Fazer ou não fazer meu próprio SO, eis a questão

Carlos Santos <unixmania@gmail.com>
Tchelinux 2019

Sobre o apresentador

- Engenheiro agrícola, acredite quem quiser (UFPel, 1988)
- Pesquisador bolsista (EMBRAPA, 1989-1993)
- Programador e gerente de redes (CPMet/UFPel, provedores de serviços Internet, 1993-1999)
- Mestre em Ciência da Computação, na área de Computação Gráfica (UFRGS, 1998-2000)
- Professor de CG, programação e redes (URI, Santo Ângelo, 2001-2005)
- Engenheiro de software (HP, 2005-2013)
- Engenheiro de manutenção de software (Red Hat, 2013-2014)
- Arquiteto de software (DATACOM, 2014-2019)
- Engenheiro de manutenção de software, de novo (Red Hat, 2019-)

Roteiro

- Vantagens e desvantagens de um SO pronto
- Vantagens e desvantagens de montar o próprio SO
- Ferramentas de automação
- Exemplo prático: geração de um SO baseado em *kernel* Linux
  - Versão para rodar no QEMU
  - Versão para rodar numa placa

Parte I – Teoria

O que é um SO *unix-like*

- Um núcleo que gerencia a máquina e a execução de processos (*kernel*)
- Um conjunto de recursos comuns usados por todos os processos (*biblioteca do sistema*)
- Um conjunto de utilitários de propósito geral ou específico (*programas de usuário*)

Vantagens de um *unix-like* pronto

- Fácil de obter uma mídia de instalação
- Funciona com grande variedade de hardware
- Documentação abundante
- Suporte (via comunidade ou comercial)
- Grande quantidade de software de terceiros
- Custo “zero” (lembrando que tempo é dinheiro)

Desvantagens dos unix-like prontos

Se visam hardware genérico (PC/servidor)

- Requerem hardware robusto (há exceções)
- Instalações ocupam muito espaço
- Requerem configurações após a instalação (automatizáveis)
- Requerem manutenção e atualização regular

Se visam hardware específico (ex. Raspberry Pi)

- Poucas opções para cada hardware
- Demora na liberação de novas versões
- Conjunto limitado de aplicações portadas
- Atendem a propósitos específicos (ex. roteador Wi-Fi).

Como contornar as desvantagens

Distribuições para hardware genérico

- Em hardware limitado, usar distribuições *leves*
- Fazer instalações parciais e/ou remover pacotes inúteis/pesados
- Usar modelos de instalação personalizados
- Automatizar configuração e atualização (dá trabalho, de início)

Distribuições para hardware específico

- Garimpar a distribuição mais adequada
- Conformar-se com versões pouco atualizadas ou limitadas
- Usar versões *alfa/beta*
- Portar/empacotar o software que falta (oportunidade de fazer fama e fortuna ☺)

Dificuldades incontornáveis

- Equipamentos de propósito específico (*switch* ethernet, telefone celular, roteador Wi-Fi, TV inteligente, …)
- Restrições severas de custo, CPU, memória, consumo de energia, etc.
- Processador/SOC não suportado
- Suporte a hardware *exótico*
- Funcionalidades *exóticas* (ex. redundância, alta disponibilidade)
- Necessidades especiais para instalação/atualização

O que fazer, então?

Hora de arregaçar as mangas e fazer do modo difícil.

Na verdade nem é mais tão difícil quanto era no passado.

Começando do zero

http://www.linuxfromscratch.org/

- Projeto iniciado por Gerard Beekmans em 1999.
- Livros que explicam como montar um sistema operacional baseado em *kernel* Linux, começando quase do zero.
- Podem ser lidos on-line ou baixados para ler localmente, em PDF e HTML.
- Não tem tradução em português, ainda (outra chance de fazer fama e fortuna).

Ferramentas de automação

Automatizam a geração de imagens de sistemas operacionais.

Cria-se *receitas* que as ferramentas usam para gerar imagens do SO.

Facilitam o trabalho em equipe, compartilhando as receitas entre os desenvolvedores.

Facilitam a manutenção. As receitas são modificadas para criar novas versões do SO.

Ferramentas de automação

- Seleção dos componentes básicos do SO
  - *Kernel* Linux
  - Biblioteca de funções (uClibc-ng, musl, GNU libc, newlib, …)
  - Sistema de inicialização (BusyBox, SystemV init, systemd, nenhum)
  - Aplicativos básicos (BusyBox, coreutils, util-linux, …)
- Geração da imagem
  - Gerar/baixar um toolchain (compilador, libc e utilitários)
  - Baixar código fonte, aplicar remendos, compilar e instalar num *diretório raiz*
  - Gerar imagem de sistema de arquivo (ubifs, ext4, ISO, …)

Cada etapa pode ser personalizada via arquivos de configuração e/ou scripts adicionais.

Ferramentas de automação

https://www.yoctoproject.org/

- Projeto da Linux Foundation (2010)
- Foco em sistemas embarcados e IoT
- Financiado por grandes corporações
- Membros e participantes contribuem financeiramente
- Gera dois produtos
  - Imagem de sistema de arquivos
  - SDK para desenvolver aplicações

Ferramentas de automação

https://buildroot.org/

- Iniciado por Eric Andersen (2001)
- Foco em sistemas embarcados e IoT
- Mantido por voluntários e apoiado por algumas empresas
- Sem financiamento corporativo
- Gera uma imagem de sistema de arquivos
- Não gera SDK
- Pode criar *toolchain* (GCC + binutils + libc) para acelerar gerações de imagens futuras

Ferramentas de automação

https://openadk.org/

- Iniciado por Waldemar Brodkorb (2009)
- Foco em sistemas embarcados e IoT
- Mantido por Waldemar Brodkorb e apoiado por algumas empresas
- Sem financiamento corporativo
- Gera uma imagem de sistema de arquivos
- Não gera SDK
- Gera sistemas Linux e não-Linux (*bare metal*, baseados em *newlib*).

Ferramentas de automação

- Simplificam o trabalho
- Uma vez pronta, a receita pode ser repetida infinitamente

- Preparar a receita requer tempo
- O SO resultante sempre mais limitado do que um genérico

Quando usar

- SO desenvolvido em equipe
- Novas versões do SO terão de ser produzidas no futuro
- Será necessário gerar mais imagens idênticas, no futuro

Parte II – Mãos à obra

Geração de um SO com Buildroot

- SO baseado em Linux
- Equipamento sem vídeo, apenas interface serial
- Servidor HTTP rodando com uma página *alô, mundo*
- Duas variantes
  - Imagem para hardware real, gravada em cartão SD/Pen Drive
  - Imagem para o QEMU, para prototipação (ver ao final)

Versão para o hardware

```
$ export BR2_DL_DIR=$HOME/src
$ cd /work
$ git clone git://git.buildroot.net/buildroot
$ git clone git://github.com/casantos/tchelinuxOS.git
$ cd tchelinuxOS
$ cp defconfig-hardware .config
$ make -C ../buildroot O=$PWD olddefconfig
$ make menuconfig
$ cat defconfig-hardware
```

```
BR2_x86_64=y
BR2_x86_atom=y
BR2_DL_DIR="$(HOME)/src"
BR2_TOOLCHAIN_EXTERNAL=y
BR2_TOOLCHAIN_EXTERNAL_DOWNLOAD=y
BR2_TOOLCHAIN_EXTERNAL_URL="https://toolchains.bootlin.com/downloads/releases/
toolchains/x86-64-core-i7/tarballs/x86-64-core-i7--glibc--bleeding-edge-
2018.11-1.tar.bz2"
BR2_TOOLCHAIN_EXTERNAL_GCC_8=y
BR2_TOOLCHAIN_EXTERNAL_HEADERS_4_14=y
BR2_TOOLCHAIN_EXTERNAL_CUSTOM_GLIBC=y
BR2_TOOLCHAIN_EXTERNAL_CXX=y
BR2_TOOLCHAIN_EXTERNAL_GDB_SERVER_COPY=y
BR2_TARGET_GENERIC_HOSTNAME="tchelinux"
BR2_TARGET_GENERIC_ISSUE="Welcome to tchelinux"
BR2_ROOTFS_DEVICE_CREATION_DYNAMIC_EUDEV=y
```

```
BR2_TARGET_GENERIC_GETTY_PORT="ttyS0"
BR2_ROOTFS_OVERLAY="$(BASE_DIR)/rootfs-overlay"
BR2_ROOTFS_POST_IMAGE_SCRIPT="$(BASE_DIR)/post-image-efi-gpt.sh"
BR2_LINUX_KERNEL=y
BR2_LINUX_KERNEL_CUSTOM_VERSION=y
BR2_LINUX_KERNEL_CUSTOM_VERSION_VALUE="4.18.10"
BR2_LINUX_KERNEL_USE_CUSTOM_CONFIG=y
BR2_LINUX_KERNEL_CUSTOM_CONFIG_FILE="$(BASE_DIR)/linux.config"
BR2_PACKAGE_THTTPD=y
BR2_TARGET_ROOTFS_EXT2=y
BR2_TARGET_ROOTFS_EXT2_4=y
BR2_TARGET_ROOTFS_EXT2_SIZE="120M"
# BR2_TARGET_ROOTFS_TAR is not set
BR2_PACKAGE_HOST_DOSFSTOOLS=y
BR2_PACKAGE_HOST_MTOOLS=y
```

Aplicar as correções no Buildroot

- Ao preparar a apresentação descobriu-se erros na inicialização do servidor HTTP (thttpd)
- Foram mandadas as correções para o projeto
- Caso elas já não tenham sido aceitas é preciso aplicá-las manualmente.
- Como dito anteriormente, encare isso como mais uma oportunidade de alcançar fama e fortuna.

As correções estão no *patchwork*

- Verificar o status no *patchwork*
  - https://patchwork.ozlabs.org/patch/1154138/
  - https://patchwork.ozlabs.org/patch/1154138/

```
$ cd buildroot
$ ls package/thttpd/
Se não houver um arquivo chamado “S90thttpd”, aplicar as correções
$ wget -O - https://patchwork.ozlabs.org/patch/1154138/mbox/ | git am
$ wget -O - https://patchwork.ozlabs.org/patch/1154139/mbox/ | git am
```

Geração do SO (finalmente!)

```
$ cd tchelinoxOS
$ make clean
$ ../buildroot/utils/brmake
```

Enquanto gera o SO

- Configuração do kernel (linux.config)
  - Suporte a UEFI
  - Suporte aos dispositivos (rede, etc)
  - Console serial
- rootfs-overlay
- Script de *pos-image* (post-image-efi-gpt.sh)
  - Criação de uma imagem de disco
  - Particionamento da imagem
  - Criação da partição 1 (EFI System)
  - Descarga da imagem do sistema de arquivos raiz para a partição 2

```
2019-08-23T06:50:06 >>>   Finalizing target directory
2019-08-23T06:50:06 >>>   Sanitizing RPATH in target tree
2019-08-23T06:50:07 >>>   Generating root filesystems common tables
2019-08-23T06:50:07 >>>   Generating filesystem image rootfs.ext2
2019-08-23T06:50:07 >>>   Executing post-image script /work/tchelinuxOS/post-image-efi-gpt.sh
[...]
2019-08-23T06:50:07 >>> Created a new GPT disklabel (GUID: 9A635C7E-F2AE-4306-9CB7-E7007D133AB0).
Done in 10min 45s
$ fdisk -l images/disk.img
Disk images/disk.img: 136 MiB, 142656512 bytes, 278626 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: gpt
Disk identifier: 9A635C7E-F2AE-4306-9CB7-E7007D133AB0

Device           Start    End Sectors  Size Type
images/disk.img1    64  32831   32768   16M EFI System
images/disk.img2 32832 278591  245760  120M Linux root (x86)
$ dd if=/work/tchelinuxOS/images/disk.img of=/dev/sdb bs=64k iflag=fullblock oflag=direct status=progress
```

Versão QEMU (omitida no evento)

```
$ export BR2_DL_DIR=$HOME/src
$ cd /work
$ git clone git://git.buildroot.net/buildroot
$ git clone git://github.com/casantos/tchelinuxOS.git
$ cd tchelinuxOS
$ cp defconfig-qemu .config
$ make -C ../buildroot O=$PWD olddefconfig
$ make menuconfig
$ cat defconfig-qemu
```

Para rodar no QEMU

Usar o script fornecido

```
$ ./run-qemu.sh --http
```

O script contém um macete para redirecionar a porta 80/TCP da máquina virtual para a porta 8080 do hospedeiro:

```
$ qemu […] -net user,hostfwd=tcp:127.0.0.1:8080-:80
```

Para acessar a máquina virtual

- No console da VM
  - Logar como *root*, sem senha (é só uma demostração)
  - Usar “ps” para exibir os processos
  - Usar “poweroff” para desligar a máquina
- Acesso ao servidor web rodando na VM
  - Acessar http://localhost:8080 na máquina hospedeira

Obrigado!