ESTUDO E CONFIGURAÇÃO DO SISTEMA OPERACIONAL LINUX PARA SISTEMAS EMBARCADOS

MATHEUS LEITZKE PINTO[1]; JULIO CARLOS BALZANO DE MATTOS[2]

[1]Universidade Federal de Pelotas, Curso de Engenharia de Computação
[2]Universidade Federal de Pelotas, Centro de Desenvolvimento Tecnológico
{mlpinto,julius}@inf.ufpel.edu.br

1. INTRODUÇÃO

Os sistemas embarcados estão presentes no coditiano das vidas das pessoas nas mais diversas aplicações. Os sistemas embarcados são sistemas computacionais que possuem uma funcionalidade dedicada dentro de um sistema mais complexo (MARWEDEL, 2006).

Esses sistemas embarcados diferem de um computador de propósito geral pois é desenvolvido para uma aplicação especifica. Exemplos de sistemas embarcados são encontrados nas mais diversas aplicações, sistemas aviônicos e em automóveis, caixas eletrônicos, etc. Tais sistemas geralmente possuem restrições como memória disponível, processadores com um desempenho suficiente as aplicações e também questões de consumo energético. Isso deve-se ao fato de geralmente estes sistemas estarem alimentados por bateria. Além disso, os sistemas embarcados possuem outras restrições como *time-to-market*, o próprio tamanho entre outras.

No passado, a grande maioria dos sistemas embarcados eram desenvolvidos com processadores simples e linguagem de baixo nível (*assembly*) pois as aplicações não possuíam um alto grau de complexidade. Atualmente, devido o aumento da complexidade desses sistemas e também a disponibilização de mais recursos de hardware, existe a necessidade de fazer uso de um sistema operacional (SO), assim como em computadores de propósito geral. Um SO tem como objetivo abstrair recursos de hardware do desenvolvedor, tornando a programação mais rápida e produtiva. Dessa forma é possível criar aplicações que seriam muito difíceis de desenvolver criando um firmware (software que se comunica diretamente com o hardware).

Os sistemas operacionais embarcados devem, além das características de um SO tradicional, ser capazes de manter um baixo consumo energético e bom processamento computacional em uma determinada estrutura de hardware. Além disso, normalmente são customizados, onde os módulos inseridos são somente os necessários para as aplicações dedicadas Estes sistemas, por restrições de hardware, precisam manter um controle ainda maior dos recursos disponíveis, eliminando, por exemplo, ao máximo qualquer consumo gerado por algum dispositivo que não esteja sendo usado ou que não seja de importância vital para sua aplicação.

No mercado, existem muitos SOs disponíveis para desenvolvimento embarcado, exemplos são TinyOS (TinyOS, 2013), Contiki (CONTIKI, 2013), VirtuOS (MICROBASE, 2013), QNX (QNX, 2013), Windows CE (MICROSOFT, 2013), entre outros. O Linux é um kernel de SO, ou seja, é programa com funcionalidades mínimas de um SO (LINUX, 2013). O Linux pode ser customizado para aplicações especificas de sistemas embarcados, tem portabilidade para mais de 20 arquiteturas de processadores, e seu uso é livre sob a licença GNU General Public License version2 (GPLv2, 2013).

Nesse artigo descreve o estudo e configuração de um sistema operacional Linux em uma plataforma independente de desenvolvimento para projeto de sistemas embarcados.

2. METODOLOGIA

A arquitetura básica de um sistema com Linux consiste no hardware, que é a parte física do sistema, no bootloader que é um programa responsável pela inicialização do hardware e pela carga do sistema operacional, de um kernel Linux, de um sistema de arquivos raiz que contém uma biblioteca C e aplicações e bibliotecas do usuário e de um toolchain que é um conjunto de ferramentas para gerar o binário do sistema. O kernel Linux disponibiliza para as aplicações e bibliotecas de usuário acesso ao hardware através de funções da biblioteca C na qual o kernel foi compilado. Dessa forma, para escrever uma string em um terminal é necessário o uso, por exemplo, da função printf() que contém na sua implementação chamadas de sistema (system calls) do kernel, basicamente são funções disponibilizadas pelo kernel, que se comunicam diretamente com o kernel. Um toolchain consiste basicamente de um compilador C, carregador, ligador e uma biblioteca C.

Para projetar um sistema embarcado é necessária uma plataforma host (um computador pessoal (PC)) para desenvolver o sistema e uma plataforma target (ou plataforma de desenvolvimento), o hardware do sistema embarcado. A comunicação entre esses dispositivos pode se dar por um cabo RS232, USB, de rede, etc. As aplicações desenvolvidas para um PC são geradas através de um toolchain nativo da própria maquina. Como o target geralmente possui poucos recursos de hardware, é uma tarefa bastante dificil gerar o binário do sistema, tal como o do kernel e das aplicações dentro do próprio target. É necessário um toolchain, chamado cross-compiling toolchain, dentro do host que gere o binário para a arquitetura do target e envie a ele.

Além das ferramentas mencionadas, o toolchain necessita dos arquivos de cabeçalho da versão do kernel em uso. A biblioteca C, tanto a usada no sistema como a que gera o sistema, usa constantes, estruturas de dados e chamadas de sistema definidas do kernel e depende de qual a versão do Linux em uso. Para compilar a biblioteca C são necessários os arquivos de cabeçalho da versão do kernel que será portada. Depois de compilada a biblioteca com o cabeçalho apropriado, se compila o Linux correspondente usando a biblioteca C. Dessa forma, nota-se que a biblioteca C e o kernel são interindependentes.

Gerar um toolchain é uma tarefa difícil, pois é preciso se preocupar com detalhes minuciosos do sistema. Apesar de muitos fabricantes de plataformas disponibilizarem toolchains prontos, muitas vezes se quer algo mais personalizado. Dessa forma, existem ferramentas que automatizam o processo de geração de toolchains. O processo de geração é transparente, porém pode-se selecionar quais componentes devem ser gerados. Como exemplo, se tem o crosstool-ng (CROOSTOOL-NG, 2013). Após a geração do toolchain, pode-se através de comandos, compilar uma aplicação para arquitetura target.

É possivel testar a aplicação com a ferramenta Qemu (QEMU, 2013) que emula a arquitetura alvo. Concluida esta etapa é possível gerar o binário do sistema. Primeiramente, gera-se o binário do bootloader. Um bootloader bastante usado em sistemas embarcado é o U-boot (U-BOOT, 2013). No diretório dos códigos fontes do bootloader usa-se comandos específicos para configuração, com um menu semelhante ao do croostool-ng, e compilação dos fontes. O processo de gravação do U-boot depende da plataforma que se esteja usando.

Após esta etapa, configura-se e compila-se o kernel. Os fontes do kernel estão disponíveis em site mantenedor do kernel (LINUX, 2013). O binário do kernel é um código fonte único, porém alguns recursos, tal como drivers de dispositivos, podem ser compilados em binários separados chamados de módulos. No menu de configuração é possível definir o que será compilado como módulo ou será integrado ao código do kernel. Dentro do diretório dos códigos fontes, o binário do kernel é salvo em uma pasta especifica, enquanto os módulos são salvos nos respectivos diretório dos drivers com formato .ko.

Alguns parâmetros devem ser passados ao kernel antes de sua inicialização. O mais importante é o tipo de sistema de arquivos que será montado no diretório raiz do sistema e em qual dispositivo de armazenamento aquele será montado. Um sistema de arquivos é o conjunto de aplicações, bibliotecas de usuários e biblioteca C organizados dentro do dispositivo de armazenamento seguindo algum padrão de sistemas de arquivos. Essa organização dentro do dispositivo é invisível ao usuário, o qual enxerga apenas um modelo de hierarquia através de uma camada entre o sistema de arquivos e o usuário e que seque o padrão Filesystem Hierarcy Standard (FHS, 2013).

Um sistema de arquivos completo com uma grande variedade de ferramentas pode ser obtida através do Busybox (BUSYBOX, 2013). O Busybox pode ser configurado dentro do seu diretório de fontes através de um menu, assim como o kernel. Após isso, é feita a compilação e em seguida a instalação do sistema de arquivos em um diretório qualquer, onde será copiado para algum dispositivo de armazenamento do target. Após o bootloader, o kernel e o sistema de arquivos criados, é necessário transferi-los para um dispositivo de armazenamento do target. A forma como esses binários serão gravados dependerá do dispositivo de armazenamento.

Finalmente, é possível gravar bibliotecas e aplicações no sistema de arquivos raiz do sistema embarcado de acordo com a necessidade do projeto. O Linux possui diversas bibliotecas e aplicações livres que podem ser encontradas na internet.

3. RESULTADOS E DISCUSSÃO

Os métodos expostos nesse artigo para gerar um sistema Linux embarcado são necessários para um melhor entendimento do processo de geração do sistema como um todo, porém são pouco produtivos na prática. Uma opção melhor para gerar um sistema Linux embarcado é através de um Build System (PRADO, 2013). Um Build System é uma ferramenta que gera o toolchain, o bootloader, o kernel e o sistema de arquivos, através de uma interface de configuração única. Algums exemplos de ferramentas Build System são o Buildroot (BUILDROOT, 2013), Yocto (YOCTO, 2013) e OpenEmbedded (OPENEMBEDDED, 2013).

4. CONCLUSÕES

É uma tarefa bastante trabalhosa gerar os binários de um sistema Linux embarcado individualmente. Porém, o entendimento do processo de geração é útil para entender as partes do sistema individualmente a assim poder customizar o que for necessário quando preciso. Apesar disso, a melhor escolha na maioria dos casos é usar um Build System para gerar todos os binários do sistema.

5. REFERÊNCIAS BIBLIOGRÁFICAS

MARWEDEL, P. **Embedded Systems Design**. Dordrecht: Springer, 2006.

TinyOS. **TinyOS home page**. Disponível em: http://www.tinyos.net/. Acessado em Outubro de 2013.

CONTIKI. **Contiki home page**. Disponível em: http://www.contiki-os.org/. Acessado em Outubro de 2013.

MICROBASE. **Sistema Operacional Virtuos**. Disponível em: http://www.microbase.com.br/produtos/produtos_virtuos.htm. Acessado em Outubro de 2013.

QNX Software Systems Limited. **QNX Operating systems**. Disponível em: http://www.qnx.com/. Acessado em Outubro de 2013.

MICROSOFT. **Windows CE 5.0**. Disponível em: http://msdn.microsoft.com/en-us/library/ms905511.aspx. Acessado em Outubro de 2013.

LINUX. **Linux Kernel Organization**. Disponível em https://www.kernel.org/. Acessado em Outubro de 2013.

CROOSTOOL-NG. **Crosstool-ng.org**. Disponivel em: http://www.crosstool-ng.org/. Acessado em Outubro de 2013.

QEMU. **Qemu**. Disponível em: http://wiki.qemu.org/Main_Page. Acessado em Outubro de 2013.

PRADO, Sérgio. **Treinamento Linux Embarcado**. Disponível em: http://dl.dropboxusercontent.com/u/1452751/Trainning/Embedded-Linux/embedded-linux-slides.pdf. Acessado em Outubro de 2013.

GPLv2. **gnu.org**. Disponivel em: http://www.gnu.org/licenses/gpl-2.0.html. Acessado em Outubro de 2013.

U-BOOT. **DENX Software Engineering**. Disponivel em: http://www.denx.de/wiki/U-Boot. Acessado em Outubro de 2013.

BUSYBOX. **Busybox**. Disponivel em: http://www.busybox.net/. Acessado em Outubro de 2013.

BUILDROOT. **Buildroot**. Disponivel em: http://buildroot.uclibc.org/. Acessado em Outubro de 2013.

YOCTO. **Yocto Project**. Disponivel em: https://www.yoctoproject.org/. Acessado em Outubro de 2013.

OPEMEMBEDDED. **Openembedded.org**. Disponivel em: http://www.openembedded.org/wiki/Main_Page. Acessado em Outubro de 2013.

FHS. **Linux Foundation** .Disponivel em: http://www.linuxfoundation.org/collaborate/workgroups/lsb/fhs. Acessado em Outubro de 2013.