Firmware para um Sistema de Comutação de Pacotes Utilizando Arquitetura x86 e MIPS-el

Pedro Arthur P. R. Duarte[1], **Alysson M. Oliveira**[1] **e Diogo P. F. Pedrosa**[2]

[1]Laboratório de Redes e Sistemas Distribuídos
[2]Laboratório de Otimização e Inteligência Artificial
Departamento de Informática - Universidade do Estado do Rio Grande do Norte
Caixa Postal 70 – 59.633-010 – Mossoró – RN – Brasil

pedroarthur.jedi@gmail.com,{alysson, diogopedrosa}@uern.br

Abstract. *Embedded systems are composed of a combination of hardware and software specially developed for the accomplishment of a specific task. Among the sofwares used for the development of embedded devices, the Linux operating system has a prominent role. However, was identifyed that the use of Linux as the software layer for Ethernet packet switching devices is not as exploited as in other types of devices. This paper describes the development of a Linux based packet switching firmware to be used in different architectures.*

Resumo. *Sistemas embarcados são compostos da combinação de hardware e software especialmente desenvolvidos para a execução de uma tarefa específica. Dentre os* sofwares *utilizados para o desenvolvimento de dispositivos embarcados, o sistema operacional Linux tem um papel de destaque. Contudo, percebe-se que a utilização do Linux em comutadores de pacotes Ethernet não é tão explorada quanto em outros tipos de equipamentos dedicados. Este artigo descreve o desenvolvimento de um* firmware *baseado em Linux para ser utilizado em equipamentos de comutação de pacotes com arquiteturas distintas.*

1. Introdução

Sistemas embarcados de processamento de informação consistem em uma combinação de *hardware*, *software* e outros elementos adicionais. Eles são voltados para a execução de uma ação dedicada [Barr and Massa 2006]. Tais sistemas são dotados geralmente de um sistema operacional e um, ou mais, *softwares* que são responsáveis pela função exercida pelo equipamento [de Oliveira 2003].

A utilização do sistema operacional Linux em sistemas embarcados tem crescido devido às vantagens tecnológicas e econômicas [Hallinan 2006]. O Linux oferece eficiência e confiabilidade de desenvolvimento pois provê toda a camada de gerência de *hardware*, restando apenas o porte das aplicações relacionadas a seu funcionamento. Aliado a isso, tem-se que o núcleo está em uma licença livre, o que reduz os custos de desenvolvimento e comercialização [Yaghmour 2008]. O Linux também apresenta alto grau de escalabilidade, existindo suporte para arquiteturas que vão desde o MIPS aos processadores 64 bits mais modernos [Raghavan et al. 2006].

Há uma quantidade significativa de aplicações de sistemas embarcados utilizando *softwares* baseados em código aberto [Yaghmour 2008]. Dentre elas, uma que se mostra particularmente interessante é a área de comutação de pacotes em redes Ethernet.

Comutadores de pacotes de rede *Ethernet* são dispositivos que permitem que cada estação de trabalho tenha uma porta dedicada, evitando colisões de pacotes. Além disso, também permitem conexões *full-duplex* fim-a-fim, o que aumenta a performance total da rede [Doherty et al. 2007]. Os sistemas de comutação de pacotes de redes *Ethernet* têm evoluido e agregado recursos nos últimos anos. Porém, muitos deles ficam limitados a um conjunto específico de fornecedores e a um *hardware* de alto custo, estando os mais simples limitados a sua função original.

É relativamente comum encontrar projetos de roteadores que utilizem o núcleo do Linux como sistema operacional (LinkSys[1], Quagga[2], Zebra[3]). No entanto, verificou-se que a utilização de tais sistemas como camada de *software* para comutadores de pacotes não foi ainda devidamente explorada. Desta forma, este artigo apresenta uma camada de *software* para a utilização de arquiteturas x86 e MIPS como dispositivos de comutação de pacotes em redes Ethernet. Esta camada usa o núcleo do sistema operacional Linux como plataforma alvo e as ferramentas do ambiente GNU como plataforma de desenvolvimento. Além disso, o projeto desta camada é realizado de tal forma que o *firmware* seja portável para outras arquiteturas, tais como ARM e PowerPC.

2. Desenvolvimento e Implementação de Sistemas Embarcados

O desenvolvimento e a implementação de sistemas embarcados utilizando o sistema operacional Linux envolve, segundo [Yaghmour 2008], a criação do sistema alvo, escolha de ferramentas de desenvolvimento e a implementação do sistema.

A fase de criação do sistema alvo pode ser subdividida em quatro etapas: (1) determinação dos componentes do sistema, (2) otimização do núcleo, (3) construção do sistema de arquivos e (4) a escolha do gerenciador do *boot*.

Na primeira etapa os componentes do sistema são determinados em tempo de análise do projeto. Uma das principais tarefas desta fase é a escolha e o congelamento da versão do núcleo do sistema operacional, o que provê abstrações do *hardware* da máquina. Além disso, o núcleo torna o processo de desenvolvimento mais eficiente visto que os desenvolvedores necessitam apenas preocupar-se com os detalhes da aplicação. Ainda nesta fase, são selecionados módulos relacionados com os dispositivos do sistema (processador, recursos necessários a aplicação, etc.).

A otimização do núcleo visa tornar o sistema mais leve do ponto de vista do armazenamento e da carga de memória. Ela envolve ativar recursos relativos ao desenvolvimento de sistemas embarcados e desativar recursos desnecessários.

A construção do sistema de arquivos consiste na definição da organização do sistema raiz e na escolha dos pacotes de *softwares* que serão usados para o desenvolvimento. Somente pacotes essenciais são selecionados devido à limitação de armazenamento da maioria dos sistemas embarcados.

O gerenciador de *boot* se mostra como uma camada de *software* responsável por carregar o sistema operacional e desviar o fluxo de processamento para esse. Estando o

[1] http://www.linksys.com/
[2] http://www.quagga.net/
[3] http://www.zebra.org/

processo de *boot* intimamente ligado à arquitetura do sistema, ele deve ser então escolhido conforme a capacidade da arquitetura utilizada.

3. Desenvolvimento do Sistema de Comutação de Pacotes

Seguindo as etapas citadas na seção 2, a criação do ambiente de desenvolvimento para o sistema de comutação de pacotes proposto consistiu na escolha de compiladores, depuradores e ambientes de desenvolvimento integrado, formando assim o ambiente de compilação cruzada. Essa não foi uma tarefa trivial, visto que as diferenças dos níveis ISA das arquiteturas de desenvolvimento e alvo exigem ferramentas diversificadas. A etapa seguinte tratou da compilação do núcleo e pacotes, além do embarque do sistema para o desenvolvimento e teste da aplicação final.

O levantamento de requisitos foi uma etapa fortemente ligada à aplicação proposta (comutação de pacotes). Estes requisitos foram obtidos nos *sites* de fabricantes de equipamentos da área de comutação de pacotes como Cisco, 3Com e IntelBrás. Dentre as características dos produtos pesquisados se destacaram VLANs, priorização de tráfego, interface de gerenciamento via WEB, interface de gerenciamento/monitoramento SNMP e filtro de *frames* de camada de enlace.

3.1. Arquiteturas de Teste

A arquitetura selecionada para o sistema embarcado de comutação de pacotes *Ethernet* foi a x86. Para que testes reais pudessem ser realizados e validados, foi montado um protótipo de comutador de pacotes utilizando-se *hardware* convencional. Esse protótipo consistiu de um processador Pentium II, com 433 Mhz, 128 MB de RAM, 8 GB de armazenamento magnético e 5 interfaces de rede 3Com modelo 3c905c-TX.

Além deste protótipo, foi adquirida uma outra arquitetura com processador MIPS-el de 175 Mhz, 32 MB de memória RAM e 64 MB de armazenamento NAND. Ela é equipada com o chipset de rede ADM 5120 de 5 interfaces. Esse equipamento se mostrou importante pois possibilitou uma primeira experiência com o porte do *firmware* para outras arquiteturas distintas da x86.

3.2. Escolha do Núcleo do Sistema

O Linux versão 2.6.26.8. foi escolhido como o núcleo do sistema. Ele apresenta suporte a uma vasta gama de interfaces de rede, dispositivos de armazenamento de bloco e arquiteturas alvo, sendo também uma versão considerada estável pelos desenvolvedores.

3.3. Criação do Ambiente de Desenvolvimento

O ambiente de desenvolvimento foi composto pelo compilador GCC versão 4.1.2, depurador dinâmico GDB 6.6 e do ambiente de desenvolvimento integrado Kate. Dois ambientes de desenvolvimento diferentes foram criados pois o projeto contemplou duas arquiteturas de testes distintas, a x86, objetivo principal do trabalho; e MIPS-el, o que garantiria a portabilidade do *firmware*.

3.4. Criação do *Firmware*

Foram gerados dois *firmwares*: o primeiro foi destinado a arquiteturas x86 e o segundo para arquiteturas MIPS-el. Neles foram instalados ferramentas de configuração do sistema, bibliotecas para a aplicação a ser desenvolvida e ferramentas auxiliares, como editor de texto ASCII, decodificadores de pacotes, analisadores de desempenho, dentre outros.

A organização do sistema raiz do *firmware* seguiu o padrão internacional *Filesystem Hierarchy Standard* [Russel et al. 2004], definido pela LSB[4] (*Linux Standard Base*), com pequenas adaptações para adequação aos propósitos do sistema.

3.5. Escolha do Gerenciador de *Boot*

Para a arquitetura x86, o gerenciador de *boot* escolhido foi o LILO em sua versão 22.8. Esse gerenciador apresenta grande adaptabilidade podendo carregar o sistema operacional a partir de vários tipos de mídias, tais quais discos rígidos, CDs e memórias flash. Além disso, ele permite a utilização dos diversos sistemas de arquivos.

Para a arquitetura MIPS, o gerenciador de *boot* foi o RouterBOOT, instalado por padrão pelo fabricante do dispositivo. Esse gerenciador possibilita somente a carga do sistemas de dispositivos de armazenamento NAND e do sistema de arquivos YAFFS.

4. Implementação do Mecanismo de Comutação Básico

O mecanismo de comutação de pacotes foi implementado utilizando o módulo 802.1d *Ethernet Bridge*, a camada de enlace da pilha de protocolos TCP/IP do núcleo do Linux e ferramentas de espaço de usuário. A abordagem utilizada foi a *store-and-forward*, a qual consiste em receber todo o frame, realizar verificações sobre ele e encaminhá-lo para seu destino [Doherty et al. 2007]. Além de elevar a confiabilidade do mecanismo de comutação, essa abordagem foi escolhida devido a uma futura implementação de um filtro de *frames* de enlace.

Essa implementação se mostrou bastante portável para as duas arquiteturas pois utilizou-se somente componentes inerentes ao núcleo do sistema operacional Linux e de ferramentas de fácil porte.

Vale ressaltar que esse mecanismo implementado é básico. Ele possui somente os recursos apresentados por dispositivos de comutação de pacotes não-gerenciáveis. Muitas das funções levantadas durante a análise dos requisitos ainda não foram implementadas.

4.1. Teste do *Firmware*

A fase de testes consistiu no embarque dos *firmwares* em suas respectivas arquiteturas e na implantação do mecanismo de comutação de pacotes criado. O embarque do *firmware* foi composto da cópia do sistema de arquivo para o dispositivo de armazenamento da arquitetura alvo, da instalação do núcleo do sistema e na configuração e instalação do gerenciador de *boot*.

O embarque do *firmware* na arquitetura x86 se mostrou uma tarefa simples e não ocorreram problemas relevantes. Já na arquitetura MIPS-el, o sistema de arquivo o qual o gerenciador RouterBOOT é compatível não estava presente no núcleo do sistema operacional Linux. Outro agravante foi que o dispositivo não permitia a alteração do código do seu gerenciador de *boot*. Para solucionar esse problema, foi utilizado o guia criado pela Aleph One, empresa mantenedora do YAFFS [Bane 2005], e basicamente consistiu na aplicação de *patches* no núcleo do sistema.

Também houve problemas relativos ao alinhamento do núcleo do sistema à memória. Ou seja, as instruções não se encontravam na organização esperada pelo gerenciador, o que acarretava na busca e decodificação de intruções de forma incorreta,

[4]http://www.linuxfoundation.org/

levando o sistema a estados de incosistência. Esse problema foi tratado com o suporte da equipe de desenvolvimento do OpenWRT[5]. De forma geral, a solução para este problema foi obtida através da modificação do endereço de carga do núcleo do sistema operacional no arquivo fonte relacionado a arquitetura MIPS.

4.2. Teste do Mecanismo de Comutação

O teste do mecanismo de comutação se deu em duas etapas. A primeira consistiu na operação em ambiente de testes e a segunda consistiu em testes no ambiente real, utilizando ambas as arquiteturas propostas.

No ambiente controlado foi realizado a troca de dados entre dois sistemas finais. A topologia consistiu em dois computadores de propósito geral tendo como enlace de dados o dispositivo criado. Esse arranjo está ilustrado na figura 1. Inicialmente foram realizados envio de *frames* brutos com intuito de verificar a entrega dos mesmos [Bradner and Mcquaid 1999]. Para tal, foi realizada a transferência de um arquivo de 800 *MB* utilizando os protocolos HTTP, SSH e um pequeno procolo de troca de arquivos sobre UDP. A assinatura MD5 gerada nos nós de destino e origem mostraram que o arquivo foi transferido corretamente.

No ambiente real, o dispositivo desenvolvido foi colocado em cascata com dois outros comutadores, cada qual com mais de 4 sistemas finais conectados. Essa topologia pode ser vista na figura 2. Esse teste foi realizado com o intuito de observar o comportamento em ambiente de produção [Bradner and Mcquaid 1999], principalmente verificar a ocorrência de congelamentos e *deadlocks*. Assim o dispositivo permaneceu operante durante uma semana, não apresentando instabilidades nesse período.

Devido o sistema ainda estar em fase de desenvolvimento os testes relativos à vazão, taxa de perdas e latência não foram realizados. O projeto encontra-se atualmente em uma etapa de finalizção, no qual dados quantitativos estão sendo adquiridos para que se possa averiguar a eficiência do *firmware* implementado.

Figura 1: Topologia no ambiente de testes controlados.

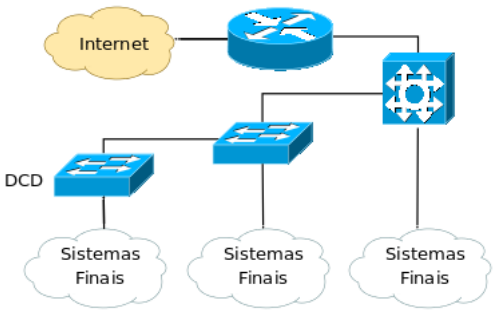

Figura 2: Topologia no ambiente de produção.

5. Conclusões e Trabalhos Futuros

Este artigo apresentou as etapas de desenvolvimento de um *firmware* baseado em Linux para ser aplicado em equipamentos de comutação de pacotes *ethernet*. Percebeu-se que,

[5]http://www.openwrt.org/

embora bastante utilizado em sistemas embarcados, tais *firmwares* desenvolvidos sob plataforma livre não são tão comuns de serem encontrados nesses equipamentos. A camada de *software* desenvolvida foi embarcada em dois equipamentos com arquiteturas distintas, sendo cada uma delas com as devidas adaptações (gerenciamento do *boot*, por exemplo).

Dois testes foram realizados com ambas as arquiteturas. No primeiro deles, utilizou-se um ambiente controlado onde os equipamentos com o *firmware* desenvolvido viabilizavam a troca de dados entre dois sistemas finais. No segundo teste, os equipamentos foram incorporados em um ambiente real de comutação, sendo conectado em cascata com outros comutadores. Os resultados obtidos em ambos os testes foram satisfatórios e instabilidades no sistema não foram observadas.

Para trabalhos futuros, pretende-se proceder com a finalização do *firmware* onde serão incorporados os requisitos ainda não implementados. Além disso, também pretende-se testá-lo com outras arquiteturas não contempladas nesta fase do trabalho, tais como a ARM e PowerPC.

6. Agradecimentos

Os autores agradecem ao Departamento de Pesquisa da Universidade do Estado do Rio Grande do Norte pelo apoio financeiro deste projeto.

Referências

Bane, N. C. (2005). *Brief HOWTO on Incorporating YAFFS as a Root FS*. Disponível em http://www.yaffs.net/howto-incorporate-yaffs. Acessado em 20/03/2009.

Barr, M. and Massa, A. (2006). *Programming Embedded Systems*. O'Reilly.

Bradner and Mcquaid (1999). *Benchmarking Methodology for Network Interconnected Devices*. RFC 2544. http://www.rfc-editor.org/rfc/rfc2544.txt.

de Oliveira, A. M. (2003). *Sistema de Gateway com Interface Wireless/Wired para Plataforma Embarcada x86*. Mossoró, RN, Brasil.

Doherty, J., Anderson, N., and Magiora, P. D. (2007). *Cisco Networking Simplified*. Cisco Press.

Hallinan, C. (2006). *Embedded Linux Primer: A Pratica, Real-World Approach*. Prentice Hall.

Raghavan, P., Lad, A., and Neelakandan, S. (2006). *Embedded Linux System Design and Development*. Auerbach Publications.

Russel, R., Quinlan, D., and Yeoh, C. (2004). *Filesystem Hierarchy Standard*. Filesystem Hierarchy Standard Group. http://proton.pathname.com/fhs/.

Yaghmour, K. (2008). *Building Embedded Linux Systems*. O'Reilly, 2 edition.