embarcados. As linhas contínuas descrevem a abordagem *Top-down* enquanto as linhas tracejadas destacam a abordagem *Bottom-up*.

Figura 2 – Principais etapas do processo de desenvolvimento de sistemas embarcados. Fonte: (Wolf 2005).

Berger (Berger 2002) descreve em seu livro praticamente as mesmas etapas, destacando, porém, as etapas de teste, refinamento, manutenção e atualizações. As etapas descritas por Berger (Berger 2002) são:

- Especificação do produto
- Divisão de desenvolvimento em componentes de software e hardware
- Iteração e refinamento da divisão
- Desenvolvimento independente de hardware e software
- Integração dos componentes de software e hardware
- Teste e lançamento do produto
- Atualizações e manutenções

De acordo com Wolf (Wolf 2005), o uso de uma metodologia de desenvolvimento é importante, pois permite que o desenvolvimento seja acompanhado, garantindo a execução de todas as atividades. Além disso, a utilização de uma metodologia permite a automatização de etapas, o uso de ferramentas de auxílio ao desenvolvimento (CAD) e a melhoria na comunicação e coordenação do time de projeto.

Segundo Ganssle (Ganssle 1999), quando estamos desenvolvendo um produto, estamos balanceando requisitos de cronograma, qualidade e funcionalidades. A Figura 3 mostra essa tríade. O desequilíbrio nesses requisitos usualmente coloca o desenvolvimento do produto em risco.

Figura 3 – Balanceamento dos requisitos no desenvolvimento de embarcados. Fonte: (Ganssle 1999).

De acordo com dados de 2006, de pesquisa realizada com desenvolvedores (CMP United Business Media 2006), 52% dos projetos são entregues tardiamente, sendo o atraso em média de 4,3 meses. Segundo dados da mesma pesquisa, 61% dos desenvolvedores trabalham em mais de um projeto ao mesmo tempo e os projetos duram em média 14 meses.

A pesquisa também destaca que 76% dos desenvolvimentos apresentam características de tempo real, 59% apresentam capacidade de comunicação em rede e 47% dos projetos são desenvolvidos com foco na robustez e resistência ao ambiente de operação (CMP United Business Media 2006).

Capítulo 4

Software Embarcado

1. Introdução

Atualmente, o software é um componente chave no desenvolvimento de sistemas embarcados, sendo raro encontrar produtos controlados apenas por hardware (Charrette 2005).

O tamanho dos softwares em produtos eletrônicos cresce vertiginosamente. Os automóveis atuais contêm cerca de milhões de linhas de código e o número de linhas de código em um celular que em 2005, era de aproximadamente 2 milhões, tinha previsão de aumentar em 10 vezes até 2010 (Charrette 2005) (Genuchten 2007).

Uma pesquisa realizada com desenvolvedores em 2006 (CMP United Business Media 2006) comprova a grande importância do software no projeto de sistemas embarcados. Segundo os dados, 65% dos projetos consomem mais tempo no desenvolvimento de software do que no desenvolvimento de Hardware.

De acordo com a mesma pesquisa, o gasto no desenvolvimento de software é maior que em hardware em 41% dos projetos. Além disso, 52% dos projetos têm um número maior de pessoas trabalhando no desenvolvimento de software do que no desenvolvimento de hardware (CMP United Business Media 2006).

A computação embarcada não consiste apenas do gerenciamento de *timers* e contadores. Sistemas embarcados envolvem muito mais recursos de computação, e com natureza diferente da computação tradicional (Wolf 2007). Segundo Lee (Lee 2002), a principal função do software embarcado não é a transformação de dados, e sim a interação com o ambiente físico.

Ainda segundo Lee (Lee 2002), o software embarcado não é apenas o software em um pequeno computador, apresentando características especiais como:

- Precisão temporal: softwares embarcados precisam interagir com o ambiente com restrições de tempo.

- Concorrência: o software embarcado deve reagir simultaneamente a estímulos de redes de comunicação e de uma variedade de sensores, e ao mesmo tempo, controlar atuadores.
- Operação contínua: o software embarcado não deve interromper a operação ou ficar bloqueado aguardando eventos que não ocorreram.
- Heterogeneidade: o software embarcado combina diferentes estilos de implementação, linguagens de programação e deve conviver com eventos periódicos e esporádicos.
- Reatividade: reagem continuamente ao ambiente na velocidade do ambiente.

Em "*The Good News and the Bad News*", Wolf (Wolf 2007) destaca mais algumas características do software embarcado como:

- Restrições de tempo real: comportamento de tempo real define a computação embarcada.
- Baixo consumo de energia: a maioria dos sistemas opera dentro de restrições de energia.
- Balanço entre hardware e software: desenvolvedores de embarcados devem sempre ponderar as vantagens e desvantagens da implementação de uma função em software ou hardware.

Além disso, Knight (Knight 2002) ressalta que o software exerce papel importante na confiabilidade, na segurança, na integridade, na manutenção e na confidencialidade do sistema embarcado.

A maioria dos projetos de software em sistemas embarcados é desenvolvida utilizando linguagem C e C++. Em 2006, segundo dados de pesquisa realizada com desenvolvedores (CMP United Business Media 2006), a linguagem C era utilizada em 51% dos projetos e C++ em 30% dos projetos. A Tabela 3, detalha o uso das diferentes linguagens nos projetos.

Outra característica dos projetos de software embarcado é o reuso de algum tipo de código, evidenciado na Figura 4.

Existem diferentes arquiteturas para o desenvolvimento de software embarcado. As características do sistema determinam a escolha da arquitetura mais apropriada. Entre as principais arquiteturas podemos destacar: *Pooled Loop*, *Foreground/Background* e os sistemas operacionais embarcados.

As seções posteriores trazem detalhes dessas arquiteturas, bem como de sistemas operacionais, importantes para a compreensão do presente trabalho.

Tabela 3 – Linguagens de programação utilizada no desenvolvimento de projetos embarcados. Dados de 2006. Fonte: (CMP United Business Media 2006)

Linguagem	Uso (%)
C	51
C++	30
Assembly	8
Java	3
BASIC	1
UML, MatLab ou outra linguagem de modelagem	3
LabView	2
Outras	5

Figura 4 - Reuso de código nos projetos de software embarcado. Dados de 2006. Fonte: (CMP United Business Media 2006)

2. Arquitetura *Pooled Loop*

A arquitetura *Pooled Loop* é a mais simples existente. Esta arquitetura consiste de um único *loop* principal executado repetidamente, que simplesmente verifica entradas e

saídas ou *flags* que indicam a ocorrência de eventos, além de executar os serviços necessários (Simon 1999) (Singh 2002).

Nesta arquitetura não há interrupções, compartilhamento de dados e preocupações com latência (Simon 1999).

Segundo Simon (Simon 1999), esta arquitetura apresenta como vantagem apenas a simplicidade, frente a diversas desvantagens que a tornam inviável para vários sistemas. Algumas das desvantagens dessa arquitetura são:

- O tempo de resposta para qualquer processo é o tempo gasto, no pior caso, para que o microprocessador execute todo o *loop* principal.
- O sistema não apresenta boa resposta na presença de processos longos que deterioram a resposta de outras tarefas.
- Fragilidade da arquitetura: a adição de novas tarefas ou de novos requisitos pode degradar sensivelmente o desempenho do sistema.

Esta arquitetura é viável apenas para sistemas muito simples, como por exemplo, multímetros digitais (Simon 1999).

3. Arquitetura *Foreground/Background*

A arquitetura *Foreground/Background* ou *Superloop* é a solução mais comum utilizada para aplicações embarcadas, principalmente em sistemas de pequena complexidade (Labrosse 2002) (Singh 2002).

Essas aplicações consistem de um conjunto de rotinas de tratamento de interrupções ou processos de tempo real chamados de *Foreground* e uma coleção de chamadas de módulos (funções) em *loop* infinito para executar as operações desejadas, chamada *Background* (Labrosse 2002) (Singh 2002).

O *Foreground* também é chamado de nível de interrupção e o *Background* de nível de tarefas. Operações críticas devem ser executadas no nível de interrupção para garantir que atendam as restrições de tempo. Isso faz com que as ISR's (rotinas de tratamento de interrupção) consumam mais tempo do que o recomendado (Labrosse 2002).

Efetivamente, todos os processamentos executados em ISR's apresentam maior prioridade que tarefas do *loop* principal (Simon 1999).

As informações geradas por interrupções para o nível de tarefas não são processadas até que o *background* retorne a execução da tarefa, o que no pior caso, representa a execução de todas as tarefas do *loop* do *background* (Labrosse 2002). A Figura 5 exemplifica o funcionamento dessa arquitetura.

Esta arquitetura fornece maior controle sobre as prioridades de tarefas. No entanto, devido à variação no tempo de execução das tarefas, o período para execução de cada porção de código do *loop* principal é não-determinística. Além disso, mudanças no código afetam a temporização do *loop* (Labrosse 2002) (Simon 1999).

A maioria das aplicações de alto volume que utilizam microcontroladores (microondas, lavadoras, refrigeradores, brinquedos, telefones, etc.) é desenvolvida utilizando sistemas *Background/Foreground* (Labrosse 2002).

Figura 5 - Arquitetura Foreground/Background. Fonte: (Labrosse 2002).

4. Sistemas Operacionais

4.1.Introdução

Os sistemas operacionais surgiram com o desenvolvimento da computação, como base para sistemas computacionais, promovendo desenvolvimentos mais modulares e a abstração entre hardware e código de aplicação (Li 2003).

Segundo Noergaard (Noergaard 2005), sistemas operacionais (OS's) são um conjunto de bibliotecas de software que atendem dois propósitos principais:

- Prover uma camada de abstração, tornando o software mais independente do hardware, facilitando o desenvolvimento do software de aplicação.
- Gerenciar os vários recursos de software e hardware garantindo confiabilidade e eficiência na operação do sistema.

O sistema operacional, como descrito por Cooling (Cooling 2004), junto com o hardware associado suporta a estrutura de tarefas dos programas, o paralelismo (concorrência de operações), o uso de recursos aleatoriamente e em tempos pré-determinados, a implementação de tarefas sem conhecimento do hardware e abstração de tarefas que permite que elas sejam implementadas como unidades independentes.

Os sistemas operacionais dividem-se em dois grupos principais: os de propósito geral, chamados de GPOS (*General-purpose Operating System*) e os de tempo real, conhecidos como RTOS (*Real-Time Operating System*) (Li 2003).

Os RTOS's e GPOS's apresentam semelhanças como: abstração do software de aplicação com relação ao hardware, gerenciamento de recursos de software e hardware, fornecimento de serviços de baixo nível para aplicação e alguns níveis de multitarefas (Li 2003).

Atualmente, GPOS's são utilizados principalmente em computação de propósito geral e executam em sistemas como PCs, *mainframes* e *Workstations*, enquanto RTOS's se adaptam melhor a sistemas embarcados com restrições de tempo real (Li 2003).

RTOS's apresentam melhor confiabilidade para aplicações embarcadas: escalonabilidade para atender requisitos de aplicação, baixos requisitos de memória,

políticas de escalonamento mais adaptadas a sistemas embarcados de tempo real, melhor portabilidade para diferentes plataformas e melhor desempenho (Li 2003).

De acordo com Ortiz Jr. (Ortiz Jr. 2001), o crescimento da demanda por sistemas operacionais embarcados torna o mercado atrativo para empresas consolidadas e novos fornecedores, o que deixa o mercado amplamente aberto, diferentemente do mercado de sistemas operacionais para PC's.

Ortiz Jr. (Ortiz Jr. 2001) destaca também opiniões de observadores do mercado, que afirmam que esta competição entre diferentes OS's é saudável e que, com o tempo, o mercado deve convergir para um número restrito de soluções. Os observadores também expressam preocupação devido à necessidade dos desenvolvedores serem obrigados a interagir com diferentes sistemas, bem como afirmam que a existência de inúmeros fornecedores é um sinal de imaturidade do mercado.

4.2.Real-Time Operating Systems - RTOS

Um RTOS é um programa que escalona a execução de maneira temporal, gerencia recursos e provê uma base consistente para o desenvolvimento de códigos de aplicação (Li 2003). A Figura 6 mostra a composição típica de um RTOS.

O uso de RTOS em aplicações embarcadas expandiu muito devido ao notável aumento de tamanho e complexidade dos softwares embarcados.

Segundo Cooling (Cooling 2004), o RTOS surge como um mecanismo de controle de execução e do comportamento do software (aplicação).

Para Vetromille et al. (Vetromille, et al. 2006), o sistema operacional, sem hesitação, é o software mais importante de todo sistema de programas em um sistema embarcado de tempo-real. Conforme o sistema, um RTOS que gerencie tanto tarefas *hard* quanto *soft real-time* é extremamente necessário para garantir a eficiência do sistema.

Ganssle (Ganssle 2006) afirma que OS's bem construídos simplificam o desenvolvimento quando múltiplas atividades executam concorrentemente, além de eliminar problemas criados por variáveis globais através de estruturas de sincronização (*mailboxes* e listas de mensagens). Contudo, faz ressalvas ao uso desses sistemas que, segundo ele, podem trazer custos e perigos. Afirma também que a maioria dos artigos

científicos gera a impressão de que todos os sistemas embarcados necessitam um RTOS.

Figura 6 - Composição do RTOS e abstração entre Hardware e Software. Fonte: (Cadeño e Laplante 2007).

Ganssle (Ganssle 2006) levanta problemas importantes dos RTOS's embarcados como falta de suporte, alto consumo de memória e falta de certificação. Também ressalta que o RTOS's deve ser usado apenas quando há benefícios tangíveis.

O uso de RTOS's pode trazer grandes benefícios como os ressaltados por Tan e Anh (Tan e Anh 2009):

- Otimização do desenvolvimento de software: o uso de RTOS propicia ciclos de desenvolvimentos menores e redução do *time-to-market*, pois permite um melhor gerenciamento de complexidade e facilita a divisão de trabalho entre desenvolvedores.
- Melhor sincronização e robustez: as tarefas podem trocar mensagens e sincronizar-se sem corrupção do sistema. Normalmente, em microssistemas o uso de variáveis globais para sincronização, acessadas por diferentes processos, aumenta a chance de corrupção de dados durante a execução do programa.

- Gerenciamento de recursos: os RTOS's apresentam API's (Interface para programas de Aplicação) para gerenciar recursos do sistema, provendo uma camada de abstração para os programadores permitindo códigos mais limpos.
- Gerenciamento de tempo: com funções de gerenciamento de tempo os RTOS's permitem aos desenvolvedores o controle do tempo, adiamento e disparo de tarefas sem conhecer profundamente o funcionamento do hardware.

Os problemas ressaltados por Ganssle (Ganssle 2006) são um grande desconforto hoje para os desenvolvedores, sendo muito importante a seleção correta da arquitetura de software e do RTOS a ser usado.

Como descrito por Tan e Anh (Tan e Anh 2009), há inúmeras vantagens no uso de um sistema operacional embarcado, principalmente com o aumento de complexidade do software, a crescente pressão por *time-to-market*, custo e qualidade na indústria, sendo imprescindível a escolha cautelosa do RTOS a ser usado. Como ressaltado por Ralph Moore, presidente da Micro Digital Inc., fabricante do SMX RTOS, em entrevista (Moore 2008), além da avaliação dos requisitos dos RTOS através de uma lista de desejos, é recomendável que o desenvolvedor escreva parte de seu código de aplicação e o teste para avaliar processadores, ferramentas e RTOS.

Segundo Tan e Anh (Tan e Anh 2009), os sistemas operacionais de tempo real são amplamente utilizados em microcontroladores de grande porte (32-bits), porém, seu uso vem se expandindo a dispositivos menores (16-bits e 8-bits) devido às inúmeras vantagens.

O RTOS é composto por um *kernel* (núcleo) que prove escalonamento, gerenciamento de recursos e lógicas mínimas, sistemas de arquivos, protocolos de comunicação, além de componentes de aplicações particulares (Li 2003).

Cadeño e Laplante (Cadeño e Laplante 2007) descrevem que atualmente existem três categorias principais de RTOS's: proprietários, RTOS's de pesquisa e extensões de tempo real dos sistemas operacionais comerciais como o UNIX.

Os RTOS's proprietários apresentam duas variedades: os comerciais e os particulares. Ambas as variedades são usadas para sistemas embarcados de pequeno porte em que execução rápida e com grande previsibilidade deve ser garantida. Além disso, ambas as variedades são versões otimizadas e com recursos reduzidos de sistemas operacionais que utilizam compartilhamento de tempo (Cadeño e Laplante 2007).

Os RTOS's particulares são altamente especializados para a aplicação, porém, vêm caindo em desuso, devido ao alto custo para desenvolvimento e manutenção, bem como pela crescente qualidade das ofertas comerciais (Cadeño e Laplante 2007).

As extensões de tempo real de produtos comerciais são RTOS's como, por exemplo, a extensão do UNIX para RT_UNIX, ou POSIX para o RT-POSIX, ou MACH para RT-MACH, ou CHORUS para a versão de tempo real. Esses RTOS's são geralmente mais lentos e menos previsíveis que os sistemas proprietários, mas apresentam maior funcionalidade e melhores ambientes de desenvolvimento. Outra vantagem significativa é que esses sistemas são baseados em interfaces familiares que facilitam portabilidade (Cadeño e Laplante 2007).

RTOS's de pesquisa são sistemas operacionais desenvolvidos nos meios acadêmicos para estudo e avaliação de novos conceitos.

Além dessas três categorias, é importante ressaltar os RTOS's de código aberto (*open-source*). Segundo Ortiz Jr. (Ortiz Jr. 2001), esses RTOS's, assim como em outras áreas de desenvolvimento de software, apresentam papel importante, uma vez que praticamente não apresentam custos e os usuários têm acesso ao código e podem customizá-lo.

Dados de pesquisa realizada com desenvolvedores em 2006 (CMP United Business Media 2006), mostram que as opções comerciais dominam o mercado, seguido pelas soluções proprietárias. Entretanto, há grande previsão para crescimento do uso de opções de código aberto. Figura 7 e a Figura 8 mostram esses dados.

Figura 7 – Tipo de OS utilizado no projeto atual. Fonte: (CMP United Business Media 2006)

Figura 8 - Tipo de OS previsto para o próximo projeto. Fonte: (CMP United Business Media 2006)

4.3. Kernel

Segundo Noergaard (Noergaard 2005), sistemas operacionais embarcados variam muito em relação a sua composição. Entretanto, todos são compostos por, no mínimo, um *kernel*. O *kernel* é o componente que contém a funcionalidade principal do OS, incluindo:

- Gerenciamento de processos: gerenciamento de outros softwares que compõem o sistema embarcado.
- Gerenciamento de memória: gerencia o acesso e alocação do espaço de memória compartilhado entre os diferentes processos.
- Gerenciamento de entradas/saídas: gerencia o acesso e a alocação dos dispositivos de entrada e saída compartilhados pelos vários processos.

Os *kernel* podem ser preemptivos ou não-preemptivos. *Kernels* não preemptivos exigem que cada tarefa execute e explicitamente libere o controle da CPU. Para manter a ilusão de concorrência, este processo precisa ser executado frequentemente (Labrosse 2002).

Kernels preemptivos são usados quando a resposta do sistema é importante. Nestes *kernels*, a tarefa de maior prioridade, pronta para executar, recebe o controle da CPU.

Escalonadores não preemptivos também são chamados de sistema multitarefas cooperativos, uma vez que as tarefas cooperam uma com as outras para dividir a CPU (Labrosse 2002).

Em *kernels* não preemptivos os eventos assíncronos continuam sendo tratados pela ISR. A ISR também pode liberar uma tarefa de alta prioridade para execução. Contudo, sempre retorna para tarefa interrompida. A nova tarefa liberada ganha o controle da CPU apenas quando a tarefa em execução dá a permissão (Labrosse 2002). A Figura 9 exemplifica a operação deste *kernel*.

Em *kernels* não preemptivos, a latência de interrupção geralmente é pequena. Esses *kernels* também podem utilizar funções reentrantes, desde que essas não realizem a liberação da CPU. O tempo de resposta de tarefas no *kernel* não preemptivo é menor que no *foreground/background*, uma vez que o tempo de resposta é dado pelo tempo da tarefa mais longa. Outra vantagem desses *kernels* é que nem todos os recursos precisam ser protegidos por semáforos uma vez que as tarefas não são preemptadas (Labrosse 2002).

Figura 9 – *Kernel* Não-preemptivo. Fonte: (Labrosse 2002).

Nesses *kernels* não é possível prever quando uma tarefa de alta prioridade irá ser executada, podendo esta esperar um longo período até que a tarefa em execução libere a CPU.

Nos *kernels* preemptivos, a execução da tarefa de maior prioridade é determinística, sendo possível determinar o momento em que ela receberá o controle da CPU. Nesta arquitetura o tempo de resposta das tarefas é minimizado (Labrosse 2002).

Nesses sistemas, funções reentrantes não devem ser usadas, a menos que o acesso exclusivo a função seja garantida (Labrosse 2002).

A Figura 10 mostra a operação dos kernels preemptivos.

Figura 10 - *Kernel* preemptivo. Fonte: (Labrosse 2002).

O *kernel*, componente principal do sistema operacional, é composto pelos seguintes componentes: escalonador, objetos e serviços (Li 2003).

4.3.1. Escalonador

O escalonador está no coração de todo *kernel*, sendo responsável pela seleção das tarefas que serão executadas e pela determinação do instante de execução, através de algoritmos de escalonamento.

A maioria dos *kernels* atuais suporta dois algoritmos de escalonamento comuns: *round-robin* e escalonamento preemptivo, baseado em prioridade (Li 2003).

Os fabricantes de RTOS tipicamente pré-definem o algoritmo, mas, em alguns casos, desenvolvedores podem criar e definir os próprios algoritmos (Li 2003).

Segundo Simon (Simon 1999), a maioria dos algoritmos de escalonamento de RTOS's são simplórios na escolha das tarefas que irão executar, diferentemente dos escalonadores de sistemas como Windows e Unix. De acordo com Simon (Simon 1999), eles apenas executam a tarefa com maior prioridade, assumindo que o desenvolvedor conhece o sistema e definiu corretamente as prioridades.

4.3.1.1. *Round-robin* e *Weighted Round Robin*.

A abordagem *round-robin* é comumente utilizada para escalonar aplicações de tempo-compartilhado (Liu 2000).

Neste algoritmo, os processos são colocados em uma fila FIFO (*First-in-first-out*) quando estão prontos para serem executados. Cada processo recebe uma fatia de tempo igual – unidade básica alocada para execução. O conjunto das fatias de tempo é chamado de *round* (Liu 2000).

O processo, no topo da fila, executa durante sua fatia de tempo e, caso o processo não finalize a sua execução durante sua fatia de tempo, é preemptado, liberando o processador para o próximo processo. O processo é então colocado no fim da fila para esperar o próximo *round* (Liu 2000).

O algoritmo *round-robin* também é conhecido como algoritmo de compartilhamento de processador (Liu 2000).

O algoritmo *weighted Round-Robin*, segue o mesmo procedimento do algoritmo *round-robin*, no entanto, o processador não é dividido igualmente entre os processos

que recebem diferentes pesos (*weights*). O *round* é igual à soma dos pesos, que podem ser usados para acelerar ou desacelerar o progresso dos processos (Liu 2000).

4.3.1.2. Algoritmos orientados à prioridade.

Algoritmos orientados à prioridade não utilizam escalas pré-computadas, ao invés disso, quando a preempção é permitida, as decisões de escalonamento são tomadas sempre que um processo é liberado ou concluído. No momento da tomada de decisão, o escalonador atualiza a lista de processos prontos para execução e escalona e executa o processo no topo da lista (Liu 2000).

Na maioria dos sistemas práticos, as prioridades atribuídas aos processos são fixas. Uma vez atribuída, a prioridade dos processos não muda (Liu 2000).

Entre os algoritmos orientados à prioridade destacam-se:

- *Rate-Monotonic (RM)*

Este algoritmo é um dos mais conhecidos com prioridade fixa, atribuindo prioridades às tarefas baseado nos períodos. Quanto menor o período maior a prioridade (Liu 2000).

O *rate* (taxa) dos processos liberados é o inverso do período da tarefa, ou seja, quanto maior o *rate* maior a prioridade (Liu 2000).

- *Deadline-Monotonic (DM)*

Este algoritmo de prioridade fixa atribui prioridade às tarefas de acordo com seus *deadlines* relativos: quanto menor o *deadline*, maior a prioridade. Sua operação é similar ao *rate-monotonic*, apresentando operação idêntica quando o *deadline* é proporcional ao período da tarefa (Liu 2000).

Quando os *deadlines* são arbitrários, o algoritmo DM apresenta resultados melhores, uma vez que, pode produzir escalas factíveis quando o RM falha. O algoritmo RM sempre falha quando o DM falha (Liu 2000).

- *Early Deadline First (EDF)*

Este algoritmo de prioridade dinâmica no nível das tarefas e prioridade fixa ao nível dos processos atribui prioridade individualmente para cada processo

de acordo com seus *deadlines* absolutos. Quanto mais próximo o *deadline*, maior a prioridade (Liu 2000).

Este algoritmo sempre produz escalas realizáveis quando usado para escalonar processos em um processador com preempção habilitada e sem disputa por recursos entre os processos (Liu 2000).

- *Least Slack Time First (LST)*

O LST é um algoritmo de prioridade dinâmica ao nível dos processos, atribuindo prioridade baseado no tempo ocioso (*slack*), ou seja, na diferença entre o *deadline* e o tempo de execução. Quanto menor o *slack*, maior a prioridade (Liu 2000).

4.3.2. Objetos

Objetos são construções especiais usados como blocos para construção de aplicações pelos desenvolvedores de sistemas embarcados de tempo real. Incluem, por exemplo, tarefas, semáforos e listas de mensagens (Li 2003).

4.3.2.1. Tarefas

Uma tarefa é um *thread* independente que compete com outras tarefas concorrentes por tempo de execução (Li 2003).

Segundo Simon (Simon 1999), a tarefa é o bloco básico de construção de um software escrito utilizando um RTOS.

Uma tarefa é definida por seus parâmetros e estrutura de dados: nome, identificação única (ID), nível de prioridade (escalonamento por prioridade), bloco de controle (TCB), pilha (*stack*) e rotina (Li 2003).

A maioria dos RTOS's não apresenta número máximo de tarefas (Simon 1999).

As tarefas apresentam certo número de estados, pelos quais transitam durante a execução do sistema. Segundo Li (Li 2003), diferentes *Kernels* podem definir diferentes grupos de estados. Contudo, geralmente três estados principais são utilizados:

- Estado *ready*: estado em que a tarefa está pronta para execução, porém não é iniciada, pois uma tarefa de maior prioridade está executando.
- Estado *running*: estado de execução da tarefa. Estado da tarefa com maior prioridade.
- Estado *blocked*: estado em que a tarefa bloqueia sua execução no caso de ter requisitado um recurso não disponível, ter adiado sua execução ou estar aguardando a ocorrência de algum evento.

A Figura 11 mostra os estados principais das tarefas e suas transições.

Figura 11 – Estado das Tarefas e transição entre eles. Fonte: (Li 2003)

Segundo Li (Li 2003), tipicamente as tarefas são estruturadas em duas formas:

- *Run-to-completion*: a tarefa executa uma única vez, quando o sistema é ligado. Este formato é muito utilizado na inicialização do sistema.
- *Endless loop*: a tarefa tipicamente executa em vários momentos enquanto o sistema está ligado. Compõe a maior parte das tarefas responsáveis pela aplicação, tratando entradas e saídas.

4.3.2.2. Semáforos

Semáforo é um objeto do *kernel* que uma ou mais tarefas podem adquirir e liberar com o propósito de sincronização ou exclusão mútua. Um semáforo é como uma chave que permite uma tarefa realizar alguma operação ou acessar um recurso (Li 2003).

Um semáforo tem associado a ele um bloco de controle (SCB – *Semaphore Control Block*), um único ID, um valor (binário ou contador) e uma lista de tarefas em espera (Li 2003).

Os principais tipos de semáforos descritos por Li (Li 2003) são:

- Binário

Semáforo que pode assumir os valores 0 (não disponível) e 1 (disponível). Na inicialização, ambos os valores podem ser assumidos. A Figura 12 mostra a máquina de estado relativa à operação deste semáforo.

Figura 12 – Máquina de Estados do Semáforo Binário. Fonte: (Li 2003).

- Contadores

Semáforo que utiliza um contador para permitir que a aquisição e a liberação sejam realizadas múltiplas vezes. O contador é inicializado no momento da criação. Quando o valor do contador atinge 0, o semáforo torna-se indisponível. Caso contrário pode ser adquirido. Uma vez em 0, o contador precisa esperar um semáforo ser liberado para se tornar disponível. A Figura 13 mostra a máquina de estado relativa à operação deste semáforo.

Figura 13 – Máquina de Estados do Semáforo Contador. Fonte: (Li 2003).

- Exclusão mútua (*mutex*)

Consiste em um semáforo binário com protocolos de exclusão mútua tais como propriedade, acesso recursivo, exclusão segura de tarefas dentre outros.

Um semáforo de exclusão mútua apresenta estados 0 (bloqueado) e 1 (desbloqueado). Na criação, o semáforo é iniciado em estado desbloqueado. A partir do momento que uma tarefa o adquire (ou bloqueia) se torna bloqueado, até ser liberado pela tarefa ou desbloqueado. A Figura 14 mostra a máquina de estado relativa à operação deste semáforo.

Figura 14 – Máquina de Estado do Semáforo de Exclusão Mútua (Mutex). Fonte: (Li 2003).

Semáforos são úteis para sincronização, execução de tarefas múltiplas ou coordenação de acesso a recursos compartilhados (Li 2003).

4.3.2.3. Fila de Mensagens

Fila de mensagem é um objeto semelhante a um *buffer* através do qual, tarefas e interrupções (ISR's) enviam e recebem mensagens para comunicar e sincronizar dados. A fila de mensagens armazena temporariamente as mensagens do remetente até que o destinatário esteja pronto para lê-las (Li 2003).

Uma fila de mensagens tem associada a ela um bloco de controle, um nome, um único ID, *buffers* de memória, tamanho da fila, tamanho máximo das mensagens e uma lista de tarefas em espera (Li 2003).

As mensagens podem conter diferentes informações como dados de sensores, imagens a serem projetadas em um *display* ou um evento de teclado (Li 2003).

4.3.2.4. Pipes

Pipes são objetos do *kernel* que provêm troca de dados não estruturados e facilitam a sincronização entre tarefas. Diferentemente de uma fila de mensagens, o *pipe* não prioriza mensagens e não armazena múltiplas mensagens, além dos dados armazenados serem não estruturados (conjunto de bytes) (Li 2003).

Pipes podem ser dinamicamente criados e destruídos. Quando criado, tem associado a ele um bloco de controle que contém o *buffer* de dados utilizado e seu tamanho, o contador de bytes e indicadores de posição de entrada e saída. O *pipe* também tem associado a ele duas listas de espera uma para tarefas que tentam escrever quando o *pipe* está cheio e outra para tarefas que tentam ler quando o *pipe* está vazio (Li 2003).

O *pipe* é comumente utilizado para transferência de dados entre tarefas ou entre interrupções e tarefas e também para sincronização de tarefas. Os dados inseridos no *pipe* são lidos na ordem de chegada (FIFO) (Li 2003).

4.3.2.5. *Registrador de Eventos*

Os registradores de eventos consistem num conjunto de *flags* binárias, que podem ser ligadas ou desligadas para sinalizar a uma tarefa a presença de um evento particular que pode controlar sua execução (Li 2003).

Quando suportado pelo RTOS, o registrador de eventos tem associado a ele um bloco de controle que é parte integrante do bloco de controle da tarefa. Este bloco é composto pelo registro de eventos desejados, pelo registro de eventos recebidos, o tempo máximo para recebimento dos eventos e a condição de notificação que pode ser usada para combinar as *flags* de eventos (Li 2003).

Os registradores de eventos não possuem dados associados a eles, sendo um mecanismo ineficiente se usado além da simples sincronizações de atividades (Li 2003).

4.3.3. *Serviços*

São operações executadas sobre os objetos ou operações gerais, como temporização, tratamento de interrupção e gerenciamento de recursos (Li 2003).

A maioria dos *kernels* fornece serviços que ajudam desenvolvedores a criar aplicações de tempo real. Os serviços incluem operações nos objetos, gerenciamento de tempo, tratamento de interrupção, dispositivos de E/S (entrada/saída) e gerenciamento de memória (Li 2003).

4.4. Arquiteturas de desenvolvimento de sistemas operacionais

Segundo Noergaard (Noergaard 2005), a maioria dos sistemas operacionais são tipicamente baseados em três modelos de desenvolvimento: monolítico, camadas (*layered*) ou *microkernel*. Em geral, esses modelos diferem quanto ao desenvolvimento do *kernel*, bem como nos componentes incorporados no OS como *drivers* e serviços.

4.4.1. OS monolítico

Nos sistemas operacionais monolíticos, tipicamente serviços e *drivers* são integrados ao sistema operacional, formando um único arquivo executável (Noergaard 2005).

Os *kernels* monolíticos são tipicamente mais difíceis de serem escalonados para pequenas aplicações, modificados e depurados em relação aos demais modelos devido a sua natureza grande, integrada e dependente (Noergaard 2005).

Para melhorar estas características, muitos OSs vêm sendo construídos dividindo as funcionalidades em módulos que são integrados em um único arquivo executável.

Devido à maior integração, esses *kernels* em geral são mais velozes, apresentando menores tempos em comunicação e no chaveamento de componentes.

Noergaard (Noergaard 2005), cita como exemplos deste tipo de OS o Linux embarcado, o Jbed RTOS e o PDOS.

4.4.2. OS em camadas

Neste modelo, o OS é dividido em camadas hierárquicas em que as camadas superiores são dependentes das funcionalidades implementadas pelas camadas inferiores (Noergaard 2005).

Assim como no OS monolítico, o *kernel* inclui os *drivers* e serviços formando um único arquivo. No entanto, devido sua estruturação, esses OS's são mais fáceis de serem desenvolvidos e mantidos (Noergaard 2005).

O *overhead* criado pelas API's fornecidas pelas camadas pode impactar tanto em tamanho quanto no desempenho (Noergaard 2005).

Noergaard (Noergaard 2005), cita como exemplos deste tipo de OS o DOS-C, DOS/eRTOS e o VTRX.

4.4.3. *Microkernel*

Microkernels são OS's reduzidos a mínima funcionalidade, sendo compostos somente pelo gerenciamento de processos e memória. Alguns desses OS's são a apenas o gerenciamento de processos, sendo conhecidos como *nanokernels* (Noergaard 2005).

Heiser, et al. (Heiser, et al. 2007) contestam a definição de conceitos como *nanokernels*, *picokernels* e *femtokernels*, que sugerem ser menores que um *microkernel*, baseando-se na definição de Jochen Liedtke. Segundo essa definição, um componente deve fazer parte do *microkernel* apenas se sua implementação externa impedir a construção de alguma funcionalidade requerida pelo sistema.

E neste sentido, Heiser et al. (Heiser, et al. 2007) definem o *microkernel* como um substrato mínimo sobre o qual serviços do sistema operacional são implementados. E também afirmam que o *microkernel* não apresenta serviços, apenas mecanismos para o seu desenvolvimento.

O conceito de microkernel surgiu na década de 70, e apesar de certo abandono deste conceito na computação de propósito geral durante os anos 90, o conceito sobreviveu em algumas áreas específicas, principalmente na indústria de sistemas embarcados (Heiser, et al. 2007).

Microkernels são tipicamente mais escalonáveis e depuráveis que os demais uma vez que componentes podem ser dinamicamente adicionados. Além disso, são mais seguros uma vez que a maioria das funcionalidades é independente do OS e há espaços de memória separados para *kernel* (servidor) e aplicação (cliente). Também são mais fáceis de serem portados a novas arquiteturas (Noergaard 2005).

Esses OS's, entretanto, são mais lentos que as demais arquiteturas como a monolítica, devido à comunicação entre componentes internos e externos ao *microkernel* e também à adição de *overhead* devido ao chaveamento entre componentes internos e externos (Noergaard 2005).

Li (Li 2003) defende esse modelo afirmando que um bom sistema operacional de tempo real deve evitar a implementação do *kernel* como um programa monolítico gigante. Ainda segundo Li, o resultado dessa abordagem é o aumento da configurabilidade do sistema, uma vez que cada sistema embarcado requer serviços específicos respeitando suas características.

Noergaard (Noergaard 2005), cita como exemplos de *microkernels* o C Executive, VxWorks, CMX-RTX, Nucleus Plus e QNX.

4.5. Padrões

Padrões são importantes para indústria e auxiliam o ataque a problemas de qualidade, custo e cronogramas, além de trazer confiabilidade aos sistemas (Cooling 2004). Segundo Aroca (Aroca 2008), a necessidade de sistemas confiáveis levou a indústria e a comunidade científica a estabelecer normas para padronizar os sistemas operacionais de tempo real.

Entre os principais padrões estão o POSIX de tempo real, o OSEK da indústria automobilística, o APEX para sistemas aviônicos, e o μ ITRON para eletrônica de consumo (Aroca 2008).

4.5.1. RT- POSIX

O POSIX (*Portable Operating System Interface*) é um padrão de sistemas operacionais produzido pelo IEEE, com o objetivo de melhorar a portabilidade, a manutenção e o reuso de códigos de aplicação que utilizam serviços de sistemas operacionais (Cooling 2004).

A idéia base deste padrão é que o software de aplicação usa serviços do OS, acessando-os através de formas padronizadas.

Este padrão é baseado no sistema operacional Unix, com foco em portabilidade entre sistemas operacionais. Um programa escrito usando POSIX poderia ser simplesmente recompilado e executado em qualquer sistema operacional que implemente este padrão (Aroca 2008) (Bouyssounouse 2005).

Para suportar aplicações de tempo real, muitas extensões foram criadas. O IEEE *Portable Operating System Interface for Computer Enviroments*, POSIX 1003.1b que especifica várias regras relacionadas a sistemas de tempo real, sendo também conhecido

como RT- POSIX (*Real-Time* POSIX). Contudo, apenas um grande RTOS é capaz de atender todos os requisitos necessários. Assim, um subconjunto de interfaces do POSIX vem sendo usadas em situações específicas (Aroca 2008) (Bouyssounouse 2005) (Cooling 2004).

Alguns RTOS's implementados com base no POSIX são o QNX, VxWorks e o LynxOS (Aroca 2008) (Bouyssounouse 2005).

4.5.2. OSEK

OSEK/VDX é uma abreviação em alemão (*Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug*) que significa sistemas e interfaces abertas para eletrônica automotiva (Aroca 2008) (Bouyssounouse 2005).

OSEK é uma especificação de arquitetura de sistemas automotivos composto por unidades de controle distribuídas. Seus principais objetivos são a redução de custos, redução do tempo de desenvolvimento e simplificação da integração de unidades individuais do sistema (Cooling 2004).

A especificação do OSEK define um modelo de sistema operacional (OSEK OS), um modelo de comunicação (OSEK COM), um modelo de gerenciamento de rede (OSEK NM) e um modelo que define configuração (OSEL *implementation language*) (Cooling 2004).

O padrão OSEK OS prevê portabilidade entre processadores entre 8 e 32-bits, configuração facilitada e alocação estática de tarefas e memória (Aroca 2008) (Bouyssounouse 2005).

4.5.3. APEX

O padrão APEX (Avionics Application Software Standard Interface) foi criado pela ARINC (Aeronautical Radio Incorporated) e permite analisar softwares com requisitos

de tempo real críticos em segurança, certificá-los e executá-los (Aroca 2008) (Bouyssounouse 2005).

Sua especificação é bastante completa e formal, abrangendo detalhes como a forma com que sistemas computacionais devem trocar mensagens, como o sistema operacional deve tratar, separar e escalonar tarefas (Aroca 2008) (Bouyssounouse 2005).

O objetivo desse padrão é garantir a próxima geração de sistemas aviônicos baseados em segurança, robustez e técnicas padronizadas (Cooling 2004).

4.5.4. μ ITRON

A especificação μ ITRON foi desenvolvida no Japão para padronizar sistemas operacionais de tempo real utilizados na construção de sistemas embarcados (Aroca 2008) (Bouyssounouse 2005).

Takada (Takada 1997) descreve os seguintes princípios da especificação μ ITRON:

- Evitar excessiva virtualização do hardware: para isto, a especificação divide claramente em aspectos padronizados para diferentes processadores e aspectos dependentes de aplicação. Itens padronizados incluem escalonamento de tarefas, funções e nomes de chamadas do sistema, nome, sequência e significados de parâmetros, e nome e significado de erros. Aspectos que necessitam ser definidos separadamente para cada implementação, baseado nas considerações de desempenho, não são padronizados.
- Permitir otimização da aplicação: permite a alteração da especificação do *kernel* e a implementação interna para atender requisitos de aplicação específica. Para sistemas embarcados, a aplicação pode usar apenas as funções necessárias. Apenas os módulos (bibliotecas) necessários são inseridos na aplicação.
- Permitir otimização baseado no hardware: permite a alteração da especificação do *kernel*, a implementação interna para otimizar o desempenho do sistema e implementações específicas, por exemplo, para invocar o tratamento de interrupções.

- Ênfase em facilitar o treinamento dos engenheiros de software: a padronização facilita o aprendizado e treinamento dos engenheiros de software, permitindo ampla aplicação do conhecimento.
- Criação de séries de especificações e divisão em níveis: especificações separadas em séries tornando-as aplicáveis a grande variedade de hardwares e divisão das funções em níveis, baseado no grau de necessidade.
- Fornecer grande número de funções: ao invés de limitar as primitivas fornecidas, a abordagem é fornecer grande variedade de primitivas com diferentes funções, permitindo que os engenheiros de software melhorem o desempenho para um hardware ou uma aplicação específica.

A primeira especificação do ITRON foi liberada em 1987. Em 1989, versões para sistemas de pequeno porte (8-bits e 16-bits) e de grande porte (32-bits) foram lançadas (Takada 1997).

A especificação do μ ITRON foi implementada em vários microcontroladores de 8, 16 e 32-bits de fabricantes japoneses e usada em vários OSs proprietários. Esta especificação é utilizada em aplicações como eletrônica de consumo (TVs, eletrodomésticos etc.), equipamentos de escritório (Fax, impressoras etc.), equipamentos de comunicação (celulares, switches etc.) e outras aplicações como automóveis, PDAs e automação industrial. A especificação do μ ITRON pode, de fato, ser considerado um padrão na indústria japonesa (Takada 1997).

Capítulo 5

Metodologia

No presente trabalho propõe-se levantar as características e operação dos RTOS, discutir as características do desenvolvimento de sistemas embarcados e avaliar a viabilidade do uso de sistemas operacionais em sistemas de pequeno porte.

Para a realização desta proposta, o trabalho foi dividido em três etapas:

- *Etapa 1:* Levantamento de características sobre o desenvolvimento brasileiro de sistemas embarcados.
- *Etapa 2:* Estudo de caso comparando arquiteturas de software embarcado.
- *Etapa 3:* Análise das características de sistemas de pequeno porte e discussão da viabilidade do uso de RTOS com base na literatura e nos resultados obtidos sobre desenvolvimento de sistemas embarcados.

A primeira etapa visou avaliar o desenvolvimento de embarcados no país. Para isso, uma pesquisa foi realizada junto à indústria, tentando avaliar o perfil dos desenvolvedores, características dos projetos embarcados, características do software embarcado e uso de sistemas operacionais embarcados.

A pesquisa foi realizada com desenvolvedores de sistemas embarcados ligados à indústria em diversas áreas como linha branca, automação agrícola, comercial, residencial e industrial, indústria de equipamentos médicos, eletrônica de consumo e aeroespacial.

A disseminação da pesquisa foi feita eletronicamente, através de duas diferentes formas:

- *E-mail:* desenvolvedores em diferentes áreas da indústria foram contatados via correio eletrônico e convidados a responder o questionário.
- *Portal Embarcados:* a pesquisa foi disponibilizada no site www.portalembarcados.com, especializado em sistemas embarcados, onde diversos desenvolvedores associados têm acesso a artigos, fóruns de discussão e informações sobre o tema.

Os desenvolvedores, ao acessar a pesquisa, eram convidados a responder uma enquete composta de 25 questões, em sua maioria de múltipla escolha, criadas com o intuito de avaliar o perfil do desenvolvedor, características do projeto embarcado, características do desenvolvimento de software embarcado e do uso de sistemas operacionais. As perguntas e o fluxo de apresentação das questões são descritos no Apêndice I.

A fim de garantir sigilo e idoneidade à pesquisa, a enquete foi armazenada num servidor, o qual era acessado pelos participantes sem que houvesse necessidade de identificação ou fornecimento de dados pessoais, através de um *link* enviado por *e-mail* ou disponibilizado através do Portal Embarcados. Além disso, nenhuma das perguntas da enquete coletou qualquer tipo de informação que pudesse identificar o participante, sua empresa ou ramo de negócio. Isso assegurou à pesquisa foco único e exclusivamente científico, sem qualquer intuito comercial ou industrial, preservando os participantes da pesquisa.

As respostas recebidas foram armazenadas automaticamente em planilhas de dados, podendo ser exportadas ao Microsoft Excel, onde os dados foram posteriormente processados estatisticamente e adequadamente reunidos em gráficos e tabelas, facilitando a apresentação, discussão e compreensão dos resultados.

A segunda etapa visou estudar comparativamente arquiteturas de software (*Superloop* e RTOS) através de um estudo de caso. Este estudo utilizou um típico sistema embarcado constituído de cargas (atuadores), sensores, interface com usuário, com requisitos *Hard e Soft Real-Time*.

O sistema embarcado utilizado é constituído de 5 cargas elétricas disparadas por *Triacs*, 2 sensores digitais, detecção de passagem por zero da rede elétrica, 1 *encoder*, 5 *tactile switches*, 19 LEDs e uma comunicação serial.

Esse sistema é controlado usando o microcontrolador de 8-bits MCS908AW32 da Freescale, que apresenta 32 *Kbytes* de memória *flash* e 2 *Kbytes* de memória RAM, um típico microcontrolador de pequeno porte.

A escolha do microcontrolador foi feita devido à experiência prévia sobre a plataforma, disponibilidade do componente e escalonabilidade de memória (possibilidade de substituição do componente por dispositivos maiores de mesma pinagem – 48Kbytes e 60Kbytes de memória).

A Figura 15 mostra um diagrama do sistema, a Figura 16 mostra a pinagem e encapsulamento do microcontrolador e a Figura 17 mostra a estrutura interna do

microcontrolador através de um diagrama de blocos. Mais detalhes sobre este microcontrolador podem ser encontrados no *datasheet* do fabricante (Freescale 2006).

Figura 15 – Diagrama de blocos do sistema embarcado utilizado na análise comparativa entre as arquiteturas Superloop e RTOS.

Figura 16 – Encapsulamento de 44 pinos do Microcontrolador MCS908AW32 (Freescale 2006).

Figura 17 – Diagrama de blocos da estrutura interna do Microcontrolador MCS908AW32 (Freescale 2006)

O sistema é alimentado por uma fonte chaveada de 5 Volts e 120 miliampéres e multiplexa a leitura de *tactiles switches*, *encoder* e de um sensor digital com o chaveamento dos bancos de LED's. O chaveamento das cargas de característica indutiva é realizado através de TRIAC's disparados no cruzamento por zero da rede.

Nesse sistema o software é responsável pelos algoritmos de controle, processamento de sinais, rotinas de segurança e tratativas de erro. A aplicação gerencia estados do produto que se alteram com os eventos gerados pelo usuário. Além disso, a aplicação executa ciclos de atividades configurados pelo usuário. Cada atividade consiste em um padrão de atuação sobre as cargas do sistema baseado em temporizações e no estado dos sensores.

As principais tarefas da aplicação, bem como sua periodicidade, *deadline* e classificação são apresentadas na Tabela 4.

Tabela 4 – Tarefas do sistema com seus deadlines e período.

Tarefa	Período (ms)	Deadline (ms)	Tipo de Tarefa
Tarefa de Inicialização do Sistema	-	-	Soft
Gerenciamento Banco de LED's 1	1	0,2	Hard
Gerenciamento Banco de LED's 2	1	0,2	Hard
Gerenciamento Banco de LED's 3	1	0,2	Hard
Gerenciamento Banco de LED's 4	1	0,2	Hard
Gerenciamento Banco de LED's 5	1	0,2	Hard
Gerenciamento do Multiplexador	0,2	0,2	Hard
Acionamento dos TRIAC's	8,333	0,2	Hard
Gerenciamento do Sensor Digital 1	8,333	0,2	Hard
Detecção de Zero Cross	0,2	0,2	Hard
Controlador de Estado das Cargas	5	1	Hard
Algoritmo de controle de carga	5	1	Hard
Watchdog Service	5	2	Hard
Tarefa de gerenciamento de eventos do usuário	25	5	Hard
Gerenciamento do Sensor Digital 2	25	5	Hard
Gerenciamento de Estado 1	25	5	Soft
Gerenciamento de Estado 2	25	5	Soft
Gerenciamento de Estado 3	25	5	Soft
Gerenciamento de Estado 4	25	5	Soft
Gerenciamento de Estado 5	25	5	Soft
Gerenciamento de Estado 6	25	5	Soft
Gerenciamento de Estado 7	25	5	Soft
Gerenciador de Ciclo de Atividades	25	5	Hard
Executor de fases do ciclo.	25	5	Hard
Atividade 1	25	5	Hard
Atividade 2	25	5	Hard
Atividade 3	25	5	Hard
Atividade 4	25	5	Hard
Atividade 5	25	5	Hard
Atividade 6	25	5	Hard
Atividade 7	25	5	Hard
Atividade 8	25	5	Hard
Atividade 9	25	5	Hard
Gerenciador de LED's do ciclo	25	5	Hard
Temporizador de LED's	500	100	Soft

Os períodos e *deadlines* da aplicação foram determinados com base na especificação e na experiência e conhecimento do sistema. A classificação da tarefa foi atribuída

segundo a definição de Liu (Liu 2000), baseada na garantia da qualidade de serviço temporal do processo. Segundo este critério, um *hard deadline* é aquele cujo usuário necessita da validação de que o sistema sempre irá atendê-lo, enquanto um *soft deadline* é aquele que exige apenas uma demonstração de que o sistema vai atender com certa probabilidade o *deadline*. Contudo, também se considerou a criticidade da funcionalidade, segundo a qual um *hard deadline* é aquele que caso não seja atendido pode apresentar consequências desastrosas e um *soft deadline* é aquele em que resultados tardios não são desejados, porém não causam efeitos desastrosos, apenas deterioram o sistema.

Para análise comparativa, o software embarcado foi implementado utilizando a arquitetura *Superloop* e um RTOS. Os dois softwares foram analisados quanto à dificuldade de gerenciamento das tarefas, ao processamento, ao consumo de memória e ao atendimento de *deadline* de tarefas.

O estudo de caso iniciou-se com a implementação da arquitetura *Superloop*. O sistema foi desenvolvido utilizando *Superloop* com *slots* e temporização do *background*, como mostrado na Figura 18.

Figura 18 – Superloop com Slots e temporização do background

Baseado nos períodos e *deadlines* das tarefas, descritas na Tabela 4, o nível de *Background* foi temporizado executando um *loop* a cada 5 ms e dividido em 5 *slots*. A cada *loop* um *slot* era executado, cada um apresentando assim periodicidade de 25 ms.

A temporização do *Background* foi realizada utilizando a interrupção de tempo cuja periodicidade foi configurada para 200 μ s.

Definidas as configurações dos níveis de *Background* e *Foreground*, definiu-se a divisão das tarefas entre os níveis de prioridade. Uma vez que essa arquitetura apresenta apenas dois níveis de prioridade, as tarefas com *deadline* menores que 1 ms foram

alocadas no nível de interrupção, enquanto as tarefas com *deadlines* superiores a 1 ms foram alocadas no nível de *Background*.

Dentro do nível de *Background*, as tarefas foram divididas segundo sua periodicidade. As tarefas com período inferior a 25 ms eram executadas a todo *loop* enquanto as tarefas com maior período foram alocadas nos *slots*.

A Tabela 5 mostra a alocação das tarefas nos níveis de *Background* e *Foreground*.

Por conveniência à aplicação, os *slots* 1 e 5 foram utilizados apenas para garantir temporização do sistema.

A segunda parte do estudo de caso consistiu na implementação da aplicação utilizando o sistema operacional de tempo-real MicroC-OS II (Labrosse 2002). A escolha desse sistema operacional foi motivada pelos seguintes fatores:

- Disponibilidade de porte para microcontroladores que utilizam a CPU HCS08.
- Ampla documentação disponível.
- Código aberto para pesquisas com intuito acadêmico.
- Escalabilidade e Configurabilidade, permitindo acrescentar e remover componentes (Semáforos, *Flags*, *Message Queues* etc) e determinar número e prioridade das tarefas.
- Compatibilidade com o compilador COSMIC.

Para esta implementação, primeiramente a aplicação foi dividida em tarefas. Diferentemente da arquitetura *Superloop*, algumas tarefas foram subdivididas para facilitar a implementação e aumentar a modularidade. A Tabela 6 mostra as tarefas do sistema operacional e sua correlação com as tarefas da arquitetura *Superloop*.

Após a divisão da aplicação em tarefas, seguiu-se à priorização das tarefas. As tarefas de prioridade 0 a 3 foram reservadas ao sistema operacional como recomendado em Labrosse (Labrosse 2002).

Após a definição das tarefas e de suas prioridades, iniciou-se a implementação do porte e a codificação das tarefas.

Todas as tarefas foram implementadas no modelo *endless loop* (para detalhes verificar item 4.3.2.1), exceto a tarefa de inicialização do sistema implementada no modelo *run-to-completion* (para detalhes verificar item 4.3.2.1).

Tabela 5 – Divisão das tarefas entre os níveis da arquitetura *Superloop*

Número da Tarefa	Tarefa	Período (ms)	Deadline (ms)	Tipo de Tarefa	Foreground	Background
1	Tarefa de Inicialização do Sistema	-	-	Soft		X
2	Gerenciamento Banco de LED's 1	1	0,2	Hard	x	
3	Gerenciamento Banco de LED's 2	1	0,2	Hard	x	
4	Gerenciamento Banco de LED's 3	1	0,2	Hard	x	
5	Gerenciamento Banco de LED's 4	1	0,2	Hard	x	
6	Gerenciamento Banco de LED's 5	1	0,2	Hard	x	
7	Gerenciamento do Multiplexer	0,2	0,2	Hard	x	
8	Acionamento dos TRIAC's	8,333	0,2	Hard	x	
9	Gerenciamento do Sensor Digital 1	8,333	0,2	Hard	x	
10	Detecção de Zero Cross	0,2	0,2	Hard	x	
11	Controlador de Estado das Cargas	5	1	Hard		Todo Loop
12	Algoritmo de controle de carga	5	1	Hard		Todo Loop
13	Watchdog Service	5	2	Hard		Todo Loop
14	Tarefa de gerenciamento de eventos do usuário	25	5	Hard		Slot 2
15	Gerenciamento do Sensor Digital 2	25	5	Hard		Slot 4
16	Gerenciamento de Estado 1	25	5	Soft		Slot 3
17	Gerenciamento de Estado 2	25	5	Soft		Slot 3
18	Gerenciamento de Estado 3	25	5	Soft		Slot 3
19	Gerenciamento de Estado 4	25	5	Soft		Slot 3
20	Gerenciamento de Estado 5	25	5	Soft		Slot 3
21	Gerenciamento de Estado 6	25	5	Soft		Slot 3
22	Gerenciamento de Estado 7	25	5	Soft		Slot 3
23	Gerenciador de Ciclo de Atividades	25	5	Hard		Slot 3
24	Executor de fases do ciclo.	25	5	Hard		Slot 3
25	Atividade 1	25	5	Hard		Slot 3
26	Atividade 2	25	5	Hard		Slot 3
27	Atividade 3	25	5	Hard		Slot 3
28	Atividade 4	25	5	Hard		Slot 3
29	Atividade 5	25	5	Hard		Slot 3
30	Atividade 6	25	5	Hard		Slot 3
31	Atividade 7	25	5	Hard		Slot 3
32	Atividade 8	25	5	Hard		Slot 3
33	Atividade 9	25	5	Hard		Slot 3
34	Gerenciador de LED's do ciclo	25	5	Hard		Slot 3
35	Temporizador de LED's	500	100	Soft		Todo Loop*

* A tarefa foi implementada a todo loop apenas pela facilidade, não por requisitos do sistema

Tabela 6 – Correlação entre tarefas do *Superloop* e tarefas do sistema operacional de tempo-real

Tarefas Superloop	Tarefas RTOS
Tarefa de Inicialização do Sistema	Tarefa de Inicialização do Sistema
Gerenciamento Banco de LED's 1	Gerenciamento Banco de LED's 1
Gerenciamento Banco de LED's 2	Gerenciamento Banco de LED's 2
Gerenciamento Banco de LED's 3	Gerenciamento Banco de LED's 3
Gerenciamento Banco de LED's 4	Gerenciamento Banco de LED's 4
Gerenciamento Banco de LED's 5	Gerenciamento Banco de LED's 5
Gerenciamento do <i>Multiplexer</i>	Gerenciamento do <i>Multiplexer</i>
Acionamento dos TRIAC's	Acionamento de TRIAC's
	Desligamento de TRIAC's das cargas indutivas
	Desligamento de TRIAC's das cargas resistivas
Gerenciamento do Sensor Digital 1	Gerenciamento do Sensor Digital 1
Detecção de <i>Zero Cross</i>	Detecção de <i>Zero Cross</i>
Controlador de Estado das Cargas	Controlador de Estado das Cargas
Algoritmo de controle de carga	Algoritmo de controle de carga - Estado 1
	Algoritmo de controle de carga - Estado 2
	Algoritmo de controle de carga - Estado 3
	Algoritmo de controle de carga - Estado 4
	Algoritmo de Controle de carga - Controle principal
<i>Watchdog Service</i>	<i>Watchdog Service</i>
Tarefa de gerenciamento de eventos do usuário	Tarefa de gerenciamento de eventos do usuário
Gerenciamento do Sensor Digital 2	Gerenciamento do Sensor Digital 2
	Temporizador do Sensor Digital 2
Gerenciamento de Estado 1	Gerenciamento de Estado 1
Gerenciamento de Estado 2	Gerenciamento de Estado 2
Gerenciamento de Estado 3	Gerenciamento de Estado 3
Gerenciamento de Estado 4	Gerenciamento de Estado 4
Gerenciamento de Estado 5	Gerenciamento de Estado 5
Gerenciamento de Estado 6	Gerenciamento de Estado 6
Gerenciamento de Estado 7	Gerenciamento de Estado 7
Gerenciador de Ciclo de Atividades	Gerenciador de Ciclo de Atividades
Executor de fases do ciclo.	Executor de fases do ciclo.
Atividade 1	Atividade 1
Atividade 2	Atividade 2
Atividade 3	Atividade 3
Atividade 4	Atividade 4
Atividade 5	Atividade 5
Atividade 6	Atividade 6
Atividade 7	Atividade 7
Atividade 8	Atividade 8
Atividade 9	Atividade 9
Gerenciador de LED's do ciclo	Gerenciador de LED's do ciclo
Temporizador de LED's	Temporizador de LED's

Os softwares foram desenvolvidos utilizando o ambiente de desenvolvimento Eclipse, a linguagem de programação C e compilados utilizando o compilador *COSMIC C Cross Compiler for HC08/HCS08* na versão 4.6.5. Detalhes sobre a operação do compilador podem ser encontrados no manual de uso (Cosmic 2010).

A linguagem *Assembly* também foi utilizada, porém, apenas para tarefas intrinsecamente ligadas ao hardware que exigiam, por exemplo, manipulação dos registradores da CPU, ponteiro da pilha (*stack pointer*) e contador de programa.

Os softwares implementados foram testados e validados utilizando emulação através do Code Warrior for Microcontrollers 6.3 com o sistema embarcado conectado ao PC, através do gravador e emulador Cyclone PRO, fabricado pela P&E micro. Para detalhes sobre a operação do emulador, consultar o manual do usuário (PE Micro 2009).

O sistema de testes para o estudo de caso é mostrado na Figura 19.

Figura 19 - Sistema de Testes para o Estudo de Caso comparativo entre as arquiteturas RTOS e *Superloop*.

Os sistemas foram analisados qualitativamente por meio da avaliação da operação e execução das funcionalidades.

A análise quantitativa foi realizada usando os mapas de memória gerados pelo compilador e medidas dos tempos de execução, período e tempo de resposta, realizadas utilizando um osciloscópio.

Para facilitar as medidas, portas do microcontrolador foram ativadas (nível lógico 1) e desativadas (nível lógico 0) na inicialização e conclusão das tarefas, respectivamente.

Essas análises permitiram identificar as principais vantagens e desvantagens das arquiteturas, além das facilidades e dificuldades da implementação.

Todos os programas utilizados foram executados em um computador Pentium D – 3,0 GHz com 2,5 Gbytes de memória RAM.

A terceira etapa consistiu em discutir e avaliar características dos sistemas embarcados de pequeno porte, com foco principalmente em software embarcado, analisando tópicos como arquitetura de microprocessadores, arquiteturas de software, linguagens de programação e sistemas operacionais. Essa discussão foi realizada com base na literatura, cujos principais conceitos e dados foram resumidos nos capítulos 2, 3, 4, e também com base nos resultados obtidos nas etapas 1 e 2.

Capítulo 6

Resultados e Discussão

1. Introdução

Este capítulo apresenta e discute os resultados da pesquisa sobre o desenvolvimento de embarcados no Brasil, bem como os resultados do estudo de caso comparativo entre as arquiteturas *Superloop* e RTOS para um sistema embarcado de pequeno porte.

Além disso, este capítulo também analisa e discute com base na literatura e nos resultados obtidos diferentes características de sistemas de pequeno porte e sobre a viabilidade do uso de RTOS.

2. Desenvolvimento de Embarcados no Brasil

A pesquisa contou com a participação de 67 participantes cujas respostas foram coletadas no período de 28 de setembro de 2010 a 16 de março de 2011.

As subseções posteriores mostram e detalham as análises dos dados coletados.

2.1. Perfil dos Desenvolvedores

A pesquisa foi direcionada a diversas pessoas ligadas ao desenvolvimento de sistemas embarcados. Entre os participantes da pesquisa estão pessoas que exercem diversos cargos junto ao desenvolvimento de eletrônica embarcada, formando um público bem heterogêneo, o que traz amostras de várias visões do processo. Entre os participantes estão engenheiros de desenvolvimento, gerentes de equipes de desenvolvimento e líderes de projeto.

Os desenvolvedores representam a maioria dos participantes num total de 68%, distribuídos em engenheiros de software (48%), engenheiros de hardware (8%) e pesquisadores (12%). A Figura 20 mostra a distribuição de cargos dos participantes. A pesquisa usou a auto-intitulação para definição do cargo exercido.

Figura 20 – Cargos ocupados pelos desenvolvedores participantes da pesquisa - distribuição percentual.

Segundo os dados da pesquisa, é possível observar pessoas jovens trabalhando com desenvolvimento de embarcados. Entre os respondentes observa-se que a grande maioria está na faixa de 26 a 30 (53,8%) e que cerca de 90% tem até 35 anos. Entre os mais novos, estão os desenvolvedores (Engenheiros de Software e Hardware) e entre os mais velhos, destacam-se os líderes de projeto e gerentes de equipes. A Figura 21 mostra a distribuição percentual dos participantes em faixas etárias.

Entre as pessoas envolvidas no desenvolvimento de embarcados, 92% são engenheiros, em sua maioria eletricitas, que representam 61%. Apenas 5% dos desenvolvedores são bacharéis em computação. Esse fato se deve provavelmente a maior familiaridade dos engenheiros eletricitas e engenheiros da computação ao hardware intrinsecamente ligado ao software embarcado e muitas vezes desenvolvido em conjunto. Além disso, grande parte dos cientistas da computação é absorvida pelos mercados de tecnologia de informação e software para computadores pessoais.

Figura 21 – Perfil de Idade dos participantes da pesquisa. Distribuição percentual em faixas Etárias.

É importante ressaltar também a participação dos engenheiros mecâtrônicos que ganham espaço no mercado principalmente devido a relação próxima entre sistemas embarcados e sistemas mecânicos, como em produtos da linha branca, da indústria automobilística e da automação industrial entre outras.

A Figura 22 mostra a distribuição percentual dos participantes entre os cursos de graduação.

Figura 22 – Formação Superior dos desenvolvedores participantes da pesquisa – distribuição percentual

Quanto à formação, é importante destacar que menos da metade dos desenvolvedores possui pós-graduação (43%). Entre os cursos de pós-graduação,

destaca-se o mestrado, título possuído por 25% dos participantes. A Figura 23 mostra a distribuição percentual dos participantes segundo a formação complementar.

O número de desenvolvedores com pós-graduação, porém, deve crescer, principalmente pelo aumento da competitividade do mercado e pela maior valorização de mestres e doutores por parte da indústria que começa a buscar esses profissionais. Na pesquisa, muitos participantes se declaram cursando uma pós-graduação, o que ressalta esta tendência.

Figura 23 - Formação Complementar dos desenvolvedores participantes da pesquisa – distribuição percentual

Segundo os dados da pesquisa, é interessante destacar que 91% dos desenvolvedores tem até 10 anos de experiência em desenvolvimento, dentre os quais 50,7% tem, no máximo, 5 anos de experiência. Além disso, considerando a faixa de até 20 anos de experiência, 100% dos participantes são englobados. Vale destacar que entre os participantes com tempo de experiência entre 10 e 20 anos, estão os líderes de projeto e gerentes de equipes. A Figura 24 mostra a distribuição percentual dos participantes em intervalos de tempo referentes à experiência.

Os dados de experiência, bem como os dados etários, ressaltam uma tendência comumente verificada nas empresas, principalmente com engenheiros, que iniciam a carreira na área técnica, diretamente ligados ao desenvolvimento, e que posteriormente migram para áreas administrativas.

Figura 24 – Experiência dos participantes no desenvolvimento de embarcados. Distribuição percentual em intervalos de tempo referentes aos anos de experiência

Os desenvolvedores geralmente conduzem 2 ou mais projetos em paralelo, com cerca de 20% conduzindo mais de 3 projetos. A maioria dos desenvolvedores (39%) conduz dois projetos em paralelo. A Figura 25 mostra número de projetos desenvolvidos pelos participantes.

Esses dados mostram uma exigência de habilidade de gerenciamento da rotina ao desenvolvedor, bem como traz indícios de que alguns desenvolvedores podem estar trabalhando sobrecarregados.

Figura 25 – Quantidade de projetos simultâneos desenvolvidos pelos participantes – distribuição percentual

Comparando os dados obtidos através da pesquisa, com os dados de pesquisa realizada em 2006 com assinantes das publicações *Embedded Systems Design*,

Embedded Systems Conference, *EE Times* e *Embedded Systems Europe* (CMP United Business Media 2006), observa-se que o desenvolvedor brasileiro geralmente trabalha em mais projetos ao mesmo tempo, é mais novo e menos experiente.

Segundo os dados da pesquisa de 2006 (CMP United Business Media 2006), cerca de 60% dos participantes tem entre 35 e 49 anos, 63% têm mais de 10 anos de experiência em sistemas embarcados e em média conduzem 1.4 projetos simultaneamente.

2.2.Características dos projetos embarcados

Os projetos de sistemas embarcados estão ligados em sua maioria ao desenvolvimento de novos produtos, representando 62,7% dos projetos. Destacam-se também os projetos de alteração de produtos em campo (melhorias, manutenção etc.) que representam 17,9% e os projetos de pesquisa e inovação que respondem por 9% e 7,5% dos projetos, respectivamente. A Tabela 7 mostra os tipos de projetos desenvolvidos pelos participantes.

Esses projetos normalmente têm mais de 6 meses de duração, com cerca de 35% durando de 6 meses a um ano e cerca de 25% durando mais de 1 ano. A Figura 26 mostra a distribuição percentual dos projetos em intervalos de tempo referentes à duração.

Tabela 7 – Tipo de projeto embarcado desenvolvidos pelos participantes – distribuição percentual

Tipo de projetos	%
Desenvolvimento de novo produto	62,7
Alteração de produto (Melhoria, Manutenção, etc...)	17,9
Solução de Problemas de Qualidade	3,0
Inovação	7,5
Pesquisa	9,0

Figura 26 – Duração dos projetos de embarcados desenvolvidos pelos participantes. Distribuição percentual em intervalos de tempo referentes à duração

Esses dados mostram ciclos longos de desenvolvimento que impactam diretamente no *time-to-market* dos produtos. O prazo/*time-to-market* junto com o custo são indicados como principais desafios nos projetos embarcados, representando respectivamente 31,9% e 23,2% das respostas dos participantes. Entre os desafios, destacam-se também a inovação e a qualidade.

Esses dados mostram desafios constantes nos projetos e condizem com a tríade cronograma, qualidade e funcionalidades (Figura 3) que devem ser sempre balanceadas no desenvolvimento segundo Ganssle (Ganssle 1999). A inovação e custo estão diretamente ligados às funcionalidades presentes no sistema embarcado.

Os principais fatores indicados como desafios estão também ligados diretamente a lealdade dos consumidores que normalmente buscam qualidade, inovação e produtos com bom custo-benefício conforme suas demandas, que devem ser atendidas com o correto *time-to-market*. A Figura 27 mostra os principais desafios nos projetos de embarcados apontados pelos participantes.

Figura 27 – Desafios enfrentados pelos participantes no desenvolvimento dos projetos embarcados. Distribuição percentual das respostas nos itens citados.

Os dados levantados pela pesquisa também apontam que a maioria dos projetos embarcados sofre atrasos considerando o cronograma original. Cerca de 30 % dos projetos sofreram prorrogações de mais de 3 meses e cerca de 30 % tiveram atrasos entre 1 e 3 meses. Apenas 26,9 % dos projetos tiveram o cronograma mantido.

Esses atrasos podem ser consequência de alterações na especificação de durante a execução do projeto, fato comum em sistemas embarcados. Também estar relacionada ao grande número de projetos conduzidos simultaneamente mesmos desenvolvedores. A

Tabela 8 mostra o andamento dos projetos com base no cronograma inicial.

Tabela 8 – Andamento dos projetos de sistemas embarcados com base no cronograma inicial. Distribuição percentual em faixas de variações temporais do cronograma.

Andamento do projeto	%
O cronograma foi antecipado em menos de 1 mês	3,0
O Cronograma foi mantido	26,9
O cronograma foi prorrogado em menos de 1 mês	4,5
O cronograma foi prorrogado de 1 a 3 meses	29,9
O cronograma foi prorrogado em mais de 3 meses	32,8
O projeto foi cancelado	3,0

Comparando os dados obtidos através da pesquisa com os dados de pesquisa realizada em 2006 com assinantes das publicações *Embedded Systems Design*, *Embedded Systems Conference*, *EE Times* e *Embedded Systems Europe* (CMP United Business Media 2006), observa-se que os projetos de embarcados brasileiros estão mais ligados a novos produtos, tem ciclos mais curtos de desenvolvimento, mas atrasam com maior frequência.

Segundo os dados da pesquisa de 2006 (CMP United Business Media 2006), em geral, os desenvolvimentos estão em sua maioria ligados a alterações em produtos existentes, representando 56 %. Além disso, esses projetos normalmente duram mais de 1 ano (50%), mas, na maioria dos casos, são entregues no prazo (42%) ou com atrasos de até 2 meses (25%).

2.3. Microcontrolador embarcado

Os sistemas embarcados desenvolvidos no Brasil, segundo dados da pesquisa, utilizam em sua maioria microcontroladores de 8-bits e 32-bits que estão presentes em cerca de 30% dos projetos. Os microcontroladores de 16-bits também têm participação importante, representando 25% dos desenvolvimentos.

A Figura 28 mostra a distribuição percentual dos desenvolvimentos entre as arquiteturas de microcontroladores.

Figura 28 – Arquitetura de microprocessadores utilizados nos projetos de sistemas embarcados pelos participantes – distribuição percentual das respostas.

Esses dados mostram uma característica comum nos desenvolvimentos de países emergentes: o grande uso de microcontroladores de 8-bits. Nesses mercados mais pressionados por custos, os microcontroladores de 8-bits surgem como alternativa para os sistemas embarcados.

Entre os principais fabricantes dos microcontroladores utilizados no Brasil estão as empresas americanas Freescale, Microchip e Texas, utilizados respectivamente em 25,4%, 25,4% e 14,3% dos sistemas embarcados em desenvolvimento. A Tabela 9 mostra a distribuição percentual dos desenvolvimentos entre os fabricantes de microprocessadores.

Tabela 9 – Principais fabricantes dos microprocessadores utilizados nos projetos de sistemas embarcados. Distribuição percentual das respostas

Fabricantes	%
Freescale	25,4
Microchip	25,4
Texas Instrument	14,3
ST	12,7
Atmel	6,3
Altera	4,8
NXP	3,2
NEC/Renesas	1,6
Samsung	1,6
Fujitsu	1,6
Atheros	1,6
Broadcom	1,6

O principal fator na seleção dos microcontroladores nos desenvolvimentos brasileiros é o custo, seguido por periféricos e desempenho. Outros fatores relevantes são a disponibilidade de ferramentas, o fornecedor, o *know-how* do desenvolvedor, a estratégia da empresa, a disponibilidade de código e o reuso. A Figura 29 mostra os fatores mais citados como relevantes na seleção dos microcontroladores.

Esses dados novamente ressaltam a pressão de custo nos projetos de embarcados e justificam o grande uso dos microcontroladores de 8-bits.

Analisando esses dados comparativamente com os resultados de pesquisa realizada em 2006 com assinantes das publicações *Embedded Systems Design*, *Embedded Systems Conference*, *EE Times* e *Embedded Systems Europe* (CMP United Business Media 2006), observa-se que os desenvolvedores brasileiros usam microcontroladores de menor porte, dão mais valor ao custo na seleção e utilizam os mesmos fabricantes.

Segundo os dados da pesquisa de 2006 (CMP United Business Media 2006), os desenvolvimentos utilizam principalmente processadores de 32 bits (54%) enquanto os microcontroladores de 8-bits e 16-bits respondem por cerca de 20% do mercado cada. Além disso, o principal fator na seleção do microcontrolador são as ferramentas disponíveis para desenvolvimento de software enquanto o custo é apenas o terceiro fator mais utilizado. Entre os principais fabricantes, a única diferença é a Intel, que aparece na frente da Microchip. Contudo, essa diferença é compreensível uma vez que a Intel trabalha com processadores de maior porte.

Figura 29 – Fatores determinantes na seleção dos microprocessadores. Distribuição percentual das respostas entre os itens citados.

2.4.Características do software embarcado

Os softwares embarcados brasileiros em sua maioria são construídos utilizando linguagem C, que responde pela quase totalidade dos desenvolvimentos (83,3%). A linguagem C++ responde por cerca de 10% dos desenvolvimentos enquanto outras linguagens como *Assembly* e *Java* fazem parte apenas de 3 e 1,5% dos desenvolvimentos, respectivamente.

A Figura 30 mostra a distribuição percentual dos desenvolvimentos entre as linguagens de programação.

Figura 30 – Principais linguagens de programação utilizadas no desenvolvimento de software embarcado. Distribuição percentual de respostas entre as linguagens citadas.

Esses dados mostram um domínio do mercado por linguagens de alto nível e a tendência de obsolescência do *Assembly*. Isso provavelmente está ligado à facilidade de manutenção e reuso de código propiciado pelas linguagens de alto nível e também ao fato da grande maioria dos desenvolvedores serem jovens, comumente mais familiarizados com linguagens como C, C++ e Java.

A linguagem C também se sobressai a outras linguagens de mais alto nível como Java e C++. Isso se deve provavelmente ao fato da maioria dos desenvolvimentos serem realizados por engenheiros, mais afetos à linguagem C, bem como a facilidade que esta linguagem propicia na manipulação do hardware em comparação às demais.

Barr em *Real men program in C* (Barr 2009) descreve bem esta vantagem, afirmando que a linguagem C tem a mistura certa das funcionalidades para programação, tanto ao nível de *drivers*, quanto ao nível de aplicação das linguagens de alto e baixo nível.

Comparando os dados com pesquisa de 2006 realizada pelas publicações *EETimes* e *Embedded System Design* (CMP United Business Media 2006), a presença da linguagem C entre os desenvolvedores é maior. Os dados de 2006 mostram a linguagem C presente em 51% dos desenvolvimentos, C++ em 30 % e *Assembly* em 8%. Esses dados, todavia, mostram a mesma tendência de superioridade da linguagem C e do declínio do *Assembly*, tendência também verificada por Barr (Barr 2009).

Os dados da pesquisa atual também mostram que o reuso de código é comum, porém em pequenas proporções. O reuso ocorre em 86% dos casos, porém, apenas 30% reutilizam mais de 50% dos códigos existentes em campo.

Esses resultados mostram que mais de 50% do código dos produtos enviados para campo são criados do zero. Isso eleva o tempo de projeto e também diminui sua robustez, uma vez que códigos já testados trazem maior confiabilidade ao desenvolvimento e dificilmente possuem falhas grosseiras.

A Figura 31 mostra a distribuição percentual dos desenvolvimentos segundo a taxa de reuso de código.

Figura 31 – Taxa de reuso de código no desenvolvimento de software embarcado. Distribuição percentual de respostas em níveis de reuso.

2.5. Sistemas operacionais embarcados

A pesquisa mostra também que apenas 28% dos desenvolvimentos atuais utilizam sistemas operacionais. O uso de sistemas operacionais está mais concentrado na plataforma de 32-bits, seguido pelas plataformas de 16 e 8-bits.

A Figura 32 mostra o uso de sistemas operacionais nos projetos atuais e a Figura 33 mostra a distribuição dos projetos que usam sistema operacional entre as arquiteturas de microprocessadores.

Figura 32 – Uso de sistemas operacionais nos projetos atuais de software embarcado. Distribuição percentual das respostas.

Figura 33 – Distribuição percentual do uso de sistemas operacionais entre as arquiteturas de microprocessadores.

Comparando os dados com pesquisa de 2006 realizada pelas publicações *EETimes* e *Embedded System Design* (CMP United Business Media 2006), é possível observar que os desenvolvimentos nacionais utilizam bem menos sistemas operacionais. Segundo os

dados de 2006, 71% dos sistemas embarcados utilizam um OS, RTOS ou *kernel*. Este fato, porém, é compreensível, uma vez que essa pesquisa mostra que a maioria dos participantes utiliza microprocessadores de 32-bits, os quais se adequam melhor a sistemas operacionais (54%).

Entre os principais motivos para o uso de sistemas operacionais estão a necessidade de gerenciamento de multitarefas, exigências de tempo real e a velocidade no desenvolvimento. A modularidade, o reuso e a robustez também são apontada como um dos fatores para o uso.

A Figura 34 mostra os principais motivos para o uso de sistemas operacionais citados pelos desenvolvedores.

Entre os principais fatores para o não uso de sistemas operacionais estão: a não exigência do projeto, o consumo de memória e a estratégia da empresa.

Figura 34 – Principais fatores que motivam a utilização de sistemas operacionais no desenvolvimento de software embarcado. Distribuição percentual das respostas entre os itens citados.

Esses dados são semelhantes aos obtidos por pesquisa realizada em 2006 com assinantes das publicações *Embedded Systems Design*, *Embedded Systems Conference*, *EE Times* e *Embedded Systems Europe* (CMP United Business Media 2006), que mostra que entre os principais argumentos para o não uso de sistemas operacionais estão a falta

de necessidade do projeto, o consumo de processamento, o consumo de memória e o custo. Vale destacar que muitos desenvolvedores não utilizam OS's por opção da empresa, fato não preponderante nos resultados da pesquisa de 2006.

A Figura 35 mostra os principais fatores para o não uso de sistemas operacionais.

Os resultados da pesquisa mostram também que a maioria dos desenvolvedores (70,1%) pretende usar um sistema operacional em projetos futuros. Dentre os que já utilizam 100% tem intenção de continuar usando e entre os que não usam 50% pretendem começar a usar. Esses resultados mostram que há uma tendência de crescimento no uso de OS's que devem dominar os desenvolvimentos até a próxima década, principalmente com o aumento dos recursos dos microcontroladores.

Figura 35 - Principais fatores que motivam a não utilização de sistemas operacionais no desenvolvimento de software embarcado. Distribuição percentual das respostas entre os itens citados.

Figura 36 – Utilização de Sistemas Operacionais nos projetos de sistemas embarcados atuais versus pretensão de uso nos projetos futuros. Distribuição percentual das respostas.

A Figura 36 mostra a utilização atual e a pretensão futura de uso.

Os desenvolvedores, quando questionados quanto ao tipo de sistemas operacionais que utilizam ou utilizariam, mostraram uma preferência por sistemas operacionais *open-source* (65,1%), seguido por sistemas comerciais (17,5%) e por último, pelos desenvolvidos internamente nas empresas (17,5%). Entre os que já utilizam OS, dominam também os sistemas operacionais “*open-source*”.

Esta preferência por sistemas *open-source* provavelmente está ligada a pressão por custos sofrida pelos sistemas embarcados nacionais. Contudo, essa escolha pode trazer alguns problemas se considerarmos que muitos desses sistemas não possuem suporte e muitas vezes não são suficientemente testados.

A Figura 37 **Error! Reference source not found.** mostra a distribuição percentual dos desenvolvedores entre os tipos de sistemas operacionais.

Figura 37 – Tipos de sistemas operacionais preferidos pelos desenvolvedores brasileiros. Distribuição percentual das respostas.

Analisando esses dados comparativamente com pesquisa de 2006 realizada pelas publicações *EETimes* e *Embedded System Design* (CMP United Business Media 2006), é possível verificar que os desenvolvedores brasileiros têm uma preferência maior por sistemas *open-source*. Os dados de 2006 mostram uma preferência por sistemas operacionais comerciais (47%), seguido por sistemas operacionais *open-source* que crescem na preferência, e pelos sistemas desenvolvidos “*in-house*” (17%), que vem caindo na preferência. Destacam-se também nos dados de 2006, as distribuições comerciais de sistemas *open-source*, que representam 17% das escolhas.

Segundo os participantes da pesquisa atual, entre os principais fatores na seleção do sistema operacional estão a disponibilidade de bibliotecas, os atributos de tempo-real, a estratégia da empresa, o custo, a compatibilidade com o hardware e as ferramentas disponíveis.

Entre os que já utilizam OS's, os principais fatores para a seleção são a disponibilidade de bibliotecas, os atributos de tempo-real e a compatibilidade com o hardware, seguido pelas ferramentas disponíveis, a estratégia da empresa e o custo.

A Figura 38 mostra os principais fatores na seleção do OS citados pelos desenvolvedores.

Figura 38 - Fatores determinantes na seleção dos OSs embarcados. Distribuição percentual das respostas entre os itens citados.

Entre os desenvolvedores de pesquisa realizada em 2006 (CMP United Business Media 2006), os fatores majoritários na seleção são respectivamente os atributos de tempo real, a compatibilidade com o hardware, o custo, o suporte técnico e as ferramentas de software disponíveis.

É interessante destacar que o custo, determinante na seleção do microcontrolador e apresentado constantemente como desafio no projeto é apenas o quarto fator na seleção dos sistemas operacionais. Isso talvez se deva ao fato de muitos desenvolvedores considerarem o uso de sistemas *open-source* que, aparentemente apresentam baixo custo, mas que podem exigir o pagamento de *royalties* ou muitas vezes a filiação a comunidades de desenvolvimento.

Figura 39 – Sistemas Operacionais utilizados nos projetos de sistemas embarcados. Distribuição percentual das respostas

Entre os principais sistemas operacionais embarcados considerados estão o Linux, o FreeRTOS e o μ C-OS II. Esses dados, mostrados na Figura 39, ressaltam a preferência por sistemas *open-source*.

2.6.Real-time operational system (RTOS)

A pesquisa também tentou avaliar o uso de sistemas operacionais de tempo real no desenvolvimento de sistemas embarcados.

Figura 40 – Intenção de uso de sistemas operacionais de tempo real nos projetos de software embarcado. Distribuição percentual de respostas.

Os resultados obtidos mostram que cerca de 70 % dos desenvolvedores consideram o uso de sistemas operacionais de tempo real para desenvolvimento de projetos. Esse dado é interessante, uma vez que entre os 70% dos participantes que pretendem usar OS, 80% consideram o uso de um RTOS.

A Figura 40 mostra a consideração de uso de RTOS's pelos participantes da pesquisa.

Os usuários também foram questionados quanto aos requisitos temporais necessários ao RTOS. Os dados obtidos mostram resultados interessantes. Cerca de 60% dos participantes não sabia opinar quanto à necessidade dos requisitos do sistema operacional. Entre os demais participantes, 16,4 % necessitam de requisitos *Hard real-time*, 14,9 % necessitam de requisitos *soft real-time* e 7,5% de requisitos *Firm real-time*.

Levando em conta as pessoas que consideram a utilização de RTOS, o número de pessoas que não sabiam opinar sobre os requisitos de tempo se mantém alto, representando 54% dos participantes.

A Figura 41 **Error! Reference source not found.** mostra os requisitos temporais exigidos ao RTOS, segundo os participantes e a Figura 42 mostra os requisitos temporais necessários ao RTOS entre os participantes que consideram sua utilização.

Figura 41 – Requisitos temporais necessários ao RTOS embarcado segundo os participantes. Distribuição temporal das respostas.

Figura 42 - Requisitos temporais necessários ao RTOS embarcado segundo os participantes que consideram a sua utilização. Distribuição temporal das respostas.

Esses dados mostram contradição e ressaltam que muitos participantes, apesar de considerarem o uso de RTOS, não têm conhecimento de conceitos de sistemas de tempo real e também características do RTOS necessárias ao projeto.

Isso se deve principalmente ao fato de a grande maioria dos cursos de engenharia não cobrirem em seus currículos conceitos de sistemas de tempo real.

Isso condiz com a afirmação de Barr (Barr 2009) que diz que há muitos conceitos do desenvolvimento de sistemas embarcados como reentrância, priorização de tarefas de sistemas operacionais de tempo real, implementação de máquinas de estado, processos robustos de desenvolvimento e arquiteturas de *firmware* são aprendidos nos empregos e não na universidade, exibindo uma lacuna nos currículos.

Segundo Barr (Barr 2009) o desenvolvimento de embarcados exige uma educação especializada que, na maioria das vezes, não é coberta nem pelos currículos de engenharia elétrica nem pelos de ciência da computação. A Figura 43 retirada de Barr (Barr 2009) mostra essas lacunas. Segundo Barr (Barr 2009), infelizmente esse conhecimento aprendido durante o trabalho é pouco organizado.

Esse fato também se reflete quando os participantes da pesquisa foram questionados quanto às características necessárias às tarefas do RTOS, com 47,8% dos participantes não sabendo opinar sobre o tema. Levando em conta os desenvolvedores que consideram a utilização de um RTOS, esse número é de 39,1%.

Entre os participantes, 37,3% afirmam necessitar de tarefas preemptivas e 14,9% afirmam necessitar de tarefas colaborativas. Levando em conta apenas os participantes que consideram a utilização de RTOS, os que afirmam necessitar de tarefas preemptivas representam 45,7% e os que necessitam de tarefas colaborativas, 15,2%.

Figura 43 – Lacuna no aprendizado necessário ao desenvolvimento de embarcados. Ambos os currículos de ciências da computação e engenharia elétrica não apresentam todos os conceitos necessários ao desenvolvimento de embarcados. Fonte: (Barr 2009)

A Figura 44 mostra as características necessárias às tarefas do RTOS segundo os participantes e a Figura 45 mostra as características necessárias às tarefas do RTOS entre os participantes que consideram sua utilização.

Figura 44 - Características necessárias às tarefas do RTOS embarcado segundo os participantes. Distribuição temporal das respostas.

Figura 45 - Características necessárias às tarefas do RTOS embarcado segundo os participantes que consideram a sua utilização. Distribuição temporal das respostas.

3. Estudo de Caso: Superloop versus RTOS

3.1. Superloop

A implementação do código utilizando a arquitetura *superloop* demandou 180 horas de trabalho (1 mês), considerando o uso de um *framework* base para o desenvolvimento. Após a implementação, o sistema foi testado para garantir as funcionalidades.

O código compilado consumiu 6677 bytes (6,5 kbytes) de memória ROM e 172 bytes de memória RAM. Para a pilha foram reservados 128 bytes de RAM

A **Error! Reference source not found.** mostra o mapa de memória gerado pelo compilador. Neste mapa, as áreas de memória ROM utilizadas são representadas pelos segmentos identificados como: *text* e *.const*. Já os segmentos de memória RAM utilizados são identificados por *.bss* e *.ubsct*. O segmento *filler* representa a área de memória ROM não utilizada e preenchida com instruções seguras que forçam o *reset* do microprocessador em caso de falha.

Através dos dados da **Error! Reference source not found.**, é possível observar o baixo consumo de memória da arquitetura *Superloop*. Considerando o microprocessador MCS908AW32, a arquitetura *Foreground/Background* em conjunto com a aplicação consumiu apenas 20,3 % da memória ROM e 14,6 % da memória RAM do sistema (variáveis e pilha). Esse baixo consumo de memória permite futuras expansões do sistema ou até mesmo o uso de microprocessadores menores como o MCS908AW16 da mesma família, reduzindo o custo do sistema.

----- Segments -----					
start	00008000	end	000096ff	length	5887 segment .text
start	000096ff	end	000099e1	length	738 segment .const
start	000099e1	end	0000ffaf	length	26062 filler (0x83)
start	00000070	end	000000c2	length	82 segment .ubsct
start	00000100	end	0000015a	length	90 segment .bss
start	0000ffcc	end	00010000	length	52 segment .const

Figura 46 – Mapa de memória do Software implementado utilizando a arquitetura Superloop. Identifica os segmentos de memória ROM e RAM utilizados.

Além do consumo de memória, o atendimento aos *deadlines* foi observado e o processamento medido para avaliar o comportamento de tempo real do sistema. Quatro medidas foram realizadas para avaliação do sistema: tempo máximo de execução da interrupção, tempo máximo de execução do *loop* de *Background*, tempo máximo consumido pelas tarefas executadas a todo *loop* e tempo máximo de execução dos *slots*. A Tabela 10 Tabela 10s valores obtidos como resultado das medidas.

Tabela 10 – Medidas Temporais da Arquitetura *Superloop* usadas na avaliação do atendimento de *deadlines* e do consumo de processamento do sistema.

Grandeza	Valor
Tempo máximo de execução da interrupção	144 μ s
Tempo máximo de execução do <i>Loop</i> de <i>Background</i>	1,00 ms
Tempo máximo das tarefas executadas em todo <i>Loop</i>	400 μ s
Tempo máximo de execução do <i>slot</i>	690 μ s

Esses dados mostram o atendimento dos *deadlines* pelas tarefas, uma vez que a interrupção consome menos de 150 μ s para sua execução, tempo inferior aos *deadlines* das tarefas de 1 a 9 executadas no *Foreground* cujos *deadlines* eram de 0,2 ms. O mesmo pode ser verificado nas tarefas 11, 12 e 13 executadas a todo *loop*, que apresentam *deadlines* de 1 ms, porém são executadas em no máximo 400 μ s.

As tarefas de 14 a 34 executadas nos *slots* também têm o atendimento das restrições de tempo garantidas, já que o tempo máximo de execução do *slot* é de 1 ms frente aos *deadlines* de 5 ms.

Os dados da Tabela 10 mostram também um consumo de processamento de 75% por parte do *Foreground* e em torno de 80%, considerando ambos os níveis de *Background* e *Foreground*. Esse processamento permite futuras expansões e adição de novas funcionalidades que tipicamente ocorrem durante o ciclo de vida de um projeto embarcado.

O sistema também foi analisado qualitativamente, sobre a ótica de um usuário interagindo com o sistema. Nesse teste foi observado que todas as funcionalidades foram executadas de maneira satisfatória, atendendo aos requisitos do sistema.

Por último, devido ao espaço disponível, um teste exploratório foi realizado, substituindo o microcontrolador MCS908AW32 pelo micro MCS908AW16 e novas funções foram adicionadas, dentre elas: Interface de comunicação Serial, Rotinas de tratamento de falhas, Rotinas de diagnósticos e Salvamento de dados não voláteis.

Analisado qualitativamente, o sistema atendeu as expectativas, reagindo aos estímulos dos usuários, interação com as teclas e envio de dados via comunicação serial, sem observação de atrasos.

Quantitativamente, o tempo máximo de execução do *loop* de *background* e o tempo máximo de execução da interrupção foram medidos. Os resultados são mostrados na Tabela 11 e confirmam o atendimento dos *deadlines*, além de um pequeno aumento no consumo de processamento.

Tabela 11 - Medidas temporais da arquitetura *Superloop* usadas na avaliação do teste exploratório de adição de funcionalidades e substituição de microcontrolador.

Grandeza	Valor
Tempo máximo de execução da interrupção	145 μ s
Tempo máximo de execução do loop de Background	1,3 ms

O consumo de memória também foi avaliado considerando os novos limites do microcontrolador MCS908AW16 que possui 1024 bytes de RAM e 16 Kbytes de ROM. O tamanho da pilha foi mantido. A **Error! Reference source not found.** mostra o mapa de memória gerado após a compilação.

Segments				

start	0000c200	end	0000fbc6	length 14790 segment .text
start	0000fbc6	end	0000ff22	length 860 segment .const
start	0000ff22	end	0000ffaf	length 141 filler (0x83)
start	00000070	end	000000fb	length 139 segment .ubsct
start	00000100	end	000002fb	length 507 segment .bss
start	0000ffcc	end	00010000	length 52 segment .const

Figura 47 - Mapa de memória do software implementado utilizando a arquitetura *Superloop*. Identifica os segmentos de memória ROM e RAM utilizados.

Esses dados mostram um consumo de memória ROM de 95,8% e de memória RAM de 75,6%. Nessas condições, o sistema dificilmente permitiria expansões futuras sem atividades de otimização. Porém, os resultados mostram uma possível redução de custo e a capacidade de agregar funcionalidades à arquitetura *Superloop* devido ao baixo consumo de memória.

3.2. Sistema Operacional de Tempo Real.

A segunda parte do estudo de caso consistiu na implementação da aplicação utilizando o sistema operacional de tempo-real MicroC-OS II.

Para esta implementação, primeiramente a aplicação foi dividida em tarefas como mostrado na Tabela 6 (capítulo 5).

Após a divisão da aplicação em tarefas, seguiu-se à priorização das tarefas. Inicialmente, as prioridades foram atribuídas utilizando o algoritmo *Rate Monotonic* baseado na periodicidade das tarefas. Porém, observando o resultado e com base na experiência sobre o sistema, observou-se que algumas tarefas com períodos longos, mas com *deadlines* curtos, poderiam ser preemptadas por tarefas com maiores *deadlines*, prejudicando a resposta de tempo-real do sistema.

Baseado nesses resultados, uma segunda alternativa foi testada utilizando o algoritmo *Deadline Monotonic*, que se baseia no *deadline* para definição das prioridades.

Esse algoritmo apresentou resultados melhores, priorizando adequadamente as tarefas do sistema. Uma vez que o sistema operacional MicroC-OS II permite apenas uma tarefa por nível de prioridade, as tarefas com mesmo *deadline* tiveram as prioridades diferenciadas com base no conhecimento e experiência sobre a aplicação e sobre o sistema embarcado. A Tabela 12 mostra as prioridades atribuídas pelos algoritmos *Rate Monotonic* e *Deadline Monotonic*.

Após a definição das tarefas e de suas prioridades, iniciou-se a implementação do porte. Neste momento foi observado que o porte fornecido não operava adequadamente para o microcontrolador MCS908AW32.

Apesar de este sistema operacional apresentar porte para o *core* HCS08, funções como o salvamento de contexto e tratamento de interrupção necessitaram ser revisadas com base na aplicação. A tarefa de revisão do porte demandou 1 semana de desenvolvimento (42 horas), não inicialmente considerada no cronograma.

Tabela 12 – Prioridades atribuídas às tarefas pelos algoritmos *Deadline Monotonic* (DM) e *Rate Monotonic* (RM)

Tarefas RTOS	Período (ms)	Deadline (ms)	Prioridade DM	Prioridade RM
Tarefa de Inicialização do Sistema	-	-	4	4
Gerenciamento Banco de LED's 1	1	0,2	5	7
Gerenciamento Banco de LED's 2	1	0,2	6	8
Gerenciamento Banco de LED's 3	1	0,2	7	9
Gerenciamento Banco de LED's 4	1	0,2	8	10
Gerenciamento Banco de LED's 5	1	0,2	9	11
Gerenciamento do Multiplexador	0,2	0,2	10	5
Acionamento de TRIAC's	8,333	0,2	11	19
Desligamento de TRIAC's das cargas indutivas	8,333	0,2	12	20
Desligamento de TRIAC's das cargas resistivas	8,333	0,2	13	21
Gerenciamento do Sensor Digital 1	8,333	0,2	14	22
Deteção de Zero Cross	0,2	0,2	15	6
Controlador de Estado das Cargas	5	1	16	12
Algoritmo de controle de carga - Estado 1	5	1	17	13
Algoritmo de controle de carga - Estado 2	5	1	18	14
Algoritmo de controle de carga - Estado 3	5	1	19	15
Algoritmo de controle de carga - Estado 4	5	1	20	16
Algoritmo de Controle de carga - Controle principal	5	1	21	17
Watchdog Service	5	2	22	18
Tarefa de gerenciamento de eventos do usuário	25	5	23	23
Gerenciamento do Sensor Digital 2	25	5	24	24
Gerenciamento de Estado 1	25	5	25	25
Gerenciamento de Estado 2	25	5	26	26
Gerenciamento de Estado 3	25	5	27	27
Gerenciamento de Estado 4	25	5	28	28
Gerenciamento de Estado 5	25	5	29	29
Gerenciamento de Estado 6	25	5	30	30
Gerenciamento de Estado 7	25	5	31	31
Gerenciador de Ciclo de Atividades	25	5	32	32
Executor de fases do ciclo.	25	5	33	33
Atividade 1	25	5	34	34
Atividade 2	25	5	35	35
Atividade 3	25	5	36	36
Atividade 4	25	5	37	37
Atividade 5	25	5	38	38
Atividade 6	25	5	39	39
Atividade 7	25	5	40	40
Atividade 8	25	5	41	41
Atividade 9	25	5	42	42
Gerenciador de LED's do ciclo	25	5	43	43
Temporizador de LED's	500	100	44	44
Temporizador do Sensor Digital 2	500	100	45	45

Após o porte ser concluído, a codificação das tarefas foi iniciada e finalizada após 3 semanas (126 horas). O software foi submetido à compilação. Para a compilação inicial, as pilhas das tarefas foram fixadas em 50 bytes para posterior ajuste.

Durante a compilação, foi identificado um estouro na memória RAM em 2 Kbytes, 100% de consumo superior à capacidade de memória do microcontrolador MCS908AW32.

Nesse momento duas ações foram tomadas para garantir a implementação do sistema: o cálculo do consumo de pilhas para cada tarefa e a redução do número de tarefas e, com isso, o número de pilhas independentes e o número de variáveis relacionadas às tarefas (Bloco de controles – TCB e Timers).

Apesar de o compilador utilizado apresentar ferramentas que auxiliam o cálculo das pilhas, como a árvore de chamada de funções e cálculo do consumo máximo da pilha, o resultado dessa ferramenta é gerado apenas após a compilação bem sucedida, o que exigiu o cálculo manual das pilhas de tarefas.

O cálculo foi exigido para assegurar que a aplicação se adequaria ao tamanho do microcontrolador. No cálculo foram consideradas todas as variáveis locais criadas pela tarefa e pelas subfunções chamadas por ela, o número de funções chamadas em cadeia e a pilha da interrupção de tempo. A **Error! Reference source not found.** mostra um exemplo do cálculo realizado para determinação da pilha utilizada pelas tarefas.

Figura 48 – Exemplo de cálculo da pilha utilizada por uma tarefa. Soma das variáveis locais criadas e das chamadas locais encadeadas.

Tabela 13 – Tarefas do sistema após otimização: funcionalidades, restrições temporais e prioridades.

Tarefas RTOS	Funcionalidades	Período (ms)	Deadline (ms)	Prioridade DM
Tarefa de Inicialização do Sistema	Tarefa de Inicialização do Sistema	-	-	4
Tarefa de alta frequência	Gerenciamento Banco de LED's 1	0,2	0,2	5
	Gerenciamento Banco de LED's 2			
	Gerenciamento Banco de LED's 3			
	Gerenciamento Banco de LED's 4			
	Gerenciamento Banco de LED's 5			
	Gerenciamento do Multiplexador			
	Acionamento de Triacs			
	Desligamento de Triacs das cargas indutivas			
	Desligamento de Triacs das cargas resistivas			
	Gerenciamento do Sensor Digital 1			
	Deteção de Zero Cross			
Controlador de Cargas	Controlador de Estado das Cargas	5	1	6
	Algoritmo de controle de carga - Estado 1			
	Watchdog Service			
	Temporizador de LED's			
Tarefa de gerenciamento de eventos do usuário	Tarefa de gerenciamento de eventos do usuário	25	5	7
Gerenciamento do Sensor Digital 2	Gerenciamento do Sensor Digital 2	25	5	8
	Temporizador do Sensor Digital 2			
Gerenciamento de Estado 1	Gerenciamento de Estado 1	25	5	9
Gerenciamento de Estado 2	Gerenciamento de Estado 2	25	5	10
Gerenciamento de Estado 3	Gerenciamento de Estado 3	25	5	11
Gerenciamento de Estado 4	Gerenciamento de Estado 4	25	5	12
Gerenciamento de Estado 5	Gerenciamento de Estado 5	25	5	13
	Gerenciador de Ciclo de Atividades			
	Executor de fases do ciclo.			
	Atividade 1			
	Atividade 2			
	Atividade 3			
	Atividade 4			
	Atividade 5			
	Atividade 6			
	Atividade 7			
	Atividade 8			
	Atividade 9			
	Gerenciador de LED's do ciclo			
Gerenciamento de Estado 6	Gerenciamento de Estado 6	25	5	14
	Gerenciamento de Estado 7			

Após 6 iterações que exigiram 1 mês adicional de codificação (180 horas), as tarefas foram aglutinadas e otimizadas reduzindo o número de tarefas à 14, o que sacrificou significativamente a modularidade do sistema, porém garantiu sua implementação dentro das restrições de memória do mesmo. As tarefas, as funções executadas e as novas prioridades são mostradas na Tabela 13 **Error! Reference source not found..**

Após a otimização o código foi novamente compilado. O mapa de memória gerado pela compilação é mostrado na **Error! Reference source not found..** Neste mapa, as áreas de memória ROM utilizada são representadas pelos segmentos identificados como: *.text* e *.const*. Já os segmentos de memória RAM utilizados são identificados por *.bss* e *.ubsct*. O segmento *filler* representa a área de memória ROM não utilizada e preenchida com instruções seguras que forcem o reset do microprocessador em caso de falha.

----- Segments -----					
start	00008000	end	0000a17b	length	8571 segment .text
start	0000a17b	end	0000a55d	length	994 segment .const
start	0000a55d	end	00011daf	length	30802 filler (0x83)
start	00000070	end	000000ff	length	143 segment .ubsct
start	00000100	end	000007d3	length	1747 segment .bss
start	0000ffcc	end	00010000	length	52 segment .const

Figura 49 – Mapa de memória da aplicação implementada utilizando o sistema operacional de tempo-real MicroC-OS II.

Os dados do mapa de memória mostram um consumo de memória ROM de 29,3 % e de 98,5 % (considerando variáveis, pilhas de tarefas e pilha de inicialização do software de 128 bytes).

Esses dados mostram um consumo significativo de RAM pelo sistema operacional devido a cada tarefa possuir seu bloco de controle e pilha independente, e ao consumo significativo de estruturas como semáforos. O consumo de RAM também demonstra uma aplicação no limite, sem possibilidade de expansões, fator prejudicial a um sistema embarcado que, em geral, necessita de atualizações e inclusão de funcionalidades durante seu ciclo de vida.

Posteriormente a compilação, o sistema teve o desempenho avaliado quantitativamente e qualitativamente.

Na análise qualitativa, o sistema demonstrou alta degradação da qualidade das funcionalidades e até mesmo perda de funcionalidades, além de elevados tempo de respostas aos impulsos do usuário.

Na análise quantitativa, o tempo de execução foi medido para verificação do atendimento dos requisitos de tempo-real do sistema, sendo identificado o não atendimento às restrições de tempo, o que justifica a perda de funcionalidades observada na análise qualitativa. A Tabela 14 mostra as 3 tarefas de maior prioridade com seus respectivos máximos tempos de execução observados.

Tabela 14 – Tempos máximos de execução observados para as tarefas de maior prioridade do sistema.

Tarefa	Máximo tempo de execução (ms)	Deadline (ms)
Tarefa de alta frequência	0,354	0,200
Controlador de cargas	106	1
Tarefa de gerenciamento de eventos do usuário	275	5

Esses dados mostram tempos de respostas muito maiores que os *deadlines* estipulados ao sistema. Através da Tabela 14 é possível verificar que o processamento do sistema é totalmente consumido pela tarefa de maior prioridade, que aloca o processador aproximadamente 100 % do tempo.

Essa tarefa é disparada pela interrupção de tempo do sistema operacional, configurada para acontecer a cada 200 μ s, responsável pela geração do *tick* do sistema operacional e a liberação de um semáforo usado na sincronização da tarefa de alta frequência.

A interrupção de tempo apresenta elevado tempo de execução devido a essas atividades, indispensáveis ao sistema operacional, reduzindo o tempo de processamento disponível às tarefas. A Tabela 15 mostra o tempo de execução da interrupção, bem como os tempos de geração do *tick* do sistema operacional e de liberação do semáforo.

Tabela 15 – Tempo de execução da interrupção de tempo e das tarefas do sistema operacional.

Tarefa	Tempo máximo de execução (μ s)
Interrupção de tempo	138
<i>TimerTick</i>	98
<i>Ossempost</i>	39

Esses dados mostram um consumo de aproximadamente 70 % do processamento por tarefas relacionadas á estrutura do sistema operacional. Esse alto consumo do sistema

operacional faz com que a tarefa de maior prioridade tenha pouco tempo disponível à sua execução e não atenda seu *deadline*. Com isso, ela passa a alocar quase a totalidade do tempo do processador gerando uma reação em cadeia em que todas as demais tarefas não atendam seus *deadlines*, degradando a qualidade e a funcionalidade do sistema.

Os dados obtidos na análise do consumo de memória e na análise quantitativa mostram uma sobrecarga do sistema operacional sobre o microcontrolador, tornando pouco viável seu uso para sistemas com pouca memória RAM e aplicações *hard real-time* com *deadlines* inferiores a 1 ms.

Apesar da sobrecarga do sistema, é importante ressaltar que o sistema operacional traz vantagens ao desenvolvimento, facilitando o gerenciamento de tempo e sincronização de tarefas.

É interessante ressaltar que, diferente dos sistemas operacionais de propósito gerais e sistemas de grande porte, o RTOS embarcado não isola o hardware da aplicação, exigindo conhecimentos de hardware ao engenheiro desenvolvedor no desenvolvimento de tarefas que envolvam gerenciamento de entradas e saídas, de conversores analógico/digitais e outros periféricos.

Outro ponto importante é o padrão de desenvolvimento de código utilizado pelo sistema operacional, que muitas vezes pode diferir do utilizado pelo desenvolvedor, sendo importante considerar essa diferença antes do início da codificação para garantir o uso de um único estilo. Entre as duas opções existentes, reescrever o código do sistema operacional ou desenvolver a aplicação seguindo o estilo do RTOS, a segunda opção é a que deve gerar menos trabalho e evitar a adição de tempo extra ao desenvolvimento.

Esse ponto reforça a necessidade de adoção de padrões globais de codificação tanto por parte dos desenvolvedores de sistema operacionais quanto por parte das empresas, como por exemplo, ANSI C ou MISRA C, para garantir a modularidade e reuso de código.

4. Sistemas embarcados de pequeno porte

4.1.Arquitetura de 8-bits x Arquitetura de 32-bits

Os sistemas embarcados de pequeno porte representam hoje uma grande fatia de mercado. Construídos principalmente com microcontroladores de 8-bits, esses sistemas estão presentes principalmente em mercados de produção em massa, como na indústria automotiva, de linha branca, de eletroportáteis e brinquedos.

Nesse mercado o custo é fundamental ao sistema de produção. Custo esse que proporciona sobrevida e move o crescimento nas vendas de microcontroladores de 8-bits, contrariando as inúmeras previsões sobre o fim desse mercado.

Os microcontroladores de 32-bits, apontados como grandes competidores do mercado, apresentam crescente queda de preços, com valores inferiores à US\$ 2 para grandes volumes. Contudo, essa queda ainda não é suficiente para sobrepujar o custo dos microcontroladores de 8-bits com valores inferiores a US\$1. Essa diferença de custo, levando-se em consideração os recursos e funcionalidades disponíveis nos microcontroladores de grande porte, aparenta ser pequena, porém representa uma economia de centenas de milhares de dólares nas empresas que produzem milhões de unidades de produtos, altamente sensíveis a custo.

Além disso, a contínua evolução da tecnologia também tem influenciado o mercado de microcontroladores de 8-bits, reduzindo custos e fornecendo dispositivos com melhor desempenho, maior capacidade de memória e maior número de periféricos.

Assim, o mercado de sistemas de pequeno porte deverá continuar dominado pelos microcontroladores de 8-bits, com aumento contínuo da presença de 32-bits, que deverá assumir a liderança apenas na próxima década

A pressão de custos influencia também diretamente o desenvolvimento de software nos sistemas embarcados de pequeno porte, cada vez mais responsável por realizar funções antes designadas ao hardware, como por exemplo, filtragem de sinais e sensoriamento do ambiente, aglutinadas ao código através de algoritmos de controle.

4.2.Assembly x Linguagem C

O *Assembly* constitui uma linguagem simples, de baixo nível e econômica em termos de custo de processamento e consumo de memória. A linguagem *Assembly* também é muito prática para implementação de *drivers*. Contudo, códigos em *Assembly* são de difícil manutenção, aperfeiçoamento e reuso. Códigos escritos em *Assembly*, apesar de otimizados, são difíceis de serem compreendidos por outros desenvolvedores e dificultam a inserção de novas funcionalidades.

Entretanto, isso não inviabiliza o uso do *Assembly* e pode ser remediado por boa documentação interna e externa do código e por uma programação estruturada, dividindo o código em rotinas e procedimentos com funcionalidades bem definidas e com um fluxo de execução seqüencial bem definido (não digressivo), utilizando os princípios de seqüência, decisão e iteração.

No entanto, essas boas práticas são estritamente dependentes dos desenvolvedores de código, na maioria das vezes não muito afetos à documentação e em muitos casos, não tão cautelosos quanto à estruturação do código, principalmente no âmbito de sistemas embarcados.

A linguagem C, por sua vez, é uma linguagem de médio nível, como descrito por Schildt (Schildt 1997), incorporando características de linguagens de alto e baixo nível. Muito mais próxima da linguagem humana, a linguagem C tem vantagens como portabilidade, facilidade de reuso e manutenção do código, além de propiciar melhor estruturação do código.

Códigos construídos com a linguagem C também exigem boa documentação interna e externa. Entretanto, mesmo os códigos pobremente comentados são mais inteligíveis se comparados a códigos em *Assembly*.

Essas características tornam a linguagem C muito mais viável para a programação de sistemas embarcados onde *time-to-market* e reuso são condições imperativas ao desenvolvimento de software embarcado.

Apesar do maior consumo de memória se comparado ao *Assembly*, a linguagem C se torna cada vez mais viável mesmo para sistemas de pequeno porte devido à evolução dos compiladores que geram códigos altamente otimizados e à maior disponibilidade de recursos dos microcontroladores tanto de memória quanto desempenho.

Isso também justifica tendência a obsolescência do Assembly evidenciada nos dados da pesquisa realizada com desenvolvedores brasileiros e nos dados de pesquisa de 2006, realizada com assinantes das publicações *Embedded Systems Design*, *Embedded Systems Conference*, *EE Times* e *Embedded Systems Europe* (CMP United Business Media 2006), que deve praticamente cair em desuso até a próxima década.

Outros fatores como o domínio do desenvolvimento de embarcados por engenheiros e por pessoas mais jovens mais afetos ao C também influenciam o baixo uso de Assembly.

Apesar do desuso do Assembly isto não significa a descontinuação dessa linguagem que continuará utilizada nos softwares embarcados para driblar as limitações da linguagem C e de outras linguagens de mais alto nível, principalmente na manipulação direta de registradores e de *drivers*. Contudo, indica que o Assembly cada vez mais deixará de ser a linguagem principal do desenvolvimento.

A linguagem C também se sobrepõe a outras linguagens como o C++ e Java que permitem a implementação de paradigmas da orientação objeto no desenvolvimento de software embarcado. Essas linguagens representam fatias pequenas e estáveis dos desenvolvimentos embarcados como demonstrado por Barr (Barr 2009), pelos dados de pesquisa de 2006 realizada pelas publicações *EETimes* e *Embedded System Design* (CMP United Business Media 2006) e pelos dados mostrados na seção 1.

Essas linguagens e a implementação dos paradigmas de orientação ao objeto em geral geram grande *overhead*, consumindo muitos recursos de memória encontrando limitações para sua implementação em sistemas embarcados principalmente nos de pequeno porte. Além disso, a linguagem C permite que paradigmas como objeto, instância, interface, encapsulamento e abstração sejam emulados melhorando a organização do código, eliminando o uso de variáveis globais e aumentando a robustez do código.

O encapsulamento pode ser emulado, por exemplo, utilizando métodos para ler e alterar variáveis eliminando o uso de variáveis globais. Os objetos podem ser emulados através da divisão do código em módulos, cada um representando o módulo e as interfaces através da padronização dos protótipos de funções.

Além do uso desses paradigmas de orientação objeto no desenvolvimento de software embarcado outras boas práticas devem ser usadas para garantir um bom uso da linguagem C, como por exemplo, o uso de padrões como ANSI C ou MISRA C para a

indústria automobilística ou o de padrões internos desenvolvidos pela empresa. Isso é muito bem destacado por Ganssle (Ganssle 2004).

Esses fatores credenciam a linguagem C como a mais indicada para o desenvolvimento de software embarcado em microcontroladores de pequeno porte.

4.3. Superloops x Sistemas Operacionais de Tempo-Real

Além da pressão por custos, as indústrias cada vez mais sofrem com a pressão por *time-to-market*, apresentando janelas de poucos meses para o lançamento de novos produtos. Como ressaltado por Ganssle (Ganssle 2004) em *The Firmware Handbook*, alguns produtos podem ter janelas inferiores a 2 meses.

Nesse ambiente, o reuso de código é fundamental para acelerar o desenvolvimento do software, que vêm cada vez mais consumindo tempo de projeto, superando o desenvolvimento de hardware. O reuso também traz benefícios à qualidade do desenvolvimento, uma vez que uma grande parcela do código já foi amplamente testada no desenvolvimento de projetos anteriores e em produtos em campo.

Os sistemas embarcados de pequeno porte comumente utilizam a arquitetura *background/foreground* (ou *Superloop*) no desenvolvimento de software, sendo que para facilitar o reuso no desenvolvimento de produtos, muitas empresas vêm desenvolvendo módulos de software que compõem um conjunto base (*framework*) sobre o qual o código da aplicação é desenvolvido. Esses módulos geralmente são responsáveis pelo gerenciamento de periféricos e funcionalidades básicas como tratamentos de sinais e operações matemáticas. Muitos fabricantes de microcontroladores também disponibilizam *frameworks* (bibliotecas) para seus dispositivos.

O uso dessa arquitetura em conjunto com os *frameworks* acelera o desenvolvimento de software. Em sistemas embarcados de pequeno porte essa arquitetura também é interessante por consumir usualmente pouca memória e permitir o gerenciamento de multitarefas através de máquinas de estados.

No entanto, essa arquitetura dificulta o gerenciamento de prioridades e de tempo das tarefas. Nos *Superloops* existem apenas 2 níveis de prioridade: o da interrupção

(*foreground*), usado geralmente pelas tarefas com menores *deadlines* e o *background* usado pelas tarefas com maiores *deadlines*.

O aumento de tarefas no sistema com *deadlines* curtos exige maior processamento de tarefas na interrupção que passam a consumir maior tempo, dificultando a execução das tarefas no *background*, que passam a apresentar maiores tempos de resposta, podendo não atender os *deadlines*.

A adição de novas tarefas ao *background* também dificulta o gerenciamento de tempo, aumentando o tempo de resposta das tarefas e exigindo contínuas validações das alterações.

Alternativas como a divisão em pequenos grupos de funções conhecidos como *slots* e a temporização do nível de *background*, podem facilitar a acomodação de novas funcionalidades, reduzir o tempo de processamento do nível de *foreground* e melhorar o determinismo.

Essas alterações melhoram significativamente o desempenho do sistema. Contudo, o aumento das funcionalidades continua a deteriorar o sistema, ocasionando estouro dos *slots* com o aumento do tempo de execução, prejudicando a resposta do sistema. Além disso, apesar do aumento do determinismo, essa estrutura desperdiça processamento aguardando o tempo para a execução do próximo *slot*.

Essas desvantagens muitas vezes inviabilizam o uso dessa arquitetura no sistema, colocando em xeque a simplicidade, principal vantagem dessa arquitetura.

Os sistemas operacionais de tempo-real surgem como alternativa aos *frameworks* formando também um bloco base (*building block*) para o desenvolvimento da aplicação. Os RTOS apresentam vantagens em relação ao *Superloops* no desenvolvimento de sistemas de tempo-real, facilitando o gerenciamento de tempo, o tratamento de multitarefas e a priorização de tarefas.

Em geral, RTOS apresentam múltiplos níveis de prioridade o que permite uma melhor organização do sistema comparado à arquitetura *Foreground/Background*. Com o aumento das funcionalidades as tarefas com menores *deadlines* ou com maior frequência podem ser priorizadas evitando a deterioração da resposta do sistema, o que é difícil nos *Superloops* devido à existência de apenas 2 níveis de prioridade.

Além disso, os mecanismos de sincronização (semáforos, Fila de mensagens e eventos) em conjunto com o escalonador facilitam a coordenação da execução das tarefas. O escalonador também garante que sob demanda de carga o processador esteja sempre em utilização, evitando ociosidade.

Sistemas operacionais de tempo-real disponíveis para sistemas embarcados, no entanto não garantem o atendimento dos requisitos de tempo. Esses RTOS em geral utilizam escalonamento estático de prioridades, deixando a cargo dos usuários a alocação de prioridades. Nesses algoritmos os usuários devem garantir o atendimento dos *deadlines*, diferentemente dos algoritmos dinâmicos orientados pelos *deadlines* como EDF e LST, que alocam as prioridades em tempo de execução, garantindo os requisitos de tempo-real.

Assim, os RTOS embarcados funcionam como catalisadores no processo de desenvolvimento, facilitando o escalonamento de tarefas e agilizando a execução das tarefas com maiores prioridades. Contudo, o atendimento ou não dos *deadlines* é determinado pelo projeto do usuário.

Timmeman e Perneel (Timmerman e Perneel 2001) definem um bom RTOS como:

“Um bom RTOS é aquele que apresenta comportamento previsível sobre todos os cenários de carga (interrupções simultâneas e execução de Threads)”

Definição extremamente relevante, uma vez que o RTOS como elemento catalisador do projeto de software do sistema de tempo-real deve ser determinístico.

A garantia do atendimento dos *deadlines* deve ser assegurada por uma validação consistente do sistema, correta alocação de prioridades utilizando algoritmos apropriados como *Rate Monotonic* ou *Deadline Monotonic* e a escolha adequada do sistema operacional. O atendimento dos requisitos temporais também é diretamente influenciado pelo projeto de Hardware.

A alocação de prioridades é crucial ao sistema e a maioria dos RTOS utiliza alocação estática de prioridades, sendo responsabilidade do usuário a definição dos níveis individuais de cada tarefa. Apesar da maioria dos RTOS apresentarem mecanismos para evitar inversão de prioridades durante execução, se o usuário realizar a inversão na definição das prioridades, a resposta do sistema é definitivamente prejudicada.

Os RTOS na maioria das vezes consomem mais memória que a arquitetura *Superloop* devido à necessidade de pilhas de dados individuais para cada tarefa. O processamento também é desperdiçado durante as trocas de contexto.

Ganssle, em conjunto com Labrosse, no livro *The Firmware Handbook* (Ganssle 2004), afirma:

“Produtos utilizando microprocessadores de 8-bits podem usar os benefícios de um kernel dependendo da complexidade do produto. Desenvolvendo produtos usando

microprocessadores de 8-bits, foi possível descobrir que o uso de um kernel simplificou a programação sem adicionar muito custo”

Ganssle (Ganssle 2004) também recomenda o uso de RTOS para micros 16-bits e 32-bits e afirma que microprocessadores de 4-bits, na maioria das vezes, não suportam um *kernel*.

Essas afirmações ressaltam os benefícios dos RTOS, e reafirmam que os desenvolvimentos com 32-bits são praticamente dominados por sistemas operacionais. Contudo, as afirmações também deixam claro que os RTOS podem não ser viáveis para sistemas simples.

Os RTOS também têm custos envolvidos à sua aquisição, diferentemente da arquitetura *foreground/background*. Em geral, a maioria dos RTOS tem custos envolvidos a licença ou *royalties*, exceto os desenvolvidos internamente às empresas. Mesmo os RTOS *open-source* podem exigir a filiação a uma comunidade, o que pode envolver custos.

Assim em geral, os RTOS são mais indicados para aplicações com inúmeras tarefas *hard real-time* que exigem diferentes priorizações e para sistemas com boa oferta de memória. No caso de uso de RTOS comerciais ou *open sources* deve-se sempre analisar o custo envolvido com licenças, suporte e *royalties*, o que pode inviabilizar o uso em sistemas altamente pressionados por custo.

Já a arquitetura *Superloop* é mais indicada para sistemas mais simples, com pouca oferta de memória e processamento crítico.

Uma sugestão para avaliação do espaço de memória é o software embarcado completo não ultrapassar 80% da memória disponível em Flash e em RAM. Isso porque em geral os sistemas embarcados passam por continuas atualizações e aperfeiçoamentos durante seu ciclo de vida que provavelmente exigirão memória extra para implementação e ajustes de suas funcionalidades.

Considerando essa regra, o sistema operacional deve representar de 30 a 40 % da memória do microcontrolador, uma vez que a parte principal do código, a aplicação, que realmente implementa as funcionalidades exigidas pelo usuário deve consumir a maior parte do código e deve ter prioridade. Em alguns casos quando da utilização de RTOS monolíticos que agregam serviços de aplicação, o OS pode consumir maior espaço de memória.

O processamento também não deve exceder 70 %, uma vez que as alterações futuras também exigirão processamento. Novamente, o RTOS não deve exceder 20 a 30 % do tempo disponível, pois a aplicação é prioritária.

Ganssle destaca isso em *The Firmware Handbook* (Ganssle 2004), onde afirma que:

“Um sistema carregado a 90 % duplica o tempo de desenvolvimento sobre um sistema carregado a 70% ou menos. A 95% o cronograma triplica”

Ganssle (Ganssle 2004) também afirma que projetos de desenvolvimento de tempo real são caros principalmente se estiverem sobrecarregados.

Este fato é coerente uma vez que caso o projeto esteja utilizando os recursos ao máximo, o desenvolvedor necessitará otimizar o consumo do sistema antes de implementar as novas funcionalidades. Além disso, o sistema sobrecarregado traz mais riscos ao desenvolvimento podendo, comprometer o cronograma ou até mesmo inviabilizar o projeto.

4.4.Sistemas Operacionais Preemptivos x Cooperativos

Os sistemas operacionais de tempo-real embarcados em geral utilizam escalonamento preemptivo. A literatura normalmente prega que sistemas de tempo real devem usar tarefas preemptivas. Entretanto, isso não é necessariamente verdade. Sistemas com tarefas cooperativas também podem atender requisitos de tempo real.

Nos sistemas embarcados, a garantia de atendimento dos *deadlines* é assegurada através de extensa validação. Assim, ambos os sistemas cooperativos e preemptivos podem ser usados em sistemas de tempo real.

Sistemas preemptivos trazem vantagens como redução da latência das tarefas de maior prioridade e redução do risco de bloqueio de execução do software por uma tarefa com falhas. No entanto, as sucessivas trocas de contexto que podem ocorrer devido à preempção podem desperdiçar processamento e aumentar excessivamente o tempo de execução de tarefas, o que pode prejudicar a resposta de tempo real do sistema.

Sistemas cooperativos deixam a cargo da tarefa a liberação do processador, isso pode trazer vantagens ao desenvolvimento, evitando excessivas trocas de contexto, reduzindo o tempo de execução de tarefas e que processamentos tenham que ser re-executados devido à preempção.

Sistemas cooperativos, em alguns casos, podem permitir a redução do consumo de memória, uma vez que as trocas de contexto são determinadas pelo usuário, elas podem ser codificadas de maneira a permitir o uso de uma única pilha para as tarefas evitando a alocação de pilhas individuais para cada tarefa, como nos sistemas preemptivos.

Em geral, sistemas cooperativos e preemptivos têm suas vantagens e desvantagens, como destacado por Curtis (Curtis 2006), que afirma que as vantagens de um são o espelho das desvantagens do outro.

Em geral sistemas preemptivos trazem melhores respostas com relação à execução das tarefas de maior prioridade enquanto sistemas cooperativos podem ser mais vantajosos em sistemas com poucos recursos de memória e processamento. Ambos podem garantir requisitos de tempo real, porém exigem extensa validação do sistema

4.5.RTOS Comercial x Proprietário

Os desenvolvedores enfrentam muitos desafios na seleção das melhores soluções para o projeto de embarcados. Desafios como a seleção do microcontrolador, da arquitetura de software e do RTOS, caso um seja utilizado.

Na seleção do RTOS diversos fatores influenciam como desempenho, consumo de memória, disponibilidade de porte para o microcontrolador utilizado, número de tarefas possíveis, preempção, serviços, ferramentas, certificação, fabricante entre outros.

Em geral a seleção é realizada utilizando matrizes de decisão, através das quais diversos RTOS são comparados quanto aos requisitos da aplicação e a prioridade destes. Diversos trabalhos discorrem sobre metodologias para seleção, como os de Hawley (Hawley 1999), Melanson e Tafazoli (Melanson e Tafazoli 2003) e Reedy (Reedy 2006). Hawley (Hawley 1999) discute o uso de um *checklist* para a seleção. Melanson e Tafazoli (Melanson e Tafazoli 2003) propõem a seleção baseado numa tabela de critérios ponderados com pesos com foco para o mercado aeroespacial. Reedy (Reedy 2006) propõe o uso de algoritmos genéticos para a seleção de um RTOS cujos dados estão contidos em um banco de dados.

No processo de seleção de um RTOS uma questão que surge é: desenvolver ou comprar um RTOS.

O desenvolvimento de um RTOS é um desafio interessante ao engenheiro de software, que pode não agregar custos ao produto final e obter uma solução mais otimizada à aplicação. Contudo, geralmente o desenvolvimento de um sistema operacional consome tempo principalmente na sua validação. Segundo Barrett (Barrett 1997), o tempo de validação de um RTOS consome de 3 a 4 vezes mais tempo do que o desenvolvimento. Tempo esse quase nunca considerado no cronograma e também extenso considerando as pressões por *time-to-market*. Ganssle (Ganssle 2004) também é cético em relação aos desenvolvimentos internos e ressalta que o desenvolvimento leva meses de trabalho de um engenheiro de software e o *debug* do sistema pode até mesmo levar anos. Ausência de documentação apropriada e a dependência de suporte dos criadores também são pontos negativos ressaltados por Ganssle (Ganssle 2004).

Atualmente, existe uma vasta gama de RTOS disponíveis no mercado, sistemas comerciais, open-source e distribuições de open-source.

O uso de sistemas operacionais comerciais pode trazer grandes vantagens ao desenvolvimento do sistema embarcado, como redução do tempo de desenvolvimento, melhor suporte e disponibilidade de serviços.

Além disso, muitas empresas vêm certificando seus RTOS segundo regulamentações e padrões de diversos setores.

Contudo, os RTOS comerciais têm custos inclusos que podem impactar o valor final do produto ou do projeto. Alguns sistemas operacionais cobram *royalties* por unidade vendida que utiliza o software. Nestes casos há um impacto direto no custo direto do produto. Outros RTOS cobram licenças para o uso do software, impactando o custo do projeto.

Alternativamente a estes custos, existem os OS's *open-source*. Estes sistemas podem não impactar o custo do produto, entretanto, a ausência de suporte e de documentação adequada pode dificultar o desenvolvimento. Além disso, algumas comunidades podem exigir filiação ou contratos de suporte.

Segundo Cadeño e Laplante (Cadeño e Laplante 2007), o custo total de um RTOS depende do custo inicial, do custo de longo termo de *royalties*, de suporte, de manutenção, de treinamento e de serviços de consultoria.

Outra desvantagem dos sistemas open-source, também presente nos proprietários é ausência de testes adequados.

A Tabela 16 compara qualitativamente algumas características dos RTOS's comerciais, *open-source* e proprietários mostrando as vantagens e desvantagens de cada

sistema. A tabela mostra características em geral apresentadas pelos RTOS de cada tipo, porém não representa necessariamente uma regra, existindo exceções.

Tabela 16 – Vantagens e desvantagens dos diferentes sistemas operacionais: *open-source*, comercial e proprietário. Classificação em três níveis: Ponto Forte ●, Intermediário ◐ e Ponto Fraco ○

Características	Tipo de RTOS		
	Comercial	Open-source	Proprietário
Custo	○	◐	●
Suporte	●	○	◐
Acesso ao Código Fonte	◐	●	●
Documentação	●	○	○
Disponibilidade de Serviços	●	◐	◐*
Validação	●	○	○
Desempenho	◐	◐	●
Ferramentas	●	◐	○
Atendimento de Normas e Regulamentações	◐	○	●
Portabilidade	●	◐	○

* Em geral, apenas serviços utilizados em projetos em andamento estão disponíveis

Assim, o uso de soluções comerciais pode ser muito viável, principalmente em projetos de curto prazo. Os sistemas proprietários, em geral são mais viáveis para sistemas cujo desempenho necessita ser otimizado, na presença de regulamentações e onde o custo é preponderante, porém exige grandes períodos de desenvolvimento.

Os sistemas *open-source* podem acelerar o desenvolvimento, porém em geral apresentam lacunas em relação a suporte, documentação e validação, o que pode impactar o desenvolvimento. Uma alternativa a isso é o uso de distribuições comerciais, mas isso eleva o custo do projeto.

No caso de opções disponíveis no mercado, é indispensável uma criteriosa seleção, considerando não apenas fatores ligados à aplicação, mas também quesitos comerciais, como suporte, custos e documentação. Após a seleção é de extrema valia executar um “*test-drive*” para garantir que o RTOS realmente satisfaz as demandas do projeto como muito bem ressaltado por Moore (Moore 2008).

Capítulo 7

Conclusões

1. Considerações Finais

1.1. Desenvolvimento de Embarcados no Brasil

Os resultados mostram que o desenvolvedor brasileiro de embarcados é em geral, jovem, com até 35 anos de idade, até 10 anos de experiência, é formado em engenharia elétrica/eletrônica, não possui pós-graduação e trabalha em mais de um projeto ao mesmo tempo.

Os dados também mostram uma tendência de migração dos engenheiros com maior experiência para áreas administrativas como liderança de projetos ou de equipes de desenvolvimento. Verifica-se também uma tendência do aumento do número de desenvolvedores cursando mestrado.

Os projetos desenvolvidos normalmente são de novos produtos, duram mais de 6 meses, têm como maiores desafios o *time-to-market* e o custo e têm o cronograma inicial prorrogado, na maioria das vezes, em mais de 3 meses.

Observa-se também que os principais desafios dos projetos estão ligados à lealdade do consumidor: qualidade, inovação, custo-benefício no momento correto (*time-to-market*).

Os desenvolvimentos são dominados pelos microprocessadores de 32-bits e 8-bits, com os microprocessadores de 16 bits também representando fatias importantes. Geralmente o custo é o principal fator na seleção, seguido pelos periféricos disponíveis e pelo desempenho. O uso de microprocessadores de 8-bits é mais significativo que em outras partes do mundo, se comparado ao resultado de pesquisa realizada em 2006 (CMP United Business Media 2006)

Os desenvolvimentos brasileiros são dominados pelas fabricantes americanas Freescale, Microchip e Texas Instrument.

O software embarcado nos projetos brasileiros é escrito em linguagem C e reusa menos de 50% de código, o que pode afetar o seu tempo de desenvolvimento e prejudicar a sua robustez, uma vez que códigos já testados aumentam a confiabilidade do software e apresentam pouca chance de apresentar falhas críticas.

Os dados mostram que apenas 28% dos sistemas embarcados utilizam sistemas operacionais. Entre os principais fatores para o uso estão a necessidade de gerenciamento de multitarefas, exigências de tempo real e a velocidade no desenvolvimento. Entre os principais fatores para o não uso estão a não exigência do projeto, o consumo de memória e a estratégia da empresa.

Os desenvolvedores brasileiros preferem os sistemas operacionais *open-source*. Os principais sistemas operacionais considerados para desenvolvimento são o Linux, o FreeRTOS e o MicroC-OS II.

Os principais fatores para seleção do OS são a disponibilidade de bibliotecas e serviços, os atributos de tempo real e a estratégia da empresa.

A pesquisa mostra também que muitos participantes desejam usar sistemas operacionais de tempo real apresentando, no entanto, dúvidas quanto aos conceitos de sistemas de tempo real e RTOS.

Essa confusão é gerada principalmente devido à maioria dos cursos de engenharia não abrangerem em seus currículos disciplinas sobre sistemas de tempo real e RTOS. Grande parte dos conceitos do desenvolvimento de sistemas embarcados é aprendida pelo trabalho nas empresas.

1.2. Superloop versus RTOS

O estudo de caso ressalta a simplicidade da arquitetura *Superloop*, sua principal vantagem em relação aos sistemas operacionais. Sua estrutura simples estabelece uma base para desenvolvimento de códigos embarcados, sem consumir demasiada memória e processamento.

A arquitetura *Superloop* apresenta alto tempo de resposta da interrupção devido ao grande número de tarefas executadas no nível de *Foreground*, o que pode vir a ser um gargalo ao sistema na presença de muitas tarefas *hard real-time* com *deadlines* curtos.

O sistema operacional mostrou-se muito prático no desenvolvimento de códigos modulares bem como facilitou o gerenciamento de tempo. Contudo, a maior complexidade de sua estrutura eleva o consumo de memória e processamento do sistema embarcado, dificultando sua implementação em microcontroladores pequenos.

O sistema operacional tende a facilitar a implementação do código, porém é importante garantir, antes do início do projeto, a correta operação do porte, evitando surpresas que possam vir a atrasar o cronograma de desenvolvimento.

A alta complexidade da estrutura do sistema operacional torna seu uso viável para sistemas com no mínimo 4 Kbytes de memória RAM e *deadlines* superiores a 1 ms, condições essas que evitam a sobrecarga do microcontrolador.

O uso de sistemas operacionais em microprocessadores com memória RAM inferior a 4 Kbytes exige otimizações ao código que deterioram as vantagens agregadas pelo RTOS, aproximando seu comportamento da arquitetura *Superloop* e agregando custo ao sistema embarcado.

Nesses sistemas o uso de arquiteturas como a *Superloop* que operam como pseudo-sistemas operacionais é mais viável devido a sua simplicidade, que permite a adição de mais funcionalidades ao sistema sem agregar custos, além de não sobrecarregar o microcontrolador.

O estudo de caso também mostra a importância da escolha adequada da arquitetura para o sistema embarcado. A escolha incorreta pode atrasar a entrega ou até mesmo comprometer a viabilidade do projeto.

Vale também ressaltar a importância do uso de padrões globais. Esses padrões facilitam o reuso e aumentam a modularidade do sistema, fatores indispensáveis aos atuais projetos de software embarcado.

1.3.Sistemas Embarcados de pequeno porte

Os microcontroladores de 8-bits continuam e continuarão a representar uma fatia importante no desenvolvimento de embarcados, principalmente nos sistemas de pequeno porte, presentes em produtos produzidos em massa e altamente sensíveis a custo. Apesar da evolução da tecnologia de 32-bits, o custo ainda é fator altamente

relevante na seleção, favorecendo os dispositivos de 8-bits, que continuarão a dominar o mercado de sistemas de pequeno porte até pelo menos a próxima década.

O desenvolvimento de software embarcado, inclusive nos sistemas de pequeno porte, é dominado pela linguagem C, principalmente por suas características mistas de alto e baixo nível.

O uso do Assembly tende a ser reduzido a pequenas partes do código onde há restrições ao uso da linguagem C, sendo praticamente eliminada como linguagem principal para desenvolvimento.

O uso de paradigmas da orientação objeto como encapsulamento, abstração, interface, objeto e instância adaptados às restrições da linguagem C traz vantagens aos desenvolvimentos de sistemas embarcados de pequeno e grande porte, aumentando robustez e estruturação do código, além de facilitar manutenções e reusos futuros.

Nos sistemas de pequeno porte o uso da arquitetura *Superloop* é eficiente em sistemas simples e com restrições de memória. Porém, com aumento do número de tarefas *hard real-time*, o uso de RTOS's mostra-se altamente viável.

O uso de RTOS cooperativos é mais eficiente em sistemas com restrições de memória e processamento, enquanto os sistemas preemptivos reduzem a latência das tarefas com maior prioridade.

O atendimento de requisitos de tempo-real independe da arquitetura utilizada - *Superloop*, RTOS's Cooperativo ou Preemptivo, porém pode ser facilitado através do uso de sistemas operacionais de tempo-real. Todas as arquiteturas exigem validações para assegurar as restrições temporais.

O uso de sistemas operacionais comerciais em geral reduz o tempo de desenvolvimento podendo, no entanto, o seu custo inviabilizar o seu uso em sistemas de pequeno porte, dominados por restrições financeiras.

Os sistemas desenvolvidos internamente às empresas podem atender as restrições de custo, apresentar melhor desempenho, porém, aumentam significativamente o tempo de desenvolvimento, o que pode impactar o *time-to-market*.

Os sistemas operacionais *open-source* podem trazer redução de custo mas podem também trazer problemas devido ao baixo suporte e também a ausência de testes adequados.

Em geral, os sistemas operacionais apresentam boas vantagens que favorecem seu uso, cabendo ao desenvolvedor ponderar e equilibrar a equação *custo x time-to-market*, mais um desafio no árduo processo de seleção de RTOS's.

O processo de seleção é uma das etapas mais importantes do processo de desenvolvimento do software embarcado e deve considerar não apenas requisitos técnicos da aplicação, mas também fatores comerciais como suporte, custos de curto e longo prazo, documentação, além da credibilidade e reputação do fornecedor.

2. Conclusões

O desenvolvimento de sistemas de pequeno porte é dominado por pressões de custo e *time-to-market*.

Neste cenário de pressão de custos, os microprocessadores de 8-bits se mostram uma alternativa altamente viável ao desenvolvimento, o que justifica o crescimento de suas vendas principalmente em países emergentes como o Brasil, conforme verificado na pesquisa.

A pressão por *time-to-market* torna o reuso imperativo nos desenvolvimentos de software. No Brasil, onde geralmente os desenvolvedores, na maioria jovens com pouca experiência, trabalham em 2 ou 3 projetos simultaneamente e com ciclos curtos de desenvolvimento, o reuso é uma prática extremamente necessária para aumentar a robustez dos softwares e garantir a entrega dos projetos.

Nesse cenário de reuso, o uso da linguagem C se sobrepõe ao *Assembly*, facilitando o reaproveitamento, a leitura e a manutenção do código. A linguagem C também se sobrepõe ao uso de linguagens de alto nível devido às restrições de memória dos sistemas de pequeno porte.

O uso de padrões globais como ANSI C ou MISRA C também é altamente recomendável para facilitar o reuso de código e aumentar a modularidade do sistema.

O atendimento de requisitos de tempo real nos sistemas de pequeno porte é garantido através de projeto cuidadoso e de extensa validação do sistema, independente da arquitetura utilizada: *Superloop*, RTOS cooperativo ou preemptivo.

Os *Superloop*, devido a sua estrutura simples, são mais viáveis para sistemas com altas restrições de memória e de processamento, como por exemplo, microprocessadores de 8 e 16 bits com menos de 2 Kbytes de memória RAM e menos de 32 Kbytes de memória ROM. Nesses sistemas, o uso de arquiteturas que operam como pseudo-

sistemas operacionais é mais viável por permitir a adição de mais funcionalidades ao sistema sem agregar custos, além de não sobrecarregar o microcontrolador.

Os RTOS são mais eficazes em sistemas com maior número de tarefas *hard real-time*, facilitando a priorização e o atendimento dos requisitos de tempo real. O sistema operacional propicia o desenvolvimento de códigos modulares e facilita o gerenciamento de tempo. Contudo, a maior complexidade de sua estrutura eleva o consumo de memória e processamento do sistema embarcado com microcontroladores de 8-bits e *clock* de até 20MHz, sendo seu uso viável para sistemas com no mínimo 4 Kbytes de memória RAM e *deadlines* superiores a 1 ms, condições essas que evitam a sobrecarga do microcontrolador.

No Brasil, o uso destes sistemas é pequeno, porém atrai os desenvolvedores que consideram sua utilização em projetos futuros.

Os RTOS comerciais com vasta oferta no mercado podem acelerar os desenvolvimentos além de apresentarem melhor suporte, documentação e serem mais amplamente validados. Contudo, seu custo final para o produto pode inviabilizar seu uso em sistemas de pequeno porte, obrigando empresas a desenvolver internamente soluções ou utilizarem sistemas *open-source*.

No entanto, essas opções normalmente apresentam menor suporte e pouca documentação, além de serem menos testadas.

A seleção do tipo de RTOS a ser utilizado traz mais um desafio ao desenvolvedor no difícil processo de escolha das melhores soluções para aplicação. No caso da opção por RTOS comercial, esse sistema deve ser avaliado não apenas segundo os requisitos da aplicação, mas também segundo quesitos comerciais, como suporte, reputação e credibilidade do fornecedor e custos de curto, médio e longo prazo.

É de suma importância também garantir a funcionalidade do porte do sistema operacional escolhido antes do início do projeto para evitar atrasos ao cronograma de desenvolvimento.

A escolha incorreta da arquitetura de software para o sistema embarcado pode atrasar a entrega ou até mesmo comprometer a viabilidade do projeto.

Apesar da ampla presença de sistemas de tempo real e do crescimento do uso de RTOS no Brasil, os conceitos de sistemas reais ainda não são familiares aos desenvolvedores. Em geral, esses desenvolvedores, na sua maioria engenheiros, não têm contato com esses conceitos nos cursos de graduação, adquirindo com a prática, dentro

do mercado de trabalho, os conhecimentos necessários ao desenvolvimento de embarcados.

3. Contribuições

Entre as contribuições geradas por este trabalho estão:

- Levantamento das características do desenvolvimento de sistemas embarcados no Brasil;
- Análise das melhores soluções para sistemas embarcados de pequeno porte.
- Recomendações para a seleção da arquitetura de software baseado nas configurações do microcontrolador.
- Publicações:
 - (Borges e Rodrigues 2011) “Embedded System Design: An Overview of Brazilian Development” 2011 IEEE International Workshop of Ubiquitous Media and Embedded Systems (UMES2011). Buzan, Coréia do Sul, 2011.

4. Trabalhos Futuros

Entre os Trabalhos Futuros propostos estão:

- Aumento do espaço amostral da pesquisa sobre desenvolvimento de embarcados ao âmbito internacional
- Estudo de Caso comparando as arquiteturas *Superloop* e Sistemas Operacionais de tempo-real em micros de 32-bits.
- Estudo de Caso comparando a arquitetura *Superloop* a outros sistemas operacionais.

Referências

Aroca, R. V. *Análise de Sistemas Operacionais de tempo real para aplicações de Robótica e automação*. São Carlos, SP: EESC USP, 2008.

Bannatyne, R. *Don't write-off 8-bit microcontrollers*. Agosto de 2009. <http://www.electronicsworld.com/Articles/2009/08/24/46802/dont-write-off-8-bit-microcontrollers.htm> (acesso em Setembro de 2010).

Barbiero, A. A., e R. A. Hexsel. “Simulação de Sistemas Embarcados utilizando ArchC.” *VII Workshop em Sistemas Computacionais de Alto Desempenho (WSCAD 2006)*. Ouro Preto, MG, 2006.

Barr, M. *Real men program in C*. Agosto de 2009. <http://www.eetimes.com/electronics-blogs/industrial-control-designline-blog/4027479/Real-men-program-in-C> (acesso em Novembro de 2010).

Barrett, T. “HOW TO WRITE AN RTOS – A Guide for Anyone Pondering The "Make-or-Buy" Decision.” *Real-Time Magazine*, 1997.

Berger, A. *Embedded Systems Design*. CMP Books, 2002.

Bouyssounouse, B. “Real-Time Operating Systems.” In: *Embedded Systems Design*, 258-286. Berlin/Heidelberg: Springer, 2005.

Buttazzo, G. “Research trends in real-time computing for embedded systems.” *ACM SIGBED Review*, Julho de 2006.

Cadeño, W., e P. A. Laplante. “An Overview of Real-Time Operating Systems.” *Journal of the Association for Laboratory Automation* Ed. 12 (2007): pg. 40-45.

Charrette, R.N. “Why Software Fails.” *IEEE Spectrum*, Setembro 2005: pg. 42-49.

CMP United Business Media. *2006 Embedded System Design – State of Embedded Market Survey*. Pesquisa de Mercado, CMP United Business Media, 2006.

Cooling, J. “Real-Time Operating Systems for Embedded World: The importance and role of standards.” *IEEE*, 2004.

Cosmic. *C Cross Compiler User's Guide for Freescale HC08/HCS08*. 2010.

Curtis, K. “Doing embedded multitasking with small microcontrollers.” *EE Times – India*, Dezembro 2006.

Freescale. *MC9S08AW60 Datasheet – HCS08 Microcontrollers*. Vers. Rev.2. Dezembro de 2006. www.freescale.com (acesso em Dezembro de 2010).

Friedrich, L.F. “A Survey on Operating System Support for Embedded Systems Properties.” *VI Workshop de Sistemas Operacionais (WSO2009)*. Bento Gonçalves, RS, 2009.

Ganssle, J. *RTOS Dissatisfaction*. 2006. <http://www.embedded.com/columns/embeddedpulse/188100957> (acesso em Julho de 2010).

- . *The art of designing embedded systems*. Newnes, 1999.
- . *The Firmware Handbook*. Elsevier, 2004.
- Genuchten, M. “The Impact of Software Growth on the Electronics Industry.” *Computer*, Janeiro de 2007: pg. 106-108.
- Hawley, G. “Selecting a Real-Time Operating System.” *Embedded Systems Programming*, Março de 1999.
- Heiser, G., K. Elphinstone, I. Kuz, G. Klein, e S. Petters. “Towards Trustworthy Computing Systems: Taking Microkernels to next level.” *ACM SIGOPS Operating Systems Review* Vol. 41, n. Issue 4 (Julho 2007).
- Henzinger, T. A., e J. Sifakis. “The Embedded Systems Design Challenge.” *Formal Methods* (Springer), 2006.
- Knight, J. C. “Dependability of Embedded Systems.” *Proceedings of the 24th International Conference on Software Engineering*. Orlando, Flórida, 2002.
- Labrosse, J. J. *MicroC/OS-II – The Real Time Kernel*. Segunda Edição. San Francisco, Califórnia: CMP Books, 2002.
- Lee, E. A. “Embedded Software.” *Advances in Computers* Vol. 56 (2002): Pg. 55-95.
- Li, Q. *Real Time concepts for Embedded Systems*. CMPBooks, 2003.
- Liu, J. W. S. *Real time Systems*. Upper Saddle River, New Jersey: Prentice Hall, 2000.
- Melanson, P., e S. Tafazoli. “A Selection Methodology for the RTOS Market.” *Data Systems in Aerospace (DASIA 2003) conference*. Praga, República Tcheca, 2003.
- Mokhoff, N. *Report has microntrollers grow to \$12 billion this year*. Março de 2010. <http://eecatalog.com/8bit/2009/04/28/conflicting-requirements-drive-the-8-bit-microcontroller-market/> (acesso em Setembro de 2010).
- Moore, R., entrevista feita por J. Mc Donald. “Selecting an embedded RTOS.” *Presidente da Micro Digital, Inc.* www.eg3.com, (Dezembro de 2008).
- Murphy, C. “This New House”. Setembro de 2005. http://money.cnn.com/magazines/fortune/fortune_archive/2005/09/19/8272889/index.htm (acesso em Dezembro de 2009).
- Nery, M. *Reduzindo a dispersão dos atrasos em sistemas de tempo real soft com restrições de média de tempo de resposta*. São Carlos, SP: EESC USP, 2009.
- Noergaard, T. *Embedded System Architecture – A comprehensive Guide for Engineers and Programmers*. Elsevier, 2005.
- Ortiz Jr., S. “Embedded OSs Gain the Inside Track.” *Computer* (IEEE Computer Society) Vol. 34, n. Issue 11 (Novembro 2001).

- PE Micro. “P&E Microcomputer Systems.” *Cyclone PRO*. Maio de 2009. http://www.pemicro.com/downloads/main_downloads_temp/201105020809421484868/cyclone%20pro%20rev.%20c%201_12.pdf (acesso em Abril de 2011).
- Ramamritham, K., e J. A. Stankovic. “Scheduling Algorithms and Operating Systems Support for Real-Time Systems.” *Proceedings of The IEEE* Vol. 82, n. 1 (Janeiro 1994).
- Reedy, S. R. “Selection of RTOS for an Efficient Design of Embedded Systems.” *International Journal of Computer Science and Network Security* Vol 6., n. 6 (Junho 2006).
- Rooijmans, J., H. Aerts, e M. Genuchten. “Software quality in consumer electronics products.” *IEEE Software*, 1996.
- Schildt, H. *C completo e Total*. 3a. Edição. São Paulo, SP: Pearson Makron Books, 1997.
- Shandle, J. *More for Less: Stable Future for 8-bit Microcontrollers*. Maio de 2004. <http://www.eetimes.com/electronics-news/4196930/More-for-Less-Stable-Future-for-8-bit-Microcontrollers> (acesso em Setembro de 2010).
- Simon, D. E. *An embedded software primer*. Addison-Wesley, 1999.
- Singh, K. *Design and Evaluation of an Embedded Real-time Micro-kernel*. Blacksburg, Virginia: Faculty of the Virginia Polytechnic Institute and State University, 2002.
- Sperling, E. *Hot market for 8-bit microcontrollers - Distributed processing and new applications are pumping new life into what was until very recently considered dated technology*. 2010. <http://eecatalog.com/8bit/2008/04/18/hot-market-for-8-bit-microcontrollers/> (acesso em Setembro de 2010).
- Stallings, W. *Operating Systems: Internals and Design Principles*. Pearson Prentice Hall, 2008.
- Takada, H. “ μ ITRON: A Standard Real-Time Kernel Specification for Small-Scale Embedded Systems.” *Real-Time Magazine*, 1997.
- Tan, Su-Lim, e T. N. B. Anh. “Real-time operating system (RTOS) for small (16-bit) microcontroller.” *The 13th IEEE International Symposium on Consumer Electronics (ISCE2009)*. 2009.
- Timmerman, M., e L. Perneel. *RTOS state of the art*. Relatório técnico, Dedicated Systems, 2001.
- Vetromille, M., L. Ost, C. A. M. Marcon, C. Reif, e F. Hessel. “RTOS Scheduler Implementation in Hardware and Software for Real Time Applications.” *Proceedings of the Seventeenth IEEE International Workshop on Rapid System Prototyping (RSP'06)*. 2006.
- Wolf, W. “The Good News and the Bad News.” *Computer*, Outubro de 2007: pg. 104-105.
- . *Computers as Components – Principles of Embedded Computer System Design*. Elsevier, 2005.

Apêndice I

Enquete Desenvolvimento de embarcados no Brasil

1. Perguntas:

1. Cargo*

- a) Engenheiro de Software
- b) Analista de Software
- c) Gerente de Software
- d) Líder de Projeto
- e) Pesquisador
- f) Outro: _____

2. Idade: _____

3. Formação*

- a) Engenharia Elétrica/Eletrônica
- b) Engenharia da Computação
- c) Ciências da Computação
- d) Engenharia Mecatrônica
- e) Outro: _____

4. Pós-graduação*

- a) Não Possui
- b) Especialização
- c) Mestrado
- d) Doutorado
- e) Outro: _____

5. Experiência no desenvolvimento de embarcados*

- a) Menos de 5 anos
- b) Entre 5 e 10 anos
- c) Entre 10 e 20 anos
- d) Entre 20 e 30 anos
- e) Mais de 30 anos

6. Em quantos projetos simultâneos você está trabalhando atualmente?*

- a) Um único projeto

- b) 2 projetos
 - c) 3 projetos
 - d) Mais de 3 projetos
 - e) Nenhum
7. Qual o tipo do último projeto que você trabalhou ou está trabalhando?*
- a) Desenvolvimento de novo produto
 - b) Alteração de produto (Melhoria, Manutenção etc.)
 - c) Solução de Problemas de Qualidade
 - d) Inovação
 - e) Pesquisa
 - f) Outro:_____
8. Qual o principal desafio do último projeto desenvolvido ou em desenvolvimento?*
- a) Custo
 - b) Qualidade
 - c) Segurança
 - d) Prazo/*Time-to-market*
 - e) Inovação
 - f) Outro:_____
9. Qual a duração do último projeto desenvolvido ou em desenvolvimento?*
- a) Menos de 2 meses
 - b) De 2 a 4 meses
 - c) De 4 a 6 meses
 - d) De 6 meses a 1 ano
 - e) Mais de 1 ano
10. Com relação ao cronograma inicial do projeto, como esta ou foi o andamento do projeto?*
- a) O Cronograma foi mantido
 - b) O Cronograma foi antecipado em menos de 1 mês
 - c) O Cronograma foi antecipado de 1 a 3 meses
 - d) O Cronograma foi antecipado em mais de 3 meses
 - e) O Cronograma foi prorrogado em menos de 1 mês
 - f) O Cronograma foi prorrogado de 1 a 3 meses
 - g) O Cronograma foi prorrogado em mais de 3 meses
 - h) O projeto foi cancelado
11. Qual a arquitetura do processador principal utilizado no último projeto desenvolvido ou em desenvolvimento?*
- a) 4-bits
 - b) 8-bits

- c) 16-bits
 - d) 32-bits
 - e) 64-bits
 - f) Outro dispositivo (SOC, FPGA's, etc.)
12. Qual o fabricante do principal microprocessador utilizado no projeto desenvolvido ou em desenvolvimento?
13. Qual o modelo do principal microprocessador utilizado no projeto desenvolvido ou em desenvolvimento?
14. Quais os principais fatores na seleção do microprocessador para o projeto?*
- ☒ Ferramentas
 - ☒ Desempenho
 - ☒ Custo
 - ☒ Compatibilidade com Sistema Operacional
 - ☒ Código/Framework disponível
 - ☒ Escalabilidade
 - ☒ Popularidade
 - ☒ Fornecedor
 - ☒ Consumo de Energia
 - ☒ Capacidade de Memória
 - ☒ Periféricos
 - ☒ Reuso
 - ☒ Número de I/Os
 - ☒ Estratégia da Empresa
 - ☒ *Know-how*
 - ☒ Outro: _____
15. Qual a porcentagem de código reusada no último projeto desenvolvido ou em desenvolvimento?*
- a) Não houve reuso
 - b) Menos de 25%
 - c) De 25% a 50%
 - d) De 51% a 75%
 - e) Mais de 75%
16. Qual a linguagem de programação usada no desenvolvimento do projeto?*
- a) Assembly
 - b) C
 - c) C++
 - d) Java

e) Outro:_____

17. Em seu último projeto, foi utilizado um Sistema Operacional Embarcado?*

- a) Sim
- b) Não

18. a. Qual o principal motivo para o uso do sistema operacional?*

- a) Necessidade de gerenciamento de multitarefas
- b) Velocidade no desenvolvimento
- c) Modularidade
- d) Estratégia da Empresa
- e) Reuso
- f) Robustez
- g) Exigências de tempo-real
- h) Outro:_____

18. b. Qual o principal motivo para o não uso do sistema operacional?*

- a) Exigências de grandes recursos de processamento
- b) Consumo de memória
- c) Custo
- d) Estratégia da Empresa
- e) Complexidade de Uso
- f) Falta de suporte
- g) Projeto não exigia
- h) Outro:_____

19. Você considera a utilização de um Sistema operacional embarcado para o próximo projeto?*

- a) Sim
- b) Não

20. Que tipo de Sistema Operacional Embarcado você utiliza ou utilizaria?

- a) Comercial
- b) Desenvolvido Internamente pela Empresa
- c) Open-source
- d) Outro:_____

21. Qual o principal fator na escolha do Sistema Operacional Embarcado?

- a) Atributos de tempo-real
- b) Custo
- c) Consumo de memória
- d) Customização
- e) Suporte Técnico
- f) Compatibilidade com Hardware

- g) Ferramentas Disponíveis
- h) Popularidade
- i) Fornecedor
- j) Experiência do Time
- k) Disponibilidade de bibliotecas e serviços
- l) Estratégia da Empresa
- m) Outro: _____

22. Você considera a utilização de um Sistema operacional de tempo-real (RTOS)?

- a) Sim
- b) Não

23. O Sistema operacional de tempo-real (RTOS) necessitaria de que tipo de requisitos?

- a) Hard real-time
- b) Soft real-time
- c) Firm real-time
- d) Não possuo opinião formada

24. O Sistema operacional de tempo-real (RTOS) necessitaria de tarefas:

- a) Preemptivas
- b) Não-preemptivas (Colaborativas)
- c) Não possuo opinião formada

25. Qual sistema operacional você utiliza ou utilizaria?

- a) VxWorks
- b) eCos
- c) μ C-OS II
- d) MQX
- e) QNX
- f) μ ITRON
- g) μ T-kernel
- h) Linux
- i) FreeRTOS
- j) Outro: _____

Comentários

Sinta-se a vontade para inserir quaisquer comentários com relação ao tema abordado ou o conteúdo da empresa.

2. Fluxo da Pesquisa

Figura I - 1 - Fluxo das perguntas da enquete Desenvolvimento de Embarcados no Brasil