UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA
E INFORMÁTICA INDUSTRIAL

MAIKO ROSSANO MOROZ

CRITÉRIOS PARA ADOÇÃO E SELEÇÃO DE SISTEMAS OPERACIONAIS EMBARCADOS

DISSERTAÇÃO

CURITIBA
2011

MAIKO ROSSANO MOROZ

CRITÉRIOS PARA ADOÇÃO E SELEÇÃO DE SISTEMAS OPERACIONAIS EMBARCADOS

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre em Ciências, do Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial, da Universidade Tecnológica Federal do Paraná – Câmpus Curitiba. Área de Concentração: Telemática.

Orientador: Prof. Dr. Volnei Antônio Pedroni

CURITIBA
2011

Dados Internacionais de Catalogação na Publicação

M871 Moroz, Maiko Rossano

Critérios para adoção e seleção de sistemas operacionais embarcados / Maiko Rosssano Moroz

. – 2011.

75 f. : il. ; 30 cm

Orientador: Volnei Antônio Pedroni.

Dissertação (Mestrado) – Universidade Tecnológica Federal do Paraná. Programa de Pós-graduação em Engenharia Elétrica e Informática Industrial. Curitiba, 2011.

Bibliografia: f. 72-75.

1. Sistemas embarcados. 2. Sistemas operacionais (Computadores) – Avaliação. 3. Processamento eletrônico de dados em tempo real. 4. Simulação (computadores). 5. Engenharia elétrica – Dissertações. I. Pedroni, Volnei Antônio, orient. II. Universidade Tecnológica Federal do Paraná. Programa de Pós-graduação em Engenharia Elétrica e Informática Industrial. III. Título.

CDD (22. ed.) 621.3

Biblioteca Central da UTFPR, Campus Curitiba

Título da Dissertação Nº 581:

"Critérios Para Adoção e Seleção De Sistemas Operacionais Embarcados"

por

Maiko Rossano Moroz

Esta dissertação foi apresentada como requisito parcial à obtenção do grau de MESTRE EM CIÊNCIAS – Área de Concentração: Telemática, pelo Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial – CPGEI – da Universidade Tecnológica Federal do Paraná – UTFPR – Câmpus Curitiba, às 10h do dia 30 de novembro de 2011. O trabalho foi aprovado pela Banca Examinadora, composta pelos professores:

Prof. Volnei Antônio Pedroni, Dr
(Presidente - UTFPR)

Prof. Altair Olivo Santin, Dr
(PUC-PR)

Prof. Douglas Paulo Bertrand Renaux, Dr
(UTFPR)

Visto da coordenação:

Prof. Fábio Kurt Schneider, Dr
(Coordenador do CPGEI)

Tu és, Senhor, o meu pastor.

Por isso nada em minha vida faltará.

(Salmo 23)

AGRADECIMENTOS

Agradeço a Deus por tudo.

À minha família, por todo o apoio e o incentivo proporcionados durante a realização desse trabalho.

Ao Professor Volnei Antônio Pedroni, meu orientador, pela atenção, paciência e confiança depositadas.

Ao Ricardo Jasinski, por todas as contribuições que melhoraram muito este trabalho.

Aos colegas do laboratório LME, pela amizade e companhia em todos os momentos que compartilhamos juntos.

Aos colegas de mestrado, pelos vários momentos de estudo e pelas valiosas discussões que me fizeram aprender mais.

À UTFPR, pelas instalações e os materiais consumidos durante a realização desse trabalho.

Ao CNPq, pela bolsa de estudos concedida.

"*If we knew what it was we were doing, it would not be called research, would it?*"

Albert Einstein

RESUMO

MOROZ, Maiko Rossano. Critérios para adoção e seleção de sistemas operacionais embarcados. 2011. 75 f. Dissertação - Programa de Pós-Graduação em Eng. Elétrica e Informática Industrial, Universidade Tecnológica Federal do Paraná. Curitiba, 2011.

Sistemas embarcados são sistemas computacionais projetados para aplicações específicas, os quais estão presentes em praticamente todos os dispositivos eletrônicos atuais. A utilização de um sistema operacional (SO) é uma maneira de simplificar o desenvolvimento de software, livrando os programadores do gerenciamento do hardware de baixo nível e fornecendo uma interface de programação simples para tarefas que ocorrem com frequência. A alta complexidade dos computadores pessoais atuais torna a utilização de um SO indispensável. Por outro lado, sistemas embarcados são arquiteturas limitadas, geralmente com muitas restrições de custo e consumo. Devido às demandas adicionais impostas por um SO, os desenvolvedores de sistemas embarcados enfrentam a crítica decisão quanto à adoção ou não de um SO. Nesta dissertação, apresenta-se uma série de critérios a fim de auxiliar os projetistas de sistemas embarcados na decisão quanto ao uso ou não de um SO. Além disso, outros critérios são apresentados com o intuito de guiar a seleção do SO mais adequado às características do projeto. Adicionalmente, escolheu-se 15 sistemas operacionais para serem analisados de acordo com os critérios apresentados, os quais podem ser utilizados como base para o processo de seleção de um SO. Adicionalmente, a fim de avaliar o impacto da adoção de um SO em um projeto embarcado, apresenta-se um estudo de caso no qual uma aplicação modelo (uma estação meteorológica embarcada) foi desenvolvida em três diferentes cenários: sem um SO, usando um SO de tempo real (µC/OS-II), e usando um SO de propósito geral (uClinux). Uma FPGA e um SoPC foram utilizados para obter uma plataforma flexível de hardware apta para acomodar as três configurações. A adoção de um SO proporcionou uma redução de até 48% no tempo de desenvolvimento; em contrapartida, isto aumentou os requisitos de memória de programa em pelo menos 71%.

Palavras-chave: Sistemas embarcados, Sistemas operacionais, Sistemas de tempo real.

ABSTRACT

Moroz, Maiko Rossano. Criteria for adoption and selection of embedded operating systems. 2011. 75 f. Dissertação - Programa de Pós-Graduação em Eng. Elétrica e Informática Industrial, Universidade Tecnológica Federal do Paraná. Curitiba, 2011.

An embedded system (ES) is a computing system designed for a specific purpose, present essentially in every electronic device. The use of an operating system (OS) is advocated as a means to simplify software development, freeing programmers from managing low-level hardware and providing a simpler programming interface for common tasks. The high complexity of modern desktop computers makes an OS indispensable; embedded systems, on the other hand, are limited architectures, usually severely cost- and power-constrained. Because of the additional demands imposed by an OS, embedded developers are faced with the crucial decision of whether to adopt an OS or not. In this work, we introduce a set of criteria to help determine whether an OS should be adopted in an embedded design. We then go further and establish a series of rules to help decide which OS to pick, if one should be used. In addition, we present a case study in which a sample application (an embedded weather station) was developed under three different scenarios: without any OS, using the μC/OS-II real-time OS, and using the uClinux general-purpose OS. An FPGA and a SoPC were used to provide a flexible hardware platform able to accommodate all three configurations. The adoption of an OS provided a reduction of up to 48% in development time; on the other hand, it increased program memory requirements in at least 71%.

Keywords: Embedded systems, Operating systems, Real-time systems.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

1.1 MOTIVAÇÃO

Sistemas embarcados são sistemas computacionais projetados para uma aplicação específica, ao contrário dos computadores pessoais, os quais são projetados para atenderem bem às mais diversas aplicações. Os sistemas embarcados estão presentes em praticamente todos os dispositivos eletrônicos, como câmeras fotográficas digitais, telefones celulares, sistemas de controle em automóveis e aviões, equipamentos de controle de processos industriais, e muitos outros. A complexidade dos projetos embarcados vem aumentando significativamente nos últimos anos, assim também a pressão no tempo de desenvolvimento para a maioria desses produtos.

Devido a esta ter um enorme impacto sobre as características fundamentais do projeto, como desempenho, custo e tempo de desenvolvimento, a decisão de adotar ou não um sistema operacional (SO) é uma das mais importantes em um projeto embarcado. No entanto, apesar da sua importância, não há uma abordagem sistemática ou um método consistente para ajudar o projetista em tal decisão.

A escolha do sistema operacional embarcado é uma decisão ainda mais complexa, sobretudo devido à quantidade de opções disponíveis. O sistema operacional escolhido terá grande impacto nas decisões subsequentes quanto aos recursos de hardware e software e metodologias de desenvolvimento. Mesmo assim, as razões para a sua escolha não são muito claras e, frequentemente, são baseadas em critérios que não levam em consideração os reais requisitos do projeto.

De acordo com a pesquisa realizada pelo grupo EETimes (EETIMES GROUP, 2010) em 2010, 70% dos projetos pesquisados utilizaram algum tipo de sistema operacional. A Figura 1 mostra a distribuição entre os projetos que utilizaram um sistema operacional, classificados nas categorias *comercial*, *open source* ou desenvolvido *in-house*. Nota-se que

em 32% dos projetos baseados em um sistema operacional os projetistas preferiram desenvolver um sistema operacional *in-house* ao invés de utilizar um existente. Esta pesquisa revela uma deficiência no mercado de sistemas operacionais, visto que, apesar de existirem vários sistemas operacionais disponíveis, nenhum foi escolhido em um terço dos projetos. Segundo a mesma pesquisa, os critérios considerados mais importantes na seleção de um sistema operacional foram a capacidade de tempo real e o suporte técnico oferecido pelo fabricante, com 45% e 42%, respectivamente. A Figura 2 mostra os dez principais critérios utilizados para seleção de um sistema operacional, segundo a mesma pesquisa, em comparação com os valores obtidos no ano de 2007. Ainda, de acordo com a mesma pesquisa, os cinco principais critérios considerados foram capacidade de tempo real, suporte técnico, ferramentas de desenvolvimento disponíveis, processadores suportados e custo total.

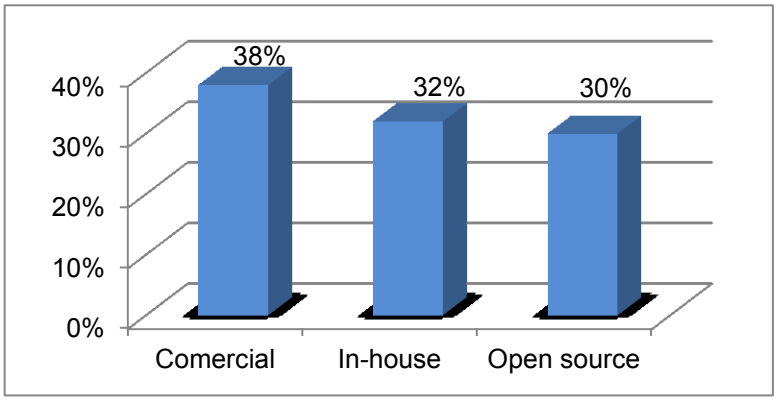


Figura 1 - Utilização de sistemas operacionais comercial, *open source* e *in-house* (adaptado de EETIMES GROUP, 2010).

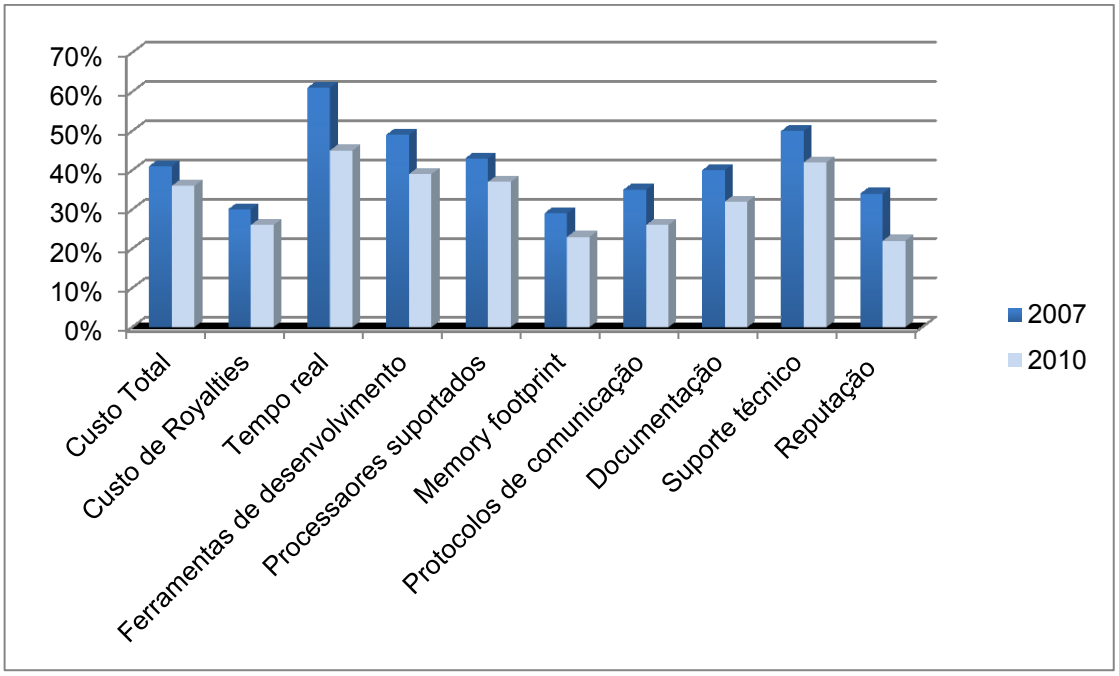


Figura 2 - Fatores mais influentes na escolha de um sistema operacional (adaptado de EETIMES GROUP, 2010).

Entre os dez principais critérios utilizados para a escolha de um sistema operacional apresentados na Figura 2, a capacidade de tempo real, os processadores suportados, os requisitos de memória e protocolos de comunicação são características funcionais do projeto, e os demais são características não-funcionais. Apesar de serem os critérios mais utilizados, estes não são suficientes para a seleção do sistema operacional mais adequado em todos os casos.

1.2 OBJETIVOS

O objetivo principal desta dissertação é definir uma série de critérios, a fim de auxiliar os desenvolvedores de sistemas embarcados nas decisões a respeito da adoção e seleção de sistemas operacionais embarcados. São apresentados critérios para avaliar a necessidade da utilização de um sistema operacional e critérios focando a seleção do sistema operacional mais adequado às necessidades do projeto. Para isso, foram definidos os objetivos específicos listados a seguir.

- Analisar projetos de sistemas embarcados, a fim de identificar as principais características que possam ser utilizadas como critérios para adoção de um sistema operacional.
- Analisar alguns sistemas operacionais embarcados, a fim de identificar as características relevantes que podem ser utilizadas como critérios de seleção.
- Realizar um estudo de caso, a fim de avaliar o impacto da adoção de um sistema operacional em um projeto embarcado. Isto será feito através do desenvolvimento de uma aplicação implementada com e sem a utilização de um sistema operacional.

1.3 ESTRUTURA DA DISSERTAÇÃO

Esta dissertação está organizada em cinco capítulos. O primeiro capítulo introduz este trabalho; o segundo apresenta uma breve revisão da literatura sobre sistemas embarcados e sistemas operacionais. O terceiro capítulo apresenta o desenvolvimento deste trabalho, ou seja, os critérios para adoção e seleção de sistemas operacionais embarcados. No quarto capítulo são apresentados os resultados obtidos, a análise de alguns sistemas operacionais e a aplicação desenvolvida para avaliar os impactos da adoção de um sistema operacional embarcado. No quinto capítulo é feita a discussão dos resultados e são apresentadas as conclusões do trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 SISTEMAS EMBARCADOS

Um sistema embarcado é qualquer sistema microcontrolado ou microprocessado dedicado a um propósito específico, o qual é um sistema computacional especializado e, geralmente, parte de um sistema maior (BASKIYAR e MEGHANATHAN, 2005). Por ser especializado, um sistema embarcado não pode ser facilmente utilizado para outras funções além daquelas para as quais foi projetado.

Sistemas embarcados apresentam comportamento autônomo na maior parte do tempo, realizando o seu trabalho com pouca ou nenhuma interferência do usuário. Em muitos casos a função do usuário é apenas gerar um evento, o qual inicia uma sequência de atividades no sistema.

Todos os sistemas embarcados são compostos por hardware e software. O hardware compreende processador, memórias e periféricos. O software é o programa executado pelo processador, podendo incluir um sistema operacional.

O processador pode ser qualquer microcontrolador, microprocessador, DSP ou uma combinação desses. Ao contrário do que ocorre com os computadores pessoais, existem muitas famílias diferentes de microcontroladores, microprocessadores e DSPs para o mercado embarcado e isso é necessário para atender a ampla gama de aplicações disponíveis. Por exemplo, a Texas Instruments possui em sua linha de processadores embarcados, incluindo o MSP430 (16 bits, de 4 a 25 MHz), ARM Cortex-M (32 bits, 20 a 80 MHz, USB, CAN, Ethernet), ARM9 e ARM Cortex-A8 (275 MHz a 1.5 GHz, USB, CAN, Ethernet, SATA, PCIe), DSP C6000, C5000 e DaVinci (até 1.5 GHz, aceleradores de vídeo, USB, CAN, Ethernet, SATA, PCIe), além da família de processadores OMAP, largamente utilizado em *smartphones* e *tablets* (até 2 GHz, multinúcleo).

De acordo com as necessidades do sistema, as memórias podem estar presentes no próprio *chip* do microcontrolador ou em *chips* externos. O armazenamento do software e dados constantes é feito em uma memória não volátil do tipo ROM ou Flash ou em um disco rígido. A memória RAM é utilizada para armazenamento de dados necessários durante a execução dos programas.

Um sistema embarcado é inserido em um ambiente para monitorar ou controlar as funções deste. A interface com esse ambiente para a leitura de informações é feita através de sensores (por exemplo, temperatura, umidade, pressão, movimento, tensão e corrente elétrica, etc.) e o controle através de atuadores (motores, solenoides, relés, válvulas, etc.).

Em muitos casos, um sistema embarcado deverá se comunicar com outros sistemas computacionais, o que ocorre através de interfaces e protocolos de comunicação como Ethernet, USB, I2C, SPI, RS-232, RS-485, Bluetooth, e ZigBee, entre outros, ou deve comunicar-se com usuários, através de botões, teclados, displays, telas sensíveis ao toque, etc.

A Figura 3 apresenta um exemplo de um sistema embarcado, expressado na forma de um digrama em blocos. Neste exemplo é utilizado um microcontrolador MSP430 da Texas Instruments, que possui memórias Flash e RAM, ADC, RTC, I2C, SPI, UART e USB integrados ao *chip*. A memória Flash é usada tanto para armazenamento do programa quanto para os dados medidos. As interfaces com o ambiente são através dos sensores de temperatura, pressão e velocidade, um motor para controle da válvula, *display* e interface de comunicação para transferência dos dados medidos para a central de monitoramento.

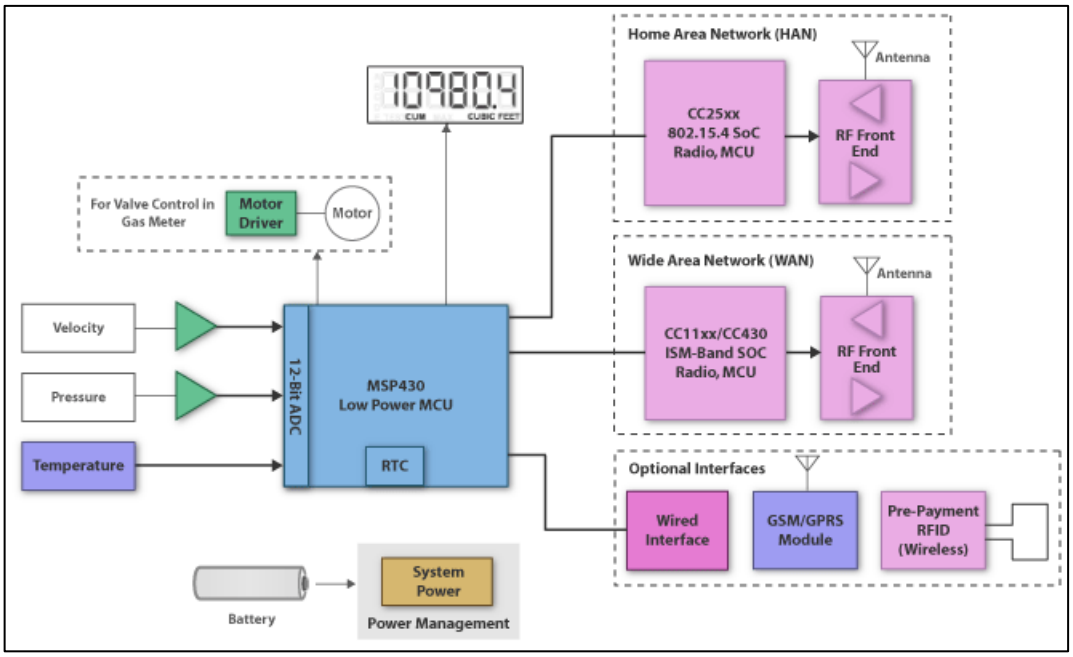


Figura 3 - Diagrama em blocos de um medidor de gás (TEXAS INSTRUMENTS, 2011b).

2.1.1 Requisitos de sistemas embarcados

Sistemas embarcados podem ser bastante complexos e possuir vários requisitos (FRIEDRICH, 2009a), os quais afetam as definições do hardware e software utilizado.

Os requisitos de um sistema são descrições dos serviços fornecidos e das suas restrições, classificados em funcionais e não-funcionais (SOMMERVILLE, 2007).

- Os requisitos funcionais definem a utilidade do sistema, isto é, são as descrições das atividades que o sistema deve ser capaz de executar. De maneira geral, requisitos funcionais especificam o que o sistema deve fazer.
- Os requisitos não-funcionais estão relacionados às propriedades do sistema, em vez de funções específicas deste. Geralmente, requisitos não-funcionais definem as qualidades do sistema, ou como este deve ser.

Os requisitos funcionais são particulares a cada sistema, pois definem suas funções, entradas e saídas.

Os requisitos não-funcionais especificam as restrições e limitações do sistema, podendo ser requisitos de produto, requisitos organizacionais ou requisitos externos.

- Requisitos de produto especificam características como tamanho, peso, aparência, desempenho e usabilidade.
- Requisitos organizacionais possuem relação com o processo de desenvolvimento, especificando padrões ou linguagens de programação.
- Requisitos externos estão relacionados à legislação, interoperabilidade e outros fatores externos ao sistema.

Friedrich (FRIEDRICH, 2009b) descreveu alguns requisitos não-funcionais que devem ser considerados em sistemas embarcados. Entre eles estão:

a) Limitação de recursos: Sistemas embarcados devem utilizar eficientemente os recursos disponíveis, pois frequentemente possuem sérias restrições.
b) Tempo real: Muitas aplicações dependem de tarefas que devem ser executadas em um tempo específico e cumprir requisitos temporais determinados.
c) Confiabilidade: Esta propriedade está relacionada à probabilidade de trabalhar continuamente por um tempo, com certa disponibilidade.
d) Robustez: Habilidade de funcionar aceitavelmente quando o sistema operar fora das suas condições nominais. Algumas aplicações devem suportar vibração, altas temperaturas ou flutuações na fonte de alimentação.
e) Estabilidade: Habilidade de conter falhas e impedir que estas afetem todo o sistema, especialmente em ambientes de rede.
f) Gerenciamento de falhas: Habilidade de produzir resultados coerentes mesmo em presença de falhas. Compreende detecção, confinamento, recuperação e tratamento.
g) Proteção: Capacidade de prevenir que informações ou o sistema sejam alterados ou utilizados por usuários não autorizados. Um sistema seguro é aquele no qual apenas uso intencional é permitido.
h) Segurança: Esta propriedade está relacionada com a prevenção da perda de vida ou danos às pessoas, aos equipamentos ou ao ambiente.
i) Privacidade: Habilidade de o sistema isolar informações sobre si mesmo e revelar estas informações seletivamente. Também está relacionado ao anonimato, podendo permanecer despercebido ou não identificado em um ambiente público.
j) Escalabilidade: Capacidade de lidar com quantidades crescentes de trabalho ou de ser facilmente ampliado.

k) Atualização: Capacidade de ter suas características melhoradas pela adição ou substituição de componentes.
l) Consumo de energia: Muitos sistemas embarcados são alimentados por bateria, de forma que seus componentes devem minimizar o consumo de energia.
m) Custo: A produção em larga escala e a competição industrial exigem produtos de baixos custos; assim, cada componente deve ser avaliado e seu custo minimizado.
n) Manutenção: Possibilidade de substituição de componentes defeituosos de hardware ou partes do software.
o) Usabilidade: Facilidade de utilização do sistema pelo usuário final.
p) Portabilidade: Está relacionada com a capacidade de migração do sistema para outra plataforma de hardware ou software.

2.1.2 Classificação quanto aos requisitos temporais

Os sistemas embarcados podem ser classificados em três categorias quanto aos requisitos temporais: sem tempo real (*non-real-time*), tempo real brando (*soft real-time*) e tempo real rígido (*hard real-time*).

Uma aplicação sem tempo real é aquela que não possui restrições temporais. Mesmo assim, é desejável que o processamento aconteça rápido, porém, qualquer atraso não causa prejuízos ao sistema.

Um sistema embarcado sem tempo real é projetado para ter um bom desempenho médio. Eventualmente, o sistema pode ficar sobrecarregado e apresentar uma degradação no seu desempenho, porém, sem causar danos. Em geral, esses sistemas são de tempo compartilhado, no qual os recursos são compartilhados entre os diversos processos, objetivando minimizar o tempo de resposta global, ou atender o maior número possível de processos dentro do prazo.

Entre os sistemas embarcados, aqueles que possuem restrições temporais são ditos sistemas embarcados de tempo real e o seu correto funcionamento depende não somente da exatidão da resposta, mas também da sua precisão temporal (LAPLANTE, 1996). Uma resposta no tempo errado é uma resposta errada.

Um sistema de tempo real não é obrigatoriamente rápido. Na verdade, um sistema de tempo real tem que responder a um estímulo ou executar suas tarefas no tempo pré-determinado. Respostas adiantadas ou atrasadas são indesejáveis. Por exemplo, um robô de solda em uma linha de montagem de carros. Esse robô deve se aproximar do carro a ser soldado e parar em uma posição específica. Se ele parar antes ou ficar afastado do carro, a solda ficará comprometida.

Um sistema embarcado possui tempo real rígido (*hard real time*) quando seu correto funcionamento depende da execução de tarefas com exatidão temporal. Sendo que o atraso na execução de uma tarefa leva ao mau funcionamento do sistema. Por exemplo, a unidade de controle do *airbag* em um carro deve continuamente ler os valores de vários sensores e precisamente decidir o momento correto de inflá-lo. Se o *airbag* demorar a ser inflado, este não estará exercendo sua função corretamente (Efficient Airbag ECU Tests, 2011).

Quando o sistema admitir a ocorrência de atrasos com certa probabilidade, ele possui requisitos brandos de tempo real (*soft real time*). Estes sistemas são capazes de continuar exercendo suas funções mesmo na presença uma falha temporal. Por exemplo, em um sistema de transmissão de vídeo, caso ocorra a perda de alguns quadros, apenas a qualidade será degradada.

2.2 SISTEMAS OPERACIONAIS

O sistema operacional suporta a execução das tarefas úteis para o sistema e para a aplicação. Ele facilita a atividade de programação, fornecendo e gerenciando o ambiente para a execução de cada aplicativo e implementando grande parte das funções comuns necessárias para os aplicativos. A utilização de um sistema operacional é uma maneira de simplificar o desenvolvimento do software, livrando os programadores do gerenciamento do hardware de baixo nível e fornecendo uma interface de programação simples para tarefas que ocorrem com frequência.

Segundo Tanenbaum (TANENBAUM, 2003), o sistema operacional consiste em uma camada de software que oculta o hardware e fornece ao programador um conjunto de instruções mais adequado.

Programar diretamente os dispositivos físicos exige que cada programador conheça a fundo os detalhes de cada componente. Para agilizar o desenvolvimento, os sistemas operacionais implementam a abstração do hardware. Isso consiste em criar uma interface de programação para acesso aos recursos do hardware, escondendo os detalhes de como isso é implementado. Como o sistema operacional está localizado entre os aplicativos e o hardware, a única maneira dos aplicativos acessarem o hardware é através de chamadas às funções do sistema operacional.

Outra função do sistema operacional é gerenciar e coordenar o acesso aos dispositivos de hardware. Caso mais de um processo queira acessar um mesmo dispositivo, cabe ao sistema operacional coordenar e permitir o acesso a apenas um processo de cada vez.

O sistema operacional também deve fornecer recursos para facilitar as tarefas de programação. Entre esses recursos estão o sistema de arquivos, a execução de programas na memória, a comunicação entre processos e o acesso aos periféricos.

2.2.1 Classificação quanto aos requisitos temporais

Os sistemas operacionais podem ser classificados de diversas maneiras. A classificação mais conveniente para esta dissertação é quanto à relação do sistema operacional com os requisitos temporais: propósito geral ou de tempo real.

2.2.1.1 Sistema operacional de propósito geral

Um sistema operacional de propósito geral (GPOS) é construído para fazer a divisão justa dos recursos entre todos os processos. O objetivo de um GPOS é atender todos os processos e dar ao usuário uma boa experiência do sistema.

Em geral os GPOS são sistemas de tempo compartilhado. O tempo do processador é dividido em pequenas fatias de tempo. O escalonador dá a um processo o direito de usar o processador durante a duração de uma fatia de tempo. Após o fim desse tempo, o escalonador retoma o processador e o entrega para outro processo. O sistema operacional pode dividir o tempo igualmente entre todos os processos ou implementar filas com diferentes prioridades.

Nesse último caso, o escalonador atenderia primeiro os processos presentes na fila de maior prioridade, para depois atender os processos de menor prioridade. Em alguns casos, a fila de maior prioridade pode apresentar comportamento de tempo real brando (SILBERSCHATZ, GALVIN e GAGNE, 2004).

Os GPOS são sistemas interativos, pois permitem a interação dos usuários por meio de periféricos como o mouse, teclado ou vídeo. Devido às suas características, possuem tempos de resposta na ordem de segundos, e são projetados para fornecer aos usuários uma boa experiência do sistema.

Os principais sistemas operacionais utilizados nos computadores pessoais são GPOS. Por exemplo, a família de produtos Windows da Microsoft destinada a computadores pessoais e as distribuições Linux. Esses sistemas operacionais devem ser GPOS porque devem suportar uma grande quantidade de dispositivos de hardware e aplicações. Quando um GPOS é desenvolvido, o fabricante não sabe exatamente qual a configuração do hardware no qual o sistema operacional será utilizado nem quais aplicativos serão utilizados pelos usuários, de forma que ele deve ser compatível com todas as configurações e aplicações.

2.2.1.2 Sistemas operacionais de tempo real

Os sistemas operacionais de tempo real (RTOS) possuem relação intrínseca com o tempo e são construídos para permitirem o cumprimento de requisitos temporais específicos. Quando o RTOS é determinístico e atende requisitos rígidos de tempo real, é dito ser *hard real time*. Quando o RTOS apresenta comportamento probabilístico, é dito ser *soft real time*.

Hard e *soft real time* são os dois pontos extremos no espectro de classificação dos sistemas operacionais quanto ao seu desempenho de tempo real (MELANSON e TAFAZOLI, 2003). Na prática, os RTOS são distribuídos entre esses dois pontos.

Para suportar múltiplas tarefas em aplicações de tempo real, o escalonador deve ser preemptivo baseado em prioridades (BASKIYAR e MEGHANATHAN, 2005). Nesse caso, o escalonador deve ser capaz de interromper qualquer tarefa para permitir a execução de outra com maior prioridade. Também, deve possuir vários níveis de interrupção para permitir o tratamento de interrupções com diferentes níveis de prioridade.

Para permitir que várias tarefas se comuniquem entre si em um tempo hábil, é necessário que a sincronização e a comunicação entre os processos sejam previsíveis. O tempo para iniciar uma tarefa, latência entre troca de tarefas e latência de interrupção devem ser bem definidos e consistentes mesmo com o aumento da carga do sistema (ANH e TAN, 2009).

Existem muitos RTOS. Exemplos conhecidos são o VxWorks, da Wind River, Windows Embedded Compact 7, da Microsoft, e Nucleus, da Mentor Graphics. Diferentemente dos GPOS, os desenvolvedores de sistemas embarcados configuram e personalizam o RTOS para cada aplicação. Desta forma, depois de configurado para uma aplicação, o RTOS terá suporte apenas àqueles serviços e hardware que forem requeridos.

2.2.2 Padrão POSIX

O POSIX (*Portable Operating System Interface*) (IEEE COMPUTER SOCIETY, 2008) é um conjunto de normas produzido pelo IEEE e aceito mundialmente, que especifica uma interface do sistema operacional para os aplicativos, com o objetivo de garantir a portabilidade do código fonte de um programa entre sistemas operacionais que adotem esse padrão. Os serviços abrangidos pelo POSIX 1003.1b (BASKIYAR e MEGHANATHAN, 2005) incluem:

a) Operações assíncronas de E/S: Habilidade em sobrepor o processamento dos aplicativos e operações de E/S. Para suportar operações de E/S em nível de usuário, um sistema embarcado deve suportar a entrega de interrupções externas de um dispositivo de E/S para um processo de uma maneira previsível e eficiente.
b) Operações síncronas de E/S: Habilidade de bloquear a utilização do processador pela tarefa que solicitou a operação de E/S, até que esta operação seja completada.
c) Bloqueio de memória: Habilidade de garantir a residência em memória de uma porção específica do espaço de endereçamento de um processo.
d) Semáforos: Habilidade de sincronizar o acesso aos recursos compartilhados por vários processos.
e) Memória compartilhada: Habilidade de mapear a memória física em espaços distintos para cada processo.

f) Escalonamento de tarefas: Aplicação de uma política para selecionar a próxima tarefa que terá direito ao uso do processador. Métodos de escalonamento incluem *round-robin*, cooperativo e preemptivo baseado em prioridade.
g) Temporizador: Um relógio particular que pode notificar uma tarefa quando o tempo chegou ou ultrapassou um valor específico, ou após uma quantidade determinada de tempo. Um sistema deve ter ao menos um temporizador (*system clock*) para fornecer serviços de tempo real de qualidade.
h) Comunicação entre processos: Mecanismo pelo qual os processos trocam as informações necessárias para a aplicação. Incluem *mailboxes* e *queues*.
i) Arquivos de tempo real: Habilidade de criar e acessar arquivos com desempenho determinístico.
j) Processos de tempo real: Entidades escalonáveis de uma aplicação de tempo real que possuem requisitos temporais individuais.

2.2.3 Gerenciamento de memória

A memória é um recurso limitado e caro. É função do gerenciador de memória prover os mecanismos para que os diversos processos em execução coexistam e compartilhem a memória de forma segura e eficiente.

O gerenciamento de memória pode ser estático ou dinâmico (BASKIYAR e MEGHANATHAN, 2005). No gerenciamento estático a memória do sistema é dividida em partições de tamanho fixo, que são alocadas aos processos. Cada processo utiliza apenas uma partição de memória. Quando um processo termina de utilizar uma partição de memória, esta é devolvida para o sistema. No gerenciamento dinâmico de memória, esta é dividida em partições de tamanho variável, sendo que cada processo ocupa apenas a quantidade de memória necessária.

Gerenciamento dinâmico de memória inclui *swapping*, *overlay*, multiprogramação e paginação por demanda. Multiprogramação consiste em alocar mais de um programa na memória principal. Para permitir que um programa maior que a memória disponível seja executado é empregada a técnica de *overlay*, a qual consiste em dividir o programa em blocos e permutar os blocos entre as memórias principal e secundária de acordo com a necessidade do sistema. *Swapping* consiste em transferir da memória principal para a secundária a área de

memória de um processo que não esteja em execução; quando for necessário, a região de memória é novamente transferida para a memória principal. Paginação por demanda consiste em dividir a memória em regiões de tamanho fixo que são alocadas aos programas sob demanda, sendo que estas regiões não precisam formar um espaço contínuo de memória.

2.2.4 Escalonamento de processos

O propósito do sistema operacional determina as características que devem ser observadas na escolha da política de escalonamento. Um GPOS tende a tratar todos os processos de maneira semelhante, dividindo o tempo de processamento igualmente entre todos os processos. Já um RTOS deve priorizar a execução das tarefas mais críticas, de maior prioridade, em detrimento de outros processos (MACHADO e MAIA, 2002).

O escalonador está intimamente relacionado aos comportamentos e objetivos do sistema operacional e deve seguir alguns princípios que podem variar de acordo com cada SO. Em todos os casos, o escalonador deve ser justo, aplicar as políticas do sistema e fazer o melhor uso dos recursos existentes (TANENBAUM, 2003). Em sistemas de tempo real, o escalonador deve cumprir os prazos de cada processo e garantir a qualidade do sistema. Em sistemas interativos, o escalonador deve responder rapidamente as requisições para minimizar os tempos de resposta e satisfazer as expectativas dos usuários.

O escalonamento dos processos pode ser preemptivo ou não-preemptivo (MACHADO e MAIA, 2002). No escalonamento preemptivo, o sistema operacional concede o direito de execução a um processo e pode retirar esse direito a qualquer momento. No escalonamento não-preemptivo, o processo em execução pode executar até que seja bloqueado ou intencionalmente libere o processador.

2.2.5 Sincronização e comunicação entre processos

Sistemas computacionais podem ser modelados como vários processos em paralelo, cada um executando uma tarefa específica. Em muitos casos, os processos precisam trocar informações ou acessar um recurso compartilhado; mecanismos de sincronização e comunicação entre processos dão suporte a estas operações.

A comunicação entre processos pode ser feita utilizando uma região compartilhada de memória. Essa abordagem, apesar de parecer simples, é muito suscetível a erros, e deve ser implementada pelo programador da aplicação.

Uma maneira mais eficiente de efetuar a comunicação entre processos é através da troca de mensagens. Esse recurso fornecido pelo sistema operacional é chamado de IPC - comunicação entre processos (SILBERSCHATZ, GALVIN e GAGNE, 2000). Nesse caso, os processos podem sincronizar suas ações e se comunicarem sem a utilização de memória compartilhada. A comunicação pode ser direta ou indireta. Na comunicação direta é criado um canal de comunicação contendo os dois processos envolvidos (*pipes*). Na comunicação indireta são utilizadas caixas de correio (*mailbox*). Cada caixa de correio tem uma identificação única, e é um local onde os processos podem colocar e retirar mensagens.

2.3 DESENVOLVIMENTO DE SISTEMAS EMBARCADOS

Diversas maneiras existem para o desenvolvimento de um sistema embarcado. Entre elas, destacam-se as três formas apresentadas na Figura 4: (a) sem sistema operacional, (b) com sistema operacional de tempo real, e (c) com sistema operacional de propósito geral.

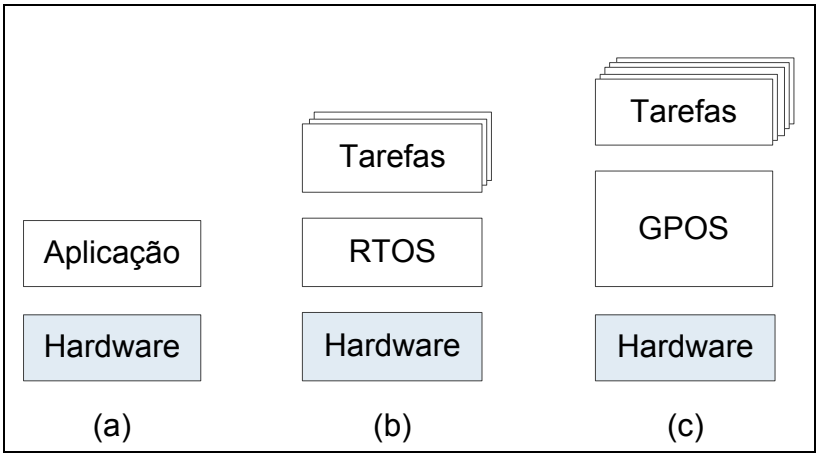


Figura 4 - Diferentes formas de implementação de um sistema embarcado (adaptado de WILLIAMS, 2006).

2.3.1 Sem sistema operacional

Aplicações simples, que não requerem muito suporte em tempo de execução, podem ser implementadas sem a utilização de um SO (Figura 4(a)). Todas as inicializações do hardware, configuração do software e gerenciamento dos dispositivos de hardware deverão ser incluídos no código da aplicação. Alguns programadores de sistemas de tempo real preferem esta abordagem porque, apesar dela requerer que mais software seja produzido, ela revela todos os aspectos do sistema, não escondendo nada atrás das complexidades de um sistema operacional (WILLIAMS, 2006).

A aplicação desenvolvida sem um SO chamada *Super Loop* ou *Endless Loop* (PONT, 2002) pode ser dividida em duas partes: inicialização e *loop* infinito. Na primeira parte, são configurados todos os dispositivos de hardware, temporizadores e variáveis da aplicação. A segunda parte consiste em um *loop* infinito contendo todas as tarefas que devem ser executadas.

Nessa metodologia, as tarefas são executadas sequencialmente, sempre na mesma ordem em que foram escritas. Quando chamada, cada tarefa é executada até o fim e o processamento somente retorna para o *loop* principal após o término da tarefa.

Para as atividades periódicas, que precisam ser chamadas com uma frequência mínima, é responsabilidade do programador garantir que isso ocorra, visto que o Super Loop não oferece nativamente nenhum mecanismo para facilitar a programação de atividades periódicas, sendo que o mesmo ocorre para as atividades de tempo real.

A Figura 5 contém um pseudocódigo representando o *loop* principal do jogo Space Invaders®, conforme apresentado por Laplante em (LAPLANTE, 1996). O jogo envolve, basicamente, ler o estado de três botões (mover para esquerda, mover para direita e atirar), mover os Aliens, verificar a ocorrência de colisões e atualizar a tela. É importante notar que a função que verifica o pressionamento das teclas ocorre três vezes mais frequentemente do que as demais, a fim de proporcionar respostas mais rápidas aos comandos do jogador.

```
for (;;;)
{
    check_for_keypressed();
    move_aliens();
    check_for_keypressed();
    check_for_collison();
    check_for_keypressed();
    update_screen();
}
```

Figura 5 - Exemplo de Super Loop (adaptado de LAPLANTE, 1996).

Se as funções chamadas no Super Loop forem relativamente curtas, aplicações eficientes e de boa qualidade poderão ser facilmente implementadas dessa forma.

O Super Loop não oferece garantias quanto à temporização, e o tempo entre as chamadas consecutivas a uma mesma função pode variar durante a execução, tempo esse que é dependente de todas as outras funções. Para garantir o tempo de execução de uma determinada atividade, pode ser utilizado um dispositivo de hardware que cause uma interrupção no sistema, fazendo com que essa atividade seja executada a partir da rotina de tratamento da interrupção. Porém, a utilização de interrupções diminui a previsibilidade do sistema e aumenta a sua complexidade. Além disso, geralmente a quantidade de níveis de interrupção é limitada.

A grande vantagem do Super Loop é o pouco consumo dos recursos de hardware, como memória e poder de processamento, além de não necessitar de nenhum periférico específico, como temporizadores. Dessa forma, ele costuma ser utilizado em aplicações mais simples, que executam poucas atividades e possuem pouca complexidade, ou quando o hardware utilizado não oferece muitos recursos. Outro ponto forte do Super Loop é sua simplicidade, o que o torna muito simples de desenvolver, depurar, testar e manter (PONT, 2001).

Além do Super Loop, outra abordagem possível para implementação de sistemas embarcados sem um sistema operacional é através do uso de interrupções (LAPLANTE, 1996). Nesse modelo de implementação, o *loop* principal consiste em apenas uma instrução *jump-to-self*. As várias tarefas do sistema são controladas via interrupções de hardware ou software nas rotinas de tratamento de interrupções. Quando o escalonamento por hardware é usado, um temporizador ou outro dispositivo externo gera sinais de interrupções, que são direcionados a um tratador de interrupções. O controlador de interrupções atende às

interrupções de acordo com a ordem de chegada ou prioridade das mesmas. A rotina de tratamento da interrupção será responsável pela decisão de qual tarefa será executada.

2.3.2 Sistema operacional de tempo real

Outra maneira de implementar um sistema embarcado é através da utilização de um sistema operacional de tempo real (Figura 4(b)). Um RTOS é um tipo de sistema operacional desenvolvido para cumprir os requisitos temporais da aplicação. As principais funções de um RTOS são escalonar a execução das tarefas em tempo hábil, gerenciar os recursos do sistema, e fornecer uma plataforma consistente para o desenvolvimento de aplicações (LI e YAO, 2003).

As aplicações desenvolvidas em um RTOS podem ser bastante diversas, indo desde um simples relógio cronômetro até uma aplicação mais complexa para um sistema de navegação em uma aeronave, por exemplo. Um RTOS deve portanto ser escalável a fim de cumprir diferentes configurações de requerimentos para diferentes aplicações.

Alguns RTOS compreendem apenas um *kernel*, o qual é o principal componente do sistema e fornece a lógica mínima, escalonamento e gerenciamento de recursos. Embora todos os RTOS possuam um *kernel*, alguns oferecem uma combinação de vários módulos, incluindo um *kernel*, um sistema de arquivos, protocolos de comunicação e outros componentes requeridos para uma aplicação em particular.

A Figura 6 mostra uma representação em camadas de uma aplicação genérica, desenvolvida utilizando um RTOS. Estão representados alguns serviços comumente oferecidos pelo RTOS. Tanto o RTOS como a aplicação podem se comunicar com o hardware, o que é feito por meio do BSP (*Board support package*), que apresenta uma abstração do hardware e fornece funções para acesso ao mesmo.

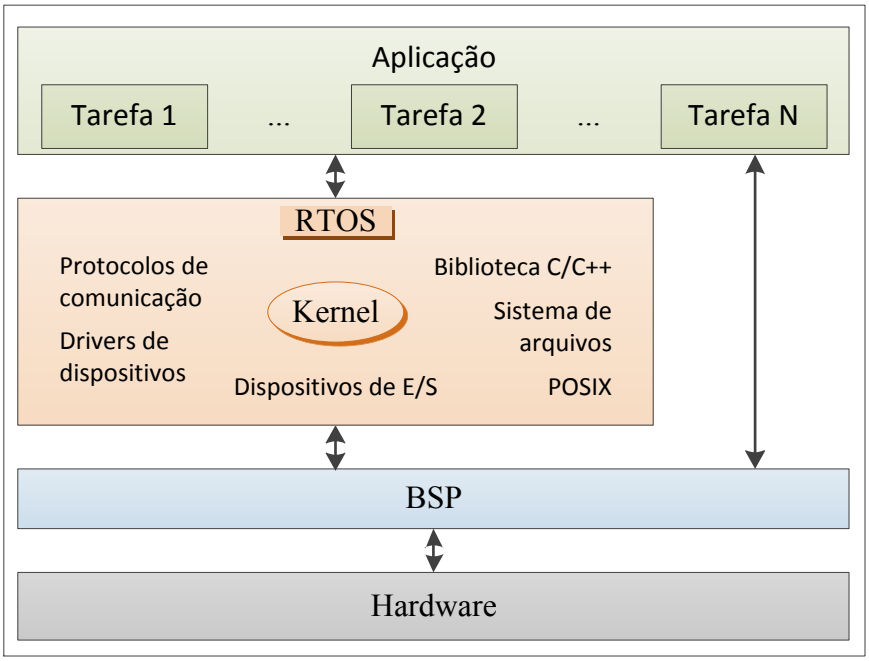


Figura 6 - Representação em camadas de um SE contendo um RTOS.

Uma aplicação desenvolvida com um RTOS pode ser dividida em várias tarefas, as quais utilizam os serviços fornecidos pelo RTOS, que também é responsável pelo seu escalonamento. Tal aplicação é composta por várias pequenas tarefas que se comunicam entre si, cada tarefa possuindo sua função específica. Essa é a vantagem mais evidente a favor da utilização de um RTOS. Além disso, os RTOS facilitam a modularização, a reutilização de código fonte e o desenvolvimento de tarefas periódicas ou de tempo real.

Exemplos de RTOS são o VxWorks, da Wind River, Windows Embedded Compact 7, da Microsoft, e Nucleus, da Mentor Graphics.

2.3.3 Sistema operacional de propósito geral

Sistemas embarcados também podem ser implementados utilizando-se um sistema operacional de propósito geral (Figura 4(c)). Um GPOS difere de um RTOS por não garantir o cumprimento dos requisitos temporais, apesar de alguns sistemas operacionais possuírem prioridades no escalonamento e conseguirem atender requisitos temporais leves. Além disso, os GPOS oferecem uma quantidade muito maior de serviços que os RTOS.

A grande desvantagem da utilização de um GPOS é o maior consumo de recursos do sistema, tais como memória de programa, memória de dados e poder de processamento, além do fato de não atenderem requisitos temporais rígidos.

Assim como os RTOS, os GPOS embarcados também são bastante configuráveis, sendo possível criar aplicações contendo apenas os serviços requeridos e com *footprint* bastante otimizado se comparado às versões não embarcadas.

Uma vantagem da utilização de um GPOS embarcado é que a forma de desenvolvimento de aplicações é muito semelhante àquela adotada para computadores pessoais, exigindo menos tempo para formação dos programadores. Frequentemente, os processos podem ser implementados como aplicações separadas; isso permite um projeto mais modular, com pequenas aplicações executando tarefas bem específicas, o que é mais fácil de desenvolver, usar e testar.

Exemplos de GPOS para sistemas embarcados são Windows Embedded Standard 7, da Microsoft, Linux e uClinux.

O Windows Embedded Standard 7 é a versão embarcada do sistema operacional Windows 7 da Microsoft para computadores pessoais. Ele apresenta a mesma interface gráfica da versão *desktop*, além de aplicativos como Internet Explorer, Windows Media Player, e suporte ao Silverlight e ao .net Framework. Entre suas vantagens constam: facilidade de portar aplicações e *drivers* originalmente desenvolvidos para a versão *desktop*; redução no *time-to-market,* devido ao aproveitamento do conjunto familiar de ferramentas e conhecimentos de desenvolvimento para *desktop*; seleção de apenas os *drivers*, serviços e aplicações necessárias, resultando em um *footprint* otimizado; suporte para as arquiteturas x86 e x64.

O sistema operacional Linux foi portado para diversas plataformas, inclusive várias embarcadas. Existem versões livres, sem suporte, além de versões distribuídas por algumas empresas, com suporte comercial. Exemplos desta última são MontaVista, Timesys e Wind River, com suporte para as arquiteturas x86, AMD64, ARM, MIPS, PowerPC e SuperH.

O uClinux (pronunciado “you-see-Linux”) é um porte do Linux para sistemas sem unidade de gerenciamento de memória (MMU) (tipicamente, microcontroladores e DSPs). Possui suporte para diversas arquiteturas, entre elas ARM, Blackfin, NIOS II, Coldfire, MIPS,

m68k e MicroBlaze. Além do *kernel*, o uCLinux possui diversas aplicações, bibliotecas e *drivers* portados do Linux padrão. O uClinux é altamente configurável e é possível, com um pouco de trabalho, incluir apenas os serviços, *drivers* e aplicações necessários, resultando em um *footprint* otimizado em relação ao Linux de *desktop*. Além disso, a maioria das ferramentas de desenvolvimento do Linux também podem ser utilizadas no uCLinux.

2.4 TRABALHOS RELACIONADOS

Esta seção lista os trabalhos que possuem relação com o tema desta dissertação. Os trabalhos de Luís Fernando Friedrich (FRIEDRICH, 2009a e FRIEDRICH, 2009b) contribuem com uma classificação de sistemas embarcados e vários requisitos que devem ser considerados durante um projeto embarcado. Além disso, alguns sistemas operacionais são avaliados quanto ao atendimento desses requisitos.

Greg Hawley (HAWLEY, 1999) apresentou, para a revista "*Embedded Systems Design*", o problema da escolha de um sistema operacional, algumas vantagens na utilização de um SO, e discutiu se é melhor comprar ou desenvolver seu próprio sistema operacional. Também foram introduzidos alguns critérios para a seleção de um RTOS.

Melanson e Tafazoli (MELANSON e TAFAZOLI, 2003), da agência espacial Canadense, propõem uma metodologia para a seleção de um RTOS para aplicações espaciais. A principal contribuição deste trabalho foi a apresentação de uma lista de critérios e um método de atribuição de pesos para auxiliar no processo de seleção e classificação de um RTOS.

A dissertação de mestrado de Moreira (MOREIRA, 2007) se assemelha a este trabalho no ideal de facilitar ao projetista a escolha entre sistemas operacionais de tempo real. Porém, o foco desse trabalho foi examinar as características dos sistemas operacionais Windows CE (Microsoft Windows Embedded CE, 2011b), RTLinux (RTLinux, 2011) e RTAI (RTAI, 2011) para plataforma x86. Esses sistemas operacionais foram comparados quanto às suas características técnicas e através de resultados obtidos de *benchmarks*. A contribuição desse trabalho foi a listagem e a análise das características técnicas dos sistemas operacionais analisados.

Finalmente, em sua dissertação de mestrado, Aroca (AROCA, 2008) fez uma análise qualitativa e quantitativa de alguns sistemas operacionais de tempo real, com o objetivo de verificar as capacidades, vantagens e desvantagens na implementação de aplicações de robótica e automação sobre a arquitetura x86. A contribuição deste trabalho foi a análise experimental de algumas características dos sistemas operacionais, como pior tempo de resposta e latência para tratamento de interrupções.

3 DESENVOLVIMENTO

3.1 INTRODUÇÃO

A utilização de um sistema operacional é uma maneira de simplificar o desenvolvimento do software, livrando os programadores do gerenciamento do hardware de baixo nível e fornecendo uma interface de programação simples para tarefas que ocorrem com frequência. Porém, devido às demandas adicionais impostas por um sistema operacional (SO), os desenvolvedores de sistemas embarcados enfrentam a crítica decisão quanto à adoção ou não de um SO.

Neste capítulo, apresenta-se uma série de critérios, baseados em requisitos comuns aos sistemas embarcados, a fim de auxiliar os projetistas de tais sistemas a decidirem pela utilização ou não de um sistema operacional. Além disso, outros critérios são apresentados com o intuito de guiar a seleção do sistema operacional mais adequado às características do projeto.

A partir deste momento, em que passa a ser considerada a utilização de um sistema operacional embarcado, considera-se que o projeto já passou por alguma etapa de planejamento e especificação. Independentemente da metodologia adotada para o gerenciamento do projeto, provavelmente há pelo menos uma etapa destinada à definição dos objetivos e elaboração dos requisitos do produto sendo desenvolvido. É nesta etapa, durante a especificação do software, que será decidido pela utilização ou não de um sistema operacional. Ainda mais, caso a decisão seja favorável à adoção, deverão ser levantados os requisitos gerais do projeto, possibilitando a escolha do sistema operacional mais adequado.

A engenharia de requisitos é a disciplina responsável pelo processo de descobrir, analisar, documentar e validar os requisitos do sistema (SOMMERVILLE, 2007). No caso de sistemas embarcados, isso inclui requisitos de hardware e software, e é a partir dos resultados

deste processo que se extraem as características do projeto que serão utilizadas neste processo de decisão pela adoção de um sistema operacional e sua seleção.

3.2 CRITÉRIOS PARA ADOÇÃO DE UM SISTEMA OPERACIONAL

A decisão pela utilização ou não de um sistema operacional embarcado será tomada a partir da análise dos requisitos do projeto. Esta seção ajuda o desenvolvedor a decidir pela utilização ou não de um sistema operacional embarcado. Para assistir nesta decisão, serão utilizados as características e os requisitos do projeto, como o processador utilizado, a quantidade de memória e as restrições temporais.

Os critérios apresentados nas seções a seguir são complementares entre si, ou seja, o contrário de tudo o que for apresentado como razão para a não utilização de um SO serve como justificativa para utilização de um SO, da mesma forma que o oposto das razões para a utilização de um SO são motivos para sua não utilização.

3.2.1.1 Razões para a utilização de um sistema operacional

Muitas razões favorecem a utilização de um sistema operacional embarcado, sendo que as principais estão listadas na Tabela 1. Os critérios apresentados foram classificados em funcionais e não-funcionais. De forma geral, a presença de apenas umas destas características não obriga a utilização de um SO; porém, quanto mais destas características estiverem presentes no projeto, maior será a recomendação pela utilização de um SO.

Sistemas computacionais são geralmente modelados como uma série de tarefas executando ao longo do tempo, frequentemente com muitas tarefas executando simultaneamente. Os sistemas operacionais fornecem mecanismos de escalonamento de tarefas, os quais são complexos e exigem grande esforço de programação para serem implementados. Dessa forma, é preferível a utilização de um sistema operacional quando há várias tarefas concorrentes. Além disso, um escalonador baseado em prioridades permite a fácil programação de tarefas com diferentes prioridades. O mesmo ocorre para as tarefas periódicas, que devem ser executadas com determinada frequência, ficando a cargo do escalonador a execução destas tarefas no momento correto.

Tabela 1 - Razões para a utilização de um sistema operacional

Natureza	Critério
Funcional	Várias tarefas concorrentes
	Tarefas com diferentes prioridades
	Tarefas periódicas
	Tarefas de tempo real
	Processador multinúcleo
	Gerenciamento de memória por hardware
	Interfaces de comunicação de alta complexidade
	Compartilhamento de recursos
	Comunicação entre processos
	Interface homem máquina
	Interface gráfica do usuário
Não-funcional	Curto tempo para desenvolvimento (time-to-market)
	Confiabilidade
	Portabilidade

Fonte: Própria.

As tarefas de tempo real são aquelas que devem ser completadas em um determinado prazo (*deadline*). Em sistemas simples, estes requisitos temporais podem ser gerenciados com temporizadores e interrupções. Entretanto, quando o número de tarefas de tempo real aumenta, torna-se mais difícil garantir que todos os prazos sejam atendidos. Para contornar estas dificuldades, deve ser utilizado um RTOS, o qual foi projetado para cumprir os requisitos temporais das tarefas.

Quando o sistema em desenvolvimento utiliza um processador multinúcleo, é recomendada a utilização de um sistema operacional por dois motivos principais. Primeiro, o SO pode distribuir processos convencionais entre os núcleos disponíveis, mesmo sem interferência do desenvolvedor. Segundo, no caso de aplicações especificamente projetadas para aproveitar mais de um núcleo de processamento, um SO pode oferecer suporte adicional

para explorar o paralelismo ao nível da aplicação e coordenar a comunicação entre os núcleos (TOMIYAMA, HONDA e TAKADA, 2008).

Em ambientes executando vários processos, deve-se tomar cuidado para evitar interferências entre regiões de memória usadas por diferentes processos. Alguns processadores possuem um hardware dedicado para auxiliar nesta tarefa, como a Unidade de Gerenciamento de Memória (MMU) e a Unidade de Proteção de Memória (MPU), as quais permitem que cada processo seja executado em uma região separada, isolada e protegida de memória, evitando conflitos de endereçamento e impedindo que uma falha em uma tarefa afete as demais tarefas. A utilização dessas unidades de hardware é uma tarefa rotineira, porém complexa, de forma que geralmente é mais bem aproveitada se for oferecida como um serviço para a aplicação, por um sistema operacional.

As interfaces de comunicação permitem a troca de informações entre sistemas e podem ser classificadas em *local*, *ponto a ponto* e *rede*. A primeira é para comunicação interna ao sistema; por exemplo, PCI, SPI e I2C. A segunda é para comunicação entre apenas dois sistemas; por exemplo, USB e RS-232. A última é para comunicação entre vários sistemas; por exemplo, Ethernet e Bluetooth. Algumas interfaces de comunicação são complexas o suficiente para desencorajar sua utilização sem um SO. Se uma ou mais destas interfaces de comunicação for necessária, um SO com suporte embutido é altamente recomendável.

Em um sistema composto por várias tarefas, pode acontecer que mais de uma tarefa precise acessar um mesmo recurso de hardware, assim será necessário proteger o acesso a este recurso para que apenas uma tarefa o acesse em um dado momento. Em sistemas compostos por poucas tarefas e poucos recursos compartilhados, a proteção do acesso pode ser feita, sem maiores complicações, pelo desenvolvedor da aplicação. Entretanto, à medida que o número de tarefas e recursos compartilhados aumentam, é conveniente a utilização dos mecanismos fornecidos pelos sistemas operacionais, tais como exclusão mútua e semáforos.

Em muitos casos, as tarefas ou processos necessitam trocar informações entre si, o que pode ser conseguido utilizando regiões compartilhadas de memória, ou através de mecanismos mais complexos fornecidos pelos sistemas operacionais. No Super Loop, a utilização de regiões compartilhadas de memória ocorre sem maiores problemas, pois a sua execução é sequencial e as escritas e leituras acontecem nos momentos definidos pelo

programador; porém, em sistemas multitarefa, faz-se necessária a sincronização das tarefas envolvidas. Dessa forma, a solução correta quando utilizando um SO é por meio dos mecanismos específicos para comunicação e sincronização entre tarefas, tais como *queues*, *pipes* e *mailbox*.

Sistemas operacionais auxiliam o desenvolvimento de interfaces homem-máquina, uma vez que cada dispositivo de entrada ou saída pode ser tratado por uma tarefa separada. Adicionalmente, sistemas operacionais lidam com os dados brutos obtidos destes dispositivos e entregam informação de alto nível para as aplicações. Por exemplo, o pressionamento de uma tecla pode ser automaticamente traduzido para um caractere disponibilizado em um *buffer* de entrada.

De forma geral, a utilização de uma interface gráfica do usuário pressupõe a utilização de um sistema operacional. Algumas interfaces simples podem ser programadas sem necessidade de um SO; porém, interfaces mais ricas em funcionalidades e visualmente atrativas fazem uso de recursos presentes em sistemas operacionais.

Time-to-market é definido como a quantidade de tempo entre a concepção inicial do produto e sua disponibilidade para venda. O tempo de vida refere-se a quanto tempo o produto permanecerá disponível no mercado. De forma geral, a utilização de um SO reduz o *time-to-market* e aumenta a vida útil dos produtos, de acordo com (GREEN HILLS SOFTWARE, 2011), (MICROSOFT, 2011a), (WIND RIVER, 2011) e (LYNUX WORKS, 2011). A utilização de um SO pode levar a uma redução no tempo de desenvolvimento por vários motivos. Primeiro, grande parte das tarefas de baixo nível é realizada de forma confiável pelo SO, isentando o desenvolvedor de lidar com tarefas onerosas e propensas a erro. Segundo, os desenvolvedores podem se beneficiar de uma base de códigos existente, facilitando a reutilização de código fonte. Finalmente, a utilização de um SO estabelece um desenvolvimento mais modular, facilitando a integração, a manutenção e a atualização do software.

A confiabilidade é a capacidade de atender a especificação, dentro de condições definidas, durante certo período de funcionamento e condicionado a estar operacional no início do período (WEBER, 2011). A utilização de um SO melhora a confiabilidade do sistema embarcado quando aquele consiste em um conjunto maduro de códigos que foram testados, verificados e validados.

A utilização de um sistema operacional, que suporte várias arquiteturas de processadores, permite que o código desenvolvido para uma arquitetura seja facilmente, com pouca ou nenhuma modificação, portado para outra arquitetura. Em muitos projetos, a portabilidade é um requisito importante e que justifica a adoção de um sistema operacional.

3.2.1.2 Razões para a não utilização de um sistema operacional

A configuração do hardware é um dos principais fatores analisados quanto à utilização de um sistema operacional. Se o hardware for muito limitado em recursos, estará impondo restrições à utilização de um sistema operacional, visto que quanto mais complexo for o SO, mais recursos de hardware serão exigidos. A Tabela 2 apresenta as principais razões para a não utilização de um sistema operacional embarcado.

Tabela 2 - Razões para a não utilização de um sistema operacional

Natureza	Critério
Funcional	Consumo de processamento pelo SO
	Quantidade de memória de programa requerida pelo SO
	Quantidade de memória de dados requerida pelo SO
Não-Funcional	Produtos de extremo baixo custo

Fonte: Própria.

O processador é o componente responsável pela execução do software, seja um sistema operacional ou não. Devido a algumas rotinas do SO deverem ser reescritas para cada arquitetura de processador, apenas alguns sistemas operacionais serão compatíveis com cada processador. Uma característica importante dos processadores é a sua frequência de operação, a qual determina a velocidade com que as instruções de software serão executadas. Normalmente, processadores mais simples operam em frequências baixas e podem ser mais bem aproveitados se o software for implementado sem um sistema operacional. Isso porque o sistema operacional consiste em rotinas de software que estarão competindo pelo uso do processador com o software da aplicação.

A fim de se verificar a viabilidade da utilização de um sistema operacional em um determinado processador, pode se avaliar o tempo de execução do SO em relação ao tempo

total da aplicação. Espera-se que o tempo gasto para a execução das tarefas rotineiras do SO seja o mínimo possível a fim de destinar a maior parte do tempo da CPU para a execução das tarefas relacionadas com a aplicação. Desta forma, a utilização de um SO não seria recomendada, por exemplo, caso este utilize mais de 10% do tempo de processamento necessário para a aplicação.

A memória de programa é onde o software e os dados constantes são armazenados, enquanto a memória de dados é utilizada durante o tempo de execução para armazenamento dos dados temporários. De maneira geral, quanto mais funcionalidades um sistema operacional oferecer, mais memória será necessário. Usualmente, aplicações simples utilizam apenas as memórias internas dos microcontroladores, as quais, geralmente, estão disponíveis em pouca quantidade.

Se forem comparadas as quantidades de memória de programa e memória de dados requeridos pelo SO, com o total disponibilizado pelo microcontrolador escolhido, obtem-se um parâmetro quantitativo para a avaliação da viabilidade de utilização de cada SO. Se os requisitos de memória de programa e memória de dados forem inferiores a 10 vezes a quantidade memória presente no microcontrolador, a utilização de um SO é possível; caso contrário, não é recomendada.

O preço-alvo do produto final é um fator não técnico com muita influência quanto à utilização de um sistema operacional. Produtos focados em baixo custo podem não comportar o licenciamento de um sistema operacional comercial. Além disso, a opção pela não utilização de um SO pode implicar na utilização de componentes mais baratos, como um processador mais simples, com menor poder de processamento e menor quantidade de memória.

3.3 CRITÉRIO PARA SELEÇÃO DE UM SISTEMA OPERACIONAL

Se após considerar os critérios apresentados na seção 3.2 uma decisão for feita pela adoção de um SO, ainda é necessário selecionar o mais adequado. Nesta seção, são apresentados na Tabela 3 os principais critérios funcionais e não-funcionais para auxiliar a seleção do SO mais adequado. Estes critérios devem ser fiéis às características do projeto a fim de que o SO mais adequado seja escolhido.

O fabricante do SO, que é responsável pelo seu desenvolvimento e manutenção, geralmente é um fator de preocupação para projetos de longa duração. Adicionalmente, políticas corporativas podem exigir que os componentes de software sejam de empresas confiáveis. Além disso, podem ser considerados o histórico e a reputação do SO, baseados na sua utilização por outras empresas e prêmios recebidos.

Outro fator com implicações técnicas e comerciais é o modelo de licenciamento do sistema operacional, o qual pode ser distribuído sob uma licença proprietária ou de software livre. A licença pode impor algumas restrições no uso, modificação e redistribuição do SO. Por exemplo, a licença pública GNU (GPL) (FREE SOFTWARE FOUNDATION, 2007) exige que todas as modificações feitas no código fonte sejam disponibilizadas aos demais desenvolvedores. Assim como as licenças de software livre, algumas licenças proprietárias também permitem o acesso ao código fonte e direitos de modificação, mas sem a necessidade de tornar públicas as alterações.

A opção por um SO comercial tem relação direta com o orçamento do projeto e o preço final do produto. De um ponto de vista, há um aumento nos custos com o licenciamento do SO e possível treinamento da equipe de desenvolvimento. Por outro lado, há uma redução nos custos com a utilização de códigos existentes e de funcionamento comprovado.

Muitos custos estão envolvidos com a adoção de um SO. No caso de um SO comercial, os custos mais evidentes são a taxa inicial e os *royalties* por unidade. Alguns sistemas operacionais são comercializados em módulos, permitindo que apenas os componentes utilizados sejam adquiridos. Além disso, podem ser incluídos os custos de suporte técnico e aquisição de ferramentas.

Quando um SO comercial é adquirido, geralmente este inclui suporte por um período inicial, além do qual, horas adicionais podem ser contratadas. Por outro lado, os sistemas operacionais gratuitos não possuem nenhuma garantia de suporte, o que pode ser considerado um risco adicional pelos gestores do projeto.

Tabela 3 - Principais critérios funcionais e não-funcionais para seleção de um sistema operacional

Natureza	Critério
Funcional	Suporte ao processador utilizado
	Suporte a processadores multinúcleo
	Requisitos de tempo real
	Gerenciamento de memória por hardware
	Memory footprint
	Comunicação e sincronização entre processos
	Protocolos de comunicação
	Sistemas de arquivos
	Suporte a cartão de memória
	Funcionalidades de interface gráfica do usuário
	Suporte à atualização do software (SO e aplicação)
	Suporte a áudio e vídeo
	Compatibilidade com APIs
Não Funcional	Fabricante
	Histórico e reputação
	Formas de licenciamento
	Custo total
	Disponibilidade do código fonte
	Ferramentas de desenvolvimento
	Suporte técnico
	Certificação

Fonte: Própria.

Ferramentas de desenvolvimento são produtos de software e hardware, auxiliares no processo de desenvolvimento. Apesar do grande impacto que tais ferramentas possuem sobre a produtividade, sua importância é frequentemente negligenciada. Além disso, para muitas famílias de processadores, há poucas ferramentas de qualidade disponíveis. As ferramentas de desenvolvimento mais comuns são o ambiente integrado de desenvolvimento (IDE), compiladores, *profilers*, depuradores, simuladores e emuladores. Se uma ferramenta adequada para solucionar um problema não estiver disponível, o desenvolvedor será obrigado a improvisar, resultando em aumento do esforço de desenvolvimento e, em alguns casos, baixa qualidade.

Em projetos nos quais existe a possibilidade da substituição do processador por um de outra arquitetura, o suporte do sistema operacional para diferentes famílias de processadores é de grande importância. Nestes casos, escolher um sistema operacional que suporta várias arquiteturas aumenta a portabilidade do projeto.

A escolha entre um sistema operacional de tempo real e outro sem tempo real depende da aplicação, e deve ser feita de acordo com os requisitos e especificações desta. Se uma aplicação possuir poucos *deadlines* e a carga da CPU for baixa durante a maior parte do tempo, é possível alcançar um desempenho razoável com um sistema operacional sem tempo real. Entretanto, à medida que o número de tarefas de tempo real e processos concorrentes aumentam, um RTOS torna-se a melhor escolha.

Em muitos casos, é possível aproveitar a existência de bibliotecas de software, ou códigos fonte escritos para outros projetos, o que pode causar impactos positivos no tempo de desenvolvimento e orçamento do projeto. A fim de melhorar a portabilidade de código fonte entre sistemas, algumas interfaces de programação de aplicativos (API) foram criadas, como, por exemplo, POSIX (1003.1-2008 - IEEE Standard for Information Technology - Portable Operating System Interface (POSIX(R)), 2009), ITRON (ITRON Project Archive) e OSEK/VDX (OSEK VDX Portal, 2011).

Uma certificação é uma declaração de conformidade com um determinado conjunto de regras. De forma geral, as certificações estão relacionadas à segurança do produto e ao impacto que este pode causar em caso de falha. Certificações são exigências para aceitação de produtos em algumas indústrias, tais como médica, transporte aéreo e energia nuclear. Exemplos de certificações relacionadas a sistemas embarcados são IEC 61508 (aplica-se a

todos os sistemas embarcados, são especificados quatro níveis de segurança - *Safety Integrity Level*, SIL – de acordo com a probabilidade de ocorrência de falhas), DO-178B (aplica-se a todos os softwares presentes em aeronaves; são classificados em cinco níveis de acordo com a gravidade causada por possíveis falhas), DO-278 (estende a certificação DO-178B para softwares de controle de tráfego aéreo), IEC 62304 e FDA-1252 (aplica-se aos softwares utilizados em equipamentos médicos, o software é classificado em três níveis de referência de acordo com severidade das consequências de uma falha na execução do software).

Muitos sistemas embarcados são construídos utilizando uma memória reprogramável, como, por exemplo, a memória Flash. Isso adiciona mais flexibilidade ao produto, permitindo que o software seja atualizado mesmo após o produto ser enviado ao cliente, o que pode aumentar a vida útil do produto. Esta atualização consiste basicamente no download do novo software para a memória não volátil e o *boot* do sistema a partir do novo software. Métodos de atualização do software incluem a utilização de um protocolo serial RS-232 (MICRIUM, 2011a), USB (MENTOR GRAPHICS, 2007), protocolo TFTP via Ethernet, ou até mesmo através de uma rede sem fio, como, por exemplo, *over-the-air programming* (LIBELIUM COMUNICACIONES DISTRIBUIDAS, 2011) e *firmware over-the-air* (QNX SOFTWARE SYSTEMS LIMITED, 2011).

Memory footprint se refere à quantidade total de memória de programa e dados utilizada por uma aplicação, neste caso, o sistema operacional. O consumo de memória de um SO está relacionado com a quantidade de funcionalidades oferecidas por ele. De maneira geral, os sistemas operacionais embarcados são bastante modulares e configuráveis, o que permite ao desenvolvedor configurá-lo de acordo com as suas necessidades, resultando em um consumo de memória bem otimizado.

De forma geral, é possível utilizar um sistema operacional sem suporte ao gerenciamento de memória em um processador com MMU, mas não o contrário. Já no caso de um processador multinúcleo, o total aproveitamento do mesmo só acontecerá com o suporte específico do SO.

Em geral, os protocolos de comunicação são complexos e exigem conhecimentos profundos do sistema operacional e do protocolo para sua implementação, sendo portanto preferível que o SO suporte todos os protocolos de comunicação necessários no projeto. O mesmo deve ocorrer para sistemas de arquivos.

O suporte a cartões de memória também inclui suporte aos sistemas de arquivos necessários. Além disso, caso seja um requisito da aplicação, deve-se verificar se o sistema operacional permite a inserção, a remoção ou a troca do cartão de memória em tempo de execução (*hot swapping*).

As interfaces do usuário são formas de comunicação entre o sistema embarcado e o usuário. Estas são geralmente implementadas como uma série de processos executando em paralelo, que continuamente leem ou atualizam um hardware dedicado. Há várias formas de se interagir com o usuário, entre elas teclados, botões, *displays* de caracteres ou gráficos e telas sensíveis ao toque.

A escolha de um sistema operacional embarcado com interface gráfica vai muito além de verificar se o fabricante indica a presença de uma interface gráfica. É necessário visualizar a mesma para avaliar se esta atende aos requisitos do produto. Adicionalmente, algumas aplicações podem necessitar de recursos de áudio e vídeo, como decodificação de música em formato MP3.

4 RESULTADOS

4.1 INTRODUÇÃO

A fim de validar os critérios apresentados na seção 3.3 referentes à seleção de sistemas operacionais, foram escolhidos 15 sistemas operacionais embarcados para serem analisados utilizando os critérios propostos. Os resultados obtidos constam na seção 4.2, servindo como início para a seleção de um sistema operacional.

Em seguida, a fim de avaliar o impacto da adoção de um sistema operacional em um projeto embarcado, é apresentado na seção 4.3 um estudo de caso no qual uma aplicação modelo (uma estação meteorológica embarcada) foi desenvolvida em três diferentes cenários: sem um SO, usando um RTOS (µC/OS-II), e usando um GPOS (uClinux). Estes sistemas operacionais satisfazem os critérios de seleção propostos anteriormente.

4.2 ANÁLISE DE SISTEMAS OPERACIONAIS

De acordo com o que foi apresentado no capitulo 3, os requisitos do projeto serão utilizados para a decisão sobre a utilização de um SO e, em seguida, caso se opte pela sua utilização, para escolher um que seja o mais adequado para o projeto. O objetivo desta seção é auxiliar no processo de seleção, avaliando alguns sistemas operacionais de acordo com os critérios propostos na seção 3.3.

Para essa avaliação, foram selecionados 15 sistemas operacionais, dentre os quais 8 são *open source*, mostrados na Tabela 4, e 7 são comerciais, mostrados na Tabela 5. A divisão entre *open source* e comerciais mostrou-se necessária devido à importância desta característica, a qual é considerada um dos principais critérios em muitos projetos.

O critério utilizado para a escolha destes sistemas operacionais foi sua popularidade, avaliada de acordo com a pesquisa realizada pelo site *EETimes* (EETIMES GROUP, 2010). Assim, foram selecionados os sistemas operacionais mais conhecidos e que podem abranger uma grande variedade de projetos.

Uma lista mais completa de sistemas operacionais pode ser encontrada em (OSDEV, 2011) e (WIKIPEDIA, 2011a). Outra lista, contendo apenas RTOS, está disponível em (WIKIPEDIA, 2011b).

Alguns fabricantes de processadores também apresentam listas de sistemas operacionais compatíveis com seus produtos. Por exemplo, a Texas Instruments fornece uma lista de RTOS para sua família de processadores MSP430 em (TEXAS INSTRUMENTS, 2011a). A ARM também fornece uma lista de sistemas operacionais compatíveis com seus processadores em (ARM, 2011).

Um dos argumentos a favor da utilização dos sistemas operacionais *open source* é o acesso ao código fonte, mas como pode ser visto na Tabela 5 vários sistemas operacionais comerciais também permitem isso. Nas Tabelas 4 e 5 é possível notar que a maioria dos sistemas operacionais possui suporte aos protocolos de comunicação em rede TCP/IP e ao sistema de arquivos FAT. Outra característica é o suporte ao tempo real, sendo que a maioria suporta requisitos temporais críticos, alguns são de tempo real brando e poucos não suportam tempo real.

Quase todos os sistemas operacionais selecionados suportam a arquitetura ARM, porém, apenas quatro destes suportam o processador Nios II da Altera, o qual foi utilizado no estudo de caso na próxima seção. Os requisitos de memória estão concentrados na faixa de baixo e médio consumo de memória, sendo que apenas os sistemas operacionais baseados em Linux apresentam alto consumo.

A compatibilidade com o padrão POSIX pode melhorar a portabilidade do código fonte e é seguida por vários sistemas operacionais, tanto comerciais quanto livres. Porém, devido às demandas adicionais impostas pelo padrão, aqueles sistemas operacionais que apresentam os menores requisitos de memória não seguem esse padrão.

O gerenciamento de memória por hardware (MMU) e a proteção de memória por hardware (MPU) facilitam o desenvolvimento de aplicações confiáveis, pois evitam que uma

aplicação defeituosa afete as demais. Por ser uma característica diretamente relacionada ao processador, apenas aqueles sistemas operacionais compatíveis oferecem suporte, sendo seu uso opcional em alguns casos. Alguns sistemas operacionais (por exemplo, µC/OS II) oferecem isso como um módulo à parte, podendo ser incluído de acordo com as necessidades do projeto.

Tabela 4 - Exemplos de sistemas operacionais *open source*

	Linux 2.6	**uCLinux 2.6**	**FreeRtos 7.0**	**eCos 3.0**	**SYS/ BIOS 6.x**	**RTEMS 4.10**	**TinyOS 2.1.1**	**Android**
Fabricante	-	-	Real Time Engineers Ltd.	Ecoscentric	Texas Instruments	OAR Corporation	TinyOS Alliance	Google
Licença	GPL	GPLv2, BSD, LGPL	GPL	GPL	BSD	GPL	BSD	GPL
IDE	Eclipse, GNU Tools	Eclipse, GNU Tools	IAR, Keil, Crossworks, AVR Studio, Codewarrior, Eclipse, MPLab	Eclipse, GNU Tools	CCStudio	Eclipse, GNU Tools	Eclipse plugin (Yeti 2, NESCDT, TinyDT)	Android SDK
CPU	x86, AMD64, ARM, MIPS, PowerPC, SuperH	ARM, Blackfin, NIOS II, Coldfire, MIPS, m68k, MicroBlaze	ARM, AVR, MSP430, PIC, x86, 8052, Coldfire, Nios II, Renesas*	ARM, x86, Coldfire, MIPS, PowerPC, SuperH	TMS320, ARM9, Cortex-M3	ARM, MIPS, Blackfin, NIOS II, AVR, x86 Coldfire, PowerPC, SuperH	MSP430, Atmega, XScale	ARM, MIPS, x86
Tempo real	não	não	soft	soft	hard	hard	soft	não
API	POSIX	POSIX	-	POSIX, uITRON	-	POSIX, uITRON, RTEID/ORKID	-	-
Memory footprint	alto	alto	baixo	baixo	baixo	baixo	muito baixo	alto
MMU	sim, obrigatório	não	sim, opcional	não	sim, opcional	não	não	sim, obrigatório
Multinúcleo	sim	sim	não	sim	sim	não	não	sim
IPC	sim	sim	sim	sim	sim	sim	sim	sim
Protocolos de comunicação	TCP, IPv4, IPv6, USB, Wi-Fi, Bluetooth, CAN	TCP, IPv4, IPv6, USB, Wi-Fi, Bluetooth, CAN	TCP, IPv4	TCP, IPv4, IPv6, USB, CAN	TCP, IPv4, IPv6	TCP, IPv4, IPv6	-	TCP, IPv4, IPv6, USB, Wi-Fi, Bluetooth
Sistemas de arquivos	JFFS2, initramFS, ramFS, LogFS, NFS, RomFS, UBIFS, YAFFS2, ext3/4, FAT, ReiserFS, ISO9660	ramFS, tmpFS, rootFS, initramFS, ext2/3, ramdisk, JFFS2, ubiFS, NFS, FAT, ISO9660	-	IMFS, FAT, JFFS2, YAFFS, MMFs, ext2	FAT	IMFS, mini-IMFS, FAT, RFS, TFTP, NFS	-	ext2/3, YAFFS2, FAT, NFS
GUI	sim	sim	sim	sim	não	sim	não	sim
Áudio e vídeo	sim	sim	não	não	Não	não	não	sim

Fonte: Própria.

Tabela 5 - Exemplos de sistemas operacionais comerciais

	VxWorks 6.9	**Windows Embedded compact 7**	**µC/OS II**	**Nucleus 2.2**	**Integrity**	**QNX Neutrino 6.5**	**RTX51 tiny**
Fabricante	Wind River	Microsoft	Micrium	Mentor Graphics	Green Hills	QNX	Keil
Código fonte	não	sim	sim	Sim	não	sim	sim
IDE	Windriver Workbench	Platform builder, Visual Studio	uC/Probe, IAR	Sourcery Tools, Inflexion UI	MULTI IDE	QNX Momentics	µVision IDE
CPU	ARM, x86, i64, MIPS, PowerPC, Coldfire	x86, ARM, MIPS	ARM, Nios II, Blackfin, Coldfire, PowerPC, AVR, x86, PIC, MIPS, MSP430, TMS320, MicroBlaze, Renesas	ARM, PowerPC, Coldfire, MIPS	PowerPC, Blackfin, ARM, MIPS, ColdFire, x86	x86, PowerPC, ARM, MIPS, SH-4	8051
Tempo real	hard	hard	hard	hard	hard	hard	soft
API	POSIX	Win32, MFC, .NET	-	POSIX, uITRON	POSIX	POSIX	-
Memory footprint	baixo	médio	baixo	baixo	médio	baixo	muito baixo
MMU	sim, opcional	sim, obrigatório	sim, opcional	sim, opcional	sim, obrigatório	sim, opcional	não
Multinúcleo	sim	sim	não	sim	sim	sim	não
IPC	sim	sim	sim	sim	sim	sim	não
Protocolos de comunicação	TCP, IPv4, IPv6, USB, Wi-Fi, CAN/OPC	TCP, IPv4, IPv6, USB, Wi-Fi, Bluetooth	TCP, IPv4, IPv6, USB, Bluetooth, CAN	TCP, IPv4, IPv6, USB, Wi-Fi, Bluetooth	TCP, IPv4, IPv6, USB, Wi-Fi	TCP, IPv4, IPv6, USB, Wi-Fi	-
Sistemas de arquivos	FAT, NFS, HRFS	FAT, BinFS, ISO9660	FAT	FAT, VFS, ISO9660, Proprietary	FFS, FAT, NFS, ISO9660, CIFS	FAT, NTFS, EXT2, CD-ROM, HFS, NFS	-
GUI	sim	sim	sim	sim	sim	sim	não
Áudio e vídeo	não	Sim	não	sim	não	sim	não

Fonte: Própria.

4.3 ESTUDO DE CASO

A fim de avaliar os impactos da adoção de um sistema operacional em um projeto embarcado, uma aplicação modelo foi concebida exibindo muitas características de um sistema embarcado típico. O equipamento a ser desenvolvido é uma estação meteorológica embarcada (EWS), a qual deve obter informações sobre a temperatura de duas fontes distintas: um sensor local e um provedor meteorológico remoto na internet. Atualmente, muitos provedores fornecem uma variedade de informações meteorológicas, tais como *weather.com*, *weather.gov*, *accuweather.com* e *climatempo.com.br*. Exemplos de informações fornecidas são a temperatura localizada (mínima, máxima e atual), a velocidade do vento, a umidade e a previsão de chuva. Por uma questão de simplicidade, este dispositivo apenas apresenta informações de temperatura para o dia atual, além da previsão para os próximos dois dias. Um diagrama em blocos do sistema é mostrado na Figura 7.

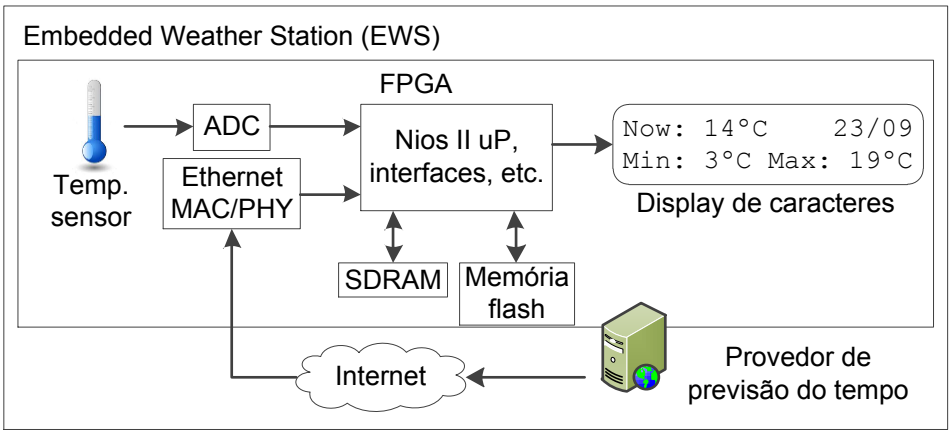


Figura 7 - Diagrama de blocos do sistema EWS.

A fim de fornecer uma plataforma de hardware flexível, porém uniforme, uma FPGA foi utilizada como o componente principal do sistema. Isso permitiu que pequenas mudanças no hardware fossem feitas de acordo com as necessidades de cada sistema operacional. O sistema foi desenvolvido em um processador Nios II de 32 bits da Altera.

4.3.1 Desenvolvimento

Além da versão sem sistema operacional, dois sistemas operacionais foram selecionados, de acordo com os critérios apresentados na seção 3.3, para o desenvolvimento da aplicação EWS.

Micrium's µC/OS-II é um RTOS, preemptivo, para microcontroladores, microprocessadores e DSPs. Seu código fonte é escrito em ANSI C e está disponível para um grande número de arquiteturas, de mais de vinte fabricantes. Uma lista completa está disponível em (MICRIUM, 2011b). Sendo um RTOS, o seu tempo de execução é constante e determinístico, e não depende do número de tarefas em execução. O seu *kernel* suporta até 250 tarefas.

uClinux (pronunciado "*you-see-Linux*") é um porte do Linux para sistemas sem unidade de gerenciamento de memória (MMU) (tipicamente, microcontroladores e DSPs). A capacidade de retirar partes do SO permite que este seja executado em sistemas muito menos potentes que o Linux original, tipicamente com apenas 2 MB de memória de programa e 4 MB de memória de dados. Além do *kernel* do sistema (atualmente na versão 2.6), o uClinux inclui uma coleção de aplicações de usuário, bibliotecas e ferramentas de desenvolvimento, adaptadas do Linux/Unix padrão. Uma vantagem importante do uClinux é a possibilidade de compilar (embora com algumas alterações) muitas aplicações e *drivers* comuns do Linux, assim se aproveitando da base de código madura deste. Uma comparação mais detalhada entre o Linux e o uClinux é mostrada em (MINTING e FAGUI, 2007).

Assim como pode ser visto nas Tabelas 4 e 5, ambos o µC/OS-II e o uClinux preenchem os requisitos para a aplicação EWS. Devido ao primeiro ser um RTOS e o segundo um GPOS, suas escolhas são complementares, e junto com o Super Loop, esta seleção aborda as três possibilidades disponíveis para um projeto embarcado.

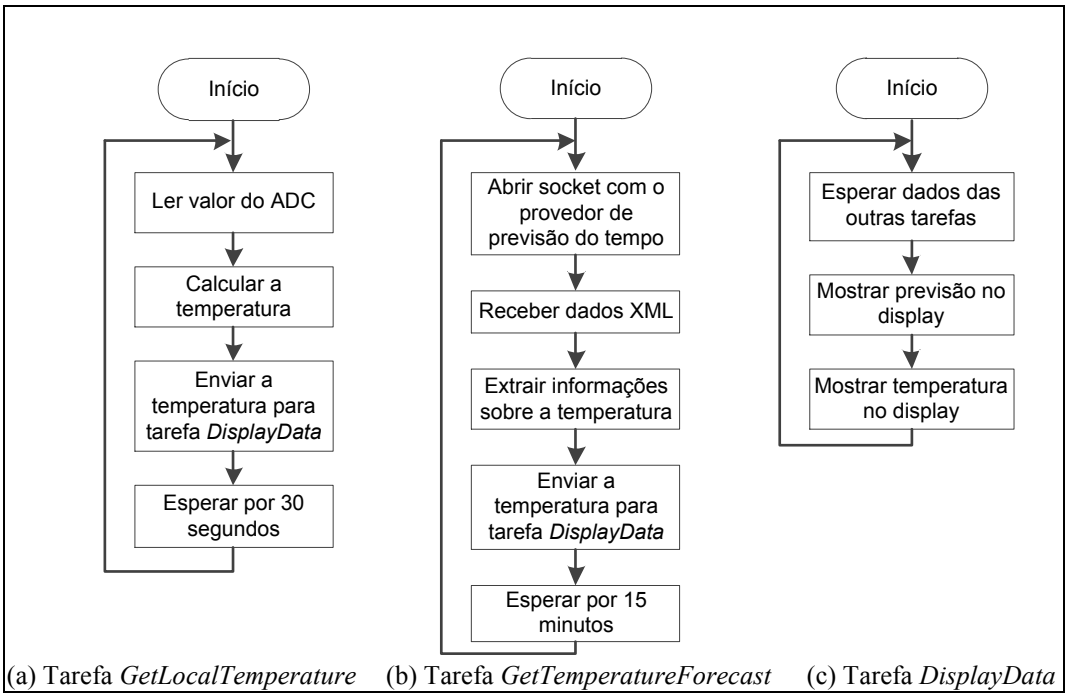
Uma visão global das tarefas de desenvolvimento é mostrada na Tabela 6. Como pode ser visto, algumas atividades são comuns às três implementações e precisaram ser realizadas apenas uma vez, enquanto outras precisaram ser desenvolvidas uma para cada implementação.

Tabela 6 - Visão geral das tarefas de desenvolvimento

Tarefa	Atividade	Comum	Super Loop	µC/OS-II	uClinux
T1	Análise do sistema e planejamento	x			
T2	Esboço geral da aplicação (fluxograma básico)	x			
T3	Desenvolvimento de uma aplicação protótipo em um computador pessoal	x			
T4	Desenvolvimento do hardware (FPGA, SoPC, interface com o ADC)	x			
T5	Configuração do ambiente de desenvolvimento		x	x	x
T6	Configuração do sistema operacional			x	x
T7	Projeto do *loop* principal		x		
T8	Particionamento do sistema em tarefas			x	x
T9	Seleção e estudo de bibliotecas de software adicionais		x	x	x
T10	Desenvolvimento do software		x	x	x

As atividades de desenvolvimento começam com uma análise detalhada do sistema e planejamento geral do projeto (*T1*). Devido à aplicação ser conceitualmente simples, estas atividades não consomem muito tempo. Informações mais detalhadas sobre o tempo de desenvolvimento serão mostradas nas próximas subseções, para cada implementação.

Depois da análise inicial, uma possível próxima etapa é esboçar o funcionamento da aplicação (*T2*), por exemplo, através do uso de um fluxograma. Em uma implementação baseada em um sistema operacional, as atividades serão mapeadas para tarefas do sistema, executando em *threads* separados. Em uma implementação sem SO, estas operações serão executadas sequencialmente, dentro de um *loop* infinito. A Figura 8 mostra o fluxograma de atividades para as três versões da aplicação EWS, as quais são representadas como tarefas do sistema.



(a) Tarefa *GetLocalTemperature* (b) Tarefa *GetTemperatureForecast* (c) Tarefa *DisplayData*

Figura 8 - Fluxograma das tarefas do sistema.

Uma prática comum no desenvolvimento de software é a criação de aplicações-protótipo descartáveis, com o objetivo de melhor entendimento ou investigação de problemas (SCHRAGE, 2004). No desenvolvimento embarcado, esta prática é ainda mais aconselhável, uma vez que é possível criar aplicações-protótipo em um ambiente *desktop*, aproveitando-se da grande quantidade de ferramentas de *debug* disponíveis. Além disso, o desenvolvimento para *desktop* é geralmente mais rápido, mais barato e as ferramentas são mais maduras se comparadas ao desenvolvimento embarcado.

Para o EWS, duas aplicações deste tipo foram desenvolvidas (tarefa *T3*): (i) uma aplicação simples desenvolvida no Linux para obter uma página de um site na internet, e (ii) uma aplicação completa, desenvolvida no Windows e capaz de fazer todas as operações requeridas pelo sistema, a qual é mostrada na Figura 9. A aplicação (i) provou ser muito útil, pois a mesma solução pôde ser utilizada tanto na implementação com uClinux quanto com µC/OS-II. A aplicação (ii) ajudou a entender a aplicação completamente e antever as dificuldades que poderiam surgir no desenvolvimento das versões embarcadas.

Figura 9 - Tela da aplicação-protótipo desenvolvida em Windows.

Devido haver interesse na influência da adoção de um sistema operacional do ponto de vista do software embarcado, a plataforma de hardware usada nos experimentos foi o kit de desenvolvimento *Nios II Development kit Cyclone II Edition* (ALTERA, 2011), contendo uma FPGA *Cyclone II* da Altera (EP2C35F672). A Figura 10 mostra uma foto da aplicação sendo executada neste kit de desenvolvimento. A utilização de uma arquitetura baseada em FPGA foi fundamental para permitir a avaliação de diferentes sistemas operacionais na mesma plataforma de hardware. Devido a alguns sistemas operacionais necessitarem de características especiais de hardware (por exemplo, MMU, temporizadores, memória cache), foi necessário adaptar a arquitetura do sistema adequadamente (tarefa *T4*). Usando uma FPGA e um *system-on-a-programable-chip* (SoPC), foi possível experimentar diferentes sistemas operacionais na mesma plataforma de hardware.

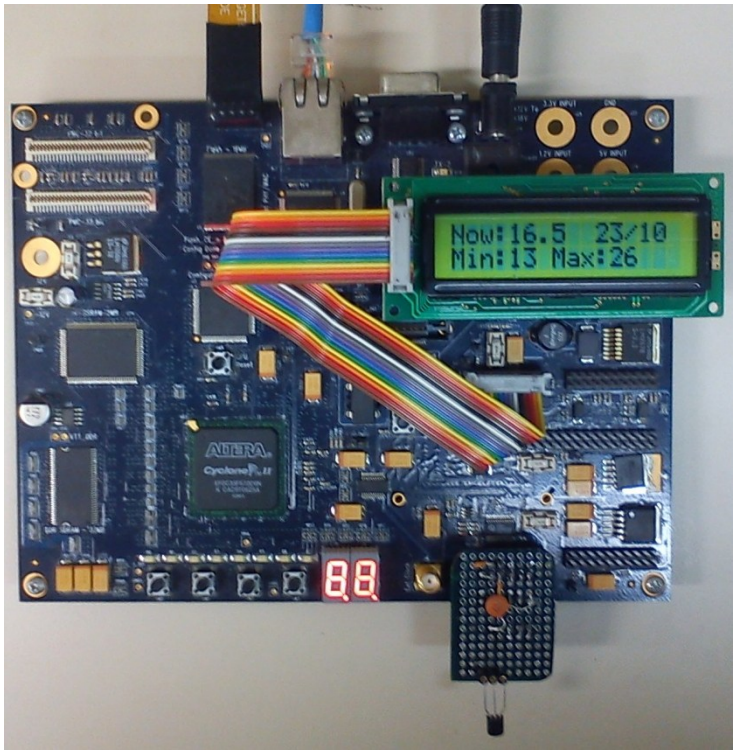


Figura 10 - Foto da aplicação EWS sendo executada no kit de desenvolvimento.

Devido às limitações típicas de memória, sistemas operacionais embarcados geralmente permitem um alto nível de customização, e muitos componentes não necessários para a aplicação podem ser retirados e deixados de lado. A atividade de configuração do sistema operacional (*T6*) consistiu na minuciosa eliminação de todas as características desnecessárias para esta aplicação. Apesar de ambos os sistemas operacionais fornecerem ferramentas gráficas interativas para ajudar nesta configuração, isso ainda é uma tarefa muito demorada, pois é preciso considerar os efeitos colaterais de incluir ou remover algum módulo específico ou característica do sistema.

Depois da configuração do sistema operacional, o projeto da aplicação torna-se uma questão de planejar uma solução de software baseada nos recursos disponíveis do sistema. Na versão sem SO, não há facilidades de escalonamento prontamente disponíveis, e o *loop* de controle deve ser implementado manualmente (*T7*). Por outro lado, em uma implementação baseada em um SO, cada atividade pode ser desenvolvida como um *thread* executando separadamente (*T8*). Em ambos os casos, é possível que a plataforma de software não forneça uma característica necessária; neste caso, o projetista deve avaliar bibliotecas de software de fontes diferentes (*T9*).

Finalmente, após todas as outras tarefas terminadas, o desenvolvimento do software propriamente dito pôde ser iniciado. Nas próximas subseções, são analisadas as vantagens particulares, desvantagens e outras dificuldades no desenvolvimento de cada implementação.

4.3.1.1 Desenvolvimento sem sistema operacional com Super Loop

O desenvolvimento com Super Loop difere das demais implementações pois o projetista é inteiramente responsável pela sequência de atividades, e deve garantir que cada função periódica seja chamada com a menor frequência exigida. Entretanto, uma vez que uma função é chamada, o controle da execução não retornará ao *loop* principal até que esta função seja completada.

A desvantagem do Super Loop é a falta de funcionalidades comuns fornecidas por um sistema operacional, como comunicação em rede e sistema de arquivos. Para a aplicação EWS sem sistema operacional, uma biblioteca TCP/IP chamada Nichestack, da empresa InterNiche, foi escolhida. Para esta biblioteca funcionar corretamente, uma função precisa ser chamada periodicamente, caso contrário a comunicação TCP/IP confiável não pode ser garantida. Contrastando com a implementação em uClinux, na qual a comunicação TCP/IP é fornecida nativamente, a inclusão de comunicação TCP/IP causou grande impacto no tempo de desenvolvimento da implementação Super Loop. Uma separação do tempo gasto nesta implementação é apresentada na Tabela 7.

Em computadores pessoais, o sistema operacional é responsável tanto por abstrair a camada de hardware quanto por executar processos simultâneos. Para projetos embarcados, há alternativas simples, tal como *hardware abstraction layers* (HAL) e *board support packages* (BSP), fornecidos pelos fabricantes. Estas bibliotecas não substituem as funcionalidades de um sistema operacional, entretanto, elas livram o programador de ter que conhecer detalhes específicos do hardware. Para a aplicação EWS, o HAL fornecido pela Altera teve suporte para todos os periféricos, exceto para o conversor Analógico digital ADC0832, para o qual uma interface de baixo nível usando os pinos de GPIO foi desenvolvida.

Tabela 7- Atividades de desenvolvimento do Super Loop

Atividade	**Tempo(h)**
Análise do sistema e planejamento	5:00*
Desenvolvimento do hardware (FPGA, SoPC, interface com o ADC)	30:00*
Desenvolvimento de uma aplicação protótipo para download de uma página da Internet em Linux	4:00*
Desenvolvimento do protótipo da aplicação EWS em Windows	8:00*
Configuração das bibliotecas do sistema, drivers e rede (BSP)	1:00
Seleção de uma biblioteca TCP/IP não dependente de um SO	9:00
Desenvolvimento de uma aplicação protótipo para avaliar a biblioteca TCP/IP	10:00
Adaptação da biblioteca TCP/IP	12:00
Integração das bibliotecas com a aplicação	2:00
Desenvolvimento da interface com o ADC0832	2:30*
Desenvolvimento da tarefa *GetTemperatureForecast*	5:00
Desenvolvimento de rotinas de software para interpretação dos dados XML	3:15*
Desenvolvimento da tarefa *GetLocalTemperature*	2:15
Desenvolvimento da tarefa *DisplayData*	3:00
Desenvolvimento do *loop* principal	2:30
Total	**99:30**
* comum para as três implementações	

Uma vantagem do Super Loop é a facilidade de comunicação entre as tarefas do sistema. Na aplicação EWS, todas as variáveis usadas para este propósito são locais ao *loop* principal. Isso significa que o valor de retorno de uma função (por exemplo, a tarefa *GetLocalTemperature*) pode ser passado como um argumento para a próxima tarefa (*displayData*), sem ser necessário recorrer a mecanismos complexos de passagem de informação e compartilhamento de recursos.

A Figura 11 apresenta a tela de configuração do BSP para o Super Loop utilizando o software *Nios II IDE* da Altera. Nela, pode ser visto o campo chamado RTOS, no qual

nenhum RTOS está selecionado, a configuração das regiões de memória e configurações da biblioteca padrão do sistema, além de outras configurações.

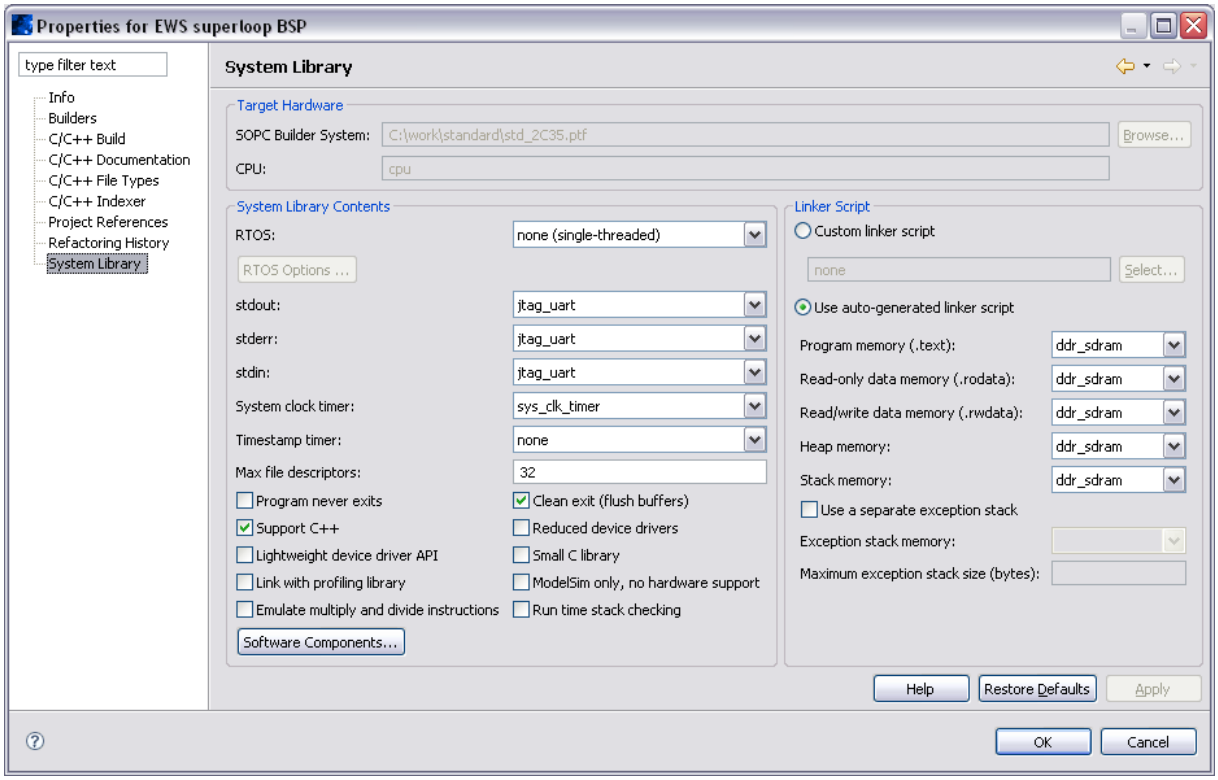


Figura 11 - Configuração do BSP para o Super Loop no software *Nios II IDE*.

4.3.1.2 Desenvolvimento em µC/OS-II

Muitos fabricantes de sistemas operacionais se juntam aos fabricantes de processadores a fim de melhor integrarem o SO com as ferramentas de desenvolvimento do processador. Devido à parceria comercial entre a Micrium e a Altera, o suporte para o µC/OS-II é integrado com o Nios II IDE, facilitando muito o desenvolvimento. Isso reflete drasticamente no curto tempo de desenvolvimento para se ter uma aplicação básica em funcionamento (apenas uma hora para se ter o BSP pronto), como mostrado na Tabela 8.

Tabela 8 - Atividades de desenvolvimento em µC/OS-II

Atividade	Tempo(h)
Análise do sistema e planejamento	5:00*
Desenvolvimento do hardware (FPGA, SoPC, interface com o ADC)	30:00*
Desenvolvimento do protótipo da aplicação EWS em Windows	8:00*
Desenvolvimento de uma aplicação protótipo para download de uma página da Internet em Linux	4:00*
Configuração das bibliotecas do sistema, *drivers* e rede (BSP)	1:00
Desenvolvimento da interface com o ADC0832	2:30*
Desenvolvimento da tarefa *GetTemperatureForecast*	5:00
Desenvolvimento de rotinas de software para interpretação dos dados XML	3:15*
Desenvolvimento da tarefa *GetLocalTemperature*	2:15
Desenvolvimento da tarefa *DisplayData*	3:00
Total	**64:00**
* comum para as três implementações	

Outra vantagem é a facilidade para o *debug* da aplicação. Devido à configuração do sistema operacional e ao desenvolvimento da aplicação serem feitos na mesma IDE, as ferramentas de desenvolvimento possuem todas as informações necessárias para a comunicação com o processador, e uma sessão de *debug* pode ser iniciada de um item de menu. O mesmo não é possível quando os sistemas operacionais não são integrados às ferramentas de desenvolvimento do processador (como é o caso do uClinux).

A Figura 12 apresenta uma tela de configuração do µC/OS-II. Esta tela em particular compreende a seleção dos componentes adicionais de software; neste caso, foi selecionada a biblioteca *altera_iniche*, uma versão da pilha TCP/IP Nichestack para o processador Nios II. Este ambiente de configuração está integrado às ferramentas de desenvolvimento de software fornecidas pela Altera (neste caso, a IDE *Nios II Software Build Tools for Eclipse*), o que facilita muito o desenvolvimento, pois em uma única ferramenta o programador possui acesso à configuração do SO, edição do código fonte, compilação e ferramentas de *debug*.

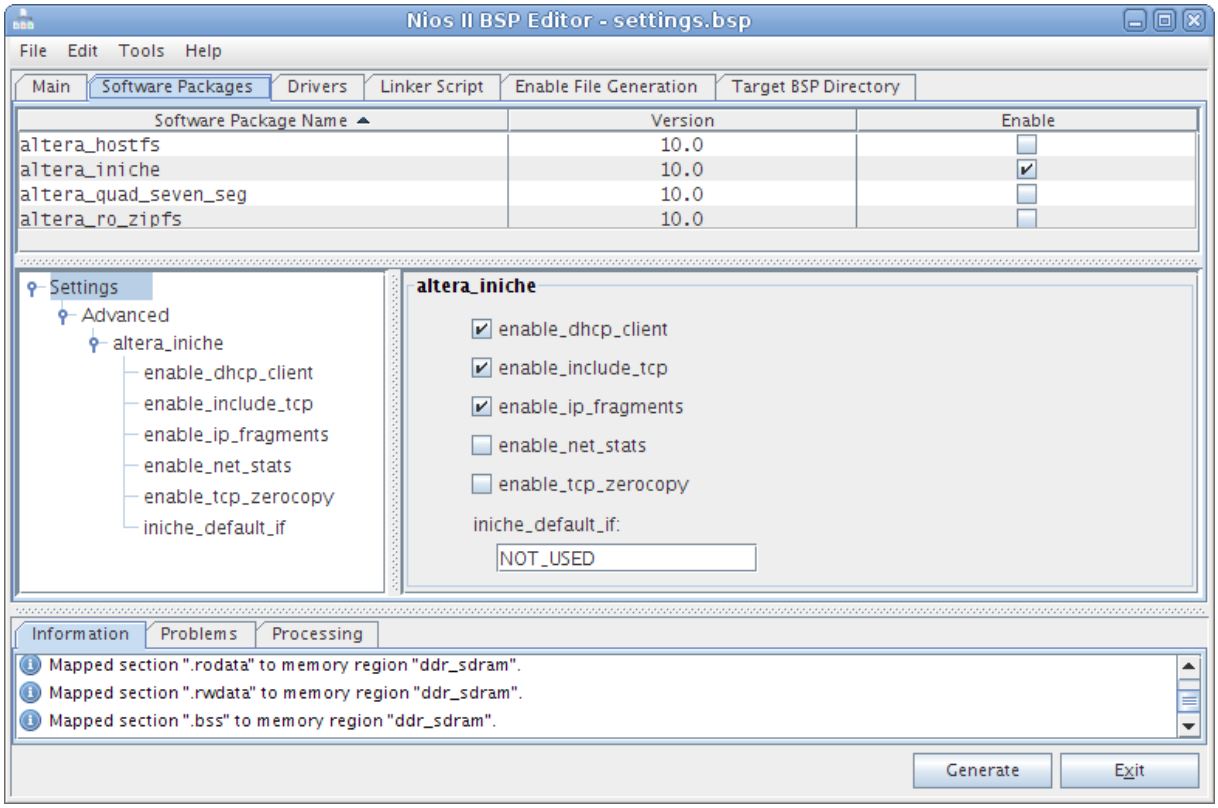


Figura 12 - Seleção de componentes de software para o µC/OS-II.

Devido à estrutura da aplicação, cada atividade foi desenvolvida separadamente das outras e mapeada para uma tarefa executando independentemente. Isso facilitou o desenvolvimento de cada tarefa, enquanto adicionou uma pequena complexidade na forma de sincronização. Para cada tarefa pôde ser atribuído um período específico de repetição e um nível de prioridade único, e ambos foram gerenciados automaticamente pelo sistema operacional. O lado negativo disso é um pequeno *overhead* na quantidade de memória de programa utilizada (no nosso caso, 2 kB por tarefa).

4.3.1.3 Desenvolvimento em uClinux

Por ser um GPOS, a implementação em uClinux pode ser estruturada diferentemente das outras duas. Aplicações em Linux são geralmente organizadas como vários processos colaborando para proporcionar uma dada funcionalidade. Frequentemente, estes processos são implementados como aplicações totalmente separadas, o que permite um projeto mais

modular, com pequenas aplicações de responsabilidades muito específicas, as quais são mais fáceis de usar, desenvolver e depurar.

A implementação em uClinux é baseada nesta abordagem. Ao invés de um único processo (*single threading*), três pequenos programas foram criados, os quais se comunicam entre si por meio de arquivos de texto puro no sistema de arquivos. Por exemplo, a aplicação *GetLocalTemperature* apenas lê o valor do ADC e escreve o valor da temperatura em um arquivo. Quando este arquivo é modificado, a aplicação *DisplayData* é notificada e atualiza as informações no *display*. Neste exemplo, fica claro que estas aplicações podem ser desenvolvidas e depuradas separadamente.

Devido a esta abordagem, o desenvolvimento de cada aplicação se torna muito simples. Como é mostrado na Tabela 9, para o desenvolvimento da tarefa *GetTemperatureForecast* foi necessário apenas meia hora, contra cinco horas nas demais versões.

Outra vantagem fornecida pelo uClinux é a grande base de software proveniente do ambiente Linux. Na aplicação EWS, foi possível utilizar a biblioteca tinyXML para parsear os dados de previsão do tempo obtidos do provedor remoto. Adicionalmente, foi possível usar o serviço *inotify* do *kernel* do uClinux para monitorar eventos no sistema de arquivo e ativar comportamentos específicos do sistema.

Claramente, a maior vantagem na utilização do uClinux foi seu suporte embutido à comunicação via rede Ethernet e implementação completa da pilha TCP/IP. Sendo um GPOS preparado para comunicação em rede, ele possui aplicativos tradicionais do universo Unix; por exemplo: *ifconfig*, um aplicativo de administração para configurar, controlar e informar sobre as interfaces de comunicação de rede; *dhclient*, um cliente para o protocolo DHCP que configura automaticamente o endereço das interfaces de rede; e *wget*, um utilitário para o download de arquivos a partir da internet, que suporta os protocolos HTTP, HTTPS e FTP.

Tabela 9 - Atividades de desenvolvimento em uClinux

Atividade	Tempo(h)
Análise do sistema e planejamento	5:00*
Desenvolvimento do hardware (FPGA, SoPC, interface com o ADC)	30:00*
Desenvolvimento do protótipo da aplicação EWS em Windows	8:00*
Desenvolvimento de uma aplicação protótipo para download de uma página da Internet em Linux	4:00*
Configuração das bibliotecas do sistema, *drivers* e rede (BSP)	7:00
Estudos da biblioteca tinyXML	3:00
Desenvolvimento da aplicação *GetTemperatureForecast*	0:30
Desenvolvimento da interface com o ADC0832	2:30*
Desenvolvimento da aplicação *GetLocalTemperature*	1:30
Desenvolvimento da aplicação *DisplayData*	7:00
Total	**68:30**
* comum para as três implementações	

Outra vantagem do uClinux é que a atual implementação é muito próxima à aplicação-protótipo desenvolvida na tarefa *T3* (Tabela 6). De fato, se uma aplicação totalmente funcional fosse desenvolvida para Linux, seriam necessárias apenas algumas adaptações para esta poder ser compilada para uClinux. Basicamente, apenas as interfaces de hardware necessitariam ser remodeladas.

Do lado negativo, uma grande desvantagem da implementação em uClinux é a maior quantidade de memória de programa necessária, a qual pode ser muito superior às implementações alternativas (tipicamente, 2 MB em uma configuração básica, contra algumas centenas de *kilobytes* de outros sistemas operacionais embarcados).

Outra desvantagem significativa é a falta de integração com as ferramentas de desenvolvimento de software da Altera. Apesar da Altera fomentar a comunidade Nios/Linux (fornecendo uma Wiki, fóruns, algum suporte especializado e algum patrocínio financeiro), a versão do uClinux para Nios II não é oficialmente suportada e, portanto, não integrada com as IDEs da Altera.

A atividade de manualmente selecionar quais módulos do sistema e *drivers* devem ser incluídos na aplicação também consumiu uma grande quantidade de tempo. Devido ao uClinux não ter sido planejado e construído desde a raiz (ao contrário, cresceu com novas características sendo adicionadas diariamente), não é possível garantir que todas as combinações de configuração funcionarão. Se uma configuração inválida for preparada, o sistema operacional pode não compilar, ou fornecer resultados errados. Na prática, isso significa que apenas algumas alterações de configuração podem ser feitas de cada vez, e precisam ser testadas no sistema antes de continuar. Por esta razão, a configuração do sistema operacional uClinux demora muito mais tempo que o µC/OS-II.

A Figura 13 apresenta a tela de configuração do *driver* do componente Ethernet no uClinux, onde está selecionado (com um asterisco) o *driver* correspondente ao controlador Ethernet LAN91C111 utilizado. Esta interface de configuração do uClinux é parte do projeto *BuildRoot* (http://buildroot.uclibc.org/) e utiliza a familiar interface do *menuconfig*. Ela permite a configuração do *kernel* e a seleção dos aplicativos que comporão o sistema.

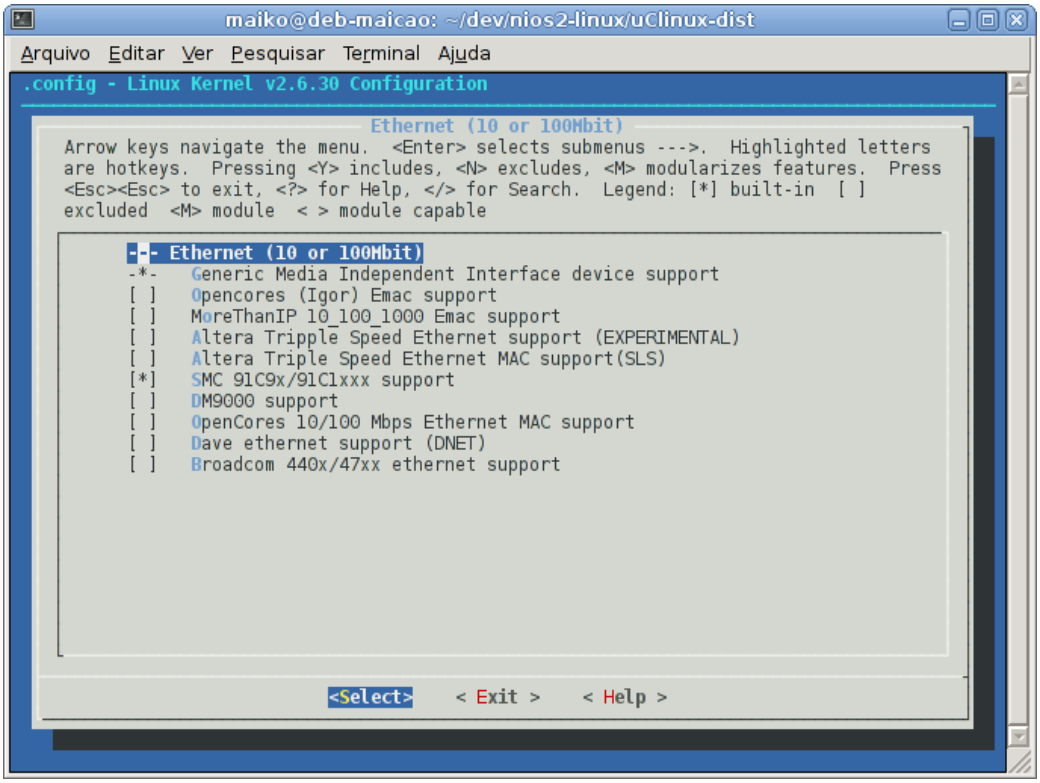


Figura 13 - Seleção do driver Ethernet no uClinux.

4.3.2 Análise dos resultados

O controle de tempo de todas as atividades de desenvolvimento permitiu fazer uma comparação objetiva do impacto da adoção de cada sistema operacional. Os resultados, extraídos das Tabelas 7 a 9, estão resumidos na Figura 14. Como pode ser notado, o tempo total de desenvolvimento foi reduzido de aproximadamente 100 horas na implementação sem sistema operacional para 64 e 68 horas nas versões com µC/OS-II e uClinux, respectivamente. Analisando os tempos decompostos nas Tabelas 7 a 9, nota-se que o fator principal para esta redução foi o suporte à comunicação em rede, fornecido por ambos os sistemas operacionais. O tempo de desenvolvimento na versão com uClinux foi um pouco superior ao tempo com µC/OS-II, principalmente devido ao tempo gasto com a configuração do sistema operacional.

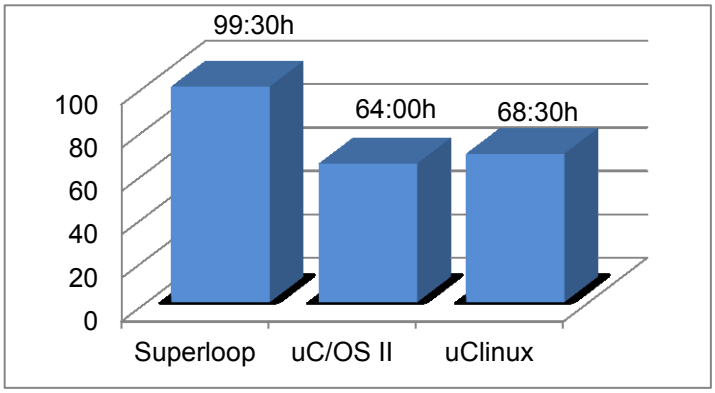


Figura 14 - Tempo de desenvolvimento.

A Figura 15 mostra o consumo de memória de programa para cada implementação, o que é uma preocupação comum em sistemas embarcados. Como era esperado, o Super Loop resultou na solução mais eficiente quanto ao consumo de memória, isso porque ele possui o menor *overhead*. O que talvez seja menos óbvio é a grande diferença entre as versões com µC/OS-II e uClinux, aproximadamente 340%. Isso pode ser explicado da seguinte forma. Uma distribuição Linux mínima de computador pessoal exige ao menos muitos *megabytes*; o uClinux é uma versão reduzida derivada do Linux, compartilhando a maioria do seu código fonte com as distribuições Linux para computadores pessoais. Por sua vez, o µC/OS-II é um RTOS, projetado desde o zero com o objetivo de pouco consumo de memória. Se for comparada a quantidade de memória de programa requerida pelo uClinux (alguns *megabytes*)

com uma distribuição Linux convencional (algumas centenas de *megabytes* ou mais), a quantidade de otimização é clara, porém, mesmo assim, não pode competir com o código extremamente otimizado do µC/OS-II.

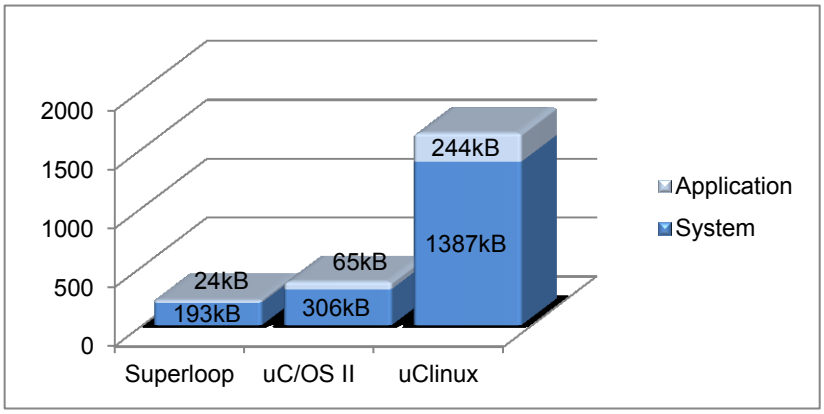


Figura 15 - Utilização da memória de programa.

A Tabela 10 apresenta com mais detalhes a quantidade de memória de programa necessária em cada implementação, onde pode ser visto o quanto foi usado pelo sistema operacional propriamente dito e o quanto foi usado apenas pela aplicação.

Tabela 10 - Detalhes da utilização de memória de programa

Implementação	Sistema	Aplicação	Total
Super Loop	193 kB	24 kB	217 kB
µC/OS-II	306 kB	65 kB	371 kB
uClinux	1387 kB	244 kB	1631 kB

Parte do *overhead* do uClinux pode ser justificado pelas características adicionais fornecidas por este GPOS. Nesta solução, é fácil enviar novos aplicativos para a aplicação, substituindo ou expandindo as funcionalidades originais do sistema. Se as principais aplicações do sistema fossem mantidas em uma memória regravável, como um cartão de memória, seria trivial obter uma versão atualizada através do protocolo FTP (usando a aplicação *wget*) e verificar a integridade do arquivo obtido (usando a aplicação *chksum*). Esta

funcionalidade seria muito mais difícil de implementar no caso de um sistema operacional que não seja GPOS. Além disso, como já foi visto, a versão com uClinux é composta por três aplicações distintas, de forma que um impacto maior na quantidade de memória de programa era naturalmente esperado. Comparado com a versão µC/OS-II, a implementação em uClinux apresentou um *overhead* de 275%.

5 CONSIDERAÇÕES FINAIS

5.1 CONCLUSÕES

A adoção de um SO em um projeto embarcado apresenta grandes impactos sobre as características fundamentais do projeto, tais como custo, desempenho e tempo de desenvolvimento. Esta decisão é complexa, especialmente para os desenvolvedores menos experientes. Geralmente, esta decisão deve ser feita logo no início do projeto, o que ressalta a sua importância, pois os custos de uma decisão errada no início do projeto podem ser expressivos. Uma vez que se opte pelo uso de um SO embarcado, a escolha do SO correto é uma tarefa ainda mais complicada, principalmente devido à grande quantidade de opções disponíveis. Mesmo assim, não foi encontrada uma abordagem satisfatória sobre o assunto, o que serviu como principal motivador para o desenvolvimento desta pesquisa.

A fim de facilitar essas decisões, foram selecionadas as características do hardware, do software e do projeto embarcado que influenciam na adoção e na seleção de um sistema operacional. Pressupõe-se, assim, que haja uma etapa destinada à definição dos objetivos e à elaboração dos requisitos do sistema embarcado sendo desenvolvido. São os resultados desta etapa que serão usados para fomentar as decisões sobre a utilização de um SO e sua seleção.

A decisão a respeito da adoção de um sistema operacional embarcado é tomada a partir dos critérios apresentados na seção 3.2. Para tanto, foram apresentados tanto critérios que favorecem quanto critérios que desfavorecem a utilização de um SO.

De forma geral, a utilização de um sistema operacional não é favorável quando os componentes de hardware, previamente definidos, oferecem recursos muito limitados de processamento e memória, como processadores de 8 ou 16 bits com baixa frequência de operação e pouca quantidade de memória de programa e dados. Isso significa que produtos de baixíssimo custo desfavorecem a utilização de um SO, pois os mesmos normalmente utilizam processadores com recursos muito reduzidos.

Aplicações simples, que podem ser modeladas em um único fluxo sequencial de atividades, e opcionalmente algumas interrupções, são facilmente implementadas sem utilização de um sistema operacional. Por outro lado, aplicações que são modeladas como várias tarefas concorrentes, possuem várias restrições temporais e diferentes prioridades, são mais bem desenvolvidas utilizando um SO.

Outro fator que favorece a utilização de um sistema operacional é a necessidade de recursos complexos, como gerenciamento de memória por hardware, uso de processadores multinúcleo, interfaces de comunicação complexas (USB e Ethernet) e interfaces com o usuário. Além disso, a utilização de um SO tende a reduzir os custos e o tempo de desenvolvimento, como foi mostrado no estudo de caso da seção 4.3.

A seleção do sistema operacional mais adequado para o projeto é feita a partir dos critérios técnicos e não-técnicos apresentados na seção 3.3. Entre os fatores considerados estão quem é o fabricante do SO e sua reputação, certificações recebidas, forma de licenciamento e custo. Também foi considerada a compatibilidade com o processador escolhido, além do suporte às demais características do hardware e atendimento aos requisitos temporais da aplicação. Outro fator importante é o conjunto de ferramentas de desenvolvimento disponíveis, as quais possuem grande impacto sobre a produtividade. Também foram consideradas necessidades especiais da aplicação, como sistema de arquivos, compatibilidade com API, interface gráfica do usuário, suporte a áudio e vídeo, e protocolos de comunicação.

A fim de investigar os benefícios e custos impostos pela adoção de um sistema operacional, foi desenvolvida uma aplicação modelo, a qual consiste em uma estação meteorológica embarcada (EWS), projetada em três diferentes cenários: sem um SO, com μC/OS-II e com uClinux. Na seção 4.3, foram apresentados os detalhes desta aplicação, assim como a descrição detalhada das atividades de desenvolvimento para cada uma das três implementações, sendo que foi medido o tempo gasto em cada atividade. Após o desenvolvimento da aplicação, também foi possível medir a quantidade de memória de programa necessária em cada implementação. O tempo de desenvolvimento da aplicação, utilizando um sistema operacional, foi reduzido em até 36% se comparado à solução sem SO. Em contrapartida, a utilização de memória de programa aumentou 71% (no caso do μC/OS-II) e 652% (no caso do uClinux). Além disso, o suporte à comunicação em rede e a integração do

sistema operacional com as ferramentas de desenvolvimento do fornecedor da FPGA tiveram grande impacto no tempo de desenvolvimento.

De acordo com os critérios propostos na seção 3.2, seria recomendável a utilização de um sistema operacional para o desenvolvimento da aplicação EWS, pois esta possui várias tarefas concorrentes, periódicas e que necessitam trocar informações entre si, além de utilizar comunicação Ethernet e a biblioteca TCP/IP. Além disso, foi utilizado um processador de 32 bits, de 100 MHz, e a quantidade de memória não era uma restrição. O tempo de desenvolvimento das três implementações mostrou que a utilização de um SO era a melhor forma de desenvolver esta aplicação. Se forem filtrados os sistemas operacionais apresentados nas Tabelas 4 e 5 pelo processador Nios II, que foi utilizado, e pelo suporte à comunicação Ethernet e TCP/IP para o kit de desenvolvimento escolhido, apenas o uCLinux e µC/OS-II atendem os requisitos.

5.2 TRABALHOS FUTUROS

Uma sugestão para continuidade deste trabalho é estudar uma metodologia de associação de pesos aos critérios de seleção e às características dos sistemas operacionais analisados. Dessa forma, os critérios mais importantes para o projeto possuiriam pesos maiores e influenciariam mais na escolha que os demais critérios. Algumas características seriam caracterizadas por valores binários (sim ou não), como, por exemplo, a presença de sistema de arquivos ou um protocolo de comunicação em particular. Outras características teriam vários níveis de ponderação, tais como o custo ou os requisitos de tempo real. Por exemplo, o requisito de tempo real poderia receber um peso entre 0 e 10, dependendo do sistema ser sem tempo real, com tempo real brando, ou com tempo real rígido.

Outra sugestão é a criação de uma aplicação *web* para viabilizar a seleção de sistemas operacionais a partir dos critérios apresentados nesta dissertação. Seria necessário criar uma base de dados com praticamente todos os sistemas operacionais disponíveis, analisados segundo os critérios utilizados. Assim, através de uma interface simples, o usuário poderia informar os requisitos do seu projeto e o sistema responderia com o SO mais adequado.

5.3 PUBLICAÇÕES

O desenvolvimento deste trabalho de mestrado gerou as seguintes publicações.

- MOROZ, M. R.; JASINSKI, R. P.; PEDRONI, V. A.. Critérios para Seleção e Adoção de Sistemas Operacionais Embarcados. In: VII Congreso Internacional de Electrónica, Control y Telcomunicaciones, 2011, Bogotá. VII Congreso Internacional de Electrónica, Control y Telcomunicaciones, 2011. v. 3.

- MOROZ, M. R. ; JASINSKI, R. P. ; PEDRONI, V. A. . Requisitos Para Adoção De Sistemas Operacionais Embarcados. Visión Electrónica: Algo más que un estado sólido, 2011. v. 9.

- MOROZ, M. R. ; JASINSKI, R. P. ; PEDRONI, V. A. . Criteria for Adoption and Selection of Embedded Operating Systems. In: Proc. 10th International Information and Telecommunication Technologies Conference, 2011, Florianópolis. 10th International Information and Telecommunication Technologies Conference, 2011.

- JASINSKI, R. P.; MOROZ, M. R.; PEDRONI, V. A. .The Impact of Operating System Adoption in an Embedded Project: a Case Study. In: Proceedings of VIII Southern Conference on Programmable Logic, Bento Gonçalves. VIII Southern Conference on Programmable Logic (SPL 2012), 2012.

REFERÊNCIAS

1003.1-2008 - IEEE Standard for Information Technology - Portable Operating System Interface (POSIX(R)). **ISO/IEC/IEEE 9945 (First edition 2009-09-15)**, Sept 2009. Disponível em: <http://standards.ieee.org/findstds/standard/1003.1-2008.html>.

ALTERA. Nios II Development Kit, Cyclone II Edition. **Altera**, 2011. Disponível em: <http://www.altera.com/products/devkits/altera/kit-nios-2c35.html>. Acesso em: 10 set. 2011.

ANH, T. N. B.; TAN, S.-L. Real-time operating systems for small microcontrollers. **Micro, IEEE**, v. 29, n. 5, p. 30-45, Set-Out 2009.

ARM. Software Enablement, 2011. Disponível em: <http://www.arm.com/community/software-enablement/index.php>. Acesso em: 10 Out. 2011.

AROCA, R. V. **Análise de sistemas operacionais de tempo real para aplicações de robótica e automação**. EESC/USP. São Carlos. 2008.

BASKIYAR, S.; MEGHANATHAN, N. A Survey of Contemporary Real-time Operating Systems. **Informatica**, 2005. 233-240.

EETIMES GROUP. 2010 Embedded Market Study. **EETimes**, 2010. Disponível em: <http://www.eetimes.com/electrical-engineers/education-training/webinars/4006580/2010-Embedded-Market-Study>. Acesso em: 04 jan. 2011.

EFFICIENT Airbag ECU Tests. **Vector Informatik GmbH**, 2011. Disponível em: <http://www.vector.com/portal/medien/cmc/press/PND/CANoe_Bosch_Airbag_HanserAutomotive_PressArticle_200910_EN.pdf>. Acesso em: 03 Abril 2011.

FREE SOFTWARE FOUNDATION. **GNU GENERAL PUBLIC LICENSE**, 2007. Disponível em: <http://www.gnu.org/licenses/gpl.html>. Acesso em: 17 jun. 2011.

FRIEDRICH, L. F. **A Survey of Operating Systems Infrastructure for Embedded Systems**. Faculdade de Ciências da Universidade de Lisboa. Lisboa. 2009a. (http://docs.di.fc.ul.pt/handle/10445/3144).

FRIEDRICH, L. F. A Survey on Operating System Support for Embedded Systems Properties. **Anais do Workshop de Sistemas Operacionais**, n. VI, Jul 2009b. 2393-2404.

GREEN HILLS SOFTWARE. Extend Your Product Life. **Green Hills Software**, 2011. Disponível em: <http://www.ghs.com/ExtendProductLife.html>. Acesso em: 1 out. 2001.

HAWLEY, G. Selecting a Real-Time Operating System. **embedded.com**, 1999. Disponível em: <http://vault.embedded.com/1999/9903/9903sr.htm>. Acesso em: 18 Março 2011.

IEEE COMPUTER SOCIETY. Standard for Information Technology - Portable Operating System Interface (POSIX), dez. 2008. ISSN ISBN 978-0-7381-5798-6.

ITRON Project Archive. **ITRON Project Archive**. Disponível em: <http://www.ertl.jp/ITRON/>. Acesso em: 17 jun. 2011.

LAPLANTE, P. A. **Real-Time Systems Design and Analysis:** An Engineer's Handbook. 2ª. ed. [S.l.]: Wiley-IEEE Press, 1996.

LI, Q.; YAO, C. **Real-Time Concepts for Embedded Systems**. [S.l.]: CMP Books, 2003.

LIBELIUM COMUNICACIONES DISTRIBUIDAS. Over the Air Programming with 802.15.4 and ZigBee - OTA, 2011. Disponível em: <http://www.libelium.com/ota>. Acesso em: 1 out. 2011.

LYNUX WORKS. High Availability (HA) RTOS Package for Embedded Systems. **Lynux Works**, 2011. Disponível em: <http://www.lynuxworks.com/rtos/high-availability.php3>. Acesso em: 1 out. 2011.

MACHADO, F. B.; MAIA, L. P. **Arquitetura de Sistemas Operacionais**. 3ª. ed. Rio de Janeiro: LTC, 2002.

MELANSON, P.; TAFAZOLI, S. A Selection Methodology for the RTOS Market. **Data Systems in Aerospace**, Praga, República Tcheca, Junho 2003.

MENTOR GRAPHICS. New Mentor Graphics Nucleus USB Class Driver Supports Firmware Upgrades, 2007. Disponível em: <http://www.mentor.com/company/news/nucleususb_firmware>. Acesso em: 1 out. 2011.

MICRIUM. Micrium Flash Loader, 2011a. Disponível em: <http://micrium.com/page/products/rtos/fl>. Acesso em: 1 out. 2011.

MICRIUM. uC/OS II, 2011b. Disponível em: <http://micrium.com/page/products/rtos/os-ii>. Acesso em: 10 set. 2011.

MICROSOFT. Benefits of Windows Embedded. **Windows Embedded**, 2011a. Disponível em: <http://www.microsoft.com/windowsembedded/en-us/evaluate/benefits-of-windows-embedded.aspx>. Acesso em: 1 Out. 2011.

MICROSOFT Windows Embedded CE. **Microsoft Windows Embedded**, 20 Janeiro 2011b. Disponível em: <http://www.microsoft.com/windowsembedded/>. Acesso em: 1 Out. 2011.

MINTING, W.; FAGUI, L. Research & implementation of uCLinux-based embedded browser. **Asia-Pacific Service Computing Conference, The 2nd IEEE**, 11-14 Dec. 2007. 504-508.

MOREIRA, A. L. S. **ANÁLISE DE SISTEMAS OPERACIONAIS DE TEMPO REAL**. UFPE. Recife. 2007.

OSDEV. Projects, 2011. Disponível em: <http://wiki.osdev.org/Projects>. Acesso em: 10 Out. 2011.

OSEK VDX Portal, 2011. Disponível em: <http://www.osek-vdx.org/>. Acesso em: 1 out. 2011.

PONT, M. **Embedded C**. London: Addison-Wesley, 2002.

PONT, M. J. **Patterns for time-triggered embedded systems:** Building reliable applications with the 8051 family of microcontrollers. [S.l.]: ACM Press / Addison-Wesley, 2001.

QNX SOFTWARE SYSTEMS LIMITED. Red Bend's Over-the-Air Software Updating Capability Ported to QNX Neutrino Operating System, 2011. Disponível em: <http://www.qnx.com/news/pr_4626_1.html>. Acesso em: 1 out. 2011.

RTAI. **RTAI**, 20 Janeiro 2011. Disponível em: <http://www.rtai.org/>. Acesso em: 1 Out. 2011.

RTLINUX. **Wind River:** RTLinuxFree, 20 Janeiro 2011. Disponível em: <http://www.rtlinuxfree.com/>. Acesso em: 1 Out. 2011.

SCHRAGE, M. Never go to a client meeting without a prototype. **IEEE Software**, v. 21, n. 2, Mar. 2004. ISSN doi:10.1109/MS.2004.1270760.

SILBERSCHATZ, A.; GALVIN, P.; GAGNE, G. **Sistemas Operacionais:** Conceitos e Aplicações. Rio de Janeiro: Campus, 2000.

SILBERSCHATZ, A.; GALVIN, P.; GAGNE, G. **Operating System Concepts**. 7th edition. ed. [S.l.]: John Wiley & Sons, 2004.

SOMMERVILLE, I. **Engenharia de Software**. 8ª. ed. São Paulo: Pearson Addison-Wesley, 2007.

TANENBAUM, A. S. **Sistemas Operacionais Modernos**. 2ª. ed. São Paulo: Prentice Hall, 2003.

TEXAS INSTRUMENTS. **MSP430 Real Time Operating Systems Overview**, 2011a. Disponível em: <http://processors.wiki.ti.com/index.php/MSP430_Real_Time_Operating_Systems_Overview >. Acesso em: 10 Out. 2011.

TEXAS INSTRUMENTS. Smart Meter: Gas and Water, 2011b. Disponível em: <http://www.ti.com/solution/smart_meter_gas_and_water>. Acesso em: 1 Out. 2011.

TOMIYAMA, H.; HONDA, S.; TAKADA, H. Real-time operating systems for multicore embedded systems. **SoC Design Conference, 2008. ISOCC '08. International**, Busan, v. 01, p. I-62 - I-67, nov. 2008. ISSN ISBN: 978-1-4244-2598-3, DOI:10.1109/SOCDC.2008.4815573.

WIKIPEDIA. List of operating systems, 2011a. Disponível em: <http://en.wikipedia.org/wiki/List_of_operating_systems>. Acesso em: 10 Out. 2011.

WIKIPEDIA. List of real-time operating systems, 2011b. Disponível em: <http://en.wikipedia.org/wiki/List_of_real-time_operating_systems].>. Acesso em: 10 Out. 2011.

WILLIAMS, R. **Real-Time Systems Development**. Oxford: Butterworth-Heinemann, 2006.

WIND RIVER. Wind River VxWorks Platforms 6.9. **Wind River**, 2011. Disponível em: <http://www.windriver.com/products/product-notes/PN_VE_6_9_Platform_0311.pdf>. Acesso em: 1 out. 2011.