QtCon BR

Construindo uma Distribuição GNU/Linux com Suporte a Qt para Dispositivos Embarcados

Sergio Prado – Embedded Labworks
sergio.prado@e-labworks.com

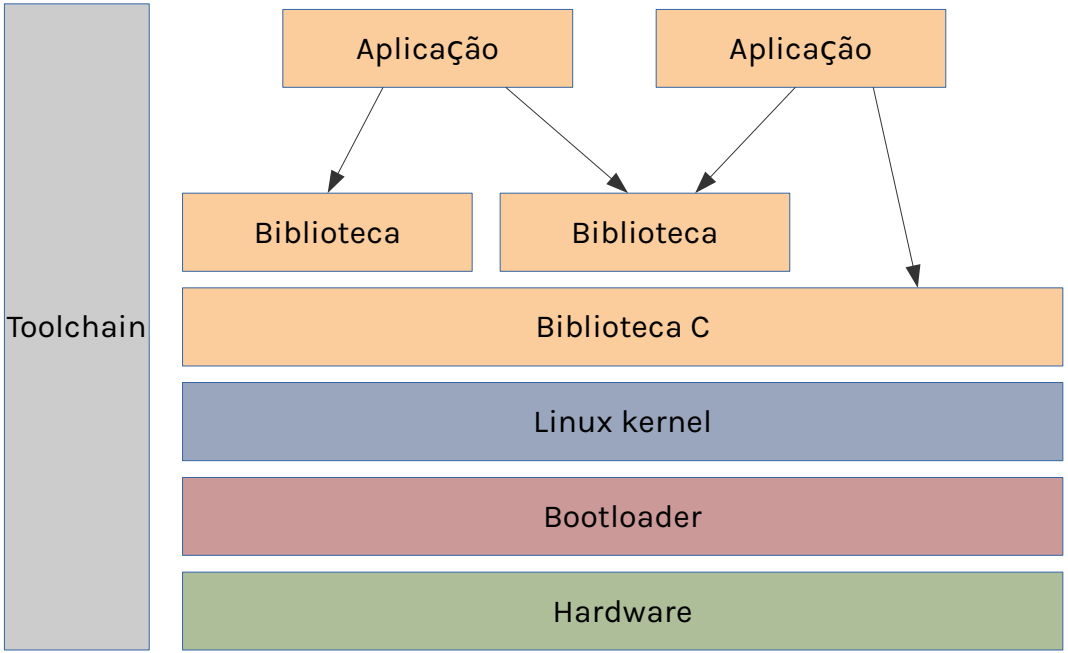
SERGIO PRADO

- Sergio Prado tem mais de 20 anos de experiência em desenvolvimento de software para sistemas embarcados.
- É sócio da **Embedded Labworks**, onde atua com consultoria, treinamento e desenvolvimento de software para sistemas embarcados.

  https:/e-labworks.com
- É ativo na comunidade de sistemas embarcados no Brasil, sendo um dos criadores do site **Embarcados**, administrador do grupo **sis_embarcados** no Google Groups, além de manter um blog pessoal sobre assuntos da área.

  http://sergioprado.org
- É colaborador de alguns projetos de software livre, incluindo o **Buildroot** e o **kernel Linux**.

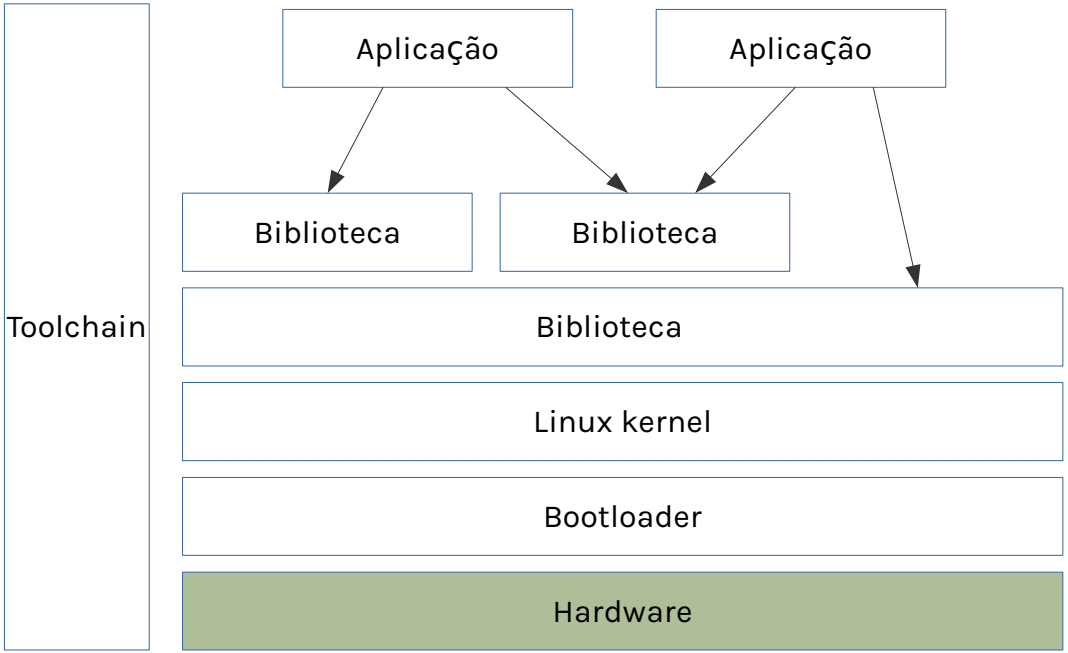
SISTEMA LINUX EMBARCADO



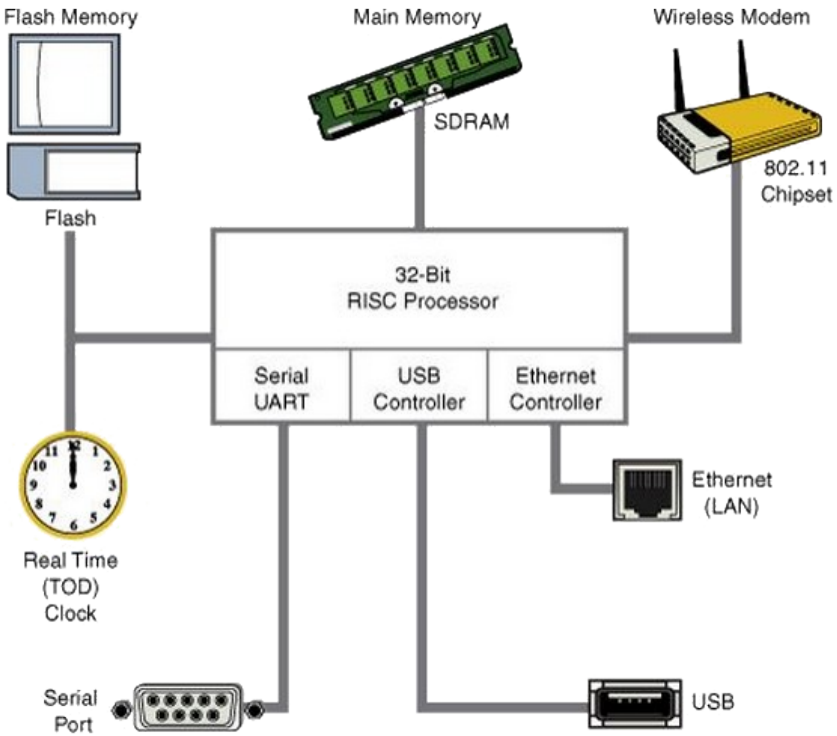
COMPONENTES DE UM SISTEMA LINUX

- **Hardware**: dispositivo de hardware ou plataforma-alvo (target).
- **Bootloader**: responsável pela inicialização básica do hardware, carga e execução do sistema operacional, no nosso caso o kernel Linux.
- **Kernel Linux**: núcleo do sistema operacional. Gerencia CPU, memória e I/O, exportando diversos serviços para a camada de usuário.
- **Rootfs**: sistema de arquivos principal, onde estão as bibliotecas e aplicações do sistema, incluindo a biblioteca C e outras bibliotecas e aplicações do usuário.
- **Toolchain**: conjunto de ferramentas para gerar os binários do sistema.

HARDWARE


Toolchain
Aplicação
Aplicação
Biblioteca
Biblioteca
Biblioteca
Linux kernel
Bootloader
Hardware

HARDWARE (cont.)



PROJETO RASPBERRY PI

- O **Raspberry Pi** é um computador do tamanho de um cartão de crédito, lançado em 2011 por alguns amigos do Laboratório de Computação da Universidade de Cambridge.

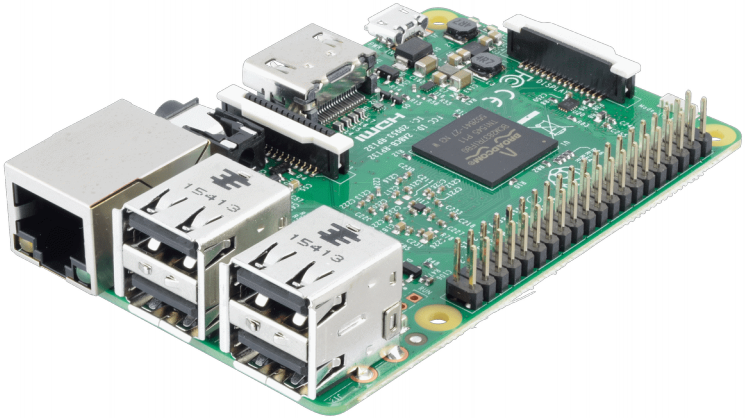
  https://www.raspberrypi.org/

- O objetivo principal do projeto é promover o ensino de programação e estudo de ciências da computação em países em desenvolvimento e escolas/universidades ao redor do mundo.

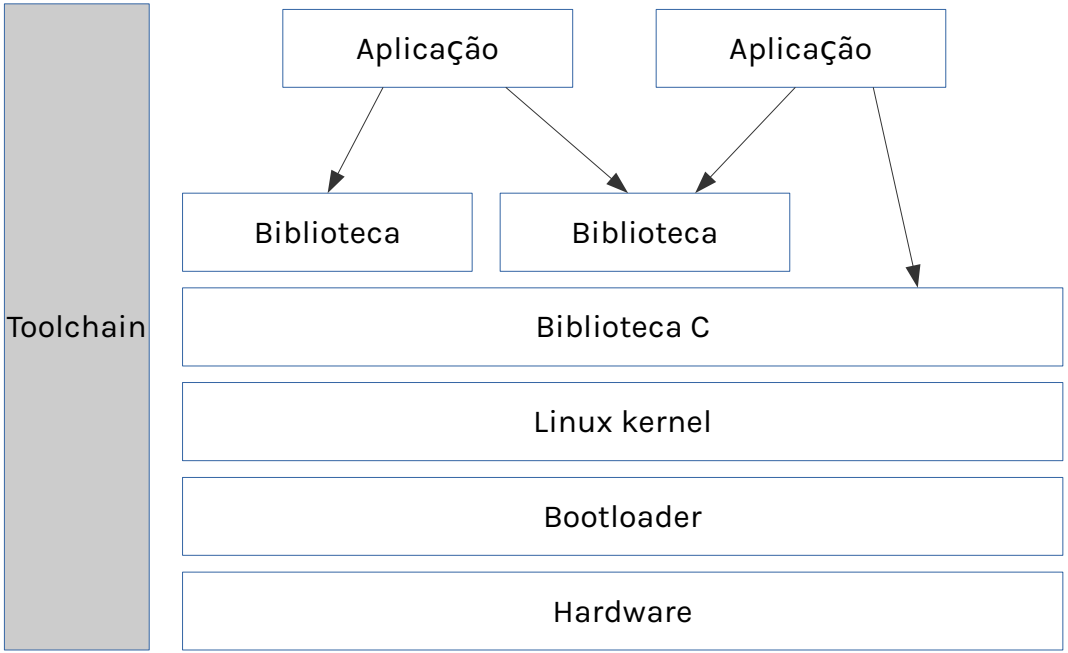
- O projeto é composto por uma série de placas de modelos diferentes, todas de baixíssimo custo (Raspberry Pi 1/2/3, Raspberry Pi Zero, Raspberry Pi Compute Module, etc).

RASPBERRY PI 3

- Baseada no SoC BCM2837 da Broadcom, com 4 processadores de 64 bits ARM Cortex-A53 rodando a 1,2GHz e GPU Videocore IV.
- Possui 1GB de memória RAM, além das interfaces Ethernet, 4 x USB, cartão SD/MMC, Wi-Fi 802.11n e Bluetooth 4.1.
- Barramento de pinos com acesso a GPIOs, UART, I2C, SPI, etc.



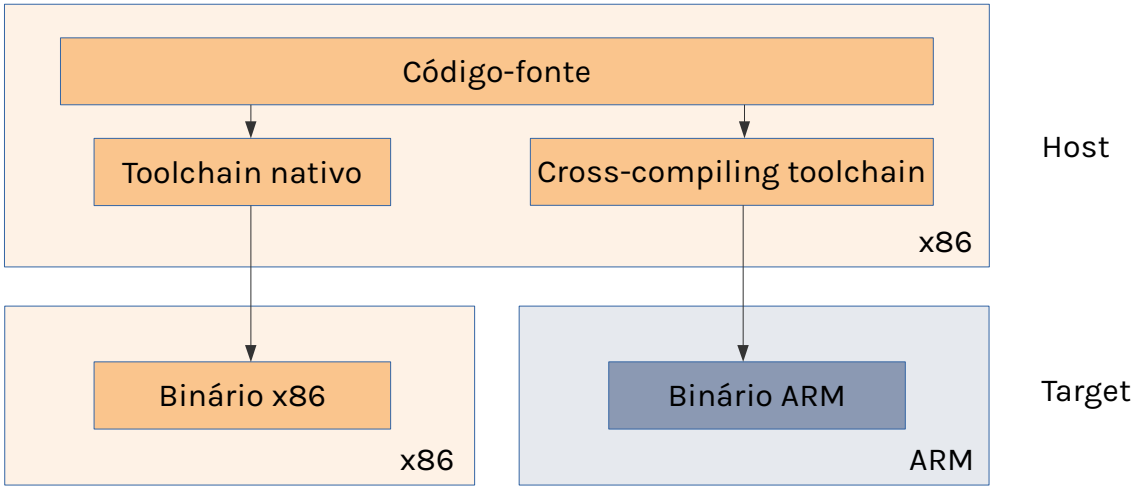
TOOLCHAIN


Aplicação
Aplicação
Biblioteca
Biblioteca
Toolchain
Biblioteca C
Linux kernel
Bootloader
Hardware

TOOLCHAIN (cont.)

- Ao pé da letra e traduzindo literalmente, toolchain é uma "corrente de ferramentas". Na prática, é um conjunto de ferramentas de compilação.
- As ferramentas de desenvolvimento normalmente disponíveis em um desktop GNU/Linux são chamadas de toolchain nativo.
- Quando a plataforma de desenvolvimento (**host**) é diferente da plataforma alvo (**target**), chamamos o toolchain de **cross-compiling toolchain** ou toolchain de compilação cruzada.

CROSS-COMPILING TOOLCHAIN



TOOLCHAIN BASEADO NO GNU

- **gcc**: compilador, com suporte a diversas linguagens como C, C++ e Fortran.

  http://gcc.gnu.org/

- **binutils**: ferramentas de manipulação de binários como o assembler e o linker.

  http://www.gnu.org/software/binutils/

- **glibc**: biblioteca C padrão do sistema.

  http://www.gnu.org/software/libc/

INSTALANDO UM TOOLCHAIN

- Existem alguns toolchains prontos disponíveis na Internet, como por exemplo o da **Linaro**.

  https://wiki.linaro.org/WorkingGroups/ToolChain

- Uma distribuição Linux pode conter toolchains em seu repositório de pacotes:

  $ sudo apt-get install gcc-arm-linux-gnueabihf

- Existem algumas ferramentas capazes de gerar toolchains, incluindo o **crosstool-ng**, **Buildroot** e **Yocto Project**.

COMPILAÇÃO CRUZADA

- Compilando uma aplicação de forma nativa:

  $ gcc teste.c -o teste

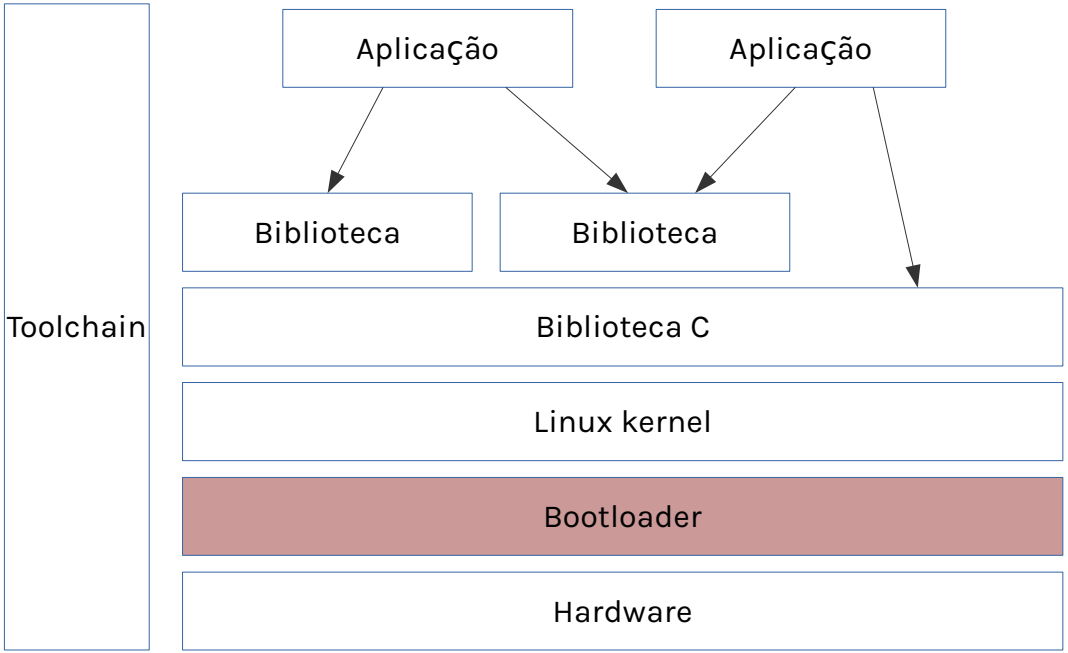
- Compilando uma aplicação de forma cruzada:

  $ PREFIX-gcc teste.c -o teste

- O prefixo depende da configuração do toolchain, e permite diferenciar toolchains nativos de toolchains para compilação cruzada.

  $ arm-linux-gnueabi-gcc teste.c -o teste

BOOTLOADER


Aplicação
Aplicação
Biblioteca
Biblioteca
Toolchain
Biblioteca C
Linux kernel
Bootloader
Hardware

BOOTLOADER (cont.)

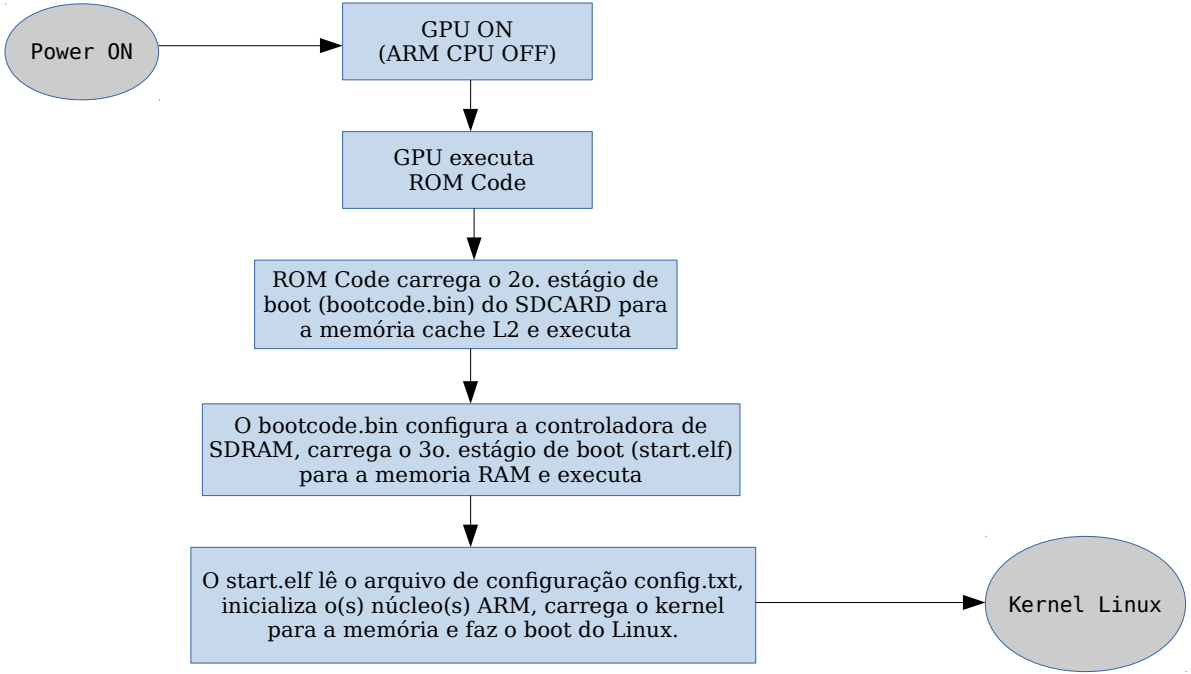
- O bootloader tem basicamente duas responsabilidades:
  - Inicializar o hardware.
  - Carregar e executar o sistema operacional.
- Normalmente o bootloader provê outras funcionalidades para facilitar o desenvolvimento do sistema, como possibilitar a gravação na memória flash ou fazer o boot pela rede.

Qt

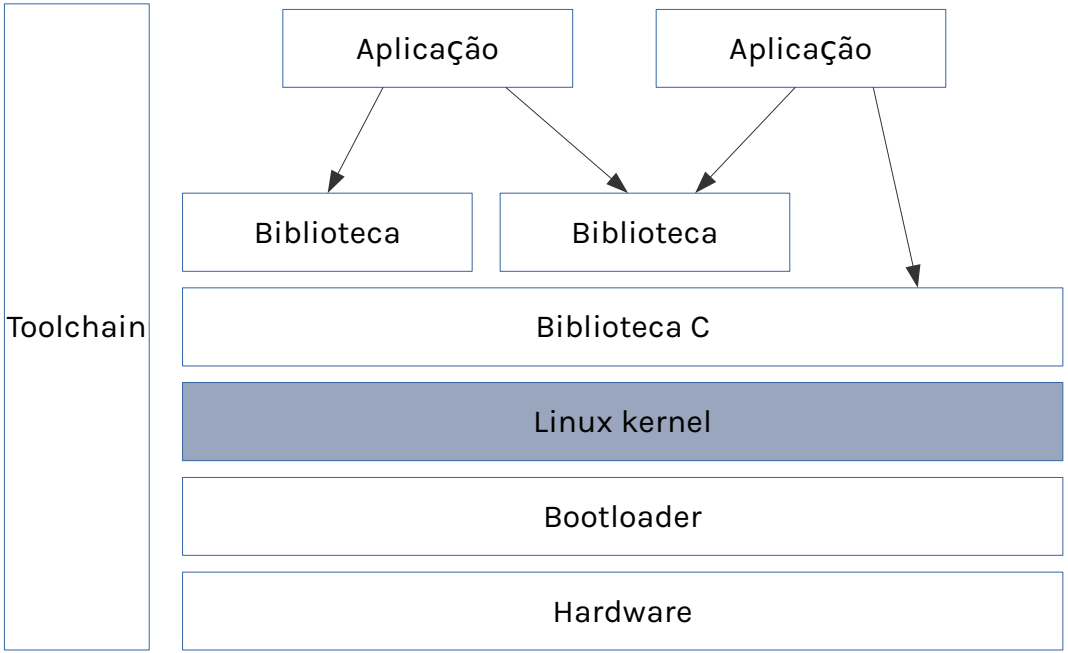
PRINCIPAIS BOOTLOADERS

- x86:
  - LILO
  - Grub
  - Syslinux
- ARM, MIPS, PPC e outras arquiteturas:
  - U-Boot
  - Barebox
  - Redboot

BOOT NA RASPBERRY PI 3



KERNEL LINUX



KERNEL LINUX (cont.)

- O Linux é um kernel!

  https://kernel.org

- As distribuições GNU/Linux (Ubuntu, Fedora, Debian, Slackware, etc) integram o kernel Linux, bibliotecas e aplicações.

- Criado em 1991 pelo estudante finlandês **Linus Torvalds**, começou a ser usado rapidamente como sistema operacional em projetos de software livre.

- Linus foi capaz de criar uma comunidade grande e dinâmica de desenvolvedores e usuários ao redor do projeto. Atualmente, centenas de pessoas e empresas contribuem com o projeto.

KERNEL LINUX (cont.)

- O kernel Linux abstrai o uso das CPUs do sistema, de forma que cada processo acredite que ele tem a CPU só para ele.
- O kernel Linux abstrai o uso da memória com a ajuda da MMU, de forma que cada processo acredite que ele tem a memória só para ele.
- O kernel Linux abstrai o acesso a dispositivos de I/O, utilizando arquivos como principal mecanismo de abstraÇão.

  $ echo “hello” > /dev/ttyS0

BAIXANDO O CÓDIGO-FONTE

- O código-fonte do kernel Linux pode ser baixado no site oficial do projeto ou no site do fabricante do hardware.
- O kernel oficial da Raspberry Pi é disponibilizado no GitHub do projeto, e pode ser baixado conforme abaixo:

```
$ wget https://github.com/raspberrypi/linux/archive/raspberrypi-kernel_1.20170703-2.tar.gz
$ tar xfv raspberrypi-kernel_1.20170703-2.tar.gz
$ cd linux-raspberrypi-kernel_1.20170703-2/
```

CONFIGURANDO E COMPILANDO O KERNEL

- Configurando o kernel:

  $ make ARCH=arm bcm2709_defconfig

- Compilando o kernel:

  $ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- zImage -j8

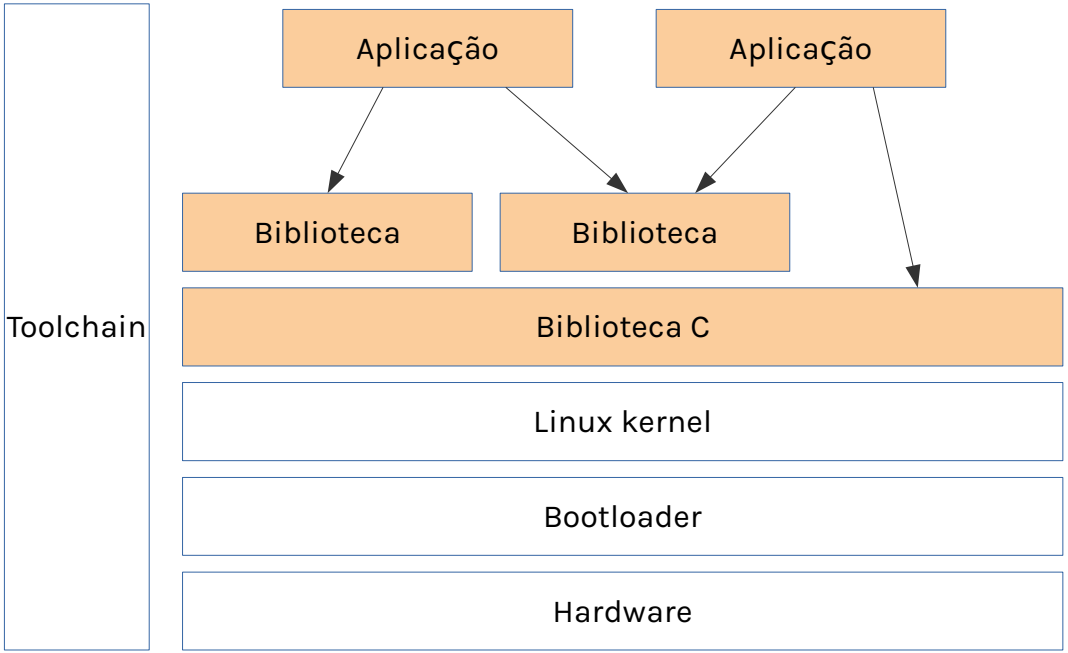
- Compilando o device tree (arquivo de descrição do hardware):

  $ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- bcm2710-rpi-3-b.dtb

- Compilando os módulos

  $ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- modules -j8

ROOTFS


Aplicação
Aplicação
Biblioteca
Biblioteca
Toolchain
Biblioteca C
Linux kernel
Bootloader
Hardware

COMPONENTES BÁSICOS

- Um sistema GNU/Linux precisa de um conjunto básico de programas para funcionar, incluindo:
  - Uma biblioteca do sistema (glibc, uClibc-ng, musl, etc).
  - Um mecanismo de inicialização (systemd, sysvinit, upstart, etc).
  - Diversas bibliotecas e aplicações (bash, cat, echo, grep, sed, useradd, vi, getty, libusb, etc).
- Normalmente estes programas são fornecidos em diferentes projetos e é trabalhoso configurar, compilar e integrar manualmente todos eles.

BUSYBOX

- O **Busybox** é uma solução alternativa, trazendo uma quantidade grande e comum de programas usados em sistemas Linux, mas com tamanho reduzido, perfeito para sistemas embarcados!

  http://www.busybox.net/

- O Busybox contém diversos componentes, incluindo um sistema de inicialização baseado no sysvinit, um terminal de comandos, além de ferramentas e utilitários diversos (cat, echo, ps, vi, etc).

- Geralmente, as ferramentas são mais limitadas em termos de funcionalidades quando comparadas às originais.

BUSYBOX – TUDO ISSO EM ~1MB!

addgroup, **adduser**, adjtimex, ar, **arp**, arping, ash, awk, basename, bbconfig, bbsh, brctl, bunzip2, busybox, bzcat, **bzip2**, cal, cat, catv, chat, chattr, chcon, chgrp, chmod, chown, chpasswd, chpst, chroot, chrt, chvt, cksum, clear, cmp, comm, cp, **cpio**, crond, crontab, cryptpw, cttyhack, cut, date, dc, dd, deallocvt, delgroup, deluser, depmod, devfsd, df, dhcprelay, **diff**, dirname, dmesg, dnsd, dos2unix, dpkg, dpkg_deb, du, dumpkmap, dumpleases, **e2fsck**, echo, ed, egrep, eject, env, envdir, envuidgid, ether_wake, expand, expr, fakeidentd, false, fbset, fbsplash, fdflush, fdformat, fdisk, fetchmail, fgrep, **find**, findfs, fold, free, freeramdisk, fsck, fsck_minix, ftpget, ftpput, fuser, getenforce, getopt, getsebool, **getty**, grep, gunzip, gzip, halt, hd, hdparm, head, hexdump, hostid, hostname, **httpd**, hush, hwclock, id, ifconfig, ifdown, ifenslave, ifup, **inetd**, init, inotifyd, insmod, install, **ip**, ipaddr, ipcalc, ipcrm, ipcs, iplink, iproute, iprule, iptunnel, kbd_mode, **kill**, killall, killall5, klogd, lash, last, length, less, linux32, linux64, linuxrc, ln, load_policy, loadfont, loadkmap, logger, login, logname, logread, losetup, lpd, lpq, lpr, ls, lsattr, **lsmod**, lzmacat, makedevs, man, matchpathcon, md5sum, mdev, mesg, microcom, mkdir, mke2fs, mkfifo, mkfs_minix, mknod, mkswap, mktemp, **modprobe**, more, mount, mountpoint, msh, mt, mv, nameif, nc, **netstat**, nice, nmeter, nohup, nslookup, od, openvt, parse, **passwd**, patch, pgrep, pidof, ping, ping6, pipe_progress, pivot_root, pkill, poweroff, printenv, printf, **ps**, pscan, pwd, raidautorun, rdate, rdev, readahead, readlink, readprofile, realpath, reboot, renice, reset, resize, restorecon, rm, rmdir, rmmod, route, rpm, rpm2cpio, rtcwake, run_parts, runcon, runlevel, runsv, runsvdir, rx, script, sed, selinuxenabled, **sendmail**, seq, sestatus, setarch, setconsole, setenforce, setfiles, setfont, setkeycodes, setlogcons, setsebool, setsid, setuidgid, sh, sha1sum, showkey, slattach, sleep, softlimit, sort, split, start_stop_daemon, stat, strings, stty, su, sulogin, sum, sv, svlogd, swapoff, swapon, switch_root, sync, sysctl, syslogd, tac, tail, tar, taskset, tcpsvd, tee, telnet, **telnetd**, test, tftp, **tftpd**, time, top, touch, tr, traceroute, true, tty, ttysize, tune2fs, udhcpc, **udhcpd**, udpsvd, umount, uname, uncompress, unexpand, uniq, unix2dos, unlzma, unzip, uptime, usleep, uudecode, uuencode, vconfig, **vi**, vlock, watch, watchdog, wc, wget, which, who, whoami, xargs, yes, zcat, zcip

CRIANDO O ROOTFS

- No rootfs do sistema GNU/Linux que criaremos nesta apresentação, incluiremos a glibc, as ferramentas básicas providas pelo Busybox, o Qt e as bibliotecas de acesso à GPU da Raspberry Pi.
- Apesar de ser possível, criar um rootfs manualmente é trabalhoso pois envolve configurar e compilar cada componente individualmente.
- Para automatizar a criação do rootfs, podemos utilizar um **build system** (sistema de build).

BUILD SYSTEM

- Um **build system** é capaz de gerar todo o sistema, incluindo o toolchain, bootloader, kernel Linux e rootfs.
- Atualmente, os dois principais projetos de sistemas de build são o **Buildroot** e o **Yocto Project**.
- Nesta apresentação, utilizaremos o Buildroot para construir a distribuição GNU/Linux para a Raspberry Pi 3 com suporte ao Qt.

BUILDROOT

- Sistema de build simples, flexível e prático!

  https://buildroot.org/

- Possibilita gerar o toolchain, o bootloader, o kernel e o rootfs com muitas bibliotecas e aplicações disponíveis.

- Mais de 2.000 aplicações e bibliotecas integradas, de utilitários básicos a bibliotecas mais elaboradas como X.org, Qt, Gtk, Webkit, Gstreamer, etc.

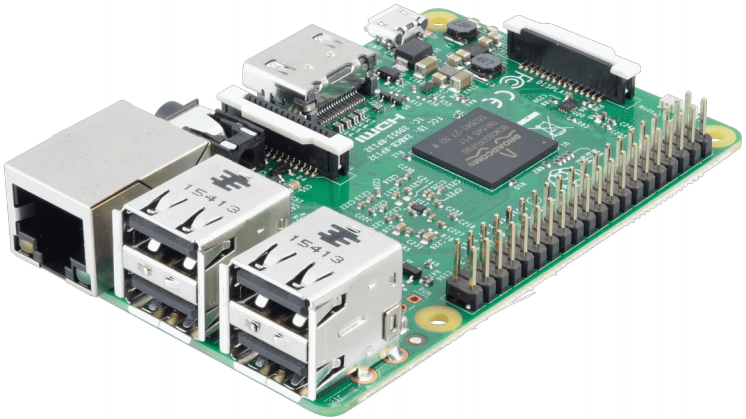
- Desde a versão 2009.02 um novo release é liberado a cada 3 meses.

BUILDROOT (cont.)

- Baixar e usar o Buildroot é muito simples:

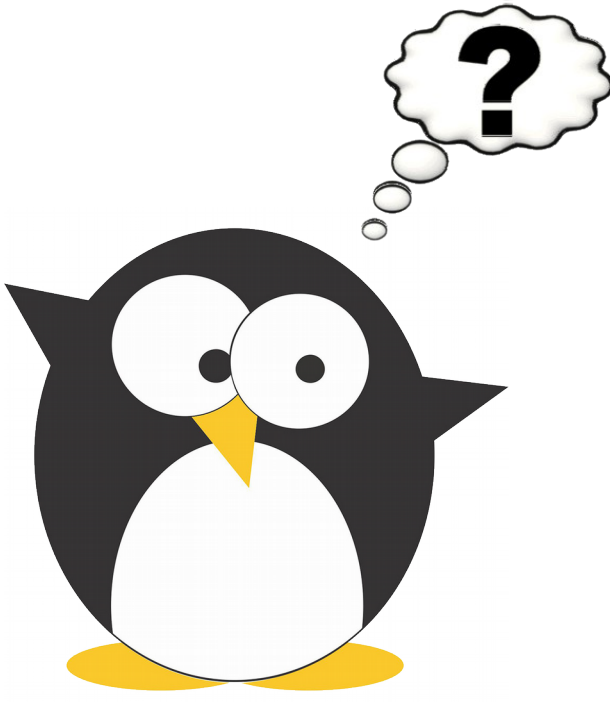
```
$ wget https://buildroot.org/downloads/buildroot-2017.02.5.tar.gz
$ tar xfv buildroot-2017.02.5.tar.gz && cd buildroot-2017.02.5
$ make menuconfig
$ make
$ ls output/images
rootfs.tar      u-boot.bin      zImage
```

HANDS-ON



Utilizando o Buildroot para construir uma distribuição GNU/Linux com suporte a Qt para a Raspberry Pi 3

DÚVIDAS?



Qt

QtCon BR

OBRIGADO!

Qt

Twitter: @sergioprado
E-mail: sergio.prado@e-labworks.com