Ferramenta para Concepção de Sistemas de Arquivos de Tempo Real Embutidos

Fábio Rodrigues de la Rocha*, **Rômulo Silva de Oliveira**

Laboratório de Controle e Micro-informática – Universidade Federal de Santa Catarina
Caixa Postal 476, CEP 88040-900, Florianópolis – SC

frr@das.ufsc.br, romulo@das.ufsc.br

Abstract. *This paper describes an environment for the development of embedded real-time file systems. It has been designed to take into account embedded real-time systems concerns such as predictability and memory footprint. Also a graphical user interface was created to act as a front-end and to simplify the custom file system creation task. The main advantages of this environment are reduction in both cost and time needed to develop a product, which leads to a smaller time to market.*

Resumo. *Este artigo descreve um ambiente de concepção para sistemas de arquivos de tempo real embutidos. Ele foi criado para tratar os problemas existentes nestes sistemas, tais como economia de memória e previsibilidade. Também foi concebida uma ferramenta visual que funciona como um front-end para facilitar a tarefa de criação de sistemas de arquivos personalizados. Como benefícios deste trabalho, temos uma redução no tempo de desenvolvimento de um projeto, no seu custo e no tempo que um produto leva até chegar ao mercado.*

1. Introdução

Com o desenvolvimento da micro-eletrônica nas décadas de 70 e 80, ocorreu uma proliferação do uso de microprocessadores e microcontroladores. Esses dispositivos substituíram milhares de componentes eletrônicos discretos e permitiram que dispositivos programáveis de baixo custo fossem produzidos e utilizados para controlar equipamentos. O software, que inicialmente era composto por centenas de linhas de código em linguagem de baixo nível, passou a ser composto por dezenas ou centenas de milhares de linhas de código em linguagem de alto nível. Desenvolver esses produtos da mesma forma que no passado não é mais possível, o grau de especialização atual requer o uso de novas ferramentas e abstrações de software [Berger, 2002].

Muitas das aplicações que existem nos dispositivos embutidos necessitam armazenar dados durante a sua execução, tais como agenda eletrônica, telefones celulares, leitores de código de barras, etc. Desenvolver um software específico para tratar o armazenamento de informações é uma tarefa que demanda tempo e fazer isto partindo do zero para cada um dos projetos é pouco produtivo. O ideal seria abstrair, do projetista da aplicação, o armazenamento de informações, escondendo-o sob uma camada de software denominada sistema de arquivos. Infelizmente, o uso de sistemas de arquivos de propósito geral, tais como os existentes nos computadores pessoais, não é vantajoso na maioria dos casos, visto que estes são concebidos sem levar em consideração as necessidades e

*Bolsista de doutorado do CNPq

características das aplicações embutidas. Além disso, os sistemas de arquivos de propósito geral fazem uso de muita de memória, recurso geralmente escasso em dispositivos embutidos.

Neste trabalho é proposta uma forma de auxiliar o desenvolvimento de software para dispositivos embutidos que necessitem de um sistema de arquivos (SA). A proposta é validada através de uma ferramenta de software para produzir sistemas de arquivos, isto é, uma ferramenta que gera código fonte em C a partir de um Sistema de Arquivos Abstrato (SAA), produzindo um Sistema de Arquivos Personalizado (SAP) para uma determinada aplicação final. Este SAP pode ser implementado em memória *RAM*/*NVRAM* ou mesmo em memória *Flash*. Caso memória *Flash* seja utilizada, deve-se prover um *device-driver* para tratar as leituras/escritas e apagamentos neste dispositivo. Com esta ferramenta, um projetista de software pode, em questão de minutos, criar um sistema de arquivos que será utilizado pela aplicação existente no dispositivo embutido. O SAP é concebido levando-se em consideração detalhes da aplicação final, com o objetivo de adaptar-se a ela e também é feito para lidar com restrições de memória e de tempo real (prevendo o seu uso em sistemas embutidos de tempo real), pois ambas estão freqüentemente presentes nos projetos atuais.

Não é do conhecimento dos autores ferramentas cujo propósito específico seja gerar sistemas de arquivos personalizados. Entretanto, vários trabalhos utilizam conceitos importantes neste sentido. Em [Friedrich et al., 2001] é feita uma discussão do conceito de componentes de software para construir sistemas personalizados, destacando-se a utilização destes em sistemas embutidos. O projeto OSKit, apresentado em [Ford et al., 1997] utiliza componentes de software para simplificar a construção de sistemas operacionais personalizados. Em [Huber, 2002] é descrita a *Run-time support library (RTS)* uma biblioteca C/C++ criada pela Texas Instruments para prover funções de entrada/saída para manipulação de arquivos. Em [qnx, 2002] [M-Systems, 2002] são descritos sistemas de arquivos em memória flash voltado para sistemas embutidos. Uma versão preliminar deste artigo foi apresentado no 8º Simpósio de Informática da PUC-RS em 2003.

Na seção 2 deste artigo serão caracterizados os sistemas embutidos de tempo real e discutidos os principais problemas existentes na implementação destes. Na seção 3 será apresentado o sistema de arquivos desenvolvido, com suas principais características. Na seção seguinte será mostrado o projeto do sistema de arquivos desenvolvido, com comentários sobre os algoritmos utilizados. Na seção 5 será mostrada a ferramenta visual desenvolvida para facilitar a criação de sistemas de arquivos personalizados. Um exemplo de uso da ferramenta desenvolvida será descrito na seção 6 e na seção 7 serão resumidas as contribuições deste trabalho.

2. Sistemas Embutidos de Tempo Real

Um sistema embutido *(embedded system)* é uma combinação de hardware e software projetados para realizar uma tarefa específica [Barr, 1999]. Geralmente, um sistema embutido é um componente contido num sistema maior, por exemplo: um sistema embutido para controlar um robô, para descompactar músicas em aparelhos de som, para controlar uma câmera fotográfica digital, etc. Uma grande parcela dos sistemas embutidos também podem ser classificados como sistemas de tempo real. Um sistema de tempo real é um sistema no qual se espera que, além de fornecer uma resposta correta para uma dada tarefa, também dê garantias quanto ao tempo de conclusão desta. Nestes sistemas as operações possuem deadlines para sua conclusão, isto é, se espera que estas tarefas sejam concluídas dentro de um tempo limite. Na maioria dos casos, um deadline perdido é tão ruim quanto uma resposta incorreta [Farines et al., 2000].

2.1. Requisitos das Aplicações Embutidas de Tempo Real

Todos os sistemas operacionais desenvolvidos ou adaptados para tempo real mostram grande preocupação com o escalonamento de tarefas visando atender os requisitos temporais da aplicação. Um bom algoritmo de escalonamento não será suficiente se as demais partes do sistema não forem cuidadosamente planejadas e codificadas. É necessário um levantamento de pior caso dos algoritmos envolvidos e também tratar problemas decorrentes da concorrência interna, tais como bloqueios e inversões de prioridade.

2.1.1. Previsibilidade

Um fator que tem grande influência sobre a previsibilidade é o tempo de computação de um algoritmo. Não seria interessante utilizar um algoritmo eficiente na maioria dos casos mas que é muito ruim em casos críticos, que mesmo tendo uma pequena probabilidade de ocorrência esta não é nula [Farines et al., 2000]. Desta forma, para tratar o problema de tempo real é necessário um estudo minucioso da complexidade dos algoritmos utilizados para estabelecer uma comparação e posterior escolha de um algoritmo com um bom tempo de computação no pior caso. Uma das notações mais utilizadas para complexidade de algoritmos usam os símbolos Ω para complexidade média, O para complexidade de pior caso (nosso principal interesse) e Θ para o melhor caso [Toscani and Veloso, 2001].

2.1.2. Concorrência

Nos sistemas concorrentes, podem existir diversas linhas de execução (*threads*) que trabalham em conjunto numa dada tarefa, utilizando uma memória comum. Neste cenário poderão existir seções críticas de código cuja execução pode levar a condições de disputa. Para proteger o acesso às seções críticas do código, deve-se utilizar mecanismos de acesso exclusivo, tais como semáforos [Tanenbaum, 1992]. Outro ponto a ser observado é quanto a granularidade da sincronização. Se utilizarmos um semáforo protegendo todo código de uma função estaremos sacrificando a concorrência, pois várias *threads* poderão ficar aguardando bloqueadas enquanto uma *thread* executa completamente seu código. Essa situação é ainda mais grave nos sistemas com políticas de escalonamento de prioridades fixas, onde poderão ocorrer inversões de prioridade.

2.1.3. Memória

Muitas plataformas de hardware para sistemas embutidos possuem pouca memória, com o propósito de reduzir o custo do equipamento. Indiretamente, esse fato influencia a construção do software utilizado no dispositivo embutido. Nitidamente soluções genéricas, muito utilizadas em computadores de propósito geral, não se aplicam. Por estas limitações, o software inserido nos dispositivos embutidos precisa apresentar um pequeno *footprint* (quantidade de memória utilizada por uma aplicação).

3. Sistema de Arquivos Abstrato

O sistema de arquivos desenvolvido neste trabalho foi projetado para tratar os problemas apresentados na seção anterior. Para tanto, em sua implementação foram utilizados algoritmos com tempos de pior caso explícitos e convenientes para tratar o aspecto da previsibilidade. A implementação das funções do sistema de arquivos foi feita de modo a minimizar as inversões de prioridade reduzindo ao máximo o código dentro de seções

críticas e dividindo estas sempre que possível. Desta forma o acesso concorrente é maximizado. Já o aspecto de economia de memória é obtido com a composição de um sistema de arquivos personalizado utilizando somente os componentes necessários para suportar a aplicação final.

O sistema de arquivos proposto é ilustrado na figura 1 como um conjunto de componentes agrupados. Ele possui o código necessário para a implementação de vários componentes internos, cada qual responsável por uma função específica no SAA. Dependendo das necessidades do sistema de arquivos a ser gerado, partes desse código serão utilizadas e outras não. Além de selecionar os componentes que serão utilizados, o programador pode também escolher entre diferentes implementações destes para melhor se adaptar aos requisitos da aplicação final. O sistema de arquivos obtido é dito personalizado e resulta num SA compacto e propício a ser utilizado em sistemas embutidos com restrições de memória.

É suposto que este sistema de arquivos não será implementado através de um disco magnético, mas sim através de memória *RAM* mantida com uso de baterias, ou memória *Flash*. Essas são as soluções mais freqüentes em equipamentos de baixo custo.

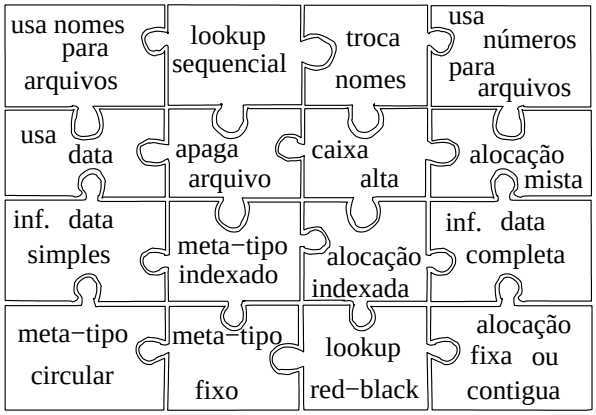

Figura 1: Visão geral do SAA

3.1. Personalização do Sistema de Arquivos

Como comentado anteriormente, o SAP foi concebido para ser composto somente com o código necessário, para tratar a economia de memória. A seleção dos componentes utilizados é feita de forma indireta, utilizando-se o conceito de **tipo** de arquivo.

Um tipo de arquivo pode ser descrito como um objeto que herda a semântica de funcionamento de um **Meta-Tipo** e alguns ou todos os métodos que este possui. A relação Meta-Tipo versus Tipo está ilustrada na figura 2. Através desses tipos de arquivos, podemos especificar qual será a implementação de determinado arquivo e escolher quais operações serão possíveis sobre ele.

Na geração de código para um SAP, somente será utilizado o código dos métodos que foram selecionados para compor os tipos de arquivos de um projeto. Dessa forma, uma escolha coerente do conjunto mínimo necessário de métodos a serem utilizados, leva a uma redução significativa do uso de memória. Como é através dos Meta-Tipos que a semântica de funcionamento e, internamente, a forma de implementação de um arquivo é escolhida, seria bastante útil se os Meta-Tipos estivessem de acordo com as aplicações geralmente encontradas em sistemas embutidos. Assim, foram implementados três Meta-Tipos, que lidam com formas muito utilizadas em aplicações embutidas. Esses Meta-Tipos estão ilustrados com seus métodos na figura 3.

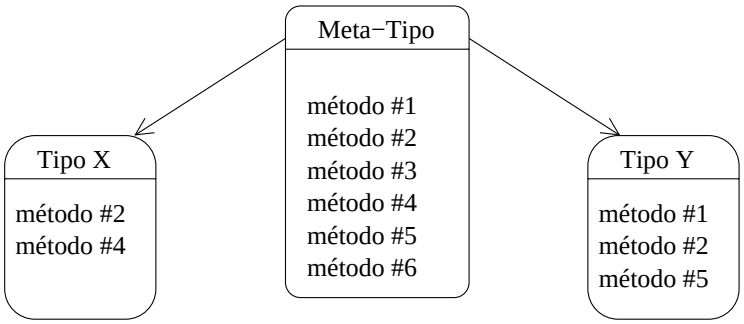

Figura 2: Relação Meta-Tipo e Tipo

Indexado		**Circular**		**Fixo**	
idx_read()	comm_ftell()	circ_read()	comm_ftell()	fix_read()	comm_ftell()
idx_trunc()	comm_setdate()	circ_write()	comm_setdate()	fix_trunc()	comm_setdate()
idx_write()	comm_getstatus()	circ_trunc()	comm_getstatus()	fix_write()	comm_getstatus()
comm_lseek()	comm_rewind()	comm_ferror()	comm_rewind()	fix_map()	comm_rewind()
comm_unlock()	comm_vlock()	comm_lock()	comm_unlock()	comm_lseek()	comm_vlock()
comm_ferror()	comm_eof()	comm_eof()		comm_unlock()	comm_lock()
comm_lock()		comm_vlock()		comm_ferror()	comm_eof()

Figura 3: Os três Meta-Tipos existentes

Na figura 3, os Meta-Tipos possuem métodos com funções semelhantes (leitura, escrita, etc.), embora a implementação desses métodos mude para cada um dos Meta-Tipos, objetivando usar os recursos das diferentes formas de se implementar arquivos. Mesmo com diferentes implementações, a interface pela qual o programador da aplicação vê o SAP é a mesma. Além destes, temos os métodos `comm` que referem-se a funções comuns, isto é, elas não são específicas de um Meta-Tipo.

Ciente de como estes Meta-Tipos são implementados e com base nos requisitos de uma aplicação alvo, o projetista do SAP pode escolher quais Meta-Tipos deve lançar mão para compor os tipos de arquivos para o seu projeto, com ganhos em desempenho, economia de memória e facilidade de uso. Sendo assim, temos os seguintes Meta-Tipos:

Meta-Tipo Indexado: Implementa arquivos utilizando uma estrutura indexada, isto é, quando o arquivo é criado ele possui tamanho nulo, a cada operação de adição de dados seu tamanho é aumentando. Internamente as escritas e leituras são realizadas em blocos de dados de tamanho fixo introduzindo um custo para estes arquivos, mas possuindo o benefício do tamanho de arquivo variável. Deve ser utilizado quando for necessário um arquivo de tamanho que não é conhecido no momento da criação.

Meta-Tipo Fixo: Implementa uma estrutura com espaço fixo pré-alocado e especificado no momento da criação do arquivo, assim não é possível aumentá-lo. Este Meta-Tipo é bastante eficiente pois o acesso aos seus dados é direto, sem o *overhead* de agrupar e desagrupar bytes em blocos de dados. Permite uma maior eficiência nas funções de *read/write*.

Meta-Tipo Circular: Implementa uma estrutura com espaço fixo pré-alocado e especificado no momento da criação do arquivo. Funciona como um *buffer* circular onde as operações de *write* progridem até o final deste e logo após retornam ao inicio.

3.2. Definições do Sistema de Arquivos

A liberdade de composição do sistema de arquivos não está presente somente na criação de tipos de arquivos e na escolha de suas funções internas. Para um maior controle e personalização deste, existem parâmetros que são usados na geração de código e que selecionam quantos arquivos o SAP deve suportar, o tamanho da memória utilizada, o tamanho de bloco, o método de gerenciamento de memória, etc.

3.3. Geração de Código

Após a etapa de análise de requisitos para o novo sistema de arquivos vem a etapa de geração de código. Nesta etapa, utiliza-se os tipos de arquivos criados para um sistema de arquivos personalizado. A figura 4 mostra os módulos envolvidos no processo de geração de código. Temos o módulo SAA (composto por uma grande quantidade de código com a implementação de vários componentes), o módulo com os tipos de arquivos criados para um projeto específico e o módulo de definições do SA (com as opções gerais do SAP).

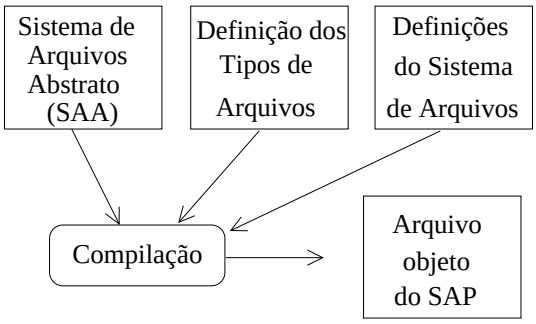

Figura 4: Módulos envolvidos na geração de um SAP

3.4. Visão do Programador de Aplicação

A interface de programação é a camada mais externa do SAP. Através dela, o programador da aplicação interage com o sistema de arquivos, criando arquivos, eliminando arquivos, lendo e escrevendo dados nestes. Por se tratar de uma parte na qual o programador da aplicação final tem contato, é importante que as interfaces da *API* sejam simples e bem conhecidas para evitar o desperdício de tempo com o aprendizado de uma nova *API*. Por esta razão, foram implementadas rotinas de acesso aos arquivos que se aproximam do padrão POSIX 1003.1, dada a sua grande aceitação. As diferenças em relação ao POSIX são encontradas nas funções `open()` e `creat()`, que neste sistema devem trabalhar com tipos de arquivos e na possibilidade de utilizar números ao invés de nomes para referenciá-los.

4. Projeto do Sistema de Arquivos

O projeto do SAA, pode ser dividido em: gerência de arquivos e alocação de memória para dados.

4.1. Gerência de Arquivos

Responsável pela manipulação e armazenamento de arquivos, esta parte pode ser decomposta em duas: Diretório e descritor de arquivos.

Diretório, nesta estrutura são inseridos, pesquisados e eliminados arquivos com base em seu nome. O sistema de arquivos abstrato provê dois componentes para esta função: um eficiente mas complexo e com grande utilização de memória (no caso é utilizada

uma árvore *red-black*, com complexidade $O(log(n))$), ou outro simples, menos eficiente mas que não utiliza muitos recursos de memória (no caso, uma pesquisa seqüencial, com complexidade $O(n)$). Caso um SAP utilize números para representar arquivos, o sistema de diretório não será utilizado e o programador acessará diretamente os descritores de arquivos.

Descritor de arquivo é a estrutura responsável por manter informações sobre cada arquivo, isto é, nome, tamanho, posição atual e dados. O SAA implementa um esquema simplificado de *i-node* [Vahalia, 1996] com algumas mudanças para lidar com as diferentes estruturas de arquivos. São implementados blocos diretos, indireção simples e dupla para dar suporte a arquivos indexados. Para arquivos fixos somente um ponteiro é utilizado.

4.2. Alocação de Memória para Dados

Quando a forma com que uma aplicação aloca memória é conhecida, pode-se implementar métodos mais eficientes e com uma menor taxa de fragmentação para uma aplicação específica. Para este trabalho, a unidade de gerenciamento de memória deve prover suporte para os três Meta-Tipos existentes.

Para dar suporte ao Meta-Tipo indexado será utilizada alocação indexada. Para os Meta-Tipos fixo e circular será utilizada alocação contígua. Nas situações onde os Meta-Tipos fixos ou circulares coexistam com os indexados será utilizada alocação mista.

Alocação indexada: Esta forma de alocação simplifica o gerenciamento de espaço livre, pois a memória é divida em blocos de dados de tamanho fixo e os blocos que constituem um arquivo não precisam ser adjacentes. Tanto a retirada quanto a inserção de blocos de um arquivo são algoritmos com complexidade $O(1)$.

Alocação contígua: Nesta forma de alocação, a memória é particionada em regiões cujo tamanho pode ser distinto. As alocações iniciam-se num dos extremos da memória e progridem em direção ao outro extremo. A desalocação de uma região de memória faz uma união das regiões de dados contínuos.

Alocação mista: Para a alocação mista é necessária a funcionalidade obtida com a alocação contígua e com a alocação indexada, para tanto os dois algoritmos anteriores são utilizados com pequenas modificações. Cada um deles iniciará num dos extremos da memória como na figura 5. Inicialmente a área disponível para alocação indexada possui tamanho nulo. Quando for requisitado a alocação de um bloco da região indexada, uma falta de memória será retornada. Neste caso, o algoritmo tentará re-estruturar a memória, isto é, ele irá reduzir o tamanho da região de memória para alocação contígua e esse espaço será utilizado para a alocação indexada. Sempre que a memória para blocos indexados se tornar insuficiente, será feita uma tentativa de empréstimo de espaço da outra região, como na figura 6.

5. Ferramenta de Concepção do SAP

O sistema de arquivos descrito na seção anterior representa um passo importante para a construção rápida de software personalizado. Infelizmente, a etapa de montagem do SAP não é trivial. Vários são os pontos chave nos quais o programador deve estar atento para evitar problemas, forçando este a conhecer o SAA mais a fundo. No entanto, a etapa de montagem do SAP pode ser facilitada. Muitos passos e decisões tomadas pelo usuário podem ser feitos automaticamente ou guiados por uma ferramenta de software visando um aumento de produtividade, qualidade e redução no tempo de conclusão de projetos.

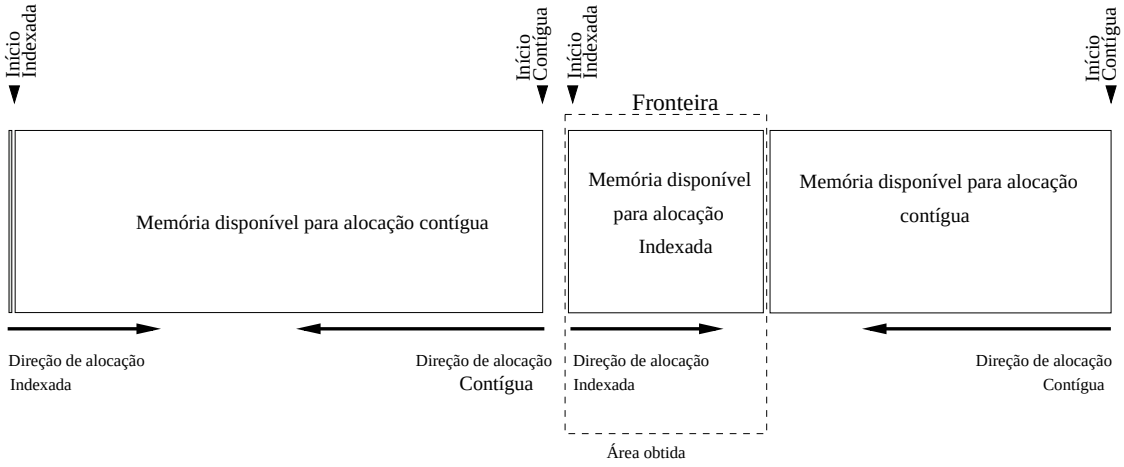

Figura 5: Estado inicial

Figura 6: Após a re-estruturação

Com essa idéia em mente, foi criada uma ferramenta de integração de componentes do SAA. A função desta ferramenta é funcionar como um *front-end* para o SAA. Os arquivos de definições do sistema de arquivos e de tipos de arquivos que anteriormente eram criados manualmente pelo programador agora serão gerados, através da simples seleção de componentes utilizando uma interface gráfica de usuário. O ambiente para concepção de sistemas de arquivos criado é um software através do qual um projetista de software pode criar um sistema de arquivos, somente informando quais recursos que ele necessita para esse sistema de arquivos (número de arquivos, tamanho máximo, operações de leitura, escrita, posicionamento, truncamento, etc.), sem a necessidade de codificar diretamente. Uma outra característica é a portabilidade da ferramenta. Para tal, optou-se pela codificação do SAP em Java para permitir sua execução em qualquer ambiente computacional para o qual exista uma máquina virtual Java.

6. Exemplo de Uso

Como uma aplicação hipotética para o sistema de arquivos, consideremos uma câmera fotográfica digital que possui o diagrama de blocos mostrado na figura 7. Neste diagrama temos um controlador central que comanda um capturador de *frames* (*frame grabber*) e este obtém uma imagem de um sensor eletrônico.

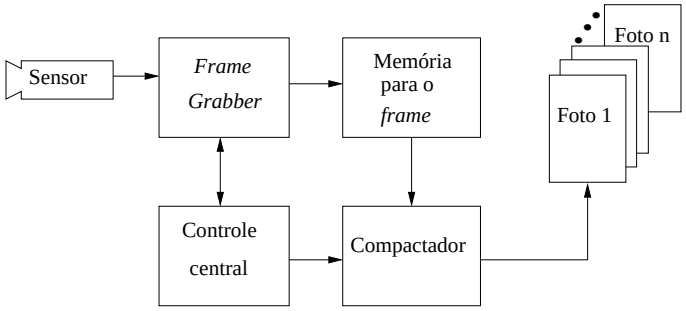

Figura 7: Diagrama de blocos de uma câmera digital

Esta imagem possui um tamanho fixo e é armazenada numa região de memória (de tamanho fixo). O controlador comanda o compactador para que este compacte a imagem adquirida e armazene junto com as demais fotos. Como o método de compactação possui uma taxa variável, um número variável de fotos pode ser obtido. Também devem

ser armazenadas algumas informações de opções/status da câmera (tais como número de fotos já tiradas).

Com base na descrição do problema e utilizando os conceitos de Meta-Tipos descritos neste trabalho, pode-se mapear as memórias utilizadas neste exemplo para arquivos, conforme ilustrado na figura 8 e na tabela 1. Nesta tabela, temos o tipo TIPO_BITMAP que é fixo e possui apenas a função `map()`. O uso da função `map()` será bastante útil neste caso, pois o algoritmo de compactação poderá operar sobre este arquivo simplesmente acessando posições de memória. Os demais tipos (TIPO_FOTOS e TIPO_OPÇÕES) foram escolhidos para descender de Meta-Tipos INDEXADO e FIXO respectivamente. Para cada um dos tipos de arquivos, seleciona-se o conjunto de funções que o tipo de arquivo deve suportar, como mostrado na figura 9. Após a criação do SAP com esses tipos, a ferramenta de concepção produz alguns arquivos: Makefile, camera.c e camera.h. A biblioteca obtida da compilação deste SAP possui cerca de 19 KB de código binário.

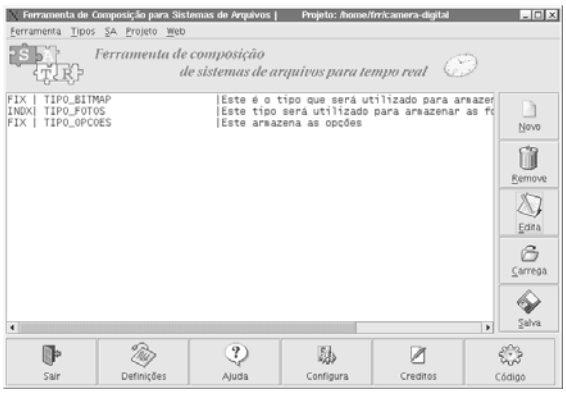

Figura 8: Tela de criação de tipos

Figura 9: Tela de edição de tipos

Tabela 1: Mapeamento de tipos

Descrição	Nome do tipo	Meta-Tipo	Funções
Memória para o *frame*	TIPO_BITMAP	Fixo	map
Memória para as fotos	TIPO_FOTOS	Indexado	read, write, eof, rewind
Opções/status	TIPO_OPÇÕES	Fixo	read, write, rewind

7. Conclusões

A principal motivação deste trabalho foi a carência de ferramentas de software automatizadas para construir sistemas de arquivos sob medida para uma aplicação e também o grande benefício que uma ferramenta destas pode produzir. Como resultado, construiu-se um sistema de arquivos composto por diversos módulos altamente configuráveis. Esse sistema de arquivos abstrato é utilizado como base para construção de sistemas de arquivos sob medida para uma determinada aplicação e foi concebido para tratar de problemas de sistemas de tempo real embutidos (tais como previsibilidade, inversões de prioridade e economia de memória). Além deste, também foi implementada uma ferramenta visual de concepção de sistemas de arquivos. Esta ferramenta funciona como um *front-end* para a composição de sistemas de arquivos. Com esta, pode-se em questão de minutos criar um sistema de arquivos personalizado para uma aplicação, com os mesmos cuidados relativos ao gasto de memória, concorrência e previsibilidade.

Todas essas vantagens reunidas neste ambiente de concepção representam um grande auxílio no desenvolvimento de software para dispositivos embutidos que necessitem de sistemas de arquivos. Além disso, representam uma vantagem competitiva para empresas da área, reduzindo o tempo de desenvolvimento de projetos, o tempo que um produto leva até chegar o mercado e conseqüentemente o custo final deste. A vantagem se torna mais aparente, quando leva-se em consideração a inexistência de ferramentas similares, deixando nas mãos do programador a tarefa de desenvolver todo o sistema de arquivos para uma aplicação.

Referências

Barr, M. (1999). *Programing Embedded Systems in C and C++*. O'Reilly.

Berger, A. S. (2002). *Embedded Systems Design*. CMP Books.

Farines, J. M., Fraga, J. d., and Oliveira, R. S. (2000). *Sistemas de Tempo Real*. Escola de Computação 2000.

Ford, B., Maren, K. V., Lepreau, J., Clawson, S., Robinson, B., and Turner, J. (1997). The flux os toolkit: Reusable components for os implementation. In *Proceedings of the Sixth Workshop on Hot Topics in Operating Systems*, pages 14–19.

Friedrich, L. F., Stankovic, J., Humphrey, M., Marley, M., and Haskins, J. (2001). A survey of configurable component-based operating systems for embedded applications. *IEEE Micro*, 21:54–68.

Huber, B. (2002). Ramdisk: A sample user-defined C I/O driver. Technical report, Texas Instruments.

M-Systems (2002). True file system (trueffs). Technical report, M-Systems.

qnx (2002). Qnx momentics non-commercial documentation roadmap. Technical report, QNX Neutrino OS.

Tanenbaum, A. S. (1992). *Modern Operating Systems*. Prentice Hall.

Toscani, L. V. and Veloso, P. A. (2001). *Complexidade de Algoritmos*. Editora Sagra Luzzato.

Vahalia, U. (1996). *Unix Internals: The new frontiers*. Prentice Hall.