Portabilidade de sistemas operacionais no domínio de sistemas embarcados*

Hugo Marcondes, Arliones Stevert Hoeller Junior,
Lucas Francisco Wanner and Antônio Augusto M. Fröhlich
Universidade Federal de Santa Catarina
Laboratório de Integração de Software e Hardware
Caixa Postal 476, 88049-900, Florianópolis, SC, Brasil
{hugom,arliones,lucas,guto}@lisha.ufsc.br

Abstract

Embedded software often needs to be ported from one system to another. This may happens for a number of reasons among which are the need for using less expensive hardware or the need for extra resources. Application portability can be achieved through an architecture-independent software/hardware interface. This is not a straight-forward task in the realm of embedded systems, since they often have very specific platforms. This work shows how an application-oriented component-based operating system was developed to allow system and application portability. Case studies present two embedded applications running in different platforms, showing that application source code is totally free of architecture-dependencies.

Keywords: Embedded Systems, Operating Systems, Portability

Resumo

Aplicações embarcadas geralmente precisam ser portadas de um sistema para outro. Isto ocorre por diversos motivos, tais como a necessidade do uso de um hardware mais barato ou pela necessidade de recursos adicionais. A portabilidade da aplicação pode ser alcançada através do uso de uma interface software/hardware independente de arquitetura, contudo a concepção de tal interface não é uma tarefa trivial de ser atingida no domínio de sistemas embarcados, visto que estes apresentam plataformas bem específicas. Este trabalho mostra como um sistema operacional orientado a aplicação e baseado em componentes foi desenvolvido para facilitar a portabilidade da aplicação e sistema. Os estudos de caso apresentam dois sistemas embarcados executando em plataformas de hardware diferentes, mostrando que o código fonte da aplicação é livre de dependências arquiteturais da plataforma.

Palavras chave: Sistemas Embarcados, Sistemas Operacionais, Portabilidade

1 Introdução

No desenvolvimento de sistemas embarcados, é comum que a aplicação seja migrada de um sistema para outro. Isto pode ocorrer por diversos motivos, entre eles a necessidade do uso de hardware de menor custo ou de recursos adicionais tais como memória e componentes presentes em plataformas específicas. Modificar a aplicação para que esta seja executada em uma nova plataforma de hardware pode ocasionar em um indesejável e elevado custo de engenharia recorrente. O uso de uma interface software/hardware altamente portável é essencial para se reduzir tais custos, sendo uma importante ferramenta para o desenvolvimento de sistemas embarcados.

A especificação e implementação desta interface não é uma tarefa trivial no contexto de sistemas embarcados uma vez que as plataformas de hardware presentes neste domínio possuem características bem específicas.

Diversas estratégias tem sido adotadas para permitir a portabilidade da aplicação, diminuindo desta forma os custos de engenharia recorrente. O uso de interfaces de chamadas de sistema (e.g POSIX, WIN32, MOSI) [8] é um exemplo. Seu uso permite que aplicações executem em sistemas operacionais que a contemplem. Para ser efetivamente

*Este trabalho foi parcialmente financiado pela FINEP - Financiadora de Estudos e Projetos

utilizada, uma interface padrão deve ser amplamente aceita pela indústria. Contudo nota-se que no domínio de sistemas profundamente embarcado, o uso de tais padrões é quase inexistente, restringindo-se apenas a sistemas de propósitos bem específicos, e que geralmente implementam apenas um subconjunto da interface. Isto ocorre principalmente porque que as interfaces aceitas pela indústria são definidas no domínio de sistemas de propósito geral, agregando desta forma, características que dificilmente podem ser acomodadas nos pequenos microcontroladores utilizados em sistemas profundamente embarcados.

Máquinas virtuais (VM) e camadas de abstração de hardware (HAL) são duas outras estratégias utilizadas para se obter portabilidade em sistemas operacionais e consequentemente, suas aplicações [8]. Embora máquinas virtuais ofereçam um bom nível de portabilidade, esta estratégia acarreta em um excessivo *overhead* de processamento e memória, restringido seu uso em sistema embarcados. Uma outra forma de se obter portabilidade em sistemas operacionais é através do uso de HALs (ex. ECOS, LINUX, WINDOWS). No entanto, nota-se que as implementações atuais sofrem a tendência de incorporarem características arquiteturais presentes na plataforma da qual foram concebidas, limitando desta forma a sua portabilidade. Isto não é desejável em sistemas embarcados, visto que estes apresentam uma grande variabilidade arquitetural.

Este trabalho mostra como um sistema operacional baseado em componentes e orientado a aplicação foi desenvolvido para permitir a portabilidade da aplicação. Isto foi possível através do uso de um cuidadoso processo de engenharia de domínio aliado ao uso de avançadas técnicas de programação como programação orientada a aspectos e meta-programação estática em um ambiente orientado a objetos [5]. O uso de *mediadores de hardware* [10] permitiu que o mesmo sistema executasse em arquiteturas bem distintas (ex. H8, AVR, ARM, POWERPC, SPARCV8, IA32).

Estudos de caso apresentam duas aplicações embarcadas executando em diferentes arquiteturas. Isto mostra como o código-fonte da aplicação é livre de dependências arquiteturais, favorecendo a portabilidade do sistema. O primeiro estudo de caso mostra a implementação de um sistema de controle de acesso utilizando um microcontrolador de 8-bits da AVR. O segundo estudo de caso mostra um multiplexador MPEG executando em uma plataforma baseada na arquitetura POWERPC de 32-bits da IBM e em uma plataforma baseada na arquitetura IA32.

A seção 2 apresenta as principais variações arquiteturais identificadas no domínio de sistemas embarcados assim como uma análise das técnicas utilizadas para a portabilidade de sistemas operacionais. A seção 3 apresenta nossa proposta de interface software/hardware para permitir a portabilidade de sistemas embarcados. A seção 4 apresenta os estudos de caso e finalmente a seção 5 apresenta um resumo do trabalho e discute seus resultados.

2 Portabilidade em Sistemas Embarcados

Sistemas embarcados utilizam uma variada gama de arquiteturas de hardware, desde simples microprocessadores de 8-bits a sofisticados processadores de 32, 64 ou 128-bits. A escolha por uma determinada arquitetura é baseada nos requisitos da aplicação-alvo com o intuito de reduzir os custos de produção.

Aplicações profundamente embarcadas geralmente não necessitam de mecanismos complexos de proteção de memória e podem ser contruídas em arquiteturas que não provém uma unidade de gerenciamento de memória (MMU). Muitos microcontroladores são baseados em uma arquitetura Harvard, com barramentos de instruções e dados separados, contrastando desta forma com a arquitetura de von Neumann, utilizada na maioria dos microprocessadores de propósito geral. Microcontroladores embarcados são baseados em arquiteturas RISC ou CISC, trocando eficiência de pipeline por densidade de código [1].

A compilação do software também é fortemente influenciada pela arquitetura-alvo. Quando uma determinada arquitetura provém uma grande quantidade de registradores, é comum que o compilador utilize parte deles para efetuar a passagem de parâmentros na chamada de funções e para retornar seus valores. Algumas arquiteturas podem utilizar estruturas mais complexas, como a janela de registradores presentes no SPARC. Arquiteturas POWERPC por exemplo, utilizam um registrador dedicado para armazenar o endereço de retorno de uma função, postergando a decisão de mover tal registrador para pilha, apenas quando um nova chamada for efetuada.

Recentes avanços em tecnologias de dispositivos de lógica programáveis (PLDS), tem tornado o uso de FPGA (*Field Programmable Gate Array*) uma alternativa viável no desenvolvimento de sistemas embarcados. Embora Hardware/Software Co-Design ainda esteja em um estágio inicial, este permite que tanto software como hardware sejam definidos de acordo com os requisitos da aplicação. Elementos de hardware podem ser recombinados, através de componentes de hardware (IP) configuráveis e formar arquiteturas especializadas para uma determinada aplicação.

Permitir a portabilidade da aplicação através de uma interface software/hardware comum de forma eficiente neste cenário não é trivial e requer a utilização de técnicas de programação eficientes. A utilização de interfaces-padrão de chamadas de sistema (ex. POSIX, WIN32) tem permitido a portabilidade entre sistemas operacionais e arquiteturas de hardware. No entanto, para um padrão ser efetivamente adotado, este deve ser baseado em tecnologias amplamente

utilizadas e estabelecidas. Este pode ser o principal motivo pelo qual alguns padrões de interfaces para sistemas embarcados (ex. MOSI - Microcontroller Operating System Interface) não tenham sido adotadas pela indústria.

O modelo de negócio da indústria de sistemas embarcados também contribui para a não aceitação de interfaces-padrão para chamadas de sistema. Na indústria de computação de propósito geral, hardware e software constituem produtos distintos e são geralmente vendidos separadamente. Sistemas embarcados são constituídos de um único produto, que geralmente faz parte de um sistema maior. Desta maneira a portabilidade entre diversos sistemas operacionais adquire uma importância muito menor, já que este é definido pelo próprio desenvolvedor do sistema, e não pelo consumidor.

Camadas de abstração do hardware (HALS) e máquinas virtuais (VMS) também são estratégias para alcançar a portabilidade da aplicação. Máquina virtuais permitem a portabilidade através da definição de uma arquitetura de hardware hipotética. Aplicações são então desenvolvidas para esta arquitetura e então traduzidas para o ambiente atual de execução. Esta abordagem permite a portabilidade binária da aplicação, permitindo que os programas executem em diversas plataformas de hardware sem qualquer tipo de modificação. No entanto, a maioria dos sistemas embarcados não podem suportar o *overhead* gerado por uma máquina virtual. Portabilidade binária da aplicação nem sempre é um requisito da aplicação embarcada, e os custos associados aos recursos computacionais necessários para suportar uma máquina virtual (ex. JAVA VM) tornaria o projeto destes sistemas economicamente inviável.

A utilização de uma camada de abstração de hardware (HAL) para abstrair o hardware da platforma, parece ser a melhor opção para atingir a portabilidade em sistemas operacionais, e é de fato a maneira mais comum entre estes (LINUX, WINDOWS, ECOS). No entanto tais sistemas, geralmente acabam incorporando peculiaridades da arquitetura original para a qual foram concebidos, permitindo apenas uma portabilidade parcial a sistemas similares.

3 Modelando a Interface Software/Hardware Portável

Diversas técnicas podem ser utilizadas afim de se obter um elevado nível de portabilidade em uma interface software/hardware. A seguir iremos apresentar as principais técnicas utilizadas na concepção da interface proposta neste trabalho, assim como a modelagem desta interface.

As HALS atuais não exploram satisfatoriamente a variabilidade arquitetural presente no domínio de sistemas embarcados. Isso ocorre por que no momento em que são projetadas, estas não passam por um profundo processo de análise de domínio. O uso de uma cuidadosa engenharia de domínio é fundamental para se atingir o nível de portabilidade necessário em sistemas embarcados. Engenharia de domínio consiste no desenvolvimento sistemático de um modelo de domínio, e sua implementação. Um modelo de domínio, segundo Czarnecki [3], é uma representação dos aspectos comuns e variantes de um número representativo de sistemas em um domínio e uma explicação para as suas variantes.

Neste contexto, diversas variações e semelhanças foram identificadas no domínio de sistemas operacionais para sistemas embarcados, tais como, políticas de escalonamento, sincronizadores (mutex, semáforos e variáveis de condição), temporização, mecanismos de gerenciamento de memória (paginação, segmentação, ambas ou nenhuma), tratamento de interrupções, tratamento de rotinas de entrada e saída (mapeamento em memória, portas de I/O, DMA, PIO). Para garantir uma ampla portabilidade do sistema, é desejável que tais características sejam configuráveis no sistema operacional, permitindo desta forma que o sistema operacional se adapte ao hardware e a aplicação.

Afim de desenvolver um sistema operacional com tais características, um processo de análise de domínio foi utilizado, guiado pela metodologia denominada *Projeto de Sistemas Orientados à Aplicação* (AOSD - *Application-Oriented System Design*) [5]. Esta metodologia propõe o uso de diversas técnicas de modelagem e programação para atingir um alto nível de configurabilidade com um overhead de processamento e memória mínimo. Para isso, técnicas como *Modelagem Orientada a Objetos* [2], *Modelagem Baseada em Famílias* [9] e *Programação Orientada a Aspectos* [6] foram utilizadas.

A figura 1 apresenta a organização geral das famílias de componentes do EPOS, um sistema operacional orientado a aplicação. Todas as unidades de hardware dependentes da arquitetura foram abstraídas através de artefatos denominados *mediadores de hardware* [10]. Tais artefatos são responsáveis em exportar toda a funcionalidade necessária as abstrações de mais alto nível do sistema operacional. Esses abstrações correspondem aos serviços tradicionais de sistemas operacionais tais como gerenciamento de memória, gerenciamento de processos, comunicação entre processos, etc. Esses componentes são discutidos em detalhes nas próximas seções, que apresentam os principais sub-sistemas do EPOS.

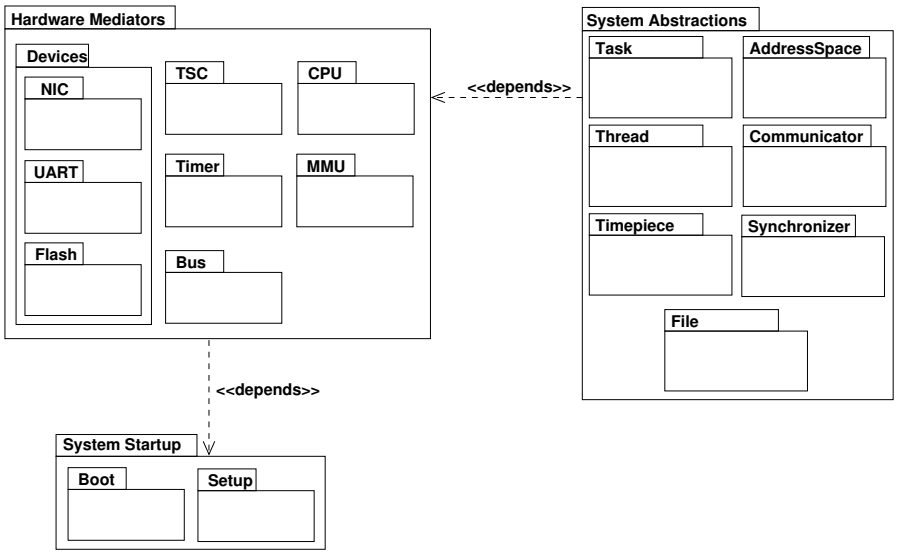

Figura 1: Visão geral dos componentes

3.1 Gerenciamento de Processos

Processos são gerenciados pelas abstrações *Thread* e *Task*. *Task* corresponde às atividades especificadas na aplicação, enquanto *Threads* são entidades que efetuam tais atividades. Os principais requisitos e dependências de tais abstrações estão relacionadas com a unidade central de processamento (CPU). O contexto de execução, por exemplo, é definido de acordo com os registradores presentes na CPU, e a manipulação da pilha é determinada pelo padrão de interface binária da aplicação, definida de acordo com a arquitetura da CPU.

Muitas das dependências arquiteturais no gerenciamento de processos são tratados pelo *mediador* de CPU (fig. 2). A classe interna *Context* define todos os dados que devem ser armazenados e que caracterizam um determinado fluxo de execução, desta forma, cada arquitetura define o seu próprio *Contexto*. O objeto *Context* é sempre armazenado na pilha da *Thread*.

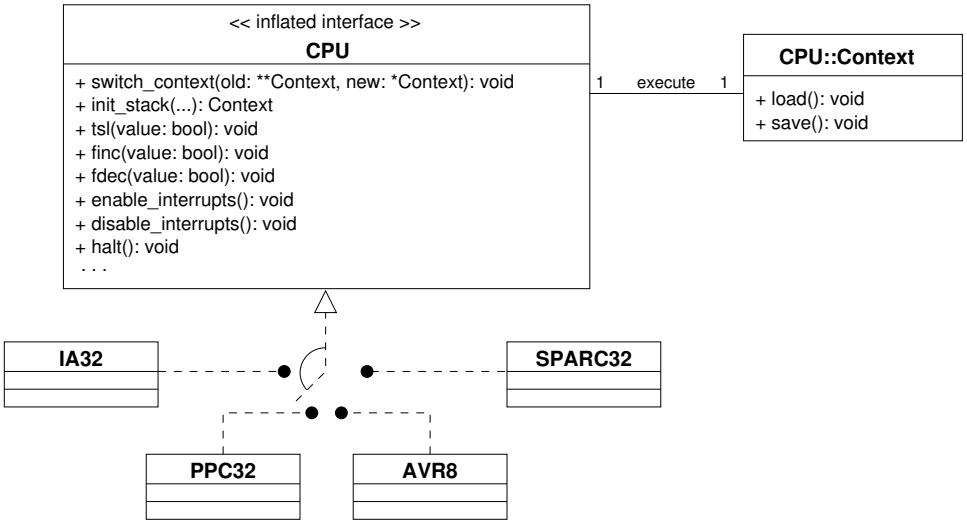

Figura 2: Mediador de Hardware CPU

Outra dependencia arquitetural no gerenciamento de processos está relacionado a inicialização da pilha de execução. O uso das estruturas de *template* da linguagem C++ foi essencial para permitir a criação de pontos de entrada flexíveis para as *Threads* sem implicar em overhead desnecessários. O construtor do objeto *Thread* recebe um poin-

teiro para uma função com um número arbitrário de argumentos de qualquer tipo. A resolução de tipo e quantidade de argumentos da função são resolvidos pelo meta-programa em tempo de compilação. Como compiladores para diferentes arquiteturas manipulam a passagem de argumentos de formas diferentes e os pontos de entrada das *Threads* são compilados como funções, a inicialização da pilha da *Thread* precisa ser efetuada pelo mediador de CPU, através do método *CPU::init_stack*, garantindo desta forma que os argumentos da função de entrada são manipulados de acordo com cada padrão de interface binária de aplicação. A figura 3 ilustra a criação (passos 1, 2 e 3) e escalonamento (passos 4 e 5) de *Threads* no sistema. Note que caso um escalonamento preemptivo esteja configurado no sistema, o passo 5 ocorre durante a criação de uma *Thread* caso esta possua uma prioridade maior que a atual *Thread* em execução.

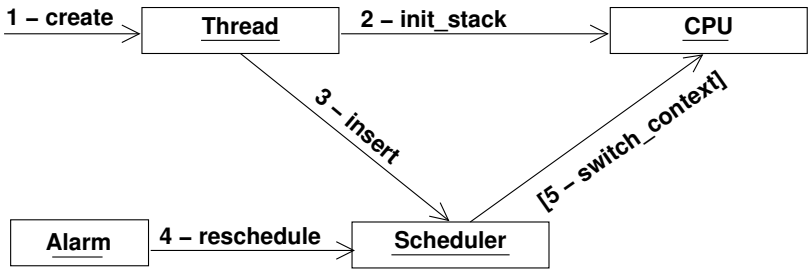

Figura 3: Criação e escalonamento de Threads

Os mediadores de CPU também implementam algumas funcionalidades para outras abstrações de sistema como transações com travamento do barramento (*Test and Set Lock*) necessárias para a família de abstrações *Synchronizer* e funções de conversão do ordenamento de bytes (ex. Host to Network e CPU to Little Endian) utilizadas pelos *Communicators* e dispositivos de E/S (Dispositivos PCI). O algoritmo de escalonamento de processos é manipulado pela família de abstrações *Timepiece*.

3.2 Sincronizadores

Processos e Threads geralmente cooperam entre si e compartilham recursos durante a execução da aplicação. Essa cooperação é efetuada através de mecanismos de comunicação entre processos ou através de dados compartilhados. Acesso concorrente a dados/recursos compartilhados pode resultar em inconsistência de dados. Sincronizadores (fig 4) são mecanismos responsáveis por garantir a consistência de dados em um ambiente de processos compartilhados.

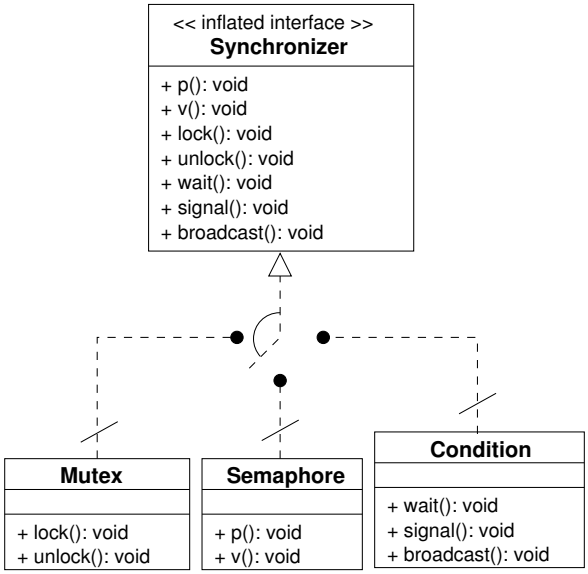

Figura 4: Família de Sincronizadores

O membro *Mutex* implementa um mecanismo simples de exclusão mútua através de duas operações atômicas: *lock* e *unlock*. O membro *Semaphore* realiza a implementação de uma variável de semáforo, que é uma variável do tipo *integer* cujo valor pode ser manipulado indiretamente pelas operações atômicas *p* e *v*. O membro *Condition* realiza uma abstração de sistema inspirada no conceito de linguagem com variáveis de condição, que permite que uma determinada Thread espere até que um determinado predicado em uma variável compartilhada se torne verdadeiro.

Afim de implementar tais mecanismos, o hardware deve prover meios para manipular dados na memória através de operações atômicas. Algumas arquiteturas provêem instruções específicas para essas operações (ex. instrução *tsl* na arquitetura IA32). Quando a arquitetura não provê tais operações, sua atomicidade é realizada através da desativação de ocorrência de interrupções na CPU. Como visto anteriormente, essas operações atômicas são implementadas no mediador de hardware CPU.

3.3 Temporização

A noção de passagem de tempo é essencial em qualquer sistema multi-processado. O tempo no EPOS é manipulado através da família de abstração denominada *Timepiece*. A abstração *Timepiece* é suportada pelos mediadores de hardware como o *Timer*, *Timestamp Counters* (TSC) e *Real-Time Clocks*.

A abstração *Clock* é responsável por armazenar o tempo corrente e está disponível apenas em sistemas que possuem dispositivos de relógios de tempo-real. A abstração *Alarm* pode ser utilizada para gerar eventos para acordar uma *Thread* ou chamar uma função. *Alarms* também possuem um evento principal com alta prioridade que ocorre a uma determinada frequência. Este evento principal é utilizado para invocar o algoritmo de escalonamento de processos a cada *quantum* de tempo, quando um escalonador está configurado no sistema. A abstração *Chronometer* é utilizada para realizar medições de tempo com alta precisão.

Temporizadores em hardware apresentam diversas funções distintas, e podem ser configurados de diversas formas diferentes. Um temporizador pode atuar como um modulador de largura de pulsos controlando um circuito analógico, um temporizador Watchdog, um temporizador de intervalo programável ou um simples temporizador de intervalo fixo. Cada um desses possíveis tipos de temporizadores possuem a suas peculiaridades de configuração. Para preservar um contrato de interface com as abstrações *Alarm*, o mediador de *Timer* apresenta uma visão simplificada do hardware, abstraindo este a um gerador de interrupções periódicas. O programador pode apenas habilitar ou desabilitar a ocorrência de interrupções do timer assim como definir uma frequência para a sua ocorrência. Outras abstrações de alto nível podem ser criadas para satisfazer funcionalidades específicas de temporizadores (ex. abstração *Pulse-Width Modulator*), criando desta forma novos contratos de interfaces com propósitos específicos.

Geralmente, arquiteturas profundamente embarcadas não dispõe de um contador de ciclos da CPU (*timestamp*). Quando isto ocorre, o mediador *Timestamp Counter* (TSC) utiliza um temporizador em hardware para contar o tempo. Geralmente, esta abordagem implica em medições de baixa precisão mas não inviabilizam o uso da abstração *Chronometer*.

3.4 Gerenciamento de Memória

Para a maioria dos sistemas operacionais, a presença de uma unidade de gerenciamento de memória (MMU) representa uma barreira que força o sistema a ser portável apenas para arquiteturas que possuem um tipo específico de MMU (ex. Paginação). No entanto, através de uma cuidadosa modelagem de abstrações e mediadores de hardware, é possível o desenvolvimento de componentes portáveis para praticamente qualquer arquitetura.

O encapsulamento dos detalhes pertinentes a proteção do espaço de endereçamento, tradução de endereços, e alocação de memória na família de mediadores de MMU, foi essencial para atingir o alto grau de portabilidade do EPOS. A abstração *Address_Space* é um *container* para regiões físicas de memória denominada *Segments*. Este não implementa nenhum mecanismo de proteção, tradução ou alocação, delegando essas responsabilidades ao mediador de MMU. Esta modelagem é retratada na figura 5, que adicionalmente ilustra o fluxo de mensagens para a criação de um segmento (1 e 2) e sua agregação ao espaço de endereçamento (3 e 4).

O espaço de endereçamento *Flat* (Fig. 5) define um modelo de memória no qual os endereços físicos e lógicos são iguais, eliminando desta forma a necessidade de uma MMU. Isto garante o cumprimento do contrato de interface entre os componentes do subsistema de memória em arquiteturas que não provêem MMU. O mediador de MMU para taisa rquiteturas é de fato um artefato em software simplificado. Regras de configuração garantem que este artefato não pode ser utilizado sem um modelo de espaço de endereçamento adequado (*Flat*). Métodos referentes a agregação de segmentos em um espaço de endereçamento *Flat* acabam se tornando vazios, com segmentos sendo agregados ao seu próprio endereço físico. Métodos referentes a alocação de memória operam sobre bytes de uma forma similiar a tradicional função *malloc* da *libc*.

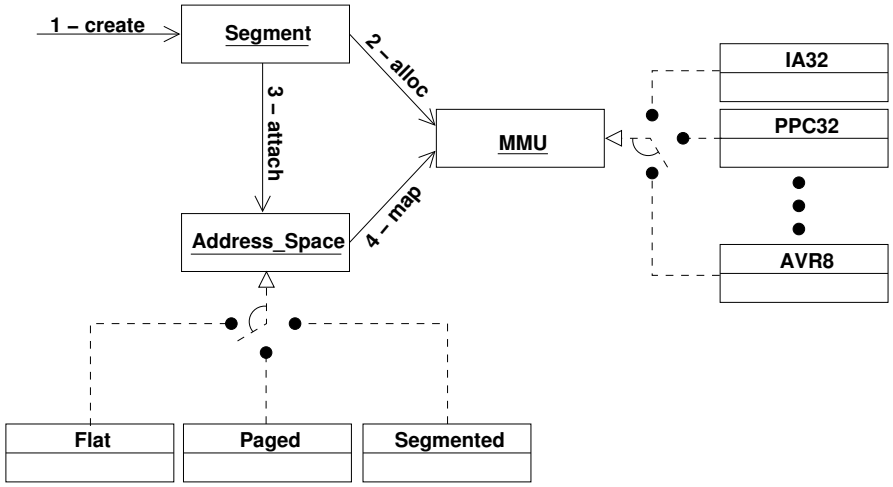

Figura 5: Gerenciamento de Memória

Conceitualmente, o modelo de memória defindo pelo membro *Flat* pode ser visto como uma degeneração do modelo de memória paginado, onde o tamanho da página é igual a um byte e a tabela de páginas mapeia endereços físicos como endereços lógicos.

3.5 Dispositivos de Entrada e Saída

Controlar os dispositivos de entrada e saída (E/S) de um sistema de computador é uma das principais funções de um sistema operacional. Sistemas profundamente embarcados geralmente não provém interfaces tradicionais de E/S (ex. teclados, monitores e mouses). Geralmente tais sistemas interagem com o ambiente em que estão inseridos através de sensores e atuadores [7]. Tais dispositivos apresentam uma grande variabilidade de interface de acesso podendo variar de acordo com arquiteturas utilizada.

Um fenômeno típico da programação de baixo nível ocorre em relação interface de um mesmo hardware em diferentes arquiteturas. Supondo que um determinado dispositivo faz parte de duas plataformas de hardware, sendo que uma plataforma utiliza I/O programado e a outra utiliza I/O mapeado em memória é bem provável que os procedimentos de acesso, configuração e interação com o hardware seja idêntico em ambas plataformas, tornando possível que o driver do dispositivo seja um componente portável. Contudo, as diferenças em relação ao modo de acesso aos dados do dispositivo poderá guiar sistemas operacionais tradicionais a configurar e utilizar dois drivers distintos e não portáveis. Um mediador de hardware meta-programado pode solucionar esse tipo de problema no modo de acesso, introduzindo um componente de *IO_Register* capaz de definir o modo de acesso ao registrador do dispositivo em tempo de compilação (utilizando especialização de templates e sobrecarga de operadores da linguagem C++). Dispositivos mapeado em memória podem ainda trazer outros problemas de portabilidade. Dispositivos conectados ao barramento PCI operam palavras com ordenamento de bytes *little-endian* e desta forma seu respectivo mediador deve levar em consideração o ordenamento de bytes utilizados pela arquitetura sendo utilizada. Isto é tratado pelo mediador de CPU através de métodos específicos para a troca da ordenação dos bytes, caso necessário.

O tratamento de interrupções é outro aspecto importante no gerencimanto de E/S, já que este evita o *polling* de registradores de hardware pela CPU. O gerenciamento de interrupções é feito pelos mediadores *Machine* e controladora de interrupção (*IC*). O *IC* abstrai o processo de habilitar e desabilitar a ocorrência de interrupções, enquanto o vetor de interrupções é gerenciado pelo mediador *Machine*.

A tabela de vetores de interrupções pode ser manipulada de diversas maneiras, de acordo com a arquitetura. A arquitetura POWERPC por exemplo, implementa dois níveis de tabelas, uma para exceções internas da CPU com tamanhos distintos para cada entrada da tabela e uma segunda tabela com entradas de tamanho fixo para as interrupções externas. Microcontroladores AVR geralmente implementam uma única tabela de interrupção, com tamanho e localização pré definidos. O sistema EPOS manipula a tabela de interrupções de uma maneira uniforme. Isto é possível graças ao uso de um componente especial de inicialização do sistema denominado *Setup*. O *Setup* é uma ferramenta não portável que executa antes do sistema operacional criar seu contexto de execução inicial. Desta maneira, estruturas complementares são criadas quando necessário pelo *Setup* afim de garantir uma manipulação uniforme de interrupções

nos mediadores de hardware *Machine* e *IC*.

4 Estudos de caso

Esta seção apresenta dois estudos de caso utilizando a interface software/hardware proposta: um sistema de controle de acesso e um multiplexador MPEG-2.

Estas aplicações foram desenvolvidas utilizando a *interface inflada* dos componentes do EPOS. O código-fonte de tais aplicações são submetidos a uma ferramenta responsável em realizar uma análise de fluxo de dados e sintática com o intuito de extrair da aplicação, seus requisitos mínimos devem ser configurados no EPOS. Quando o sistema embarcado é implementado através de dispositivos de lógica programável, uma descrição de alto nível da plataforma de hardware a ser sintetizada também é gerado por esta ferramenta. Estes requisitos são então refinados pelo programador do sistema através de uma análise de dependências guiada através de informações a respeito do cenário de execução do sistema embarcado. Este processo cria uma série de chaves de configuração que irão dirigir a compilação do sistema operacional EPOS e a síntese dos componentes de hardware nos dispositivos de lógica programável, se necessário.

4.1 Sistema de Controle de Acesso

Esse estudo de caso é um sistema de controle de acesso (fig. 6) desenvolvido utilizando um leitor de *smart cards* por aproximação. O sistema embarcado lê os dados do leitor de *smart cards* e realiza uma busca em um banco de dados presente na memória EEPROM interna. Se o dado é encontrado no banco de dados, o sistema libera a trava através da interface GPIO (por exemplo, ativa um *triac* conectado a uma tranca eletromagnética).

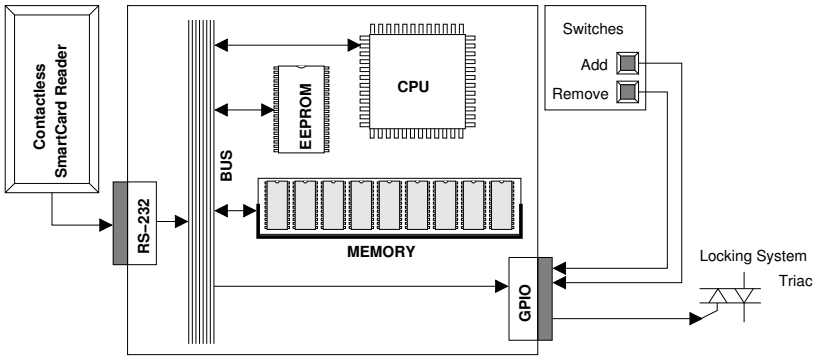

Figura 6: Sistema de controle de acesso

O leitor de *smart cards* por aproximação lê os dados do cartão e os envia através de uma interface serial. Utilizamos um microcontrolador AVR ATmega16 no gerenciamento do sistema. O ATmega16 possui um processador AVR de 8 bits com espaço de endereçamento de 16 bits. Ele possui 16Kb de memória de programa e 1Kb de memória RAM. O banco de dados foi implementado em uma memória EEPROM com capacidade de 512 bytes. O ATmega16 possui um conjunto de dispositivos, como temporizadores programáveis, controladores seriais, ADCs (*Analog Digital Converters*) e interfaces de interação com dispositivos externos (GPIO - *Global Purpose Input Output*).

As abstrações de sistema utilizadas neste estudo de caso foram o comunicador serial, para a recepção dos dados recebidos do leitor de *smart cards*, abstrações de armazenamento, para construir o banco de dados na memória (EEPROM), e GPIO para interação do sistema com sistemas externos e usuários. Uma *thread* representa o fluxo principal de execução. Uma segunda *thread* manipula o processo de inserção e remoção de dados no banco de dados em memória EEPROM. Como as duas *threads* dividem o recurso EEPROM, sincronizadores são utilizados para manter a integridade dos dados.

A aplicação foi compilada e ligada com o sistema operacional EPOS utilizando GCC 4.0.2 para a arquitetura AVR. O código objeto resultante utilizou 241 bytes da memória de dados (segmentos *.data* e *.bss*) e 13.484 bytes da memória de programa (segmento *.text*).

4.2 Multiplexador MPEG-2

O Multiplexador MPEG-2 (fig. 7) foi desenvolvido no contexto do sistema brasileiro de televisão digital (SBTVD) e consiste em um sistema embarcado responsável por criar um fluxo de dados MPEG-2, recebendo um fluxo de dados

de áudio, um de vídeo e um de dados. o fluxo de dados MPEG-2 é enviado para o sistema de modulação, sendo depois transmitido para o receptor do usuário.

A sincronização do áudio e do vídeo é fundamental para esta aplicação multimídia, por isso o sistema possui requisitos de tempo real. O uso de imagens com alta definição resulta em um grande fluxo de dados, tornando o uso de microcontroladores simples impossível.

Para a implementação do sistema foi utilizado uma placa de desenvolvimento ML310 da Xilinx que contém uma FPGA (*Field Programmable Gate Array*) modelo VirtexII-Pro XC2V30. Essa FPGA possui dois processadores *hard-cores* PowerPC 405 de 32 bits da IBM, sendo que apenas um destes foi necessário a aplicação. Todas as outras funcionalidades necessárias pelo sistema foram sintetizadas na FPGA, de acordo com os requisitos do mesmo. Uma porta UART foi utilizada para efetuar a depuração do sistema, uma ponte PCI e uma controladora de interrupções também foram instanciadas na FPGA para conectar o processador aos dispositivos de I/O externos (ex. placas de rede), responsáveis por receber e enviar os dados MPEG.

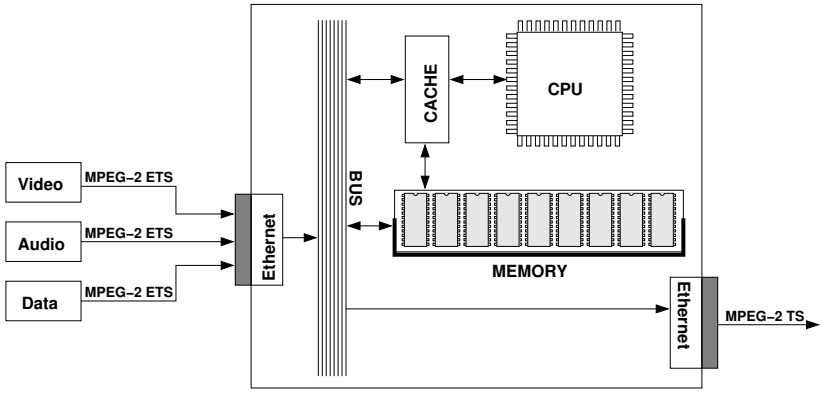

Figura 7: Plataforma do Multiplexador MPEG-2

A aplicação utiliza um número arbitrário de *threads* para manipular o fluxo de recepção (ES) de dados. Essas *threads* executam de maneira a prevenir estouros dos *buffers* em hardware. Duas *threads* de controle fornecem informações de tempo (T) e sincronização de pacotes (S). A *thread* multiplexadora recebe dados da ES, T e S e envia um fluxo de dados de transporte para saída utilizando uma *thread* de saída (OUT). Para que seja garantida consistência de áreas de dados compartilhadas, sincronizadores são utilizados nas 4 *threads*.

A aplicação do MUX foi desenvolvida também em uma plataforma de PC industrial Geode GX1 com um processador Geode SC2100. A principal razão para isso foi a restrição do tempo de execução do projeto SBTVD. Sendo essa plataforma baseada na arquitetura IA32, uma arquitetura com porte estável do sistema EPOS, os desenvolvedores puderam testar a aplicação em um PC comum enquanto o EPOS era portado para a plataforma ML310.

Isso demonstra a portabilidade da aplicação obtida utilizando a interface software/hardware proposta. A aplicação executa em ambas plataformas, sem a necessidade de modificação de seu código-fonte, e apesar de ambas plataformas serem arquiteturas de 32 bits, diversas diferenças arquiteturais podem ser identificadas entre estas. A arquitetura IA32 é uma máquina *little-endian*, possui um conjunto de registradores restrito e uma MMU imposta pelo hardware. A arquitetura PowerPC é uma máquina *big-endian* com 32 registradores (alguns utilizados para passagem de argumentos de funções), e a MMU em hardware foi desabilitada pois a aplicação não necessita de um ambiente multi-tarefa.

A aplicação foi compilada e ligada com o sistema operacional EPOS utilizando GCC 4.0.2 para a arquitetura PowerPC. O código objeto resultante utilizou 1.055.708 bytes da memória de dados (segmentos *.data* e *.bss*) e 66.820 bytes da memória de programa (segmento *.text*).

5 Conclusão

Esse trabalho apresentou o projeto de um sistema operacional altamente flexível executável em uma grande variedade de arquiteturas de hardware, desde simples microcontroladores até processadores sofisticados, de acordo com os requisitos da aplicação. Isso é possível utilizando uma refinada engenharia de domínio, projeto baseado em componentes e famílias e técnicas de programação modernas como meta-programação e programação orientada à aspectos.

Essas técnicas possibilitaram o projeto de um *contrato de interface* entre abstrações de sistema portáveis e *mediadores de hardware*, possibilitando a implementação de requisitos em software quando este não é disponibilizado

pelo hardware. O uso de uma interface hardware/software única facilita o processo de desenvolvimento. O reuso de abstrações de sistema reduz o *time-to-market* e minimiza gastos com engenharia recorrente quando a aplicação deve ser portada para outra arquitetura de hardware.

Referências

[1] Dileep Bhandarkar and Douglas W. Clark. Performance from architecture: Comparing a risc and cisc with similar hardware organization. In *ASPLOS*, pages 310–319, 1991.

[2] Grady Booch. *Object-Oriented Analysis and Design with Applications*. Addison-Wesley, 2 edition, 1994.

[3] Krzysztof Czarnecki. Beyond objects: Generative programming, 1997.

[4] G. Denys, Frank Piessens, and Frank Matthijs. A survey of customizability in operating systems research. *ACM Comput. Surv.*, 34(4):450–468, 2002.

[5] Antônio Augusto Medeiros Fröhlich. *Application-Oriented Operating Systems*. GMD - Forschungszentrum Informationstechnik, 1 edition, 2001.

[6] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-Oriented Programming. In *Proceedings of the European Conference on Object-oriented Programming'97*, volume 1241 of *Lecture Notes in Computer Science*, pages 220–242, Jyväskylä, Finland, June 1997. Springer.

[7] Peter Marwedel. *Embedded System Design*. Kluwer Academic Publishers, 2003.

[8] James D. Mooney. Strategies for supporting application portability. *IEEE Computer*, 23(11):59–70, 1990.

[9] David Lorge Parnas. On the Design and Development of Program Families. *IEEE Transactions on Software Engineering*, SE-2(1):1–9, March 1976.

[10] Fauze Valério Polpeta and Antônio Augusto Fröhlich. Hardware mediators: A portability artifact for component-based systems. In Laurence Tianruo Yang, Minyi Guo, Guang R. Gao, and Niraj K. Jha, editors, *EUC*, volume 3207 of *Lecture Notes in Computer Science*, pages 271–280. Springer, 2004.

[11] Qiming Teng, Hua Wang, and Xiangqun Chen. A hal for component-based embedded operating systems. In *COMPSAC (2)*, pages 23–24. IEEE Computer Society, 2005.