A Família de Tratadores de Interrupções do Sistema EPOS

Márcio Rodrigo de Oliveira

3 de setembro de 2002

UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE BACHARELADO EM CIÊNCIAS DA COMPUTAÇÃO

A Família de Tratadores de Interrupções do Sistema EPOS

Autor:
Márcio Rodrigo de Oliveira
Orientador:
Antônio Augusto Medeiros Frölich
Banca:
Luís Fernando Friedrich
José Mazzuco Jr

Sumário

Lista de Figuras

Agradecimentos

Gostaria de agradecer a todos que de alguma forma me ajudaram no decorrer desses anos que estive na Universidade, em especial:

- Aos meus pais que sempre me apoiaram.
- Ao meu orientador Antônio Augusto Medeiros Fröhlich com quem pude contar com seu apoio e não poupou esforços para passar seu conhecimento científico.
- Aos integrantes da banca que me ajudaram na conclusão deste trabalho.
- Ao Laboratório de Integração de Software e Hardware (LISHA) deste departamento.
- Ao grupo PET Ciências da Computação e seu tutor Luís Fernando Friedrich.

Enfim, gostaria de agradecer a todos que de alguma forma me ajudaram na conclusão deste trabalho, e obviamente a Deus por estar sempre ao meu lado.

Resumo

Este trabalho teve como finalidade o estudo dos tipos de interrupções existentes de vários sistemas operacionais e de várias arquiteturas computacionais, para se ter uma base de como funciona o mecanismo de tratamento destas interrupções.

A partir deste estudo foi possível implementar um tratador de interrupções, escolhido a partir das família de tratadores de interrupções disponíveis do sistema EPOS[1], tendo como alvo uma arquitetura de hardware de um sistema embutido.

Após a implementação deste tratador de interrupções baseado na família escolhida do framework EPOS, foi possível fazer um estudo e análise de sua performance comparada a outros tratadores, e a coerência do mesmo com o resto do sistema.

Palavras chave: Sistema operacional orientado à aplicação, sistemas paralelos e embutidos, componentes de software, engenharia de domínio, projeto baseado em famílias, microprocessador.

[1]EPOS - Embedded Parallel Operating System

Capítulo 1

Introdução

Este trabalho teve como objetivo estudar os tipos de interrupções existentes em vários sistemas operacionais e de várias arquiteturas computacionais, para se ter uma noção de como funciona o tratamento destas interrupções.

Depois deste embasamento teórico, foi possível o desenvolvimento de um dos membros da família de tratadores de interrupções existentes no sistema EPOS. Para testar o tratador de interrupções desenvolvido, foi implementada uma aplicação de controle para o microprocessador HITACHI H8/300L.

Este microprocessador faz parte de um kit chamado LEGO Mindstorms desenvolvido pela LEGO. Este microprocessador será explicado mais adiante no capítulo 2 (Conceitos Básicos).

No próximo capítulo serão apresentados alguns conceitos básicos relacionados a este trabalho. No capítulo 3 serão apresentados estudos de caso de tratamento de interrupções de alguns sistemas operacionais para contextualizar o objetivo do trabalho. No capítulo 4 será apresentada a família de tratadores de interrupções do sistema EPOS. No capítulo 5 será descrito como foi feita a implementação sobre a arquitetura escolhida.

1.1 Motivação

Engana-se quem pensa que a maioria dos processadores produzidos estão dentro de PC's ou computadores em geral. Atualmente a grande maioria destes são produzidos para sistemas computacionais dedicados, correspondendo a uma fatia de 98% do mercado de processadores [2].

Geralmente as aplicações que rodam em sistemas embutidos são um pequeno conjunto de tarefas previamente conhecidas, que deverão ser executadas satisfatoriamente para atender a necessidade do sistema.

O contrário pode ser observado em sistemas computacionais de propósito geral, onde existe a necessidade de suporte aos recursos do sistema em tempo de execução. Para estes sistemas não existe a possibilidade de antecipar quais aplicações irão ser executadas, devendo deixar disponíveis os recursos de hardware necessários para as aplicações.

Uma das principais fontes de lançamento de interrupções na maioria das arquiteturas de computadores se refere ao processamento de I/O.

Atualmente o mundo de sistemas embutidos está tendo uma forte ascensão em várias áreas da tecnologia. Até as áreas que não usavam tecnologias de microprocessadores se renderam ao chip de silício.

Hoje em dia, grandes e médios fabricantes de veículos automotivos estão cada vez mais utilizando microprocessadores para a segurança e comodidade de seus clientes. A injeção eletrônica é um exemplo concreto. Existem alguns veículos que possuem toda sua parte elétrica controlada por microprocessadores.

Está também em ascensão o número de sistemas embutidos que rodam aplicações de tempo real. Para uma aplicação controlar o trem de pouso

de um avião, é necessário que não contenha erros e que o sistema operacional possa dar uma resposta em tempo hábil. Estes tipos de sistemas são chamados de alto risco pois podem envolver a vida de muitas pessoas.

Como o mercado de processadores embutidos cresce com o passar dos dias, onde até mesmo em um liquidificador de última geração já está possuindo um microprocessador, chegará um ponto em que as funcionalidades destes objetos crescerão exponencialmente. Assim sendo, deverá existir algum software que tenha a função de gerenciar os dispositivos de hardware, isto é, um sistema operacional embutido.

As previsões que estão sendo feitas na área de comunicação de dados, dizem que em um futuro próximo todos os eletrodomésticos serão interligados em rede em uma casa. Estes deverão possuir sistemas operacionais embutidos para a realização da comunicação, além de controlar e abstrair os recursos de hardware.

Os conceitos e técnicas pertinentes ao desenvolvimento de um sistema orientado à aplicação são utilizados no sistema EPOS, que é um framework para o desenvolvimento de sistemas dedicados de alto desempenho.

As duas principais tarefas de um sistema computacional são o processamento e I/O. Dependendo da finalidade de uma tarefa, I/O pode ser mais importante que o processamento, como: Uma aplicação poderia receber dados de uma placa de rede para serem armazenados em um servidor de banco de dados.

Nota-se que o mais importante no exemplo anterior seria a identificação do recebimento dos dados pela placa de rede e o armazenamento no banco de dados.

A identificação do recebimento dos dados poderia ser feita por uma interrupção, e o processamento envolvido nesta tarefa não seria o mais importante, mas sim o tratamento da interrupção lançada pela placa de rede para o recebimento dos dados.

Como os sistemas computacionais precisam comunicar-se com o mundo exterior através dos vários dispositivos que são acoplados a eles, existe a necessidade do gerenciamento destes dispositivos, além de uma definição do modo de comunicação com o sistema.

Uma das formas que os sistemas computacionais fazem a comunicação com seus dispositivos é por meio de interrupções, que é a forma mais comum de algum dispositivo pedir algum recurso do sistema.

Por isso, existe a necessidade de um tratador de interrupções para que a comunicação entre estes dispositivos seja feita de forma organizada e com (se necessário) níveis de privilégios para a boa eficiência do sistema.

Devido a poucas pesquisas serem feitas nesta área de tratamento de interrupções em relação a outras áreas de sistemas operacionais, e sendo o sistema de I/O uma das partes mais importantes de um sistema operacional, nasceu o desejo de pesquisa nesta área.

1.2 Objetivos

O objetivo deste trabalho é o desenvolvimento de um membro da família tratadora de interrupções do framework EPOS[1], a fim de fornecer à uma aplicação teste o tratamento adequado das interrupções lançadas pelos dispositivos do microprocessador HITACHI H8/300L, de acordo com a necessidade desta aplicação.

[1]no sistema EPOS, a família de tratador de interrupções faz parte da abstração de I/O

Analisar e verificar o desempenho do tratador desenvolvido, com a finalidade de estudar novas e mais eficientes formas de executar esta tarefa que é uma das mais importantes em um sistema computacional.

O tratador de interrupções proposto visa atender os seguintes requisitos:

- Gerenciar a alocação de recursos dos dispositivos de I/O;
- Fornecer em tempo hábil uma resposta ao tratamento da interrupção gerada pelo hardware;
- Aceitar níveis de prioridades de interrupções de acordo com sua relevância;
- Ter independência de dispositivos;
- Dar suporte a novos dispositivos que poderão ser anexados ao sistema.

1.3 Problemas Abordados

Um problema que é freqüentemente observado em sistemas de geram interrupções, é que os tratadores destas não conseguem gerenciar no caso de existirem várias interrupções simultâneas (de um mesmo dispositivo ou não), onde a taxa de interrupções pode ser muito alta e o tratador pode não conseguir tratar todas estas interrupções. Se o tempo do tratamento da interrupção for maior que o tempo médio que uma interrupção é gerada, algumas interrupções podem ser perdidas.

Existem casos (mais comumente em sistemas embutidos) em que o tratador de interrupção deixa a aplicação tratar as interrupções. Isto é feito com a aplicação passando para o tratador de interrupções a rotina tratadora que deverá ser executada.

Se um tratador estiver tratando uma interrupção e for lançada outra interrupção pelo mesmo dispositivo, o tratador chamará a mesma rotina para tratar esta última interrupção e irá sobrescrever variáveis calculadas da primeira interrupção que ainda estava em execução. Assim a primeira interrupção será perdida.

No pior dos casos, pode ocorrer do sistema parar devido ao tratamento inadequado de interrupções simultâneas.

Um problema semelhante ocorre quando o tratamento da interrupção não se dá pela execução de uma rotina tratadora, mas sim por uma thread tratadora.

Se uma interrupção tentar ativar uma thread que já esta ativa (tratando uma interrupção), simplesmente não terá nenhum efeito, só que esta última interrupção será perdida.

Este trabalho tenta de alguma forma contornar estes problemas para que não se percam interrupções, que podem ser muito importantes em aplicações que exijam sistemas de tempo real.

1.4 Ferramentas Utilizadas no Desenvolvimento

Foi usado o repositório de versões CVS para o gerenciamento dos códigos fontes gerados para o sistema EPOS. Este repositório está localizado em uma das máquinas do Laboratório de Integração Hardware Software (LISHA) deste departamento.

O compilador usado para a geração do componente e da aplicação teste para o microprocessador HITACHI H8/300L, foi o cross-compiler gcc 2.95.2 para a plataforma LINUX.

Para a geração da documentação de análise de domínio, foi usado a ferramenta de modelagem Rational Rose (versão trial).

Para a modelagem deste relatório final foi utilizado o conjunto de macros LaTeX para a diagramação de texto TeX, para a mais alta qualidade tipográfica. Para a edição do LaTeX foi utilizado o editor WinEdt.

Para fazer o download do sistema embutido para o RCX foi utilizado um programa (desenvolvido para o sistema operacional LINUX) pertencente à Licença Pública Mozilla [9].

Capítulo 2

Conceitos Básicos

A seguir serão apresentados alguns conceitos relevantes para o bom compreendimento do escopo deste trabalho.

2.1 Interrupções

Interrupção é uma mensagem ou um sinal assíncrono mandado de determinados dispositivos para o processador.

Um processador deve reconhecer o dispositivo que lançou a interrupção, salvar os valores de seus registradores de dados e de controle na pilha da memória, e carregar o tratador da interrupção para ser executado. Desta forma é feito o chaveamento do fluxo de processamento para a rotina que o tratador disponibiliza, tratando esta interrupção convenientemente.

Arquiteturas de interrupções mais sofisticadas permitem que uma interrupção de mais alta prioridade seja processada na frente de outra de mais baixa prioridade. Este esquema de tratamento de interrupções através de prioridades, permite que sejam assinadas prioridades de acordo com a importância de cada interrupção.

Existem sistemas operacionais e/ou arquiteturas de hardware que desabilitam o lançamento de outras interrupções quando há o lançamento de uma interrupção. As outras interrupções são desabilitadas até que seu tratamento seja completado.

Existem dois tipos básicos de interrupções:

Síncronas - O controle é retornado ao processo do usuário depois do completo tratamento da interrupção.

Assíncronas - O controle é retornado ao processo do usuário sem esperar pelo tratamento completo da interrupção.

Uma vantagem das interrupções assíncronas é o aumento da eficiência do sistema, pois enquanto alguma interrupção está sendo tratada, a CPU pode ser usada para processar outras tarefas ou até mesmo reconhecer outras interrupções de outros dispositivos.

2.2 Tratador de Interrupções

Cabe ao tratador de interrupções saber que dispositivo lançou determinada interrupção e em que ponto do sistema operacional está o endereço inicial da rotina para que esta interrupção seja tratada. Assim o processador pode executar esta rotina no endereço especificado pelo tratador, e posteriormente permitir que o processador volte ao seu curso normal de execução após a interrupção ter sido tratada.

Geralmente em sistemas embutidos, quem possui a rotina de tratamento de interrupções é a aplicação e não o sistema operacional. Então neste caso, o sistema operacional deve fornecer mecanismos para que a aplicação possa tratar a interrupção lançada pelo dispositivo.

Os drivers de dispositivos possuem suas próprias rotinas de tratamento de interrupções, que interagem com suas placas controladoras.

Interrupções devem ser tratadas rapidamente após seu lançamento. Dado um número predefinido de possíveis interrupções, um tratador pode ser implementado usando uma tabela de ponteiros para as rotinas de tratamento das interrupções. Alguns sistemas operacionais bem conhecidos como o MS-DOS e UNIX despacham interrupções desta forma [3].

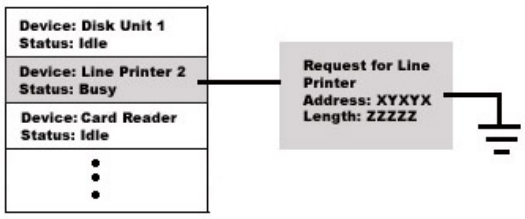

Figura 2.1: Tabela de Status dos dispositivos [4]

As placas controladoras dos dispositivos possuem uma interface bem definida de operações, da qual seu driver possa fazer operações neste dispositivo passando parâmetros ou não. Quando as operações na placa controladora do dispositivo não são simples e exigem um tempo de processamento na placa, o processador pode deixa-lá executar as operações e enquanto isso pode fazer outras coisas.

Dependendo da arquitetura de hardware envolvida, com o lançamento de uma interrupção as outras interrupções são desabilitadas até que seu tratamento seja completado.

Isto se deve ao fato do problema que poderia ocorrer se fosse lançado uma segunda interrupção simultânea, onde a segunda poderia sobrescrever algum valor que estava sendo usado pela primeira e esta seria perdida. Um caso mais grave gerado por interrupções simultâneas seria do sistema parar.

2.3 Sistema de I/O

O sistema de I/O de um sistema computacional simples, consiste de uma CPU (Central Processing Unit) e um número de controladores de dispositivos, que são conectados através de um barramento comum. Estes controladores de dispositivos podem fornecem acesso à memória compartilhada para a execução de determinada tarefa.

Cada controlador de dispositivo gerencia um tipo específico de dispositivo (discos rígido, monitor de vídeo, disquete, etc). A CPU e os controladores executam de forma concorrente, competindo por ciclos de memória. O controlador da memória deve ordenar e sincronizar o acesso a memória [3].

Os controladores de dispositivos são responsáveis por mover os dados entre os dispositivos periféricos e armazená-los em um buffer.

Quando a CPU recebe um sinal de interrupção ela pára de executar as instruções que estava executando e armazena seu "estado", ou seja, os valores de registradores de dados e de controle. Isto é necessário para que o processador volte ao seu estado normal de execução após o tratamento da interrupção recebida.

Para iniciar uma operação de I/O, a CPU examina o controlador do dispositivo para determinar que ação executar.

Vamos supor que algum controlador encontre uma requisição de leitura. Este irá começar a transferência de dados do dispositivo para seu buffer local. Uma vez terminada esta operação, o controlador informa à CPU que ele acabou sua operação, e assim a CPU pode carregar os resultados obtidos para tomar alguma decisão.

Existem sistemas de I/O que são capazes de manter várias requisições de I/O ao mesmo tempo e ainda assim assegurando que estas não se percam. Para isso todas as requisições que são feitas são armazenadas em uma tabela de dispositivo/status. Cada linha da tabela corresponde a um tipo de dispositivo bem como seu "status"(espera, ocupado) e um ponteiro para cada estrutura requisição deste dispositivo.

Estas estruturas variam de acordo com o tipo de dispositivo, e contêm atributos como nome do arquivo (para unidade de discos), operação (leitura, escrita), endereço do dispositivo, etc. Uma figura esquemática pode ser vista na figura anterior.

2.4 EPOS - Embedded Parallel Operation System

O principal objetivo do EPOS é definir um sistema operacional altamente adaptável para suportar sistemas computacionais dedicados, em particular os paralelos e embutidos.

O sistema EPOS nasceu em 1997 no Instituto de Pesquisa de Arquiteturas de Computadores e Engenharia de Software (FIRST) do centro nacional alemão de pesquisa da tecnologia da informação (GMD) como um projeto experimental[1].

Como EPOS foi desenvolvido utilizando-se de conceitos avançados de engenharia de software, existe uma estratégia de configuração do sistema operacional de acordo com a necessidade particular da aplicação.

Com isso as abstrações do EPOS resultam da modelagem de entidades de domínio correspondentes, como um conjunto de componentes reusáveis e readaptáveis.

As abstrações de um sistema operacional embutido de alta performance definidas para o sistema EPOS podem ser vista na figura abaixo. Esta decomposição do domínio computacional em entidades foi inspirado nos principais livros de sistemas operacionais tais como [3][4].

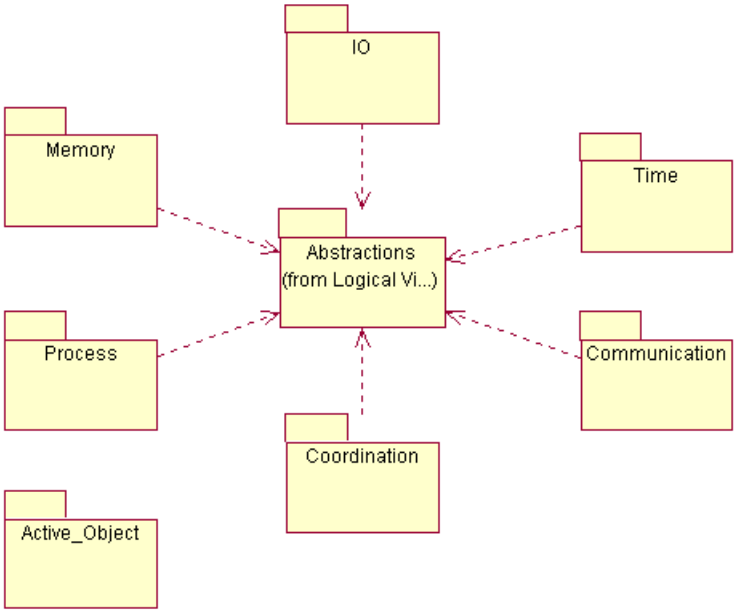

Figura 2.2: Grupo de famílias de abstrações no EPOS [6]

Para que a aplicação obtenha um sistema operacional customizado, EPOS emprega componentes de software para implementar um conjunto de abstrações independentes de cenário. Estes podem ser adaptados para uma dada execução de um cenário com a ajuda dos adaptadores de cenários.

Como a proposta é de projetar um sistema orientado à aplicação, as abstrações de EPOS foram modeladas independentes da execução em específicas arquiteturas de sistemas.

Estas abstrações são armazenadas em um repositório e são exportadas para os programadores da aplicação via interfaces infladas. Isto permite aos programadores manipular uma família de abstrações como se fosse uma única abstração.

Esta estratégia além de reduzir drasticamente o número de abstrações exportadas, capacita os programadores facilmente expressar suas solicitações das aplicações relativas ao sistema operacional[1].

A figura abaixo nos dá uma visão da estrutra do sistema EPOS.

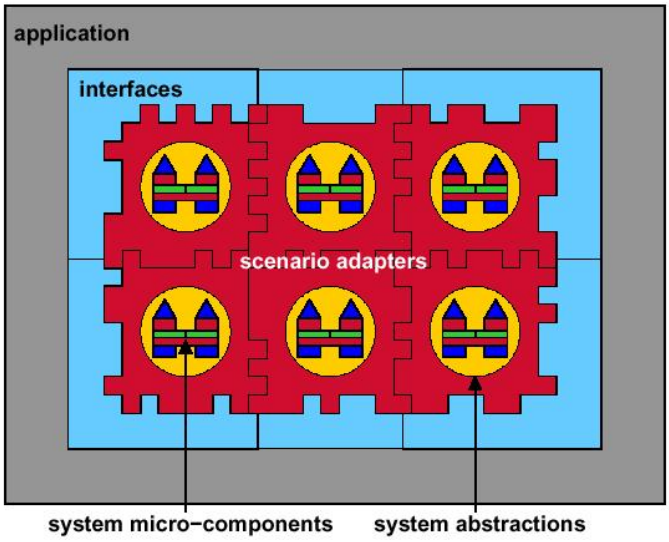

Figura 2.3: Filosofia do EPOS [2]

O projeto é então refinado pela análise de dependência contra informação sobre o cenário de execução adquirido do usuário via ferramentas visuais.

A consequência deste processo é um conjunto de chaves seletivas que suportarão a compilação do sistema operacional orientado a aplicação.

2.5 Lego Mindstorms

2.5.1 O RCX da Lego

O RCX[1] é uma peça programável que é capaz de operar três motores e três sensores simultaneamente.

O RCX comunica através de uma interface serial de infra-vermelho de um PC ou Palm Pilot. Esta é uma das peças encontradas na caixa "Lego Mindstoms Robotics Invention System".

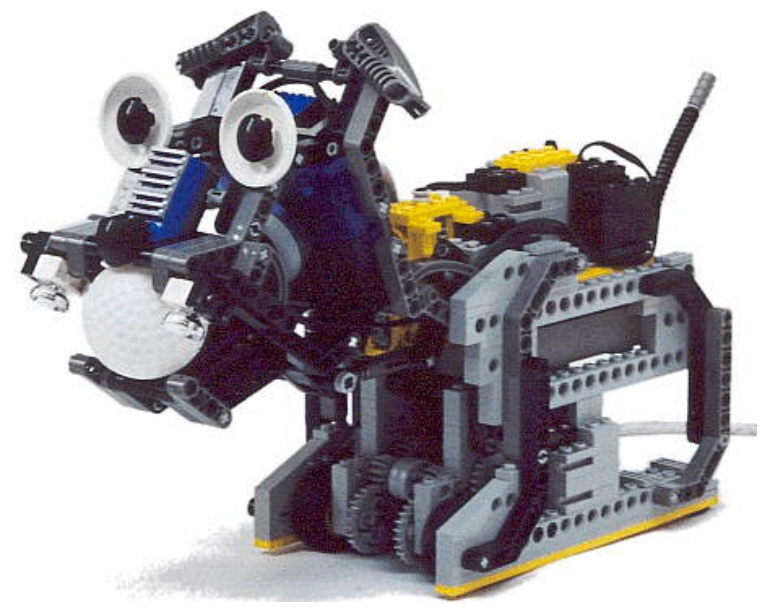

Figura 2.4: Um cão que busca uma bola[10]

Esta peça é controlada por um microcontrolador Hitachi H8/300L com 32 KB de RAM externa. Este microcontrolador é usado para controlar os módulos externos como sensores, motores e a comunicação com o PC. Além disso, contém um chip de 16 KB em memória ROM, que matém drivers e

[1]Robotic Command eXplorer

funções para o tratamento de algumas condições em que o hardware pode se encontrar.

O gerenciamento total do RCX pode ser extendido com o download de um sistema operacional embutido. O RCX também é capaz de rodar programas de usuários. Estes programas são carregados em linguagem nativa para o RCX através de uma torre de raio infra-vermelho, e armazenados em uma região da memória. Então o sistema operacional deve interpretar e executar o programa.

O programa desenvolvido é carregado no RCX através de um programa em um PC que manda os bytes para uma torre (através de uma porta serial), que emite informações por infra-vermelho ao RCX. Este possui recepção infra-vermelha e carrega as instruções na memória.

O RCX possui um display do qual diferentes informações podem ser visualizadas. Podem ser mostrados cinco letras/números, além de outros símbolos especiais para identificação do estado do mesmo.

Possui quatro botões de controle com as seguintes finalidades:

ON-OFF - É usado para ligar/desligar o RCX.

View - Visualiza a operação corrente do RCX.

Run - Roda o programa selecionado.

Prgm - Pode escolher o programa desejado (se possuir mais que um na memória!).

O RCX tem suporte para a execução de sons através de um mini alto-falante embutido.

Vem com módulos externos que podem ser conectados a ele dos quais destacam-se os motores e sensores de toque, luz e temperatura.

Figura 2.5: Foto ilustrativa do RCX[11]

2.5.2 O microprocessador HITACHI H8/300L

Algumas características de hardware encontrados no microprocessador H8/300L é que este possui em sua arquitetura 16 registradores de 8 bits (ou 8 registradores de 16 bits) que podem ser usados pelas instruções para o cálculo tanto de endereços como dados. Um destes registradores (R7) também funciona como stack pointer[2], usado implicitamente pelo hardware no processamento de interrupções e chamadas de subrotinas.

O microprocessador possui uma frequência de 3MHz e 37 fontes de interrupções, dos quais 13 são externas e 24 internas. Na sua ROM possui algumas rotinas tratadoras destas interrupções.

Pode interpretar 55 instruções básicas com suporte para operações matemáticas, movimentação dos dados, comparações e operações lógicas. A CPU[3] suporta 64KB de espaço de endereçamento (programa e dados).

[2]Ponteiro para a pilha da memória

[3]Central Processing Unit - Chip onde ocorre o processamento das instruções.

Possui dois registradores de controle que são úteis para a verificação de algumas características em determinado momento do processamento. Estes são:

PC (Program counter) - Registrador de 16 bits que indica o endereço da próxima instrução na memória.

CCR - Registrador composto por 8 bits do qual cada um possui um significado:

- **I (Interrupt mask bit)** - Quando é 1, todas interrupções são mascaradas(exceto algumas que não podem ser mascaradas).
- **U (User bit)** - Bit que o usuário pode usar para se fazer algum controle.
- **H (Half Carry)** - Este bit é usado para operações de adição, subtração e comparação e indica empréstimo.
- **U (User bit)** - Outro bit que pode ser usado para se fazer algum controle.
- **N (Negative)** - Este indica o valor do bit mais significativo (bit sinal) do resultado da instrução.
- **Z (Zero)** - Este é setado para 1 para indicar um resultado igual a 0.
- **V (Overflow)** - é setado para 1 se um overflow ocorrer.
- **C (Carry)** - empréstimo do bit mais significativo.

A memória RAM é conectada à CPU via 8 bits de barramento de dados, onde ambos, byte e word são acessados em dois tempos. Isto capacita uma rápida transferência de dados e execução da instrução.

Entre os vários tipos de timers encontrados no microprocessador, podemos destacar o "Timer Watchdog"que pode monitorar as operações de sistema. Se qualquer coisa der errada, o timer reseta a CPU, ou gera uma interrupção não-mascarada.

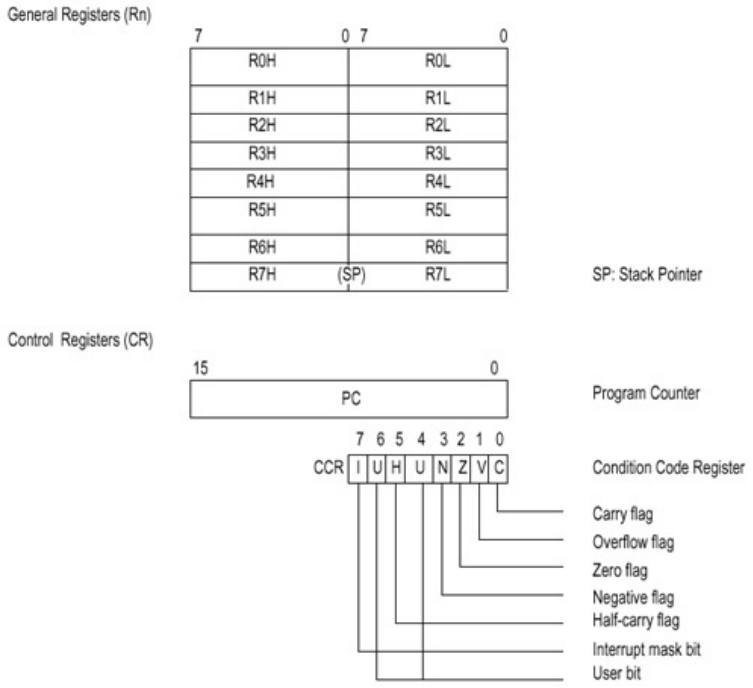

Figura 2.6: Foto ilustrativa dos registradores H8/300L[12]

Quando a função watchdog não é necessária, este timer pode ser usado para a contagem de um intervalo de tempo, mas neste modo isto requere uma flag de "overflow", onde ocorre uma interrupção cada vez que o contador der "overflow".

2.5.3 Tratamento de interrupção no Hitachi H8/300L

Quando o estado de tratamento de interrupção é iniciado, a CPU pega um endereço inicial da tabela de vetores de tratadores de interrupção que fica na ROM e começa a executar a rotina tratadora deste endereço. Durante a execução desta rotina, todas as interrupções incluindo NMI[4] são mascaradas.

[4]Not Maskarable Interrupt

Quando o tratamento de interrupção é iniciado, a CPU refere-se ao registrador sp (R7) e salva os registradores PC e CCR na pilha. O bit CCR.I é setado para 1 e o endereço inicial da rotina tratadora é adquirido da tabela de vetores de tratamento de interrupções, do qual esta rotina será executada deste endereço.

Reset tem a mais alta prioridade das interrupções do RCX. Quando uma requisição de uma interrupção é feita, o tratamento da interrupção começa depois da execução da presente instrução (completada).

Este tratador pode ser usado para controlar os motores, sendo que eles precisam de um padrão a intervalos regulares de velocidade.

Capítulo 3

Estudos de Caso

A seguir serão apresentados estudos de caso de abstrações de I/O de alguns sistemas operacionais, do qual gera grande parte das interrupções em um sistema computacional.

3.1 I/O do sistema UNIX

Um dos propósitos de um sistema operacional é esconder peculiaridades de específicos dispositivos de hardware do usuário [3].

A solução que o sistema UNIX encontrou para seu sistema de I/O é integrar todos os dispositivos no sistema de arquivos, chamados de arquivos especiais. Todos os dispositivos de I/O recebem um nome e ficam geralmente no diretório /dev.

Os arquivos especiais podem ser acessados da mesma forma que os arquivos comuns. Então se queremos imprimir um arquivo, podemos acessar a impressora com o comando:

$prompt$ cp file /dev/lp

Desta forma o arquivo será impresso (assumindo que o usuário tenha permissão). Os programas podem ler e escrever em arquivos especiais da mesma forma que em arquivos comuns. Nota-se que não é necessário nenhum mecanismo especial para a realização de I/O.

Uma outra vantagem deste sistema é que as permissões para acessar os dispositivos de I/O são as mesma usada para acessar arquivos comuns (já que os dispositivos são mapeados em arquivos especiais), ou seja, se o superusuário quiser restringir o uso de determinado dispositivo para seus usuários, ele pode fazer a modificação na permissão do arquivo que referencia aquele dispositivo.

Os arquivos especiais são divididos em duas categorias:

Arquivo especial de bloco - composto por uma seqüência de blocos numerados (geralmente 512 bytes). A propriedade deste tipo de arquivo especial é que determinado bloco pode ser endereçado e acessado individualmente. Geralmente usado para operar discos para isolar detalhes como: trilhas, cilindros, etc.

Arquivo especial de caracter - são usados por dispositivos de I/O que possuem o fluxo de dados através de um stream de caracteres. Utilizam destes arquivos dispositivos como impressoras, placas de rede, mouses, etc.

Um processo no UNIX começa com três arquivos já abertos, o de entrada padrão, o de saída padrão e o de erro, todos conectados ao terminal.

O sistema de I/O do UNIX consiste em se ter drives de dispositivos em geral e drives para dispositivos de hardware específico, onde somente o driver do dispositivo conhece peculiaridades do hardware específico.

Os drives de dispositivos devem ser compilados dentro do kernel do sistema não podendo ser instalados depois.

3.2 I/O do sistema MS-DOS

O MS-DOS também suporta arquivos especiais de caracteres tal como no sistema UNIX, para realizar I/O para dispositivos seriais. A única diferença é que estes arquivos especiais não residem em um diretório do tipo /dev.

Para copiar um arquivo do console (tela), o comando:

$prompt$ copy file con

Pode ser usado. O arquivo "con" também pode ser aberto por outros programas de maneira idêntica ao arquivo /dev/tty no UNIX.

A abertura de um arquivo especial de caracteres retorna um descritor de arquivo, o qual pode ser usado para leitura e escrita deste arquivo. Outros arquivos especiais de caracteres incluem com1 (primeira porta serial), lpt1 (porta paralela para impressora), etc.

No MS-DOS um processo começa com cinco arquivos abertos automaticamente: o de entrada padrão, o de saída padrão, o de erro, o da linha serial e o da impressora, que usam os descritores 0 a 4, respectivamente [4].

Neste sistema é permitido que usuário instale os drives dos dispositivos após o boot do sistema. Um driver de um dispositivo é instalado com a adição de um simples comando ao arquivo config.sys, fornecendo o nome do caminho onde contém o driver.

Capítulo 4

Tratamento de interrupções no EPOS

A interação entre aplicações e dispositivos periféricos é gerenciada pelas famílias de abstrações representadas na figura abaixo.

.

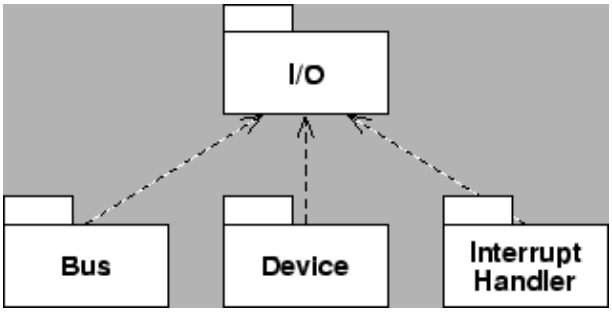

Figura 4.1: Família de abstrações para o gerenciamento de I/O [6]

Esta figura representa as famílias de abstrações para o gerenciamento de I/O no sistema EPOS. Dispositivos periféricos são abstraídos pelos membros da famíla *Device*, de uma maneira que é conveniente para as aplicações. Contudo, sistemas dedicados freqüentemente usam dispositivos dedicados que não serão encontrados nesta família.

Neste contexto, a família *Bus* é responsável por detectar e ativar dispositivos conectados ao barramento, os quais são abstraídos como membros criados da família *Device*.

A família que foi tema de pesquisa para este trabalho se trata da *Interrupt_Handler*, que trata de abstrações que permitem as aplicações tratar as interrupções geradas por um dispositivo.

O tratador de interrupções do sistema EPOS permite a processos da aplicação tratar interrupções geradas pelo hardware à nível de usuário, via uma família de abstrações como mostrado na figura abaixo.

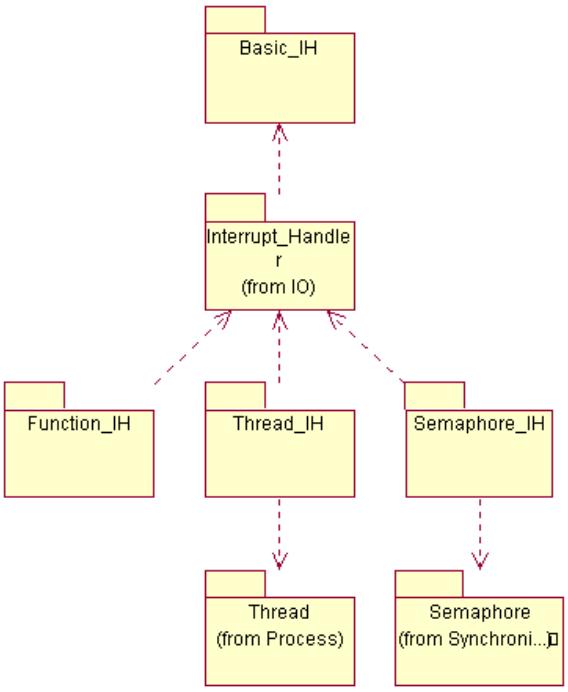

Figura 4.2: Família do tratador de interrupções[6]

Os membros desta família dissociada podem ser usados simultaneamente. O membro *Function_IH* da família associa uma função fornecida pela aplicação para tratar uma interrupção. Somente a primeira vez que uma interrupção é lançada, o sistema transforma tal função em um tratador de interrupções que é chamado. Assim, temos somente um "overhead" no sistema de tratamento da interrupção na primeira vez que o tratador é chamado.

O membro *Thread_IH* ativa uma thread para tratar uma interrupção, e esta deve ter sido previamente criada pela aplicação no estado "suspenso".

Esta thread é "acordada"toda vez que uma correspondente interrupção é gerada. Depois de tratada a interrupção, a thread deve retornar para o estado "suspenso".

Estes membros apresentados da família *Interrupt_Handler* podem apresentar problemas se interrupções são sucessivamente geradas enquanto o tratador associado está ativo. O tratador *Function_IH* passa o controle para a aplicação, que deve então assegurar que o tratador é rápido o suficiente, ou implementar uma função reentrante.

O tratador *Thread_IH* em compensação não é afetado por este problema, pois se uma interrupção tentar ativar uma thread que já esta ativa, simplesmente não tem nenhum efeito, só que esta última interrupção será perdida.

Como podemos perceber, os membros da família explicados acima sofrem algumas restrições e irão depender muito da aplicação a ser desenvolvida pela escolha de qual membro da família usar.

O framework fornece as alternativas para tipos diferentes de aplicações, basta o desenvolvedor saber qual tipo de tratador usar.

O membro proposto que contorna os problemas anteriores é o membro *Semaphore_IH*. Quando há a emissão de uma interrupção que poderia ser perdida, esta é endereçada por um *Semaphore_IH* que aponta para um semáforo previamente criado pela aplicação e inicializada com zero. O sistema operacional chama a operação "V"neste semáforo em todas as interrupções, enquanto a thread tratadora chama a operação "P"para esperar por uma interrupção.

Esta estratégia de tratamento de interrupções dá um sabor de produtor/consumidor e previne que as interrupções comecem a serem perdidas.

Entretanto, prevenir que as interrupções comecem a serem perdidas pode não ser o bastante para permitir que dados de I/O não irão ser perdidos. Um dispositivo que gera subseqüentes interrupções enquanto alguma outra já esteja sendo tratada, deve ter memória o suficiente para acumular eventuais dados.

O tratador deve compreender que é capaz de casar com a velocidade dos dispositivos operacionais, além disso, dependendo da política de escalonamento, interrupções podem não ser tratadas imediatamente. Por exemplo, se a thread responsável para tratar uma interrupção tem mais baixa prioridade do que a thread corrente que está sendo executada, o tratador irá enfileirá-lo para execução posterior.

Além destes membros já existentes da família tratador de interrupções do sistema EPOS, poderão ainda ser desenvolvidas outros membros com o intuito de fornecer uma maior gama de tipos diferentes de tratadores. Assim vários tipos de aplicações poderão encontrar o tratador de interrupção que necessitem.

Capítulo 5

Implementação

A maratona de implementação começou com a decisão de qual dos tratadores de interrupções disponíveis no sistema EPOS poderia ser utilizado para tratar as interrupções do microprocessador HITACHI H8/300L.

Com a análise dos requisitos e dos recursos disponíveis, achei melhor escolher o membro *Thread_Handler* para tratar as interrupções, levando em conta que o RCX não lança várias interrupções simultâneas e que podem ser tratadas tranqüilamente por uma "Thread" tratadora.

Além do mais a aplicação de controle tem a possibilidade de desabilitar outras interrupções enquanto outra seja completamente tratada.

Alguns dispositivos do RCX tais como alguns sensores, poderiam lançar interrupções na sua leitura, mas não lançam. Assim a aplicação que irá rodar no sistema operacional embutido, precisa fazer esta leitura dos valores lidos pelos sensores temporariamente.

Primeiramente para a utilização do membro *Thread_Hadler*, foi necessária a compreensão da família Thread para assim poder usar o tratador escolhido de forma correta.

A primeira dificuldade encontrada, foi o compreendimento de como funcionava o lançamento das interrupções no RCX e de que forma poderiam tratá-las corretamente.

A maioria das interrupções lançadas no RCX tem funções tratadoras na ROM, e para tratá-las, é necessário chamar o endereço desta rotina tratadora através de um uma instrução de desvio em assembly.

Uma grande dificuldade que obtive foi de compreender como é feita a programação de baixo nível no hardware do RCX, bem como integrar esta programação com a programação de alto nível na linguagem C++.

Com isso, o tratador desenvolvido teve que ser capaz de saber qual rotina tratadora chamar na ROM para determinada interrupção lançada.

Para que o tratador desenvolvido fosse portável para outras arquiteturas de hardware, utilizou-se um mediador que serve para chavear o comportamento de baixo nível do tratador. Assim para cada arquitetura de hardware, deve haver um mediador correspondente para sua própria arquitetura de hardware.

Uma outra dificuldade encontrada foi na forma de depurar o tratador desenvolvido, pois fica difícil ler valores em uma tela de cristal líquido e saber se este está correspondendo bem com o resto do sistema. Além de possuir vários tipos de interrupções e saber qual delas o tratador está executando.

Neste tipo de arquitetura de hardware, geralmente a aplicação precisa de ao menos dois tratadores. Um que fornece um tempo de sistema para a possibilidade de eventos de timers, e um outro para chavear eventos de dispositivos tais como sensores e botões.

Capítulo 6

Conclusão

O desenvolvimento do membro *Thread_IH* da família tratador de interrupções do sistema EPOS, foi um desafio que gerou resultados satisfatórios.

O tratador de interrupções desenvolvido forneceu à aplicação do sistema o tratamento adequado das interrupções do microprocessador HITACHI H8/300L, de acordo com a necessidade da mesma.

Com a análise e verificação dos resultados obtidos do tratador desenvolvido, pretende-se ainda estudar novas e mais eficientes formas de melhorar esta tarefa que é uma das mais importantes em um sistema computacional.

Este tratador de interrupções forneceu o tratamento adequado dos dispositivos que lançavam interrupções, tais como os botões ON/OFF e RUN, entre outras.

O tratador também pode fornecer em tempo hábil uma resposta ao tratamento das interrupções geradas pelo hardware.

Também pode aceitar níveis de prioridades de tratamento das interrupções, de acordo com sua relevância para o sistema. Assim, interrupções de mais al-

ta prioridade acabam sendo processadas prioritariamente na frente de outras de mais baixa prioridade.

Através do mediador implementado obteve a independência do hardware envolvido, mas isto merece um pouco mais de atenção para vários tipos de arquiteturas.

O desenvolvimento do membro *Thread_Handler* da família *Interrupt_Handler* do sistema EPOS foi gratificante e de um grande acréscimo intelectual usando várias áreas das ciências da computação, apesar das dificuldades encontradas.

Referências Bibliográficas

[1] **FRÖHLICH**, Antônio Augusto Medeiros; *Application-Oriented Operating Systems*, Junho 2001.

[2] **FRÖHLICH**, Antônio Augusto Medeiros; *System Software Component Engineering*, Fevereiro 2002.

[3] **GALVIN**, Peter Baer; SILBERSCHATZ, Abraham; *Operating System Concepts*, Fifth Edition, 1999.

[4] **TANENBAUM**, Andrew S. ; *Sistemas Operacionais Modernos*, 1995.

[5] **SECCHI**, Luciano; *Mecanismo para Suportar I/O Fora do Kernel do Aboelha*, Setembro de 1998.

[6] **EPOS**, Página do EPOS, *http://epos.lisha.ufsc.br*

[7] **RCX**, Manual microcontrolador HITACHI, *http://www.hitachi-eu.com*

[8] **CHAOS**, Sistema operacional para RCX, documentação disponível em *http://www.cs.auc.dk/∼nesotto/*

[9] **LEGOS**, Sistema operacional para RCX, *http://www.noga.de/legOS/*

[10] **Página Pessoal**, Robos feitos com o kit Lego Mindstorms, *http://jpbrown.i8.com/index.html*

[11] **LEGO**, Site oficial Lego Mindstorms, *http://www.legomindstorms.com*

[12] **HITACHI**; *H8/300L Series Programming Manual.*

[13] **WANNER**, Lucas; *Introdução ao LaTeX*, Julho de 2002.

[14] **ANDERSEN**, Dennis; **THOMSEN**, Casper; **KRISTENSEN**, Dennis; **OTTOSEN**, Thorsten J.; **SONDBERG-MADSEN**, Nicolaj; **DALUM**, Martin; **KALDAL**, Michael H.; *CHAOS, The RCX OS where chaos is only in the name*

Anexo 1 - Código Fonte