

Linux Embarcado

Revisão: 06/09/2024

B2Open Systems

O material está sendo disponibilizado de acordo com CC BY-SA 4.0

- **Atribuição** — Você deve dar o crédito apropriado, prover um link para a licença e indicar se mudanças foram feitas. Você deve fazê-lo em qualquer circunstância razoável, mas de nenhuma maneira que sugira que o licenciante apoia você ou o seu uso.
- **Compartilhalgual** — Se você remixar, transformar, ou criar a partir do material, tem de distribuir as suas contribuições sob a mesma licença que o original.
- **Sem restrições adicionais** — Você não pode aplicar termos jurídicos ou medidas de caráter tecnológico que restrinjam legalmente outros de fazerem algo que a licença permita.

https://creativecommons.org/licenses/by-sa/4.0/deed.pt_BR



Informações Gerais

- ❖ Proprietário da **B2Open Systems**
- ❖ Usuário Linux desde 2006 e trabalhando com Linux Embarcado desde 2013
- ❖ Participamos de projetos com Linux Embarcado em IoT, Indústria 4.0, Equipamento Médico, Agro, Telecom, Radar entre outros
- ❖ Atualmente trabalhamos muito com Yocto Project e aplicações em C, C++ e Python



Agenda

- ❖ **Dia 1** - Introdução e estudo da anatomia de um Sistema Embarcado com Linux e seus componentes Toolchain, Bootloader e Kernel
- ❖ **Dia 2** - Linux FHS, RootFS com Busybox, Gerenciador de Serviços e Ferramenta de construção de Sistemas Linux
- ❖ **Dia 3** - Aplicações Demo, Yocto Project, Ferramentas

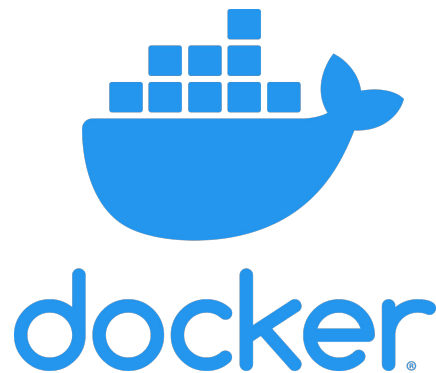


Material do Treinamento

- vi.pdf** - Ajuda sobre uso do comando vi/vim
- shell.pdf** - Principais comandos utilizados em Shell Script
- Requisitos.pdf** - Requisitos do Sistema para realizar ao treinamento
- SlidesLE.pdf** - Slides do treinamento
- LaboratorioLE.pdf** - Todas as práticas e exemplos do Treinamento
- Dockerfile** - Arquivo docker para preparar o Container do treinamento



- ❖ Container Docker
- ❖ Todas práticas e exercícios serão dentro do Container disponibilizado via Dockerfile para preparação do Treinamento
- ❖ Uma das principais vantagens possui o mesmo ambiente/versões/configurações em diferentes Distribuições Linux
- ❖ Fácil de compartilhar com demais colaboradores



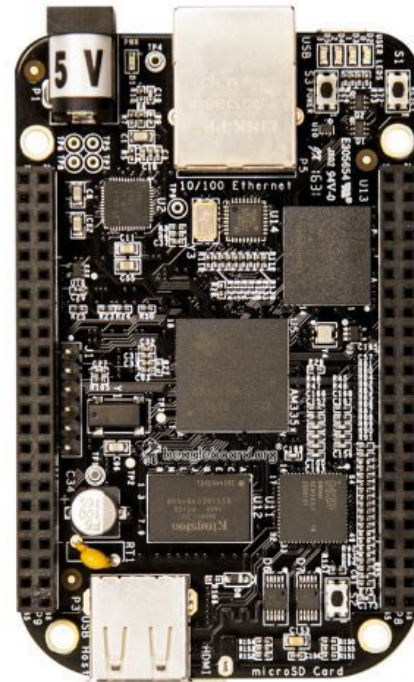
Participe!

- ❖ Não hesite em perguntar
- ❖ Sua dúvida pode ser uma questão generalizada
- ❖ Ajuda o instrutor a formatar melhor a explicação ou dar mais ênfase no tópico
- ❖ Durante os exercícios de prática qualquer erro ao reproduzir comunique o Instrutor



Hardware Utilizado - BeagleBone Black

- ▶ Processador TI Sitara XAM3359AZCZ100, 1GHZ
- ▶ 2x PRU 32-bit microcontrollers
- ▶ Memória RAM 512MB DDR3L
- ▶ 4GB 8-bit eMMC OnBoard
- ▶ Controlador Ethernet 10/100
- ▶ 46 pinos GPIO
- ▶ HDMI
- ▶ 4x USB 2.0
- ▶ USB Host
- ▶ Micro SD



Mais informações: [Beaglebone Black](#)



Hardware Utilizado - Toradex Colibri iMX6

- ▶ NXP/Freescale i.MX6S – Solo Core, 256MB RAM e 4GB eMMC
- ▶ NXP/Freescale i.MX6DL – Dual Core, 512MB RAM e 4GB eMMC
- ▶ ARM Cortex-A9 (800MHz ~ 1GHz)
- ▶ 5x UART's, 4x SPI, 3x I2C, 2x CAN, 4x PWM
- ▶ >150 GPIO's
- ▶ GPU Vivante GC880
- ▶ Video Decode (MJPEG, MPEG-4, H.264, H.263, DivX, VC1, MPEG-2)
- ▶ Video Encode (MJPEG, MPEG-4, H.264, H.263)
- ▶ Micro SD



Mais informações: [Toradex Colibri iMX6](#)



Hardware Utilizado - Toradex Colibri iMX8X

- ▶ NXP/Freescale i.MX8QX – QuadCore, 2GB RAM e 8GB eMMC
- ▶ NXP/Freescale i.MX8DX – DualCore, 1GB RAM e 4GB eMMC
- ▶ ARM Cortex-A35 (1.2GHz)
- ▶ 5x UART's, 3x SPI, 8x I2C, 3x CAN, 10x PWM
- ▶ >90 GPIO's
- ▶ GPU Vivante GC7000 Lite
- ▶ Video Decode (4K H.265 decoder*, 1080p H.264 decoder)
- ▶ Video Encode (1080p30 H.264 encoder)
- ▶ Micro SD
- ▶ Wireless Dual-Band 802.11ac / Bluetooth 5

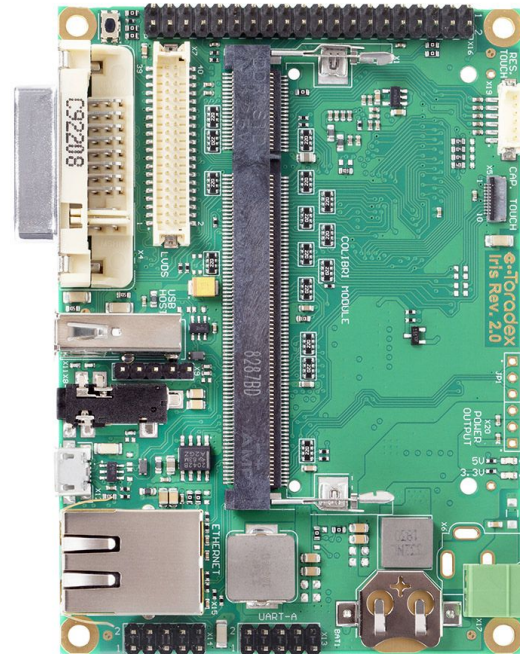


Mais informações: [Toradex Colibri iMX8](#)



Hardware Utilizado - Toradex BaseBoard IRIS

- ▶ Alimentação 6-27V DC
- ▶ 1x USB Host
- ▶ 1x USB OTG
- ▶ 3x UART's RS232
- ▶ 4x PWM
- ▶ 1x Ethernet
- ▶ 1x LVDS, 1x HDMI (Conector DVI), 1x VGA (Conector DVI)
- ▶ 1x RTC na placa
- ▶ 1x uSD
- ▶ >25 GPIO's

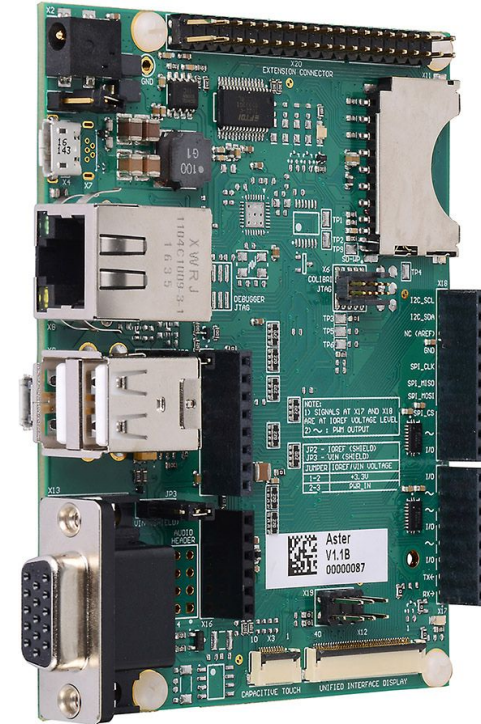


Mais informações: [Toradex IRIS Carrier Board](#)



Hardware Utilizado - Toradex BaseBoard ASTER

- ▶ Alimentação 5V DC
- ▶ 2x USB Host
- ▶ 1x USB Client
- ▶ 2x UART TTL
- ▶ 1x USB UART
- ▶ 4x PWM
- ▶ 1x Ethernet
- ▶ 1x RGB, 1x VGA (Conector DVI)
- ▶ 1x RTC na placa
- ▶ 1x SD
- ▶ >39 GPIO's



Mais informações: [Toradex ASTER Carrier Board](#)



Hardware - MYIR MYC-Y6ULX

- ▶ Processador NXP iMX 6UL/6ULL ARM Cortex-A7
- ▶ Memória RAM 256MB DDR3 SDRAM
- ▶ Memória Armazenamento 256MB Nand Flash
- ▶ Conectividade de rede: FAst Ethernet 10/100
- ▶ 8x UART's
- ▶ 2x CAN
- ▶ 1x LCD 24-bit Parallel
- ▶ 2x ADC
- ▶ 2x SDIO
- ▶ USB 2.0 HS/FS / USB OTG



Mais informações: [MYC-Y6ULX CPU Module](#)



Hardware - MYD-Y6ULX-CHMI

Alimentação 12V-24V DC/1.5A

1x USB Host

1x USB OTG

1x UART's RS232

1x Serial RS475

1x Serial Debug

1x Ethernet

Display 7" TFT-LCD (800 x 480)

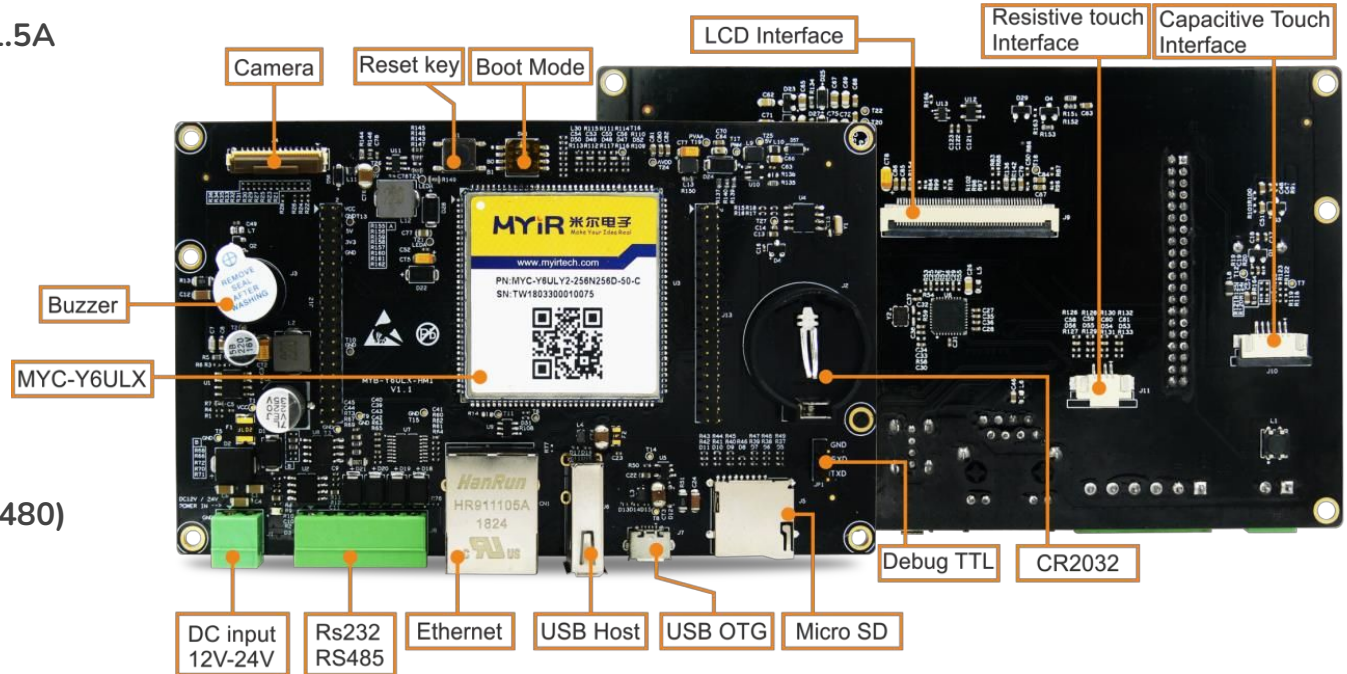
TouchScreen Capacitivo

1x RTC na placa

1x MicroSD

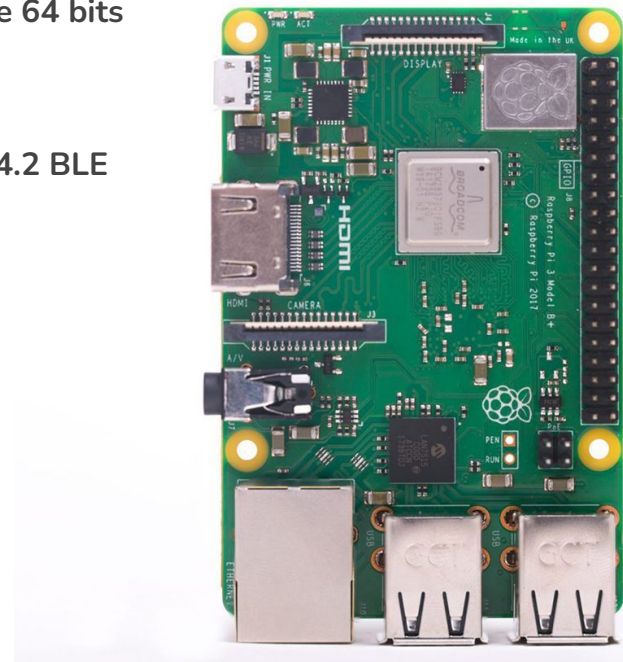
1x Buzzer

>97 GPIO's



Hardware Utilizado - RaspberryPI3

- ▶ Processador Broadcom BCM2837B0, Cortex-A53 (ARMv8) SoC de 64 bits
- ▶ Memória RAM 1GB LPDDR2 SDRAM
- ▶ Conectividade Sem Fio 2.4GHz IEEE 802.11 b/g/n/ac e Bluetooth 4.2 BLE
- ▶ Conectividade de rede: Gigabit Ethernet via USB 2.0
- ▶ Portas: GPIO 40 pinos
- ▶ HDMI
- ▶ 4x USB 2.0
- ▶ CSI (câmera Raspberry Pi)
- ▶ DSI (Display)
- ▶ Micro SD



Mais informações: [RaspberryPI Documentation](https://www.raspberrypi.org/documentation/)



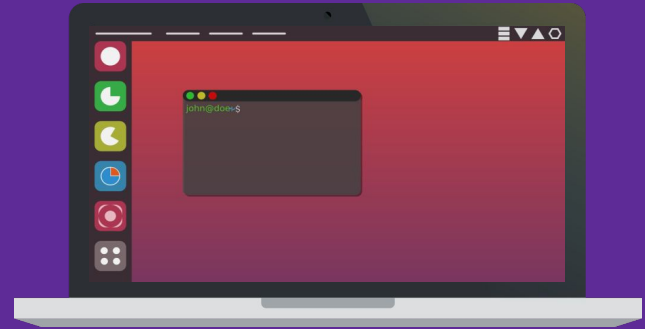
Itens adicionais

▶ Conversor USB-Serial TTL 3v3

▶ MicroSD >8GB



Ambiente de Desenvolvimento





Ambiente de Desenvolvimento

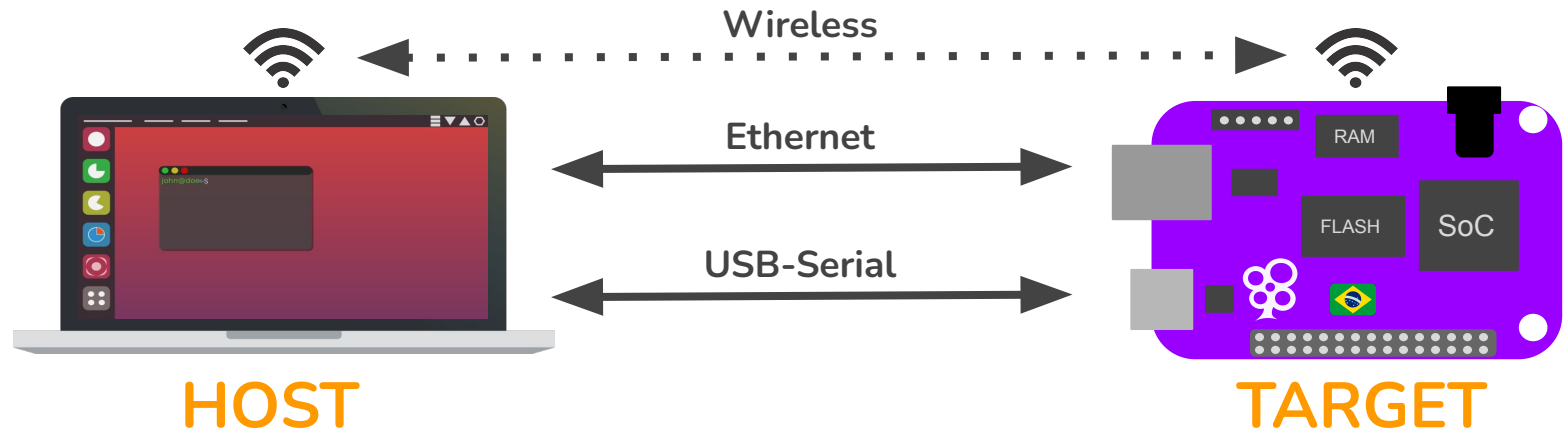
- ❖ Use a Distribuição Linux de sua preferência: Debian, Ubuntu, Mint, Fedora, CentOS, Arch, ...
- ❖ Container Docker ao invés de Máquina Virtual, fácil de reproduzir, compartilhar e com menos recursos
- ❖ Todas ferramentas utilizadas são open-source
- ❖ Ambiente 100% Linux



Ambiente de Desenvolvimento

Host - Computador de Desenvolvimento

Target - Placa Alvo, Kit de Desenvolvimento



Durante o treinamento tudo é feito no terminal, algumas orientações de uso a seguir:

- Terminal como usuário comum (restrições de comandos)



- Terminal como super-usuário (sem restrições de comandos/permissions)

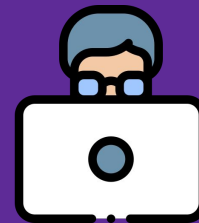


- Executando comandos root como usuário comum com **sudo**



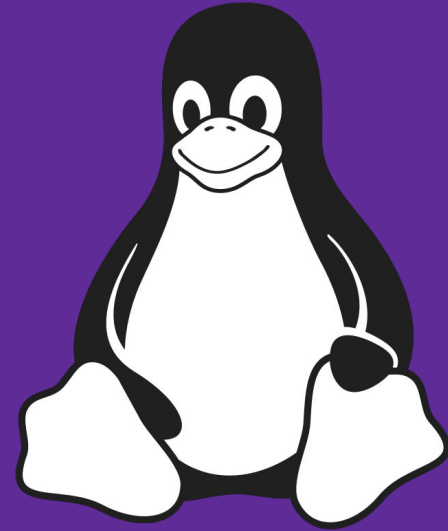
Laboratório

- ❖ Prática Terminal e Comandos
- ❖ Preparando o ambiente do Treinamento





Introdução ao Linux Embarcado



Do nascimento do Software Livre ao Linux

- ▶ **1970:** Engenheiros da Bell Labs, liderados por Ken Thompson e Dennis Ritchie, criam o sistema operacional UNIX.
- ▶ **1983:** Richard Stallman, projeto GNU e o conceito de software livre. Começa o desenvolvimento do gcc, gdb, glibc e outras ferramentas importantes.
- ▶ **1991:** Linus Torvalds, projeto do kernel Linux, um sistema operacional UNIX-like. Em conjunto com o projeto GNU, nasce o sistema operacional GNU/Linux.
- ▶ **2000:** Linux se torna mais popular em Sistemas Embarcados
- ▶ **2008:** Linux se torna popular em Dispositivos Móveis



Software Livre?

- ❖ Um programa é considerado livre quando sua licença oferece a todos os seus usuários as seguintes quatro liberdades:
 - Liberdade para executar o software para qualquer propósito
 - Liberdade para estudar o software e alterá-lo
 - Liberdade para redistribuir cópias
 - Liberdade para distribuir cópias de versões modificadas
- ❖ Essas liberdades são concedidas para uso comercial e não comercial
- ❖ Implicam a disponibilidade do código-fonte, o software pode ser modificado e distribuído aos clientes



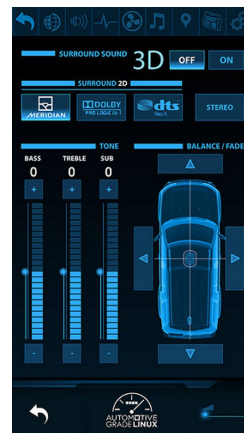
**Onde Linux
Embarcado está
presente?**







ROS



Linux Embarcado

O que é Linux Embarcado?

Falar de Linux Embarcado é o mesmo de uma Distribuição Linux utilizada em Desktop ou Servidores, é utilizado o mesmo **Kernel Linux**, combinando com diversos softwares open-source e adequações para um Sistema Embarcado, na maioria das vezes com propósito bem definido!



Vantagens de utilizar Linux e OpenSource em Sistemas Embarcados?

- Reuso de Componentes
- Baixo Custo
- Controle Total
- Qualidade
- Comunidade



Linux é o Kernel!

- ❖ Sobre o Kernel: <https://www.kernel.org/>
- ❖ Código aberto e livre de royalties
- ❖ Portável: suporte para 23 diferentes arquiteturas (versão [5.15](#))
- ❖ Escalável: o mesmo kernel roda desde Servidores até SmartWatches
- ❖ Estável: capaz de rodar por muito tempo sem precisar de um único reboot.



Linux é o Kernel!



Linux Kernel



Distribuição Cronos



Distribuição ARGO



Reuso de Componentes

- ❖ Uma das principais vantagens do Kernel Linux e softwares open-source em sistemas embarcados é a capacidade de reutilizar componentes
- ❖ A comunidade já fornece uma grande gama de componentes e recursos de suporte a hardwares, protocolos de rede, bibliotecas multimídias e etc.
- ❖ Assim que um dispositivo de hardware, ou protocolo, ou um recurso é amplamente difundido, grande chance de ter componentes de código aberto que o suportem.
- ❖ Desenvolvimento mais rápido utilizando componentes prontos
- ❖ Foco no produto/software da empresa (PoC/MVP)



- ❖ O software livre pode ser duplicado em quantos dispositivos você quiser, gratuitamente.
- ❖ Se o seu sistema embarcado usa apenas software livre, você pode reduzir o custo das licenças de software a zero. Até mesmo as ferramentas de desenvolvimento são gratuitas.
- ❖ Claro, usar o Linux não é gratuito. Você ainda precisa de tempo e esforço para estudar e aprender.
- ❖ Permite um investimento maior em hardware ou treinamento para o time especializando e melhorando as habilidades em Linux.



Controle Total

- ❖ Com o código aberto, você tem o código-fonte para todos os componentes do seu sistema.
- ❖ Permite modificações, alterações, ajustes, depuração e otimização.
- ❖ Sem bloqueio de projeto por dependência de um fornecedor terceirizado ou outras prioridades.
- ❖ Permite ter controle total sobre a parte do software do seu sistema, está tudo em suas mãos!



- ❖ Muitos componentes de código aberto são amplamente usados em milhões de sistemas.
- ❖ Normalmente qualidade superior à que um desenvolvimento interno pode produzir, ou mesmo fornecedores proprietários.
- ❖ Claro, nem todos os componentes de código aberto são de boa qualidade, tome cuidado!
- ❖ Permite desenvolver seu sistema com componentes de alta qualidade em suas bases.



- ❖ Os componentes de software livre são desenvolvidos por comunidades de desenvolvedores e usuários
- ❖ Esta comunidade pode fornecer suporte de alta qualidade: você pode contatar diretamente os principais desenvolvedores do componente que está usando.
- ❖ Muitas vezes melhor do que o suporte tradicional, mas é preciso entender como a comunidade funciona para usar adequadamente as possibilidades de suporte da comunidade.
- ❖ Permite agilizar a resolução de problemas do seu projeto!



Seleccionando Hardware para Sistemas Linux

É como selecionar um Microcontrolador

- ❖ Processador
- ❖ Memória RAM
- ❖ Armazenamento
- ❖ Periféricos
- ❖ Comunicação



- ❖ O kernel do Linux e a maioria dos outros componentes dependentes de arquitetura suportam uma ampla gama de arquiteturas de 32 e 64 bits - (x86/x86-64, ARM, RISC-V, PowerPC, MIPS, ...)
- x86 e x86-64 - Popular em Desktops e PC's Industriais
- ARM - Popular na maioria das SBC's e SoC (System on Chips)



- ❖ Ambas as arquiteturas MMU e sem MMU são suportadas, embora as arquiteturas sem MMU tenham algumas limitações.
- ❖ Linux não suporta microcontroladores pequenos (8 ou 16 bits), talvez [Zephyr](#) seja uma alternativa.
- ❖ Além do toolchain, o bootloader e o kernel, todos os outros componentes são geralmente independentes de arquitetura



Memória RAM

- ❖ Um sistema Linux muito básico pode trabalhar com 8MB de RAM, mas um sistema mais realista de um projeto atual pode requerer mais recursos.
- ❖ Influências para mais recursos: Aplicações Gráficas, Servidor Gráfico, Aplicações Multimídia, Processamento de Imagens, IA, ML, ...
- ❖ Será visto na prática o baixo consumo de RAM!



Armazenamento

- ❖ Um sistema Linux básico pode funcionar com 4MB de armazenamento, o Kernel Linux por si só ocupa algumas centenas de KBs
- ❖ Normalmente o RootFS(Bibliotecas e Aplicações) irá demandar mais espaço de armazenamento
- ❖ Suporte **Armazenamento em Bloco**(SD/MMC/eMMC/uSD) e **Armazenamento Flash RAW**(NAND e NOR)



❖ O kernel Linux tem suporte para muitos barramentos de comunicação comuns

▶ I2C

▶ SPI

▶ 1-wire

▶ SDIO

▶ PCI

▶ USB

▶ CAN



- ❖ Amplo suporte de rede
 - ▶ Ethernet, Wifi, Bluetooth, CAN, etc.
 - ▶ IPv4, IPv6, TCP, UDP, SCTP, DCCP, etc.



- ❖ Há diversas opções de Hardware no mercado, como:
 - ▶ **SBC** - Single Board Computer, normalmente placas populares, menor custo, tudo em uma única placa, limitado a customização
 - ▶ **EV-Boards** - Placa de desenvolvimento, completa de acesso a periféricos, display, custo maior
 - ▶ **CoM**(Computer On Module) / **SoM**(System On Module) - Hardware básico normalmente CPU, RAM e Flash, com placas customizáveis e personalizadas até por segmento ou área.



Critérios para seleção

- ❖ Certifique-se de que o Hardware que deseja utilizar tenha suporte ao Kernel e Bootloader Open-Source.
- ❖ Ter suporte nas versões oficiais dos projetos (kernel, bootloader) é muito melhor: a qualidade é melhor, novas versões estão disponíveis e versões de suporte de longo prazo estão disponíveis.
- ❖ Entre um hardware com suporte adequado no kernel oficial do Linux e hardware com suporte incompleto/desatualizado, poderá comprometer muito tempo de desenvolvimento e custo do projeto!



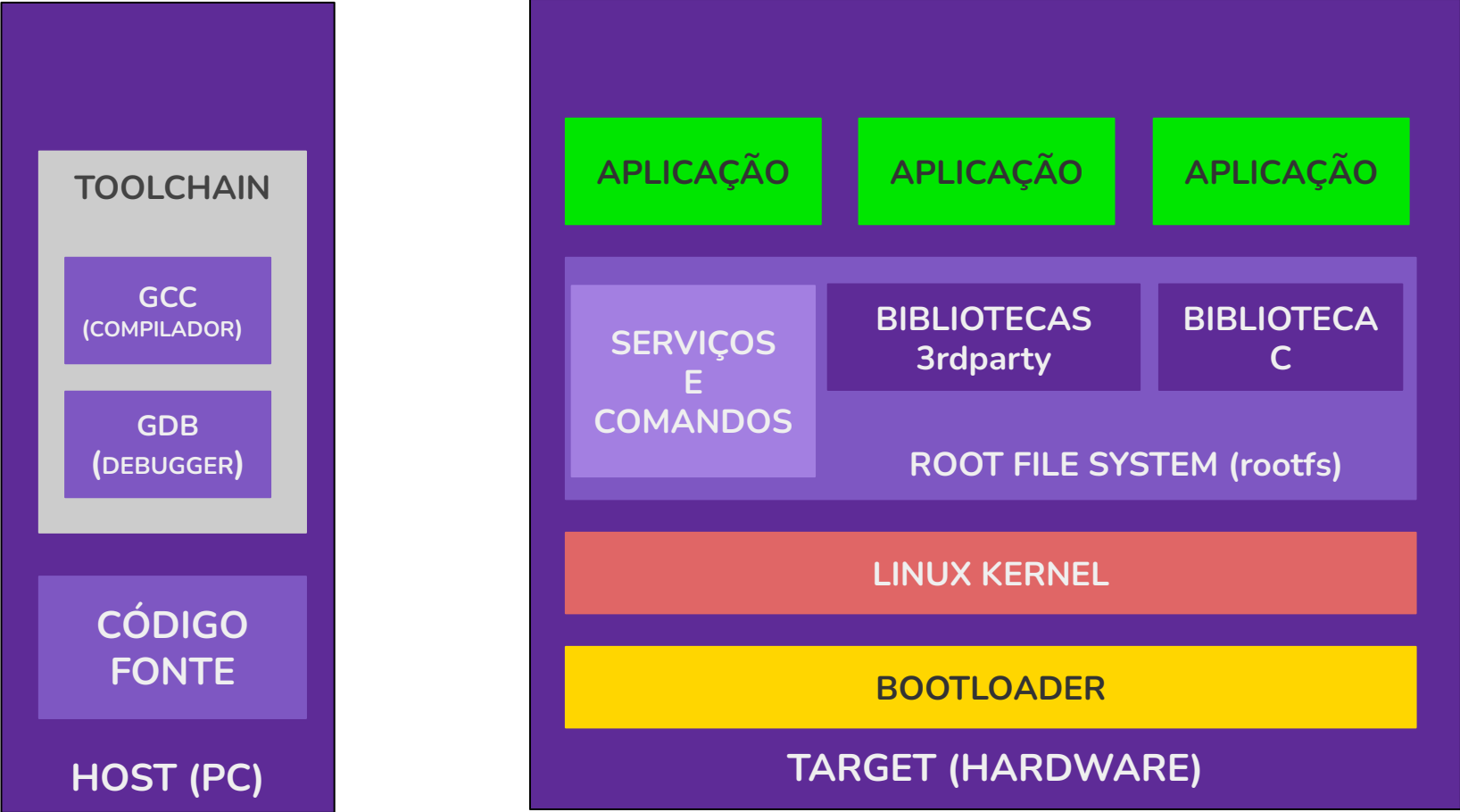
Arquitetura de um Sistema Linux Embarcado

Sistema Linux Embarcado, contempla...

- ❖ Toolchain
- ❖ Bootloader
- ❖ Kernel Linux
- ❖ Biblioteca C
- ❖ Bibliotecas e Aplicações



Arquitetura Sistema Linux Embarcado



Toolchain

Obtendo as ferramentas antes de iniciar os trabalhos de compilação cruzada

Definição de Toolchain

Conjunto de ferramentas que recebem um código-fonte e geram um programa final (binários) compatível com arquitetura desejada.

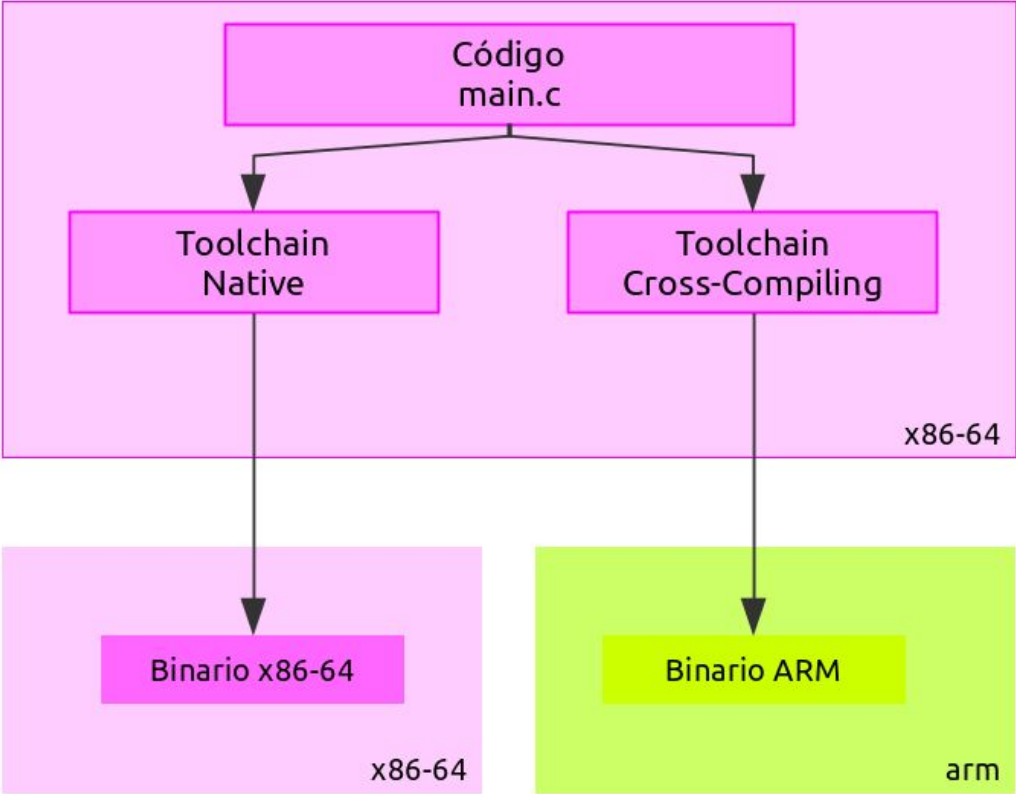


Tipos de Toolchain

- ❖ **Toolchain Nativo:** Utilizado em seu Desktop e gera código-nativo no próprio Host
- ❖ **Toolchain Cross-Compile:** Utilizado em Host mas para gerar código para outra plataforma(Target)



Toolchain Native x Toolchain Cross-Compiling



Máquinas na geração e uso de Toolchain

build

Onde o toolchain é construído

host

Onde o toolchain é executado

target

Onde os binários construídos pelo
toolchain serão executados



Diferentes Procedimentos de Build com Toolchain

x86-64

arm

mips

build

host

target

Native Build

Normalmente seu desktop, toolchain instalado e gera binário nativo

build

host

target

Cross Build

Gera toolchain para seu computador, mas gera binário para outra plataforma

build

host

target

Cross-Native Build (Canadian Cross)

Gera toolchain para outra plataforma e gera binário para esta plataforma



Diferentes Procedimentos de Build com Toolchain

x86-64

arm

mips

build

host

target

Native Build

Normalmente seu desktop, toolchain instalado e gera binário nativo

build

host

target

Cross Build

Gera toolchain para seu computador, mas gera binário para outra plataforma

build

host

target

Cross-Native Build (Canadian Cross)

Gera toolchain para outra plataforma e gera binário para esta plataforma



Componentes de um toolchain

Binutils

(as, ld, ar, ranlib, strip, objdump,
readelf, size, nm)

Kernel Headers

(Dependências da libC para
“entender” as chamadas e gerar o
binário corretamente)

Biblioteca C

(glibc, uclibc, musl, ...)

Compilador C/C++

(gcc e g++)

Cross-Compiling Toolchain

Depurador

(gdb)



- ❖ Conjuntos de ferramentas para gerar e manipular binários para uma arquitetura alvo
 - ▶ **as** - O montador, gera binário baseado em um arquivo .S
 - ▶ **ld** - O linker, ligado o arquivo objeto com as bibliotecas
 - ▶ **ar, ranlib** - Para geração de bibliotecas estáticas
 - ▶ **objdump, readelf, size, nm, strings** - Para inspecionar binarios
 - ▶ **strip** - Remover partes do binário(debugging) para reduzir tamanho

GNU Binutils: <https://www.gnu.org/software/binutils/>



A biblioteca C e os programas compilados precisam interagir com o kernel

- ❖ Chamadas de sistema disponíveis e seus números
- ❖ Definições de constantes
- ❖ Estruturas de dados, etc.

Portanto, compilar a biblioteca C requer cabeçalhos do kernel, e muitos aplicativos também os requerem.



Cabeçalhos do Kernel

- ❖ Números das chamadas de sistema em <asm/unistd.h>:

```
#define __NR_exit 1  
#define __NR_fork 2  
#define __NR_read 3
```

- ❖ Definições de constantes em <asm-generic/fcntl.h>:

```
#define O_RDWR 00000002
```

[glibc fcntl-linux](#)

- ❖ Estruturas de dados em <asm/stat.h>:

```
struct stat {  
    unsigned long st_dev;  
    unsigned long st_ino;  
    [...]  
};
```



Cabeçalhos do Kernel

Os cabeçalhos do kernel podem ser extraídos das fontes do kernel usando:



```
$ make headers_install ARCH=arm INSTALL_HDR_PATH=/usr
```



- ❖ O **GCC** (GNU Compiler Collection) é o compilador do projeto GNU.
<https://gcc.gnu.org/>
- ❖ Compatível com diversas linguagens de programação, incluindo **C**, **C++**, **Objective-C**, **Fortran**, **Ada**, **Go** e **D**.
- ❖ Pode gerar código para diversas arquiteturas, incluindo **ARM**, **AVR**, **Blackfin**, **MIPS**, **PowerPC**, **x86**, etc.

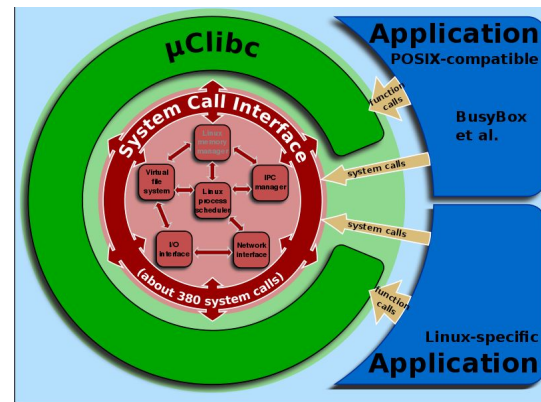


- ❖ O que é a biblioteca C?
 - ▶ Interface entre as aplicações e o kernel.
 - ▶ Toda implementação de C vem com biblioteca padrão de funções pré-definidas
 - ▶ Note que, em programação, uma biblioteca é uma coleção de funções
 - ▶ As funções comuns a todas as versões de C são conhecidas como Biblioteca Padrão C (open, close, exit, read, write, malloc, ...)



Bibliotecas C

- ❖ O que é a biblioteca C?
 - ▶ API para o desenvolvimento de aplicações.
 - ▶ O toolchain depende da biblioteca C, já que ele irá linká-la com sua aplicação para gerar os binários para a placa alvo.
 - ▶ Bibliotecas C populares disponíveis: glibc, uClibc-ng, musl, dietlibc, klibc, bionic, etc.



Fonte:
Wikipedia(<https://bit.ly/2zrGve2>)



Principais Bibliotecas C disponíveis

glibc

eglibc

LGPL

Biblioteca C do projeto GNU

Foco em performance e produtividade,
principal biblioteca em Desktop e Servidores Linux

uclibc-ng

LGPL

Mais leve e foco para sistemas embarcados

Fork do uClibc, build menor que glibc, alguns recursos podem
não estar disponíveis como funções de tempo-real

musl

MIT

Biblioteca leve, rápida e simples

Projeto escrito com foco em Sistemas Embarcados, ótima
opção para gerar binários linkado estaticamente

klibc

BSD/GPL
v2

Biblioteca extremamente pequena e limitada

Projeto escrito do zero, foco em tamanho e limitado
para uso básico como initramfs



Escolhendo o Toolchain

- ❖ Toolchain Pronto
- ❖ Gerando o próprio Toolchain



- ❖ Fácil instalação - comum instalação via gerenciador de pacotes ou download direto, configurar variável **PATH**
- ❖ Principal vantagem instalar e já sair utilizando
- ❖ Desvantagem é não permitir customização ou otimizar
- ❖ Algumas opções:
 - ▶ [Linaro](#)
 - ▶ [Code Sourcery](#)
 - ▶ [Bootlin GNU toolchain](#)



Gerando o próprio Toolchain

- ❖ Gerar o próprio toolchain não é uma tarefa fácil, muitas dependências, ferramentas, patches e configurações
- ❖ Mas hoje há ferramentas que facilitam este trabalho
- ❖ Algumas são: **Crosstool-ng**, **Buildroot** e **Yocto Project**



Principais Bibliotecas C disponíveis

Crosstool-ng

Suporta uClibc, glibc e musl

Sistema de configuração baseado em menuconfig

Buildroot

Suporta uClibc, glibc e musl

Sistema baseado em Makefiles

Yocto Project

Suporta glibc e musl

Com mais recursos, mais complexo de configuração



Principais Bibliotecas C disponíveis

Crosstool-ng

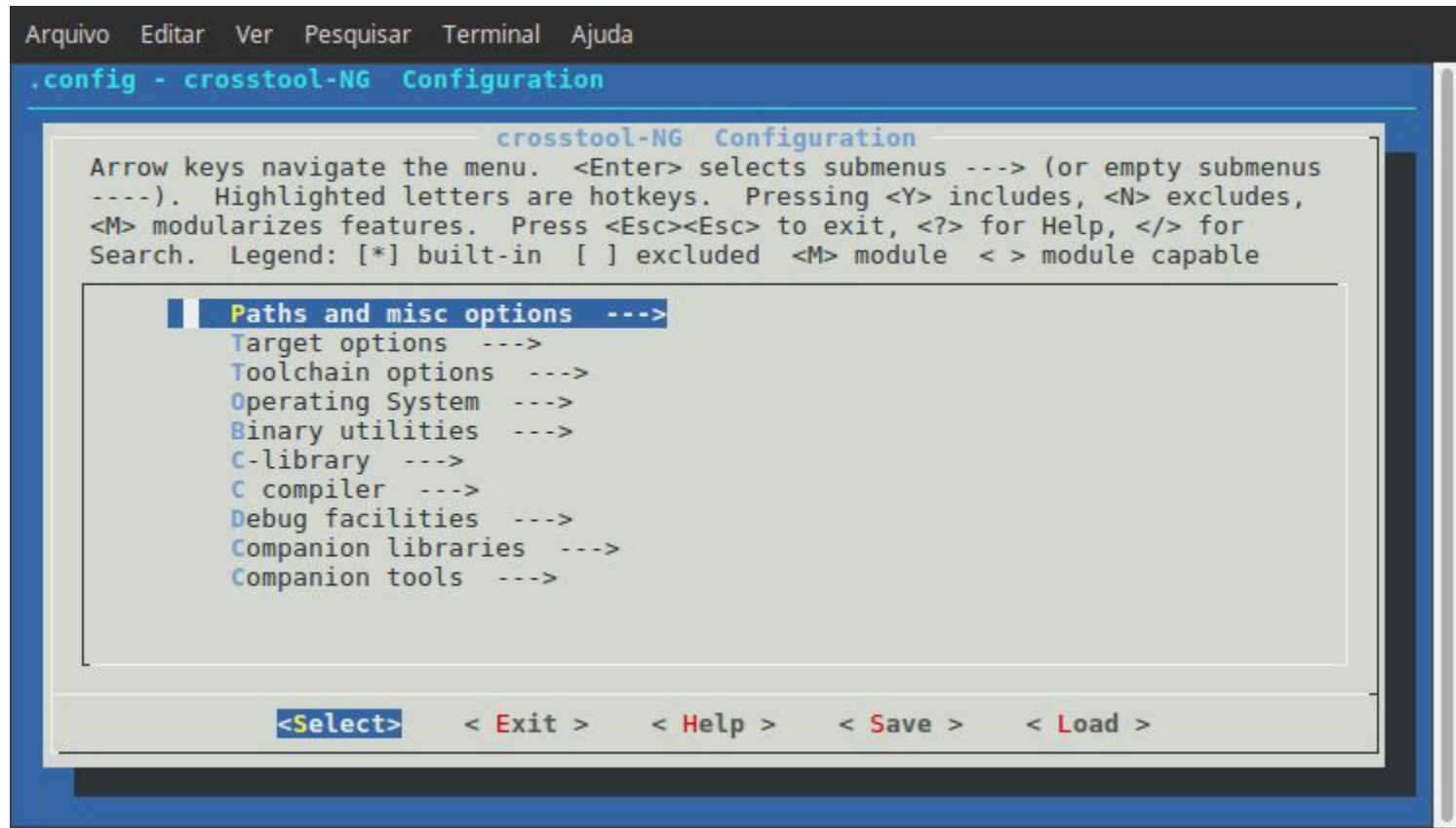
Suporta uClibc, glibc e musl

Sistema de configuração baseado em menuconfig

- ❖ Principal ferramentas open-source para geração de toolchains
- ❖ Possibilidade de gerar toolchains para diferentes arquiteturas: **x86, x86-64, MIPS, ARM**



Principais Bibliotecas C disponíveis

A screenshot of the crosstool-NG Configuration menu. The window has a title bar with the text ".config - crosstool-NG Configuration". The menu itself has a title "crosstool-NG Configuration" and a paragraph of instructions: "Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty submenus ---). Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*] built-in [] excluded <M> module < > module capable". Below this is a list of menu items, with "Paths and misc options --->" highlighted. The list includes "Target options --->", "Toolchain options --->", "Operating System --->", "Binary utilities --->", "C-library --->", "C compiler --->", "Debug facilities --->", "Companion libraries --->", and "Companion tools --->". At the bottom of the menu is a bar with five buttons: "<Select>", "< Exit >", "< Help >", "< Save >", and "< Load >".

```
Arquivo  Editar  Ver  Pesquisar  Terminal  Ajuda
.config - crosstool-NG  Configuration

                                crosstool-NG  Configuration
Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus
----).  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
<M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </> for
Search.  Legend: [*] built-in [ ] excluded  <M> module  < > module capable

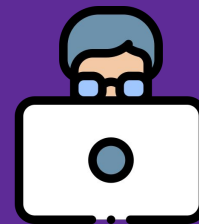
  Paths and misc options  --->
    Target options  --->
    Toolchain options  --->
    Operating System  --->
    Binary utilities  --->
    C-library  --->
    C compiler  --->
    Debug facilities  --->
    Companion libraries  --->
    Companion tools  --->

<Select>  < Exit >  < Help >  < Save >  < Load >
```



Laboratório

- ❖ Gerando o próprio toolchain com **Crosstool-NG**





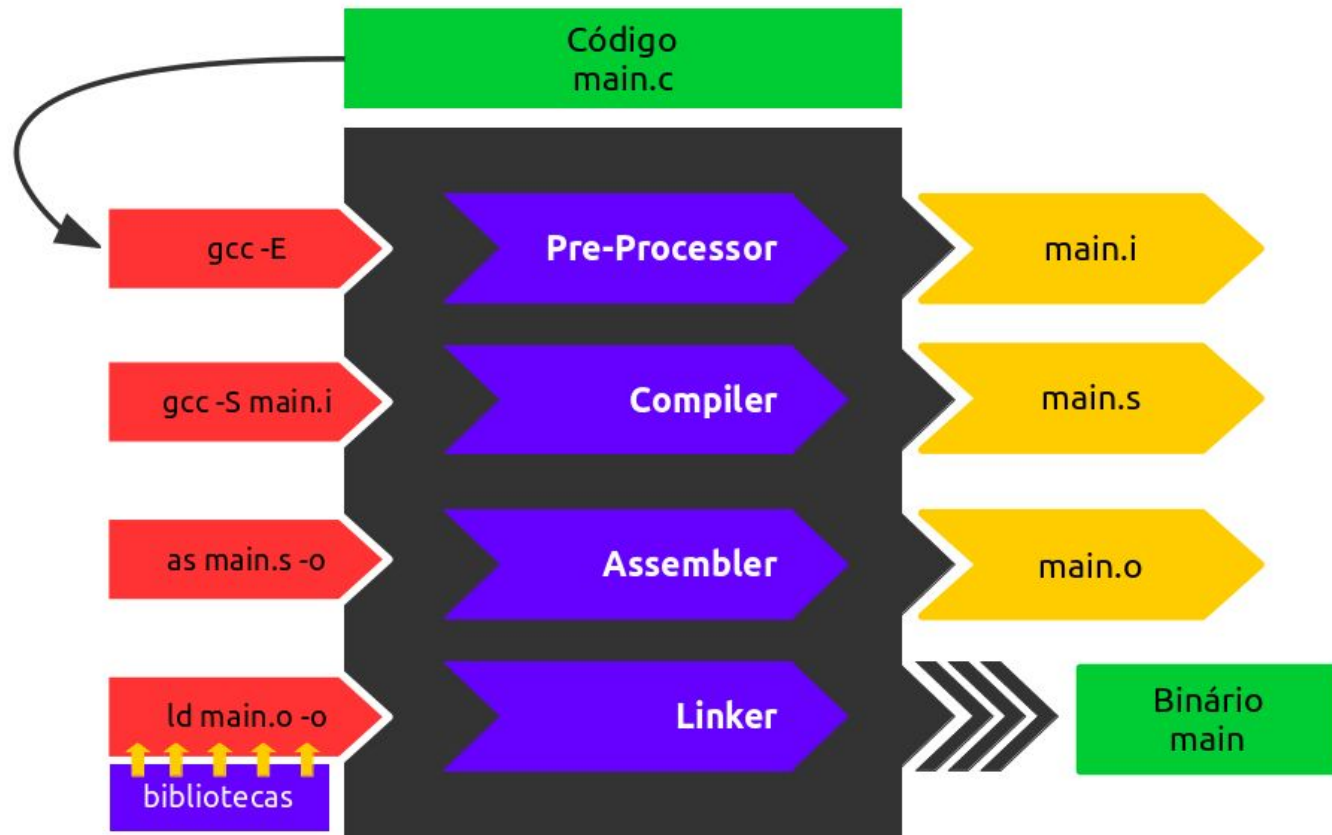
**Como um
programa em C é
compilado?**

C

—



Como um programa C é compilado?



Como um programa C é compilado?

1. **Pre-processor**(Pre-Processamento): Através do pré-processador GNU C (gcc), que inclui os cabeçalhos (`#include`) e expande as macros (`#define`).



```
$ gcc -E main.c > main.i
```



Como um programa C é compilado?

2. **Compiler**(Compilador): O compilador (gcc) compila o código-fonte pré-processado em código assembly para um processador específico. A opção `-S` específica para gerar código assembly, ao invés de código-objeto. Um arquivo `main.s` é será gerado!



```
$ gcc -S main.i
```



Como um programa C é compilado?

3. **Assembler**(Montador): O assembler (as) converte o código do assembly em código de máquina no arquivo objeto "main.o".



```
$ as -o main.o main.s
```



Como um programa C é compilado?

4. **Linker**(Ligador): Por último, o linker(ld) vincula o código do objeto ao código das bibliotecas para gerar um arquivo executável "main" para a arquitetura alvo.



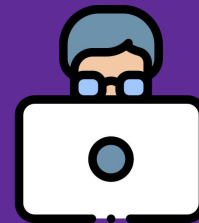
```
$ ld -o main main.o
```



Laboratório

Prática 01

- ❖ Compilando um código nativo e um para ARM





Sequência de inicialização

- ❖ **Bootloader:** Inicialização básica do hardware e carregamento do Kernel Linux.
- ❖ **Linux Kernel:** Inicializa o restante do hardware e configura de acordo com device-tree, na sequência chama o init.
- ❖ **RootFS:** Aplicações, Bibliotecas, Serviços(inclusive o init) e ferramentas.



Sequência de Inicialização

ROM
Code

BOOTLOADER
(u-boot)

KERNEL

linux kernel

device tree binary

initramfs

ROOTFS

init

services

...

services

application

(C, C++, Python, Java, Rust, Go, Lua, ...)

libc
(glibc)

libs 3rd-party
(libOpenGL.so, libsqlite3.so, ...)



Bootloader



- ❖ Uma das principais funções do bootloader é:
 - ▶ Inicialização básica do Hardware (CPU, Controladora da Memória RAM, unidades de armazenamento, etc)
 - ▶ Carregar outro binário normalmente o Kernel para continuar a inicialização
 - ▶ Passar o controle da CPU para o novo binário

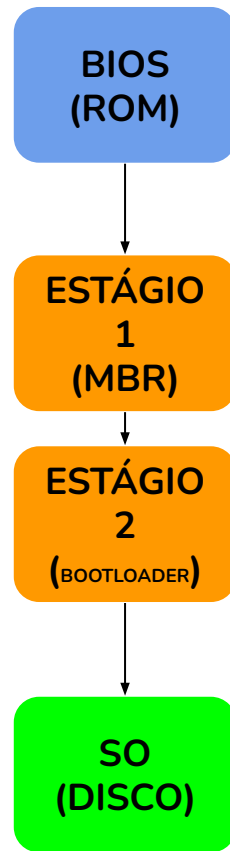


- ❖ Além dessas funções básicas, a maioria dos bootloaders fornece um shell que permite diversas operações:
 - ▶ Variáveis Ambiente, Manipulação de RAM e Flash, Inspeção de Memória, Diagnóstico de Hardware e Testes, passagem de parâmetros para o Kernel



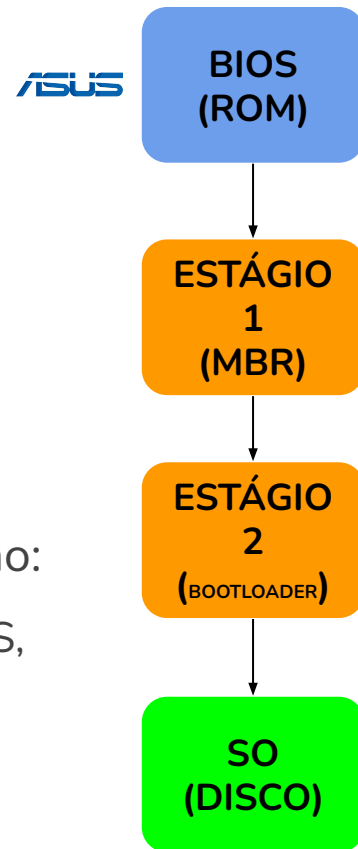
Bootloader x86

- ❖ Em plataformas x86 o bootloader reside em uma memória não-volátil, popularmente chamado de BIOS(Basic Input/Output System)
- ❖ Há BIOS está em ROM e é fornecida pelo fabricante, seria o **Estágio 1** da inicialização básica.
- ❖ Em seguida, entra o **Estágio 2** onde é carregado do Bootloader do Disco para a RAM e uma inicialização mais completa.
- ❖ O **Estágio 2** consegue carregar o Kernel e o SO do Disco e realizar a inicialização completa do SO.



Bootloader x86

- ❖ Principal Bootloader utilizado em x86/x86-64 - GRUB:
 - ▶ GRUB(Grand Unified Bootloader) - [Site GRUB](#)
 - ▶ Opção de Dual-Boot
 - ▶ Suporta Boot-Seguro e UEFI
 - ▶ Suporta diversos Sistemas de Arquivos como:
Linux(EXT2/EXT3/EXT4), DOS(FAT12/FAT16/FAT32/exFAT), NTFS,
ReiserFS, RomFS, etc



- ❖ A CPU possui um código de inicialização integrado na ROM.
 - ▶ ROM Code em iMX6
 - ▶ Cada CPU possui seus detalhes e particularidades do fabricante
- ❖ Este ROM Code é capaz de carregar um Bootloader de primeiro estágio em um dos dispositivos de armazenamento em uma SRAM interna (DRAM ainda não inicializada)
- ❖ O dispositivo de armazenamento pode ser normalmente: MMC, NAND, Flash SPI, etc.

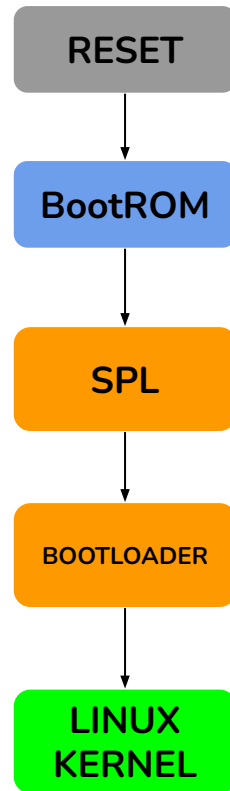


- ❖ O bootloader de **primeiro estágio** é:
 - ▶ Tamanho limitado devido a restrições de hardware (tamanho SRAM)
 - ▶ Fornecido por U-Boot (chamado de Secondary Program Loader - SPL) ou pelo Fabricante do CPU
- ❖ Este Bootloader de **primeiro estágio** deve inicializar DRAM e outros dispositivos de hardware e carregar um Bootloader de **segundo estágio** em DRAM

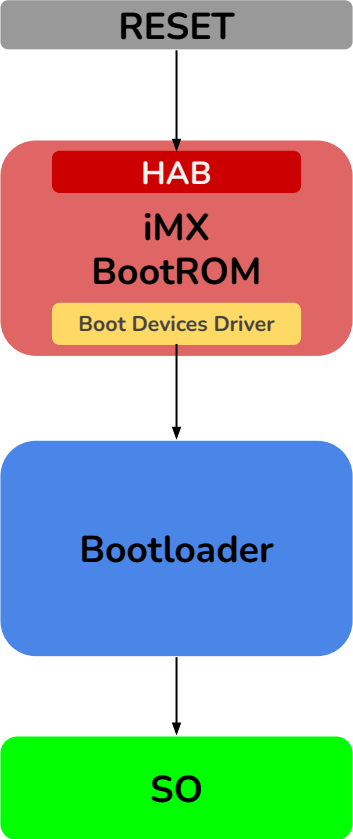


Bootloader ARM

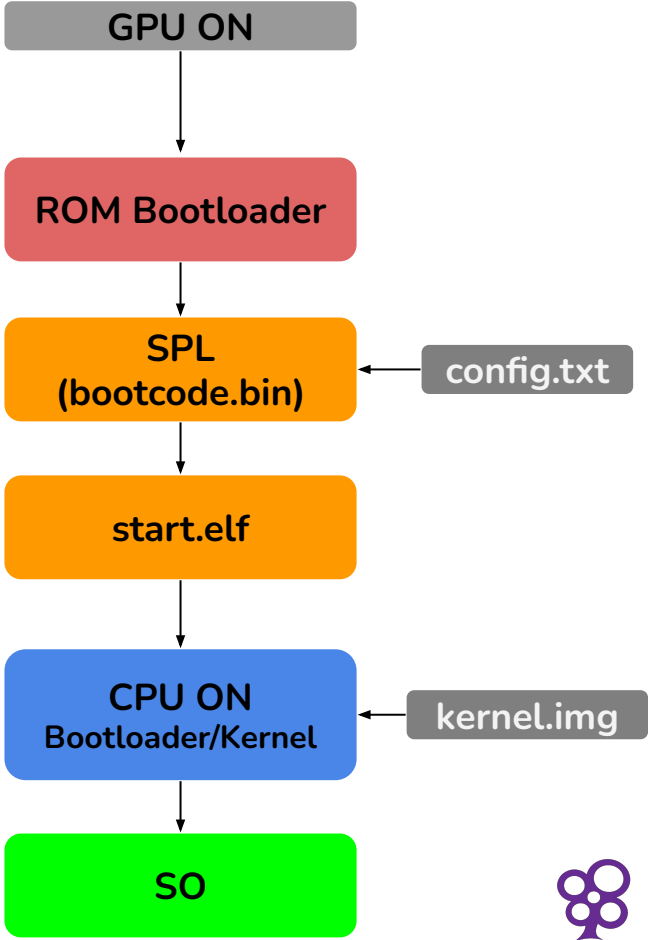
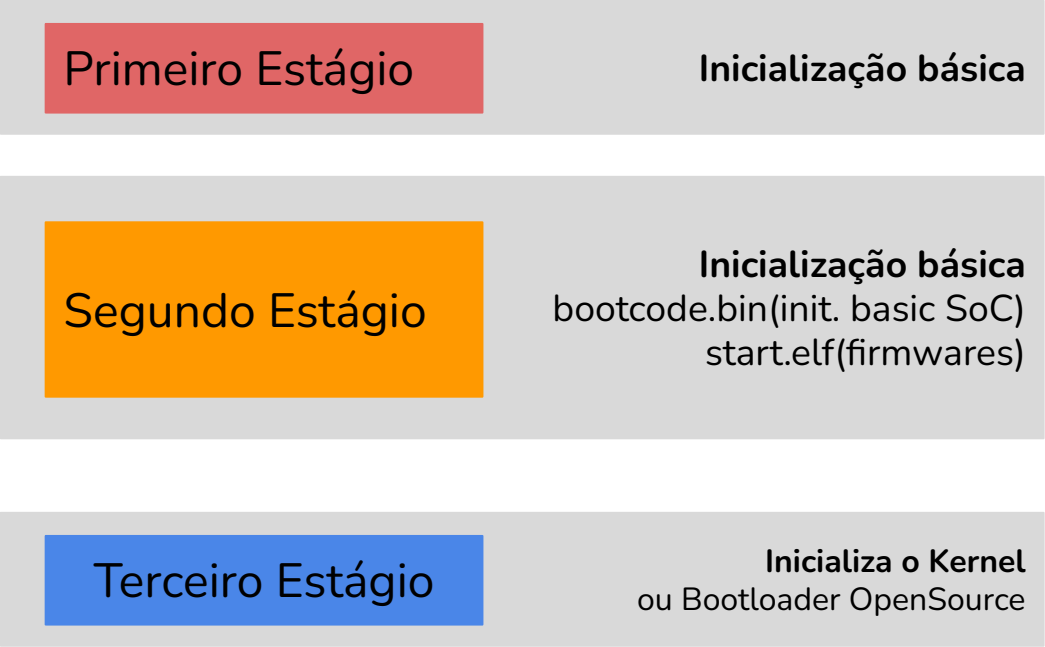
- ❖ O Bootloader de **segundo estágio** é mais poderoso, com mais recursos e até prompt para iteração.
- ❖ É com o Bootloader de **segundo estágio** que o Kernel Linux é carregado e iniciado o boot do SO, neste momento o Bootloader não atua mais.



Bootloader ARM - Família iMX



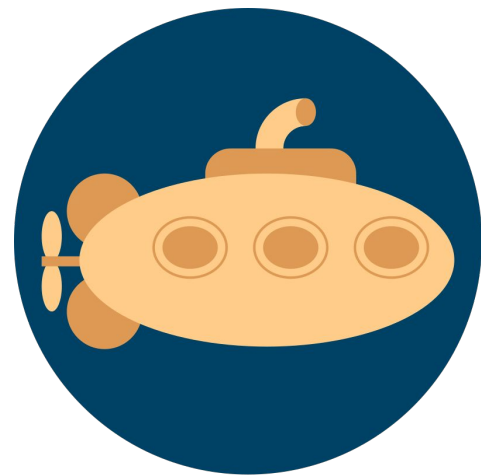
Bootloader ARM - RaspberryPI3



- ❖ Entre diversas opções open-source de Bootloader para Embarcados, podemos destacar:
 - ▶ **U-Boot** - Universal Bootloader desenvolvido e mantido pela Denx, é o mais popular e utilizar em Embarcados sendo ARM, MIPS, PowerPC e até x86
 - ▶ **BareBox** - Desenvolvido pela Pengutronix, pouca adoção no mercado mas promete uma ótima alternativa ao U-Boot.



- ❖ Licença GPLv2(mesma do Kernel Linux)
- ❖ Mantido pela comunidade e empresas
- ❖ Projeto bem estruturado/organizado
- ❖ O repositório do projeto é disponibilizado em Git
- ❖ Cada 2~3 meses uma nova versão release é disponibilizada no formato YYYY.MM

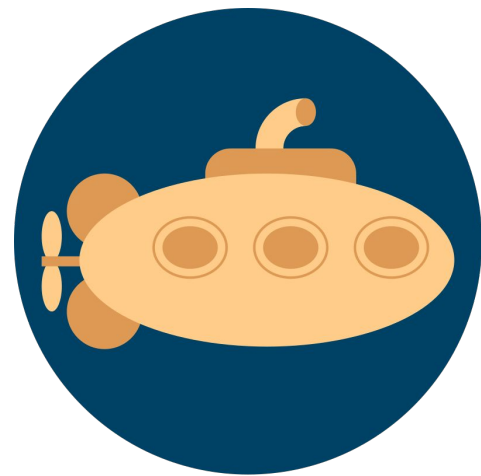


U-Boot



Bootloader - U-Boot

- ❖ Prompt Interativo
- ❖ Exibe informações do hardware ao carregar
- ❖ Acesso via Serial Console
- ❖ Boot via: Rede(TFTP), MMC, USB, Flash SPI, NAND, etc
- ❖ Manipula variáveis ambiente e script interno
- ❖ Entender Sistemas de Arquivos EXT2/EXT3/EXT4, VFAT, UBIFS, etc
- ❖ Carrega o binário Kernel e passa parâmetros de inicialização



U-Boot



Bootloader - U-Boot

❖ Estrutura do projeto

```
b2open@treinamento-b2open:~/treinamento/Lab/Bootloader/u-boot-2021.04$ ls
api          disk         include      net           u-boot       u-boot.srec
arch         doc          Kbuild       post          u-boot.bin   u-boot.sym
board        drivers      Kconfig      README        u-boot.cfg
cmd          dts          lib           scripts       u-boot.cfg.configs
common       env          Licenses     System.map    u-boot.lds
config.mk    examples    MAINTAINERS  test          u-boot.map
configs      fs          Makefile     tools         u-boot-nodtb.bin
```



- ❖ Diretório **/configs** contém um ou vários arquivos de configuração para cada placa suportada
 - ▶ Ele define o tipo de CPU, os periféricos e sua configuração, e os recursos de U-Boot que devem ser suportados. Exemplos:

configs/omap3_beagle_defconfig

configs/rpi_3_32b_defconfig



Bootloader - U-Boot

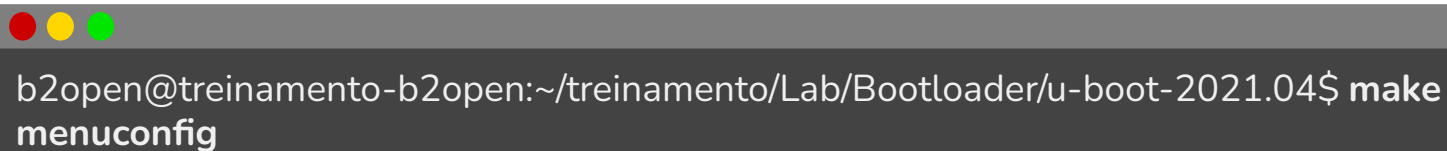
- ❖ Antes de compilar o Bootloader U-Boot deve utilizar uma configuração visto no slide anterior, como exemplo realizando a configuração para uma **RaspberryPI3**:

```
b2open@treinamento-b2open:~/treinamento/Lab/Bootloader/u-boot-2021.04$ make  
rpi_3_32b_defconfig
```



Bootloader - U-Boot <CONFIGURANDO>

- ❖ Um arquivo .config é gerado no mesmo diretório com as configurações padrões, você deverá alterar as configurações via menuconfig



```
b2open@treinamento-b2open:~/treinamento/Lab/Bootloader/u-boot-2021.04$ make menuconfig
```



Bootloader - U-Boot <CONFIGURANDO>

```
.config - U-Boot 2021.04 Configuration

                                U-Boot 2021.04 Configuration
Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenu
----).  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
<M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </> for
Search.  Legend: [*] built-in [ ] excluded <M> module < > module capable

*** Compiler: gcc (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0 ***
Architecture select (ARM architecture) --->
ARM architecture --->
General setup --->
API --->
Boot options --->
Console --->
Logging --->
Init options --->
Security support ----
v(+)

<Select>  < Exit >  < Help >  < Save >  < Load >
```



Bootloader - U-Boot <COMPILANDO>

- ❖ Para compilar o U-Boot basta utilizar o comando **make** e fornecer o prefixo do toolchain configurado, como nosso compilado é **arm-unknown-linux-uclibcgnueabi-gcc** devemos utilizar **arm-unknown-linux-uclibcgnueabi-:**

```
b2open@treinamento-b2open:~/treinamento/Lab/Bootloader/u-boot-2021.04$ make  
CROSS_COMPILE=arm-unknown-linux-uclibcgnueabi-  
...  
LDS      u-boot.lds  
LD       u-boot  
OBJCOPY  u-boot.srec  
OBJCOPY  u-boot-nodtb.bin  
COPY     u-boot.bin  
SYM      u-boot.sym
```



Bootloader - U-Boot <COMPILANDO>

- ❖ No final da compilação será gerada uma imagem do Bootloader U-Boot para ser gravada ou transferida para a placa de desenvolvimento, o arquivo gerado pode variar entre **u-boot**, **u-boot.bin**, **u-boot.img**.

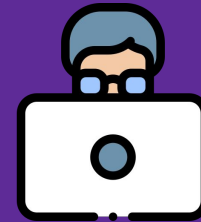
```
b2open@treinamento-b2open:~/treinamento/Lab/Bootloader/u-boot-2021.04$ file
u-boot
u-boot: ELF 32-bit LSB shared object, ARM, EABI5 version 1 (SYSV), dynamically
linked, with debug_info, not stripped

b2open@treinamento-b2open:~/treinamento/Lab/Bootloader/u-boot-2021.04$ ls -l
u-boot
-rwxr-xr-x 1 b2open b2open 4417208 Jul 23 16:05 u-boot
```



Laboratório

- ❖ Configurando e compilando o U-Boot para ARM





Kernel Linux





Um pouco de história

- ❖ O kernel Linux é um dos componentes do Sistema Operacional, que também requer bibliotecas e aplicativos para fornecer acesso aos recursos.
- ❖ O kernel Linux foi criado como hobby em 1991 por um estudante finlandês, Linus Torvalds.
- ❖ Linus Torvalds conseguiu criar uma grande e dinâmica comunidade de desenvolvedores e usuários em torno do Linux.
- ❖ Hoje em dia, mais de mil pessoas contribuem para cada lançamento do kernel, empresas grandes e pequenas.



Principais Recursos do Kernel Linux

- ❖ Portabilidade e suporte de hardware.
- ❖ Suportado pela maioria das arquiteturas (veja `arch/` no código-fonte).
- ❖ Escalabilidade. Pode ser executado em MainFrames como em Dispositivos Embarcados.
- ❖ Segurança. Seu código é revisado por muitos especialistas.
- ❖ Estabilidade e confiabilidade (pode funcionar por dias/meses sem reiniciar)

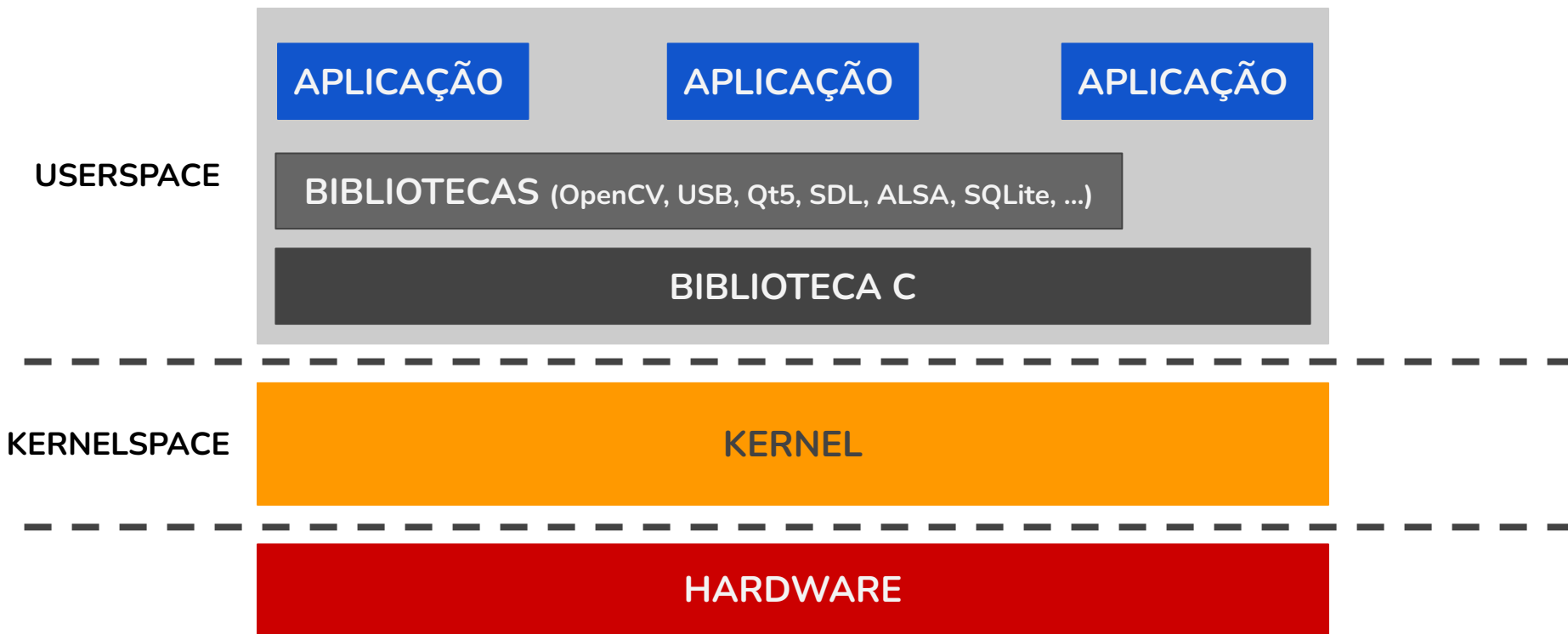


Principais Recursos do Kernel Linux

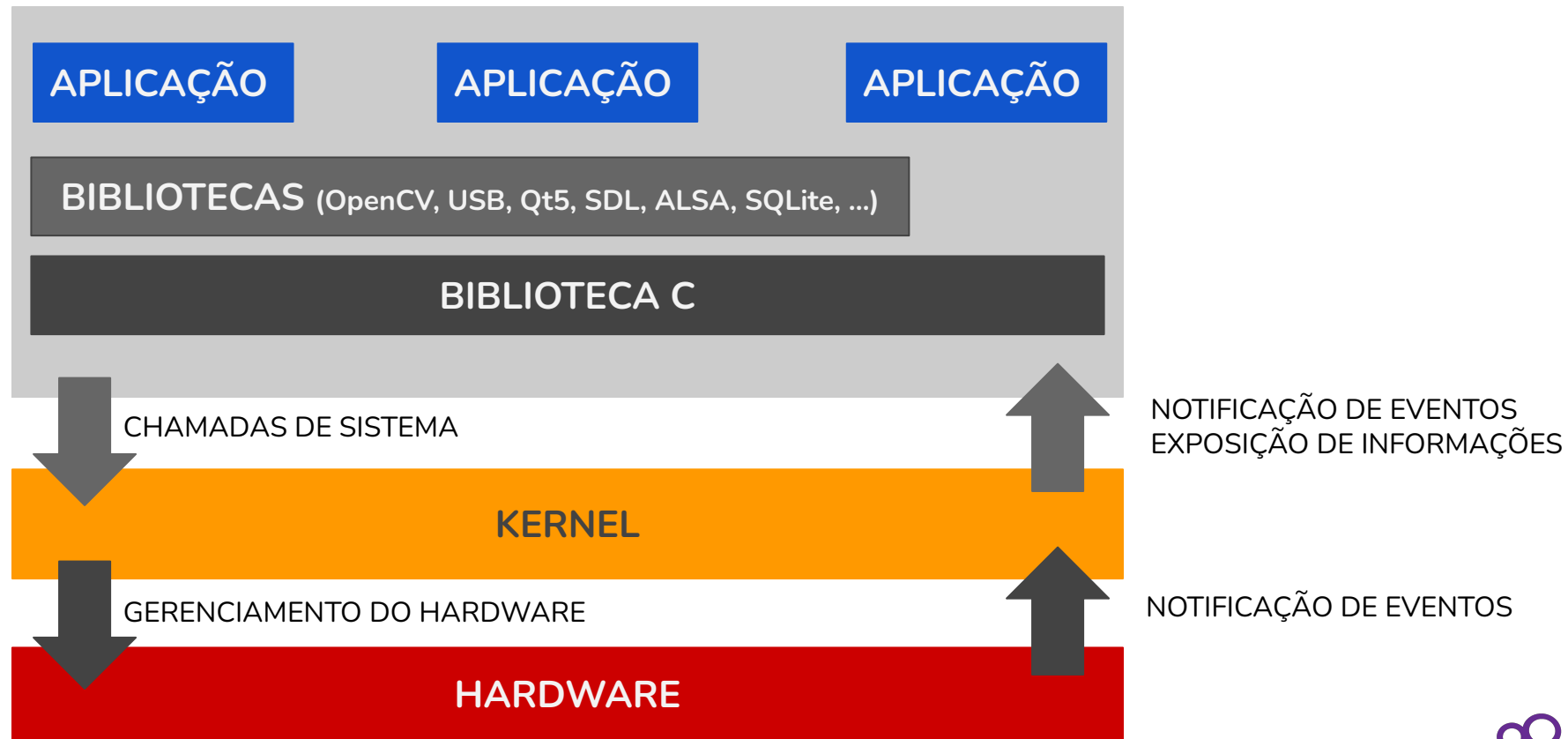
- ❖ Modularidade. Pode incluir apenas o que um sistema precisa, mesmo em tempo de execução (até mesmo o próprio kernel pode ser atualizado sem reiniciar - **kexec**)
- ❖ Todo código-fonte existente em sua maioria público, principal linguagem utilizada C, então pode aprender com o código existente, corrigir e implementar melhorias e novas funcionalidades



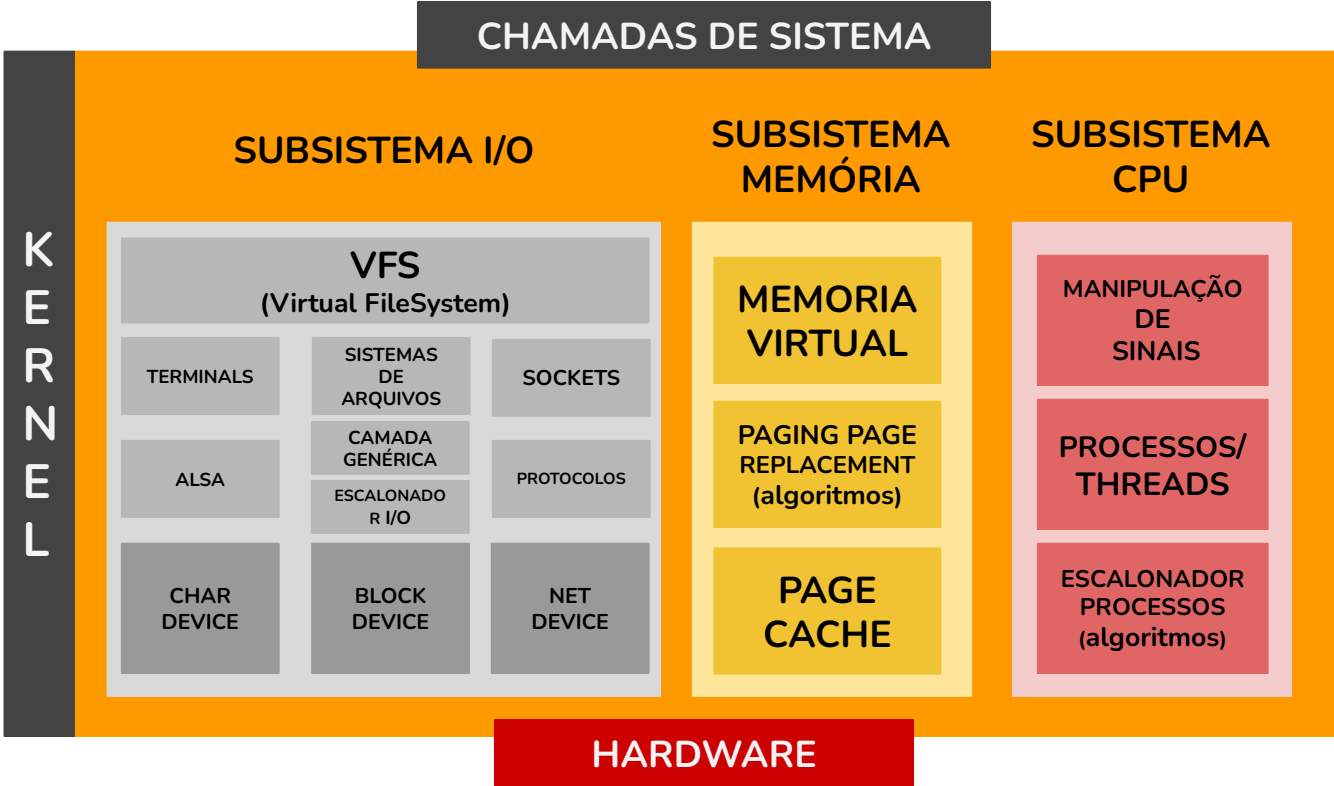
Por dentro do Kernel Linux



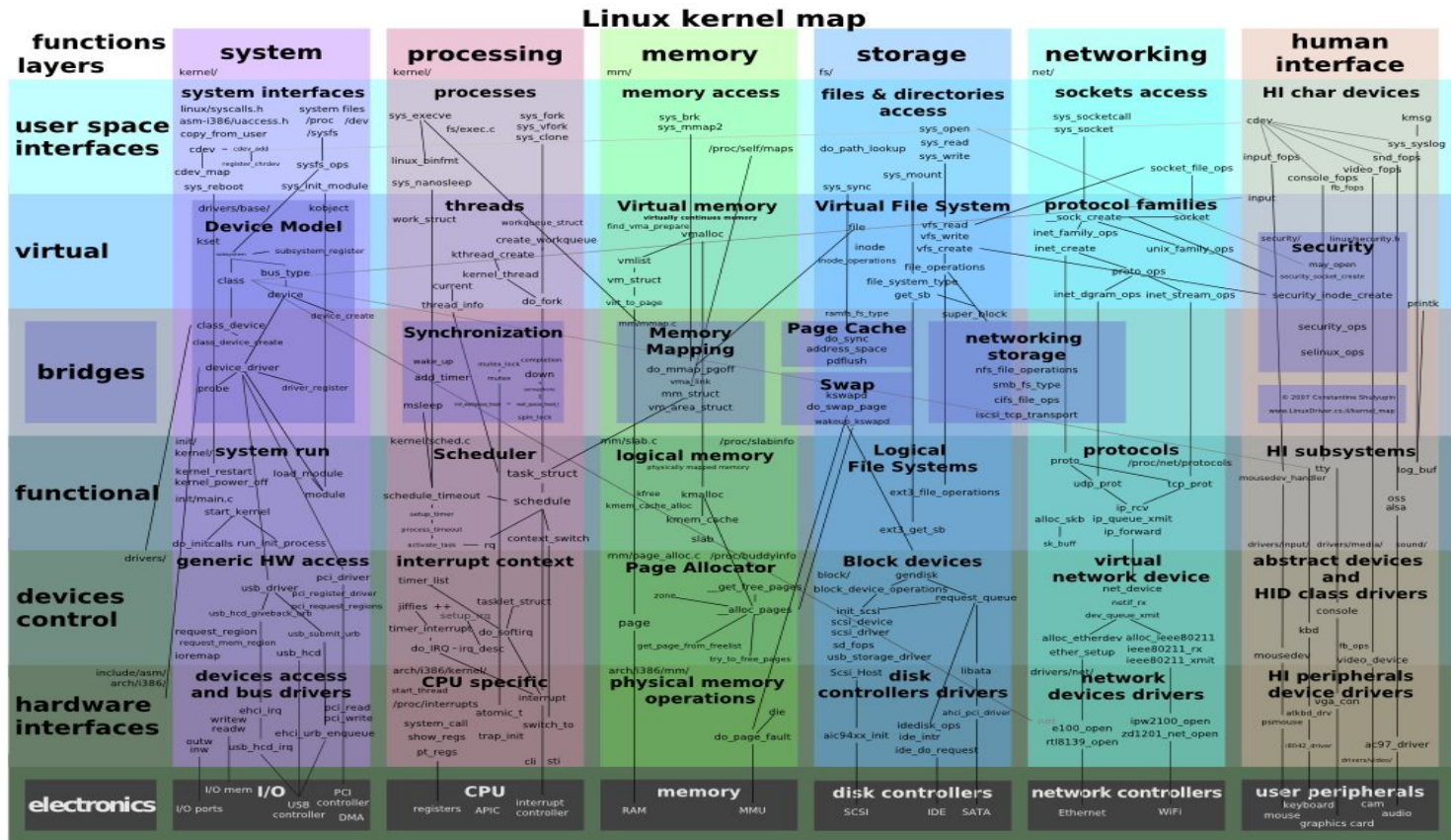
Por dentro do Kernel Linux



Por dentro do Kernel Linux



Por dentro do Kernel Linux



Referência: <https://makelinux.github.io/kernel/map/>



Por dentro do Kernel Linux

- ❖ Gerenciar todos os recursos de hardware: CPU, memória, I/O.
- ❖ Fornece um conjunto de APIs portáteis, de arquitetura e independentes de hardware para permitir que aplicativos e bibliotecas em userspace usem os recursos de hardware.
- ❖ O Kernel é responsável por multiplexar os acessos concorrente e compartilhados dos recursos de diferentes aplicações



- ❖ O Linux torna as informações do sistema e do kernel disponíveis no userspace por meio de pseudo sistemas de arquivos(VFS)
- ❖ No Linux praticamente podemos dizer que tudo é representado por um arquivo, mas com VFS podemos dizer pseudo-arquivos, com isso podemos manipular como um arquivo utilizando as chamadas **open**, **read**, **write**, **close**, **seek**, etc.



- ❖ Os dois principais VFS são:
 - ▶ **procfs** - Geralmente montado em **/proc**: Informações relacionadas ao Sistema Operacional (processos, parâmetros, gerenciamento de memória ...)
 - ▶ **sysfs** - Geralmente montado em **/sys**: Representação do sistema como uma árvore de dispositivos conectados por barramentos. Informações coletadas pelas estruturas do kernel que gerenciam esses dispositivos



- ❖ Todo o código-fonte do kernel Linux é software livre e liberado sob a licença GPLv2.

Isso significa que:

- Quando você receber ou comprar um equipamento com Linux embarcado, você tem o direito de requisitar os fontes, alterá-los e redistribuí-los!
- Quando você produzir um equipamento com Linux embarcado, você precisa liberar os fontes do kernel sob as mesmas condições, sem restrições.



Por dentro do Kernel Linux - **VERSIONAMENTO**

- ❖ Até 2003, havia um novo branch de versão estável do Linux a cada 2 ou 3 anos (2.0, 2.2, 2.4). Novos ramos de desenvolvimento levaram 2-3 anos para se tornarem estáveis (era muito lento!).
- ❖ Desde 2003, há uma nova versão estável do Linux a cada 10 semanas:

Versões 2.6 (dezembro de 2003)	a	2.6.39 (maio de 2011)
---------------------------------------	----------	------------------------------

Versões 3.0 (julho de 2011)	a	3.19 (fevereiro de 2015)
------------------------------------	----------	---------------------------------

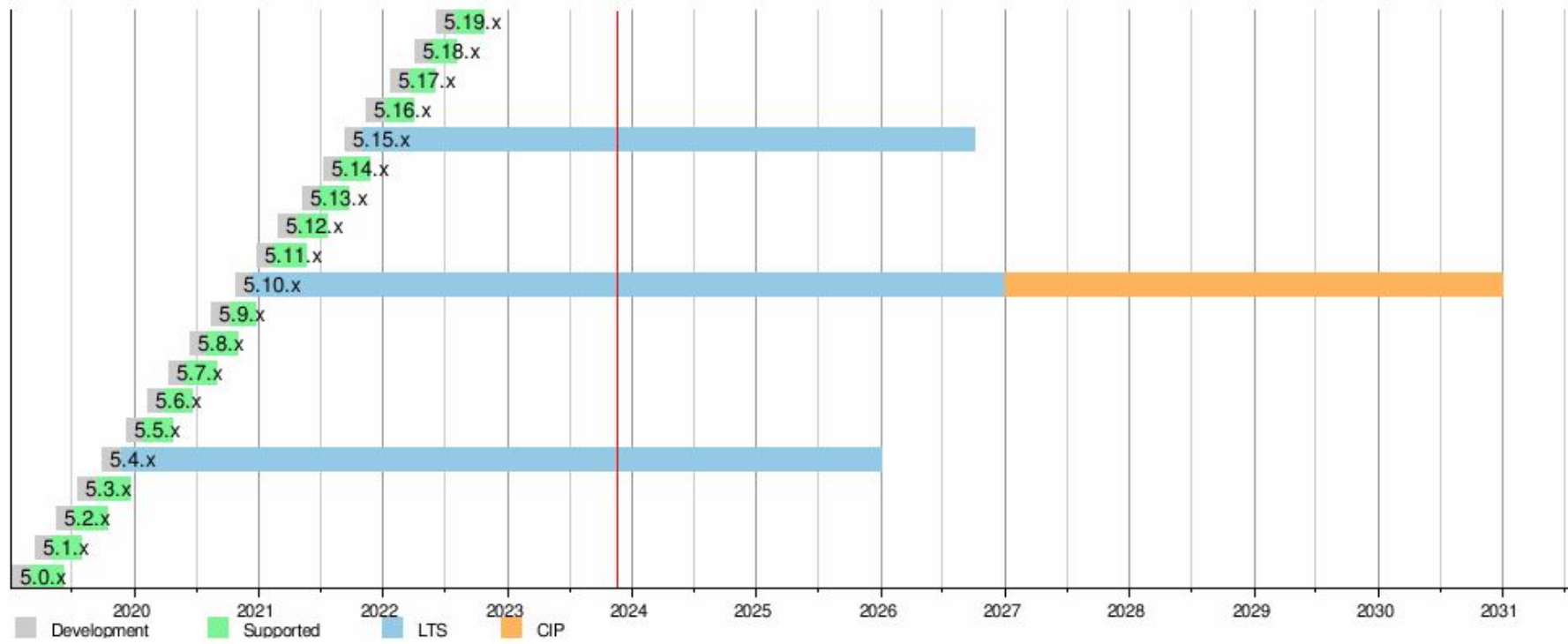
Versões 4.0 (abril de 2015)	a	4.20 (dezembro de 2018)
------------------------------------	----------	--------------------------------

Versão 5.0 (março de 2019)	
-----------------------------------	--

Versão 6.0 (outubro de 2022)

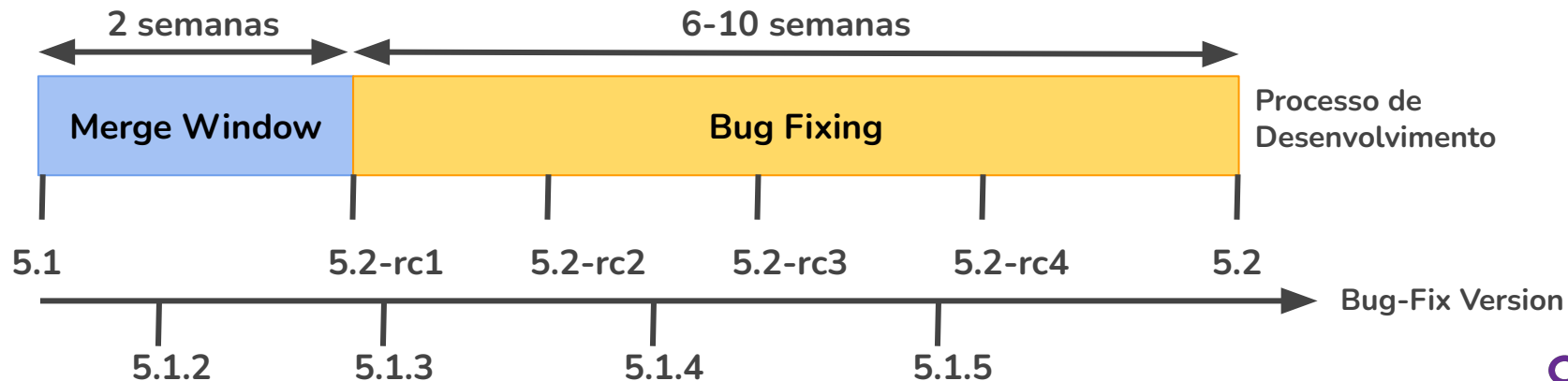


Por dentro do Kernel Linux - **VERSIONAMENTO**



Por dentro do Kernel Linux - **VERSIONAMENTO**

- ❖ Recursos são adicionados ao kernel de forma progressiva. Desde 2003.
- ❖ Processo de desenvolvimento a cada 3 meses aproximadamente, até lançar o release final X.Y.



Por dentro do Kernel Linux - **VERSIONAMENTO**

- ❖ Para acompanhar as mudanças no kernel:
 - <https://kernelnewbies.org/LinuxChanges>
 - <https://lwn.net/Kernel>



Onde baixar o Kernel?

- ❖ As versões oficiais (mainline) do kernel Linux, conforme lançado por Linus Torvalds, estão disponíveis em: <https://kernel.org>
- ❖ As fontes do kernel estão disponíveis em <https://kernel.org/pub/linux/kernel> como tarballs completos (fontes completas do kernel) e patches (diferenças entre duas versões do kernel).
- ❖ Há opção de utilizar o repositório Git:
 - [git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git](https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git)
- ❖ Opção para visualizar o repositório via Web:
 - <https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/>



Onde baixar o Kernel?

- ❖ Exemplo baixando tarball e com git:

```
$ wget https://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.10.57.tar.xz
```

```
$ git clone git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git
```



Onde baixar o Kernel?

- ❖ Comunidades e Fabricantes de hardware podem manter suas próprias versões do kernel:
 - ▶ Fabricantes de hardware fornecem suas próprias versões do kernel baseado em sua plataforma de referência (BSP)
 - ▶ Comunidade pode manter versões para arquiteturas especificar(ARM, MIPS, RISC-V) subsistemas(USB, network), Sistemas de Tempo-Real, etc



Onde baixar o Kernel?

- ❖ Kernel Toradex:

<https://git.toradex.com/cgit/linux-toradex.git/>

- ❖ Kernel RaspberryPi:

<https://github.com/raspberrypi/linux>

- ❖ Kernel Texas Instruments:

<https://git.ti.com/cgit/ti-linux-kernel/ti-linux-kernel/>



Onde baixar o Kernel?

❖ O Kernel Linux da RaspberryPI - branch **rpi-5.4.y**

- **66.405** arquivos
- **27.938.153** linhas
- **896.875.240** bytes
- Tamanho total - **1.3G**
 - **arch/** - 162MB
 - **Documentation/** - 48MB
 - **drivers/** - 621MB
 - **fs/** - 41MB
 - **net/** - 32MB
 - **sound/** - 39MB



Onde baixar o Kernel?

- ❖ O Kernel Linux é grande!
- ❖ Um dos principais motivos é o gigantesco suporte a diversos drivers e arquiteturas
- ❖ No entanto, como o Kernel é modular e configurável o binário final é extremamente pequeno podendo ser de 1 ~ 2MB, casos com drivers de GPU, Codecs totalizando 5 ~ 8MB



Como configurar o Kernel?

❖ Modo Gráfico



```
$ make xconfig
```



```
$ make gconfig
```

❖ Modo Texto



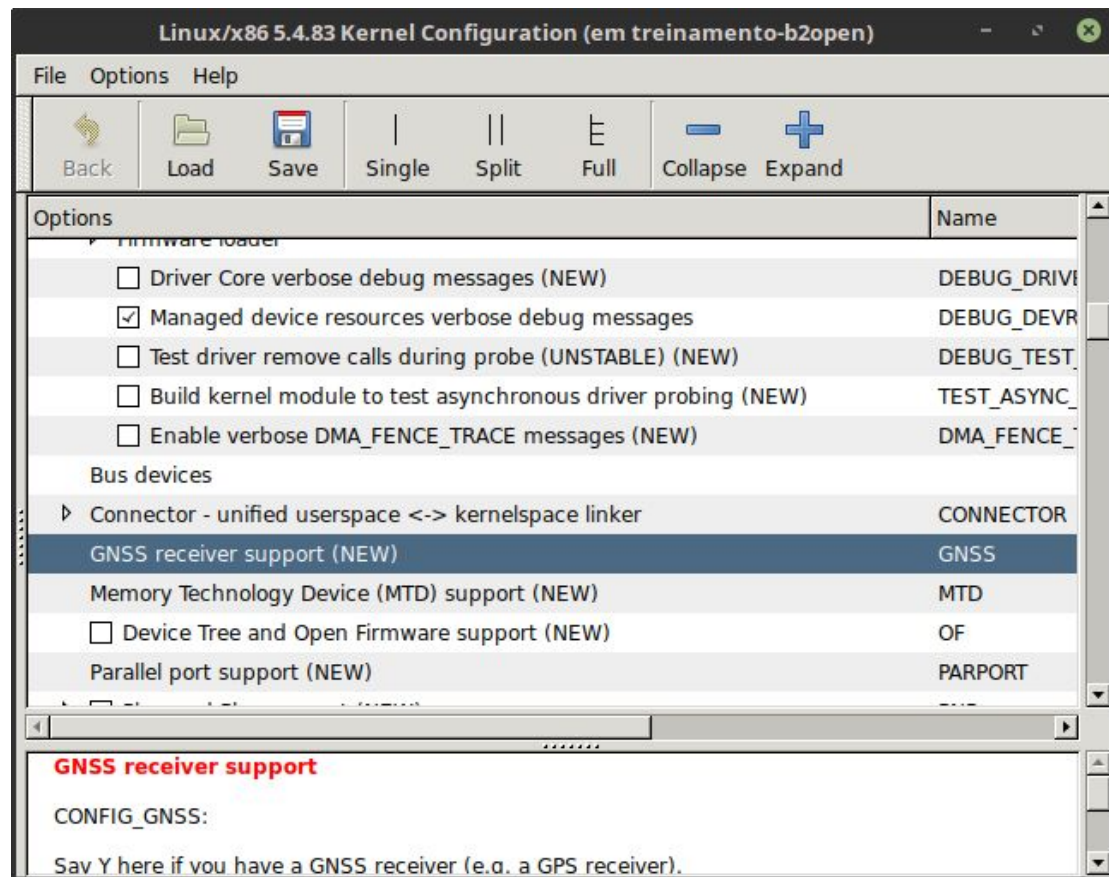
```
$ make menuconfig
```



```
$ make nconfig
```



Como configurar o Kernel?



Como configurar o Kernel?

.config - Linux/x86 5.4.83 Kernel Configuration

Linux/x86 5.4.83 Kernel Configuration

Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty submenus ----). Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*] built-in [] excluded <M> module < > module capable

*** Compiler: gcc (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0 ***

General setup --->

[*] 64-bit kernel (NEW)

Processor type and features --->

Power management and ACPI options --->

Bus options (PCI etc.) --->

Binary Emulations --->

Firmware Drivers --->

[*] Virtualization (NEW) --->

General architecture-dependent options --->

[*] Enable loadable module support --->

-*- Enable the block layer --->

IO Schedulers --->

Executable file formats --->

Memory Management options --->

v(+)

<Select>

< Exit >

< Help >

< Save >

< Load >



Como configurar o Kernel?

.config - Linux/x86 5.4.83 Kernel Configuration
Linux/x86 5.4.83 Kernel Configuration

```
*** Compiler: gcc (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0 ***
General setup --->
[*] 64-bit kernel (NEW)
Processor type and features --->
Power management and ACPI options --->
Bus options (PCI etc.) --->
Binary Emulations --->
Firmware Drivers --->
[*] Virtualization (NEW) --->
General architecture-dependent options --->
[*] Enable loadable module support --->
-*- Enable the block layer --->
IO Schedulers --->
Executable file formats --->
Memory Management options --->
[*] Networking support --->
Device Drivers --->
File systems --->
Security options --->
-*- Cryptographic API --->
Library routines --->
Kernel hacking --->
```

F1 Help F2 SymInfo F3 Help 2 F4 ShowAll F5 Back F6 Save F7 Load F8 SymSearch F9 Exit



Como compilar o Kernel?

- ❖ Deve-se especificar a variável ARCH e CROSS_COMPILE
 - ARCH - Baseado na arquitetura alvo, veja **/arch**
 - CROSS_COMPILE - Especifica o prefixo do toolchain para (gcc, ld, strip, as, ...)

```
$ export ARCH=arm  
$ export CROSS_COMPILE=arm-unknown-linux-uclibcgnueabi-
```

```
$ make ARCH=arm CROSS_COMPILE=arm-unknown-linux-uclibcgnueabi- ...
```



Configurações do Kernel

- ❖ As configurações do Kernel são armazenadas no arquivo .config
- ❖ Gerado e manipulado via **make menuconfig**, **make gconfig** ...
- ❖ Pode ser editado manualmente
- ❖ A configuração é baseado em CHAVE=VALOR
 - CONFIG_EXFAT_FS=y (Habilitado Built-in)
 - CONFIG_BT_MRVL=m (Habilitado como Módulo)
 - CONFIG_SQUASHFS=n (Desabilitado)



- ❖ O Kernel é gerado em um binário final com diversos recursos básicos e necessários para o boot completo
- ❖ Alguns recursos (drivers de dispositivo, sistemas de arquivos, etc.) podem, no entanto, ser compilados como módulos.
 - Esses são plug-ins que podem ser carregados/liberados dinamicamente para adicionar/remover recursos do kernel



Resultado da compilação do Kernel

- ❖ O **vmlinux**, uma imagem descomprimida do kernel, em formato ELF, não possível realizar boot mas utilizado para depuração
- ❖ **arch/<arch>/boot/*Image**, a imagem final do kernel comprimida e que pode dar boot, alguns formatos:
 - bzImage para x86
 - zImage para ARM
- ❖ **arch/<arch>/boot/Image**, imagem kernel sem compressão que também pode inicializar
- ❖ **arch/<arch>/boot/dts/*.dtb**, resultado da compilação do device-tree

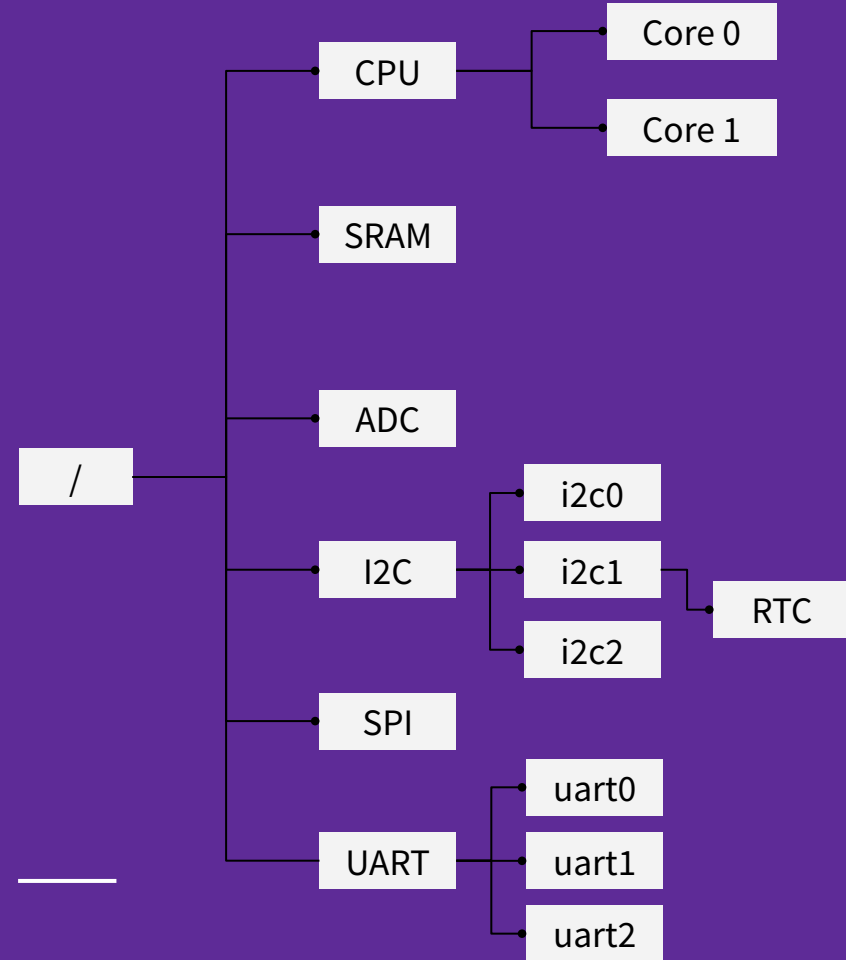


Resultado da compilação do Kernel

- ❖ Todos os módulos do Kernel(opções configuradas com =m), gerados com .ko(Kernel Object)
- ❖ Instalar os módulos requer o comando `make modules_install`
- ❖ `/lib/modules/<version>/`, diretório padrão de instalação dos módulos (.ko)



Device-Tree



Em **computadores/servidores** a inicialização conta com o auxílio da **BIOS/UEFI** e os periféricos são conectados ao processador principal por meio de um barramento(BUS), alguns barramentos suportam enumeração(discovery).

Desta forma o processador requisita ao barramento e o dispositivos retornam informações como tipo, fabricante, modelo e configurações.

O Sistema Operacional irá decidir qual driver irá utilizar para o dispositivos após receber estas informações.



Device-tree x ACPI

Em **sistemas embarcados**(ARM) não temos(no geral) BIOS/UEFI para auxiliar na inicialização, e contém barramentos que não há enumeração.

Com diversas plataformas ARM o Kernel Linux começou a ficar grande, complexo e muita duplicidade de código de diferentes fabricantes e a cada nova especificação mais código dentro do kernel. 

Alterar um simples endereço de um dispositivo na I2C teria que alterar o `platform` `device` dentro do kernel para cada plataforma ARM :/



Device-tree x ACPI

Em 2011, Linus Torvalds e alguns desenvolvedores do Kernel expressaram descontentamento com esse processo, estabelecendo um ultimato: eles se recusariam a aceitar novas implementações até que a ARM reorganizasse e estruturasse de maneira mais eficiente as implementações redundantes de plataformas, núcleos e drivers.

Esta demanda resultou na adoção contínua do **Device-Tree**, uma prática que persiste até hoje.

Referência: <https://lwn.net/Articles/443510/>



❖ Servidores/Computadores (x86) [ACPI]

Utiliza ACPI Table e possui enumeração/descoberta com barramentos como PCI, PCIe e USB

❖ Sistemas Embarcados (ARM) [DEVICE TREE]

Utiliza Device Tree para inicialização e configuração dos dispositivos com barramentos como SPI e I2C



“ O **Device Tree** (DT) no contexto de sistemas embarcados e arquiteturas **ARM** é descrever a configuração de hardware de um sistema. Ele fornece uma representação hierárquica (Root Node, Child Node, Children Node) da configuração de hardware por meio de uma estrutura de árvore.”

O Kernel se encarrega de inicializar e/ou carregar o driver corretamente se necessário.

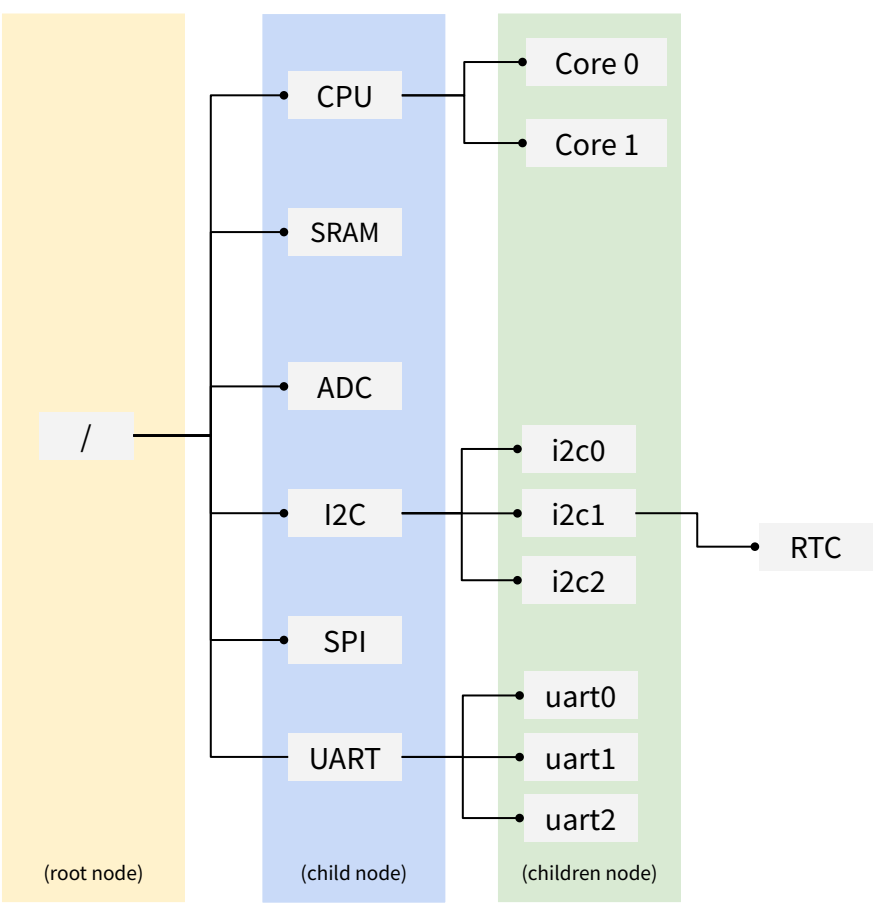


Device-tree <extensões>

DTS	Device-Tree Source	Arquivo da descrição do hardware, o código-fonte
DTSI	Device-Tree Source Include	Arquivo de origem que contém definições comuns que podem ser incluídas em vários .dts
DTB	Device-Tree Blob	Arquivo resultado da codificação binária dos dados do .dts. O .dtb é o arquivo que o kernel Linux utiliza para obter as informações do hardware durante a inicialização
DTBO	Device-Tree Blob Overlay	Permite a personalização dinâmica do Device Tree, sem recompilar o original.
DTC	Device-Tree Compiler	Ferramenta que converte o arquivo .dts (source) em um formato binário chamado .dtb (binary).

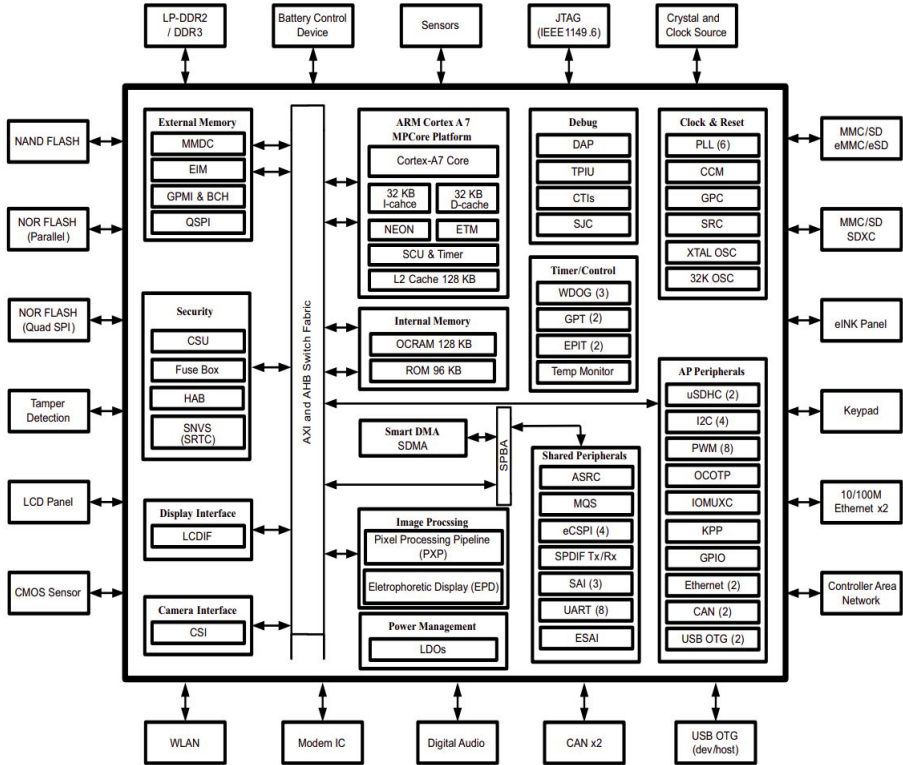


Device-tree



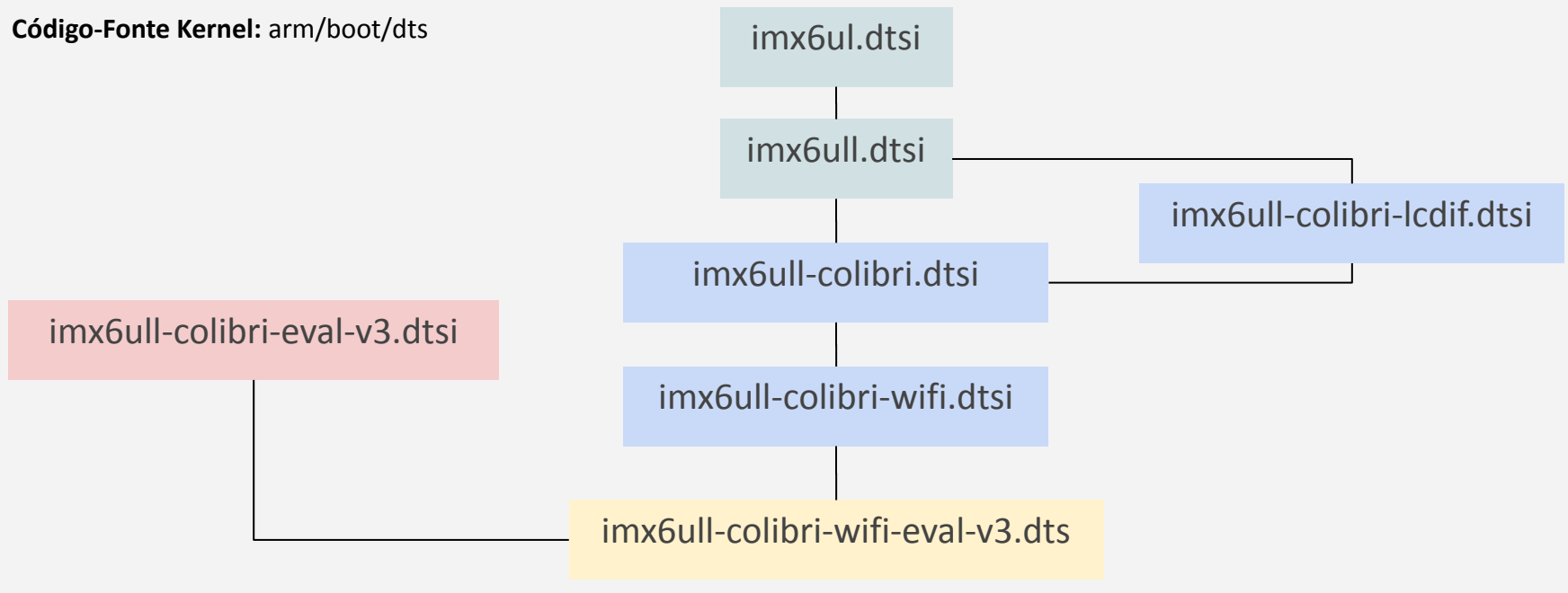
Device-tree

Para descrever um device-tree precisa de informações técnicas do Processador, SoC, Memória, Unidade de Armazenamento e CI's conectados nos periféricos.



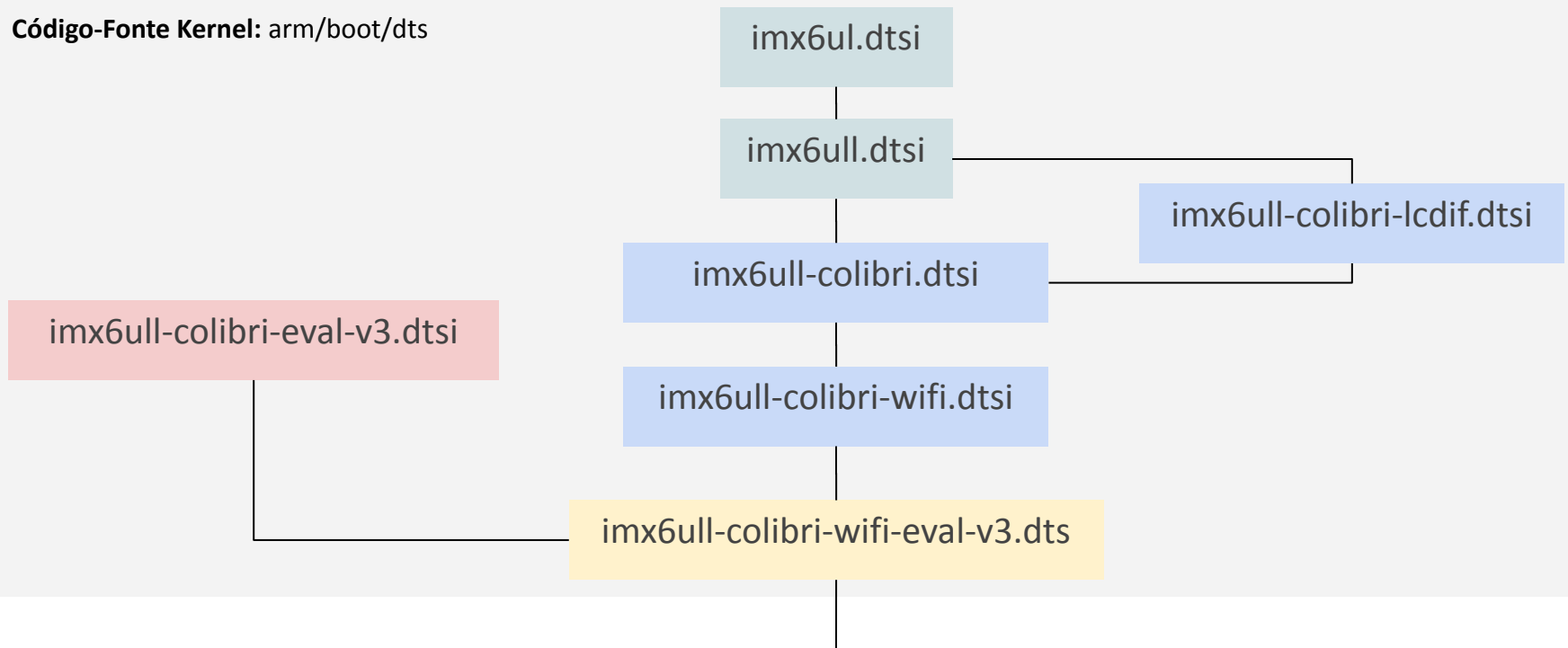
Device-tree <código-fonte e compilação>

Código-Fonte Kernel: arm/boot/dts



Device-tree <código-fonte e compilação>

Código-Fonte Kernel: arm/boot/dts

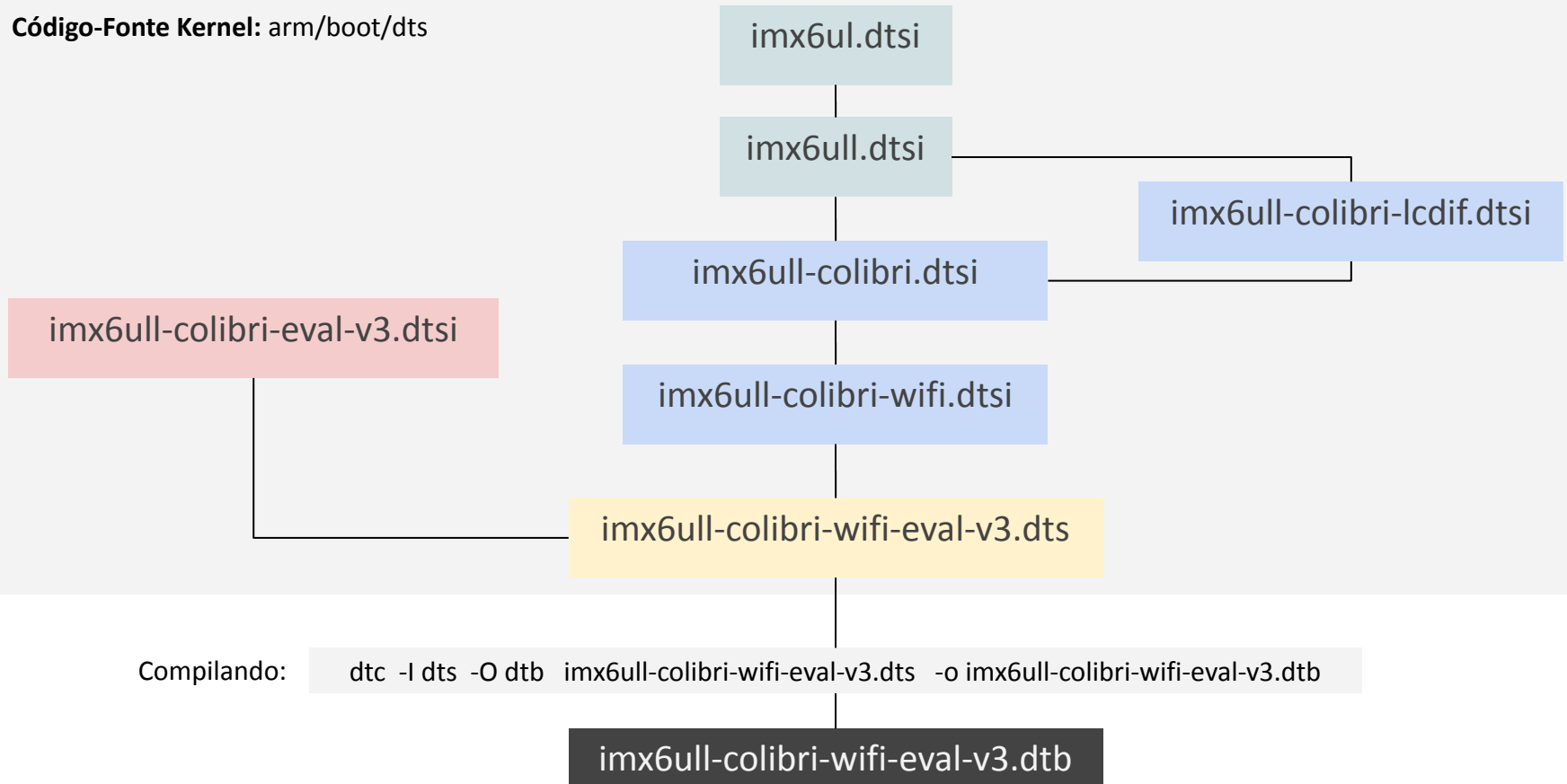


Compilando: `dtc -I dts -O dtb imx6ull-colibri-wifi-eval-v3.dts -o imx6ull-colibri-wifi-eval-v3.dtb`



Device-tree <código-fonte e compilação>

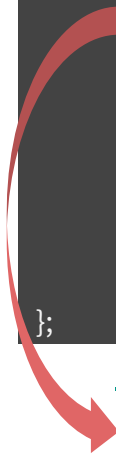
Código-Fonte Kernel: arm/boot/dts



Device-tree <exemplo de modificação>

[arch/arm/boot/dts/imx6ul.dts +30](#)

```
aliases {  
    ...  
    i2c0 = &i2c1;  
    i2c1 = &i2c2;  
    i2c2 = &i2c3;  
    i2c3 = &i2c4;  
    mmc0 = &usdhc1;  
    serial0 = &uart1;  
    serial1 = &uart2;  
    spi0 = &ecspi1;  
    ...  
};
```



[arch/arm/boot/dts/imx6ul.dtsi +1008](#)

```
i2c2: i2c@21a4000 {  
    #address-cells = <1>;  
    #size-cells = <0>;  
    compatible = "fsl,imx6ul-i2c", "fsl,imx21-i2c";  
    reg = <0x021a4000 0x4000>;  
    interrupts = <GIC_SPI 37 IRQ_TYPE_LEVEL_HIGH>;  
    clocks = <&clks IMX6UL_CLK_I2C2>;  
    status = "disabled";  
};
```

[arch/arm/boot/dts/imx6ull-colibri-eval-v3.dtsi +112](#)

```
&i2c1 {  
    status = "okay";  
  
    /* M41T0M6 real time clock on carrier board */  
    m41t0m6: rtc@68 {  
        compatible = "st,m41t0";  
        reg = <0x68>;  
    };  
};
```



Device-tree <exemplo de modificação>

[arch/arm/boot/dts/imx6ul.dts +30](#)

```
aliases {  
    ...  
    i2c0 = &i2c1;  
    i2c1 = &i2c2;  
    i2c2 = &i2c3;  
    i2c3 = &i2c4;  
    mmc0 = &usdhc1;  
    serial0 = &uart1;  
    serial1 = &uart2;  
    spi0 = &ecspi1;  
    ...  
};
```

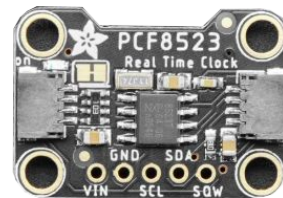
[arch/arm/boot/dts/imx6ul.dtsi +1008](#)

```
i2c2: i2c@21a4000 {  
    #address-cells = <1>;  
    #size-cells = <0>;  
    compatible = "fsl,imx6ul-i2c", "fsl,imx21-i2c";  
    reg = <0x021a4000 0x4000>;  
    interrupts = <GIC_SPI 37 IRQ_TYPE_LEVEL_HIGH>;  
    clocks = <&clks IMX6UL_CLK_I2C2>;  
    status = "disabled";  
};
```

[arch/arm/boot/dts/imx6ull-colibri-eval-v3.dtsi +112](#)

```
&i2c1 {  
    status = "okay";  
  
    /* M41T0M6 real time clock on carrier board */  
    m41t0m6: rtc@68 {  
        compatible = "st,m41t0";  
        reg = <0x68>;  
    };  
};
```

Adicionar
suporte ao
RTC
PCF8523



Device-tree <exemplo de modificação>

[arch/arm/boot/dts/imx6ul.dts +30](#)

```
aliases {  
    ...  
    i2c0 = &i2c1;  
    i2c1 = &i2c2;  
    i2c2 = &i2c3;  
    i2c3 = &i2c4;  
    mmc0 = &usdhc1;  
    serial0 = &uart1;  
    serial1 = &uart2;  
    spi0 = &ecspi1;  
    ...  
};
```

[arch/arm/boot/dts/imx6ul.dtsi +1008](#)

```
i2c2: i2c@21a4000 {  
    #address-cells = <1>;  
    #size-cells = <0>;  
    compatible = "fsl,imx6ul-i2c", "fsl,imx21-i2c";  
    reg = <0x021a4000 0x4000>;  
    interrupts = <GIC_SPI 37 IRQ_TYPE_LEVEL_HIGH>;  
    clocks = <&clks IMX6UL_CLK_I2C2>;  
    status = "disabled";  
};
```

[arch/arm/boot/dts/imx6ull-colibri-eval-v3.dtsi +112](#)

```
&i2c1 {  
    status = "okay";  
  
    /* M41T0M6 real time clock on carrier board */  
    m41t0m6: rtc@68 {  
        compatible = "st,m41t0";  
        reg = <0x68>;  
    };  
};
```

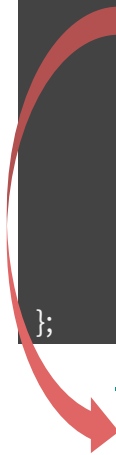
Adicionar
suporte ao
RTC
PCF8523



Device-tree <exemplo de modificação>

[arch/arm/boot/dts/imx6ul.dts +30](#)

```
aliases {  
    ...  
    i2c0 = &i2c1;  
    i2c1 = &i2c2;  
    i2c2 = &i2c3;  
    i2c3 = &i2c4;  
    mmc0 = &usdhc1;  
    serial0 = &uart1;  
    serial1 = &uart2;  
    spi0 = &ecspi1;  
    ...  
};
```



[arch/arm/boot/dts/imx6ul.dtsi +1008](#)

```
i2c2: i2c@21a4000 {  
    #address-cells = <1>;  
    #size-cells = <0>;  
    compatible = "fsl,imx6ul-i2c", "fsl,imx21-i2c";  
    reg = <0x021a4000 0x4000>;  
    interrupts = <GIC_SPI 37 IRQ_TYPE_LEVEL_HIGH>;  
    clocks = <&clks IMX6UL_CLK_I2C2>;  
    status = "disabled";  
};
```

[arch/arm/boot/dts/imx6ull-colibri-eval-v3.dtsi +112](#)

```
&i2c1 {  
    status = "okay";  
  
    /* M41T0M6 real time clock on carrier board */  
    m41t0m6: rtc@68 {  
        compatible = "st,m41t0";  
        reg = <0x68>;  
    };  
};
```



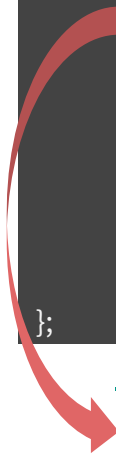
```
pcf8523: rtc@68 {  
    compatible = "nxp,pcf8523";  
    reg = <0x68>;  
};
```



Device-tree <exemplo de modificação>

[arch/arm/boot/dts/imx6ul.dts +30](#)

```
aliases {  
    ...  
    i2c0 = &i2c1;  
    i2c1 = &i2c2;  
    i2c2 = &i2c3;  
    i2c3 = &i2c4;  
    mmc0 = &usdhc1;  
    serial0 = &uart1;  
    serial1 = &uart2;  
    spi0 = &ecspi1;  
    ...  
};
```



[arch/arm/boot/dts/imx6ul.dtsi +1008](#)

```
i2c2: i2c@21a4000 {  
    #address-cells = <1>;  
    #size-cells = <0>;  
    compatible = "fsl,imx6ul-i2c", "fsl,imx21-i2c";  
    reg = <0x021a4000 0x4000>;  
    interrupts = <GIC_SPI 37 IRQ_TYPE_LEVEL_HIGH>;  
    clocks = <&clks IMX6UL_CLK_I2C2>;  
    status = "disabled";  
};
```

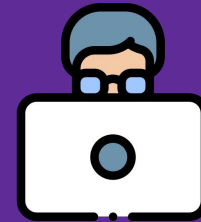
[arch/arm/boot/dts/imx6ull-colibri-eval-v3.dtsi +112](#)

```
&i2c1 {  
    status = "okay";  
  
    /* M41T0M6 real time clock on carrier board */  
    m41t0m6: rtc@68 {  
        compatible = "st,m41t0";  
        reg = <0x68>;  
    };  
  
    pcf8523: rtc@68 {  
        compatible = "nxp,pcf8523";  
        reg = <0x68>;  
    };  
};
```



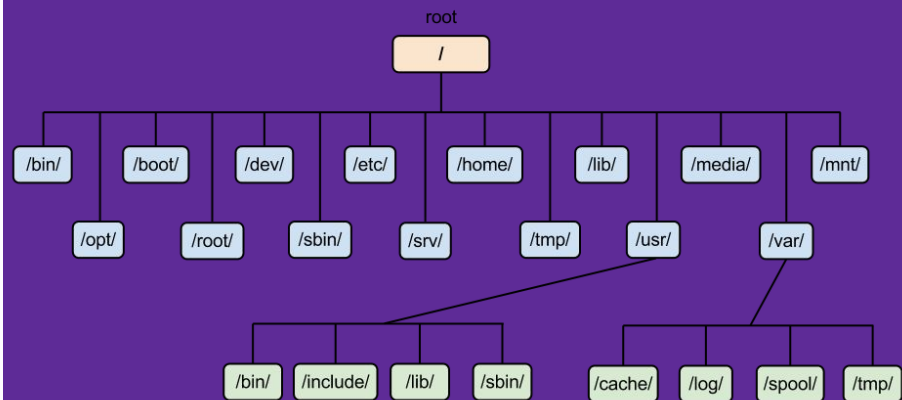
Laboratório

- ❖ Baixando, Configurando e compilando o Kernel para ARM





RootFS



- ❖ Os sistemas de arquivos são usados para organizar dados em diretórios e arquivos em dispositivos de armazenamento ou na rede. Os diretórios e arquivos são organizados como uma hierarquia
- ❖ Em sistemas UNIX, aplicativos e usuários veem uma única hierarquia global de arquivos e diretórios.
- ❖ Sistemas de arquivos são montados em um local específico nesta hierarquia de diretórios.



Root FileSystem

- ❖ Quando um sistema de arquivos é montado (comando **mount**) em um diretório (chamado ponto de montagem), o conteúdo desse diretório reflete o conteúdo do dispositivo de armazenamento
- ❖ Quando o sistema de arquivos é desmontado (comando **umount**), o ponto de montagem fica vazio novamente.
- ❖ Isso permite que os aplicativos acessem arquivos e diretórios facilmente, independentemente de seu local de armazenamento exato.



Comando mount e umount

- ❖ **mount** permite montar sistemas de arquivos

```
$ sudo mount -t ext4 /dev/sdb1 /mnt/backup
```

- **mount -t <type>** monta o dispositivo com o tipo especificado
- **/dev/sdb1** é o dispositivo de armazenamento (HD, SSD, PenDrive, MicroSD, ...)
- **/mnt/backup** é o diretório onde os arquivos do dispositivo de armazenamento serão montados
- Diversas combinações de parâmetros são permitidas com mount para diversas funcionalidades, [saiba mais sobre o mount](#)



Comando mount e umount

❖ **umount** desmonta sistemas de arquivos

```
$ sudo umount /dev/sdb1
```

- Irá desmontar o ponto de montagem do dispositivo /dev/sdb1 tornando o diretório vazio
- Isso é necessário antes de reinicializar ou antes de desconectar um pendrive, porque o kernel armazena em cache as gravações na memória para aumentar o desempenho.



Sistema de Arquivo Root

- ❖ Um sistema de arquivos específico é montado na raiz da hierarquia, identificado por `/`
- ❖ Este sistema de arquivos é chamado de sistema de arquivos raiz
- ❖ Como os programas de mount e umount estão dentro do Sistema de Arquivos Raiz eles não fazem esta montagem, ficando a função ao Kernel resolver!



Sistema de Arquivo Root

- ❖ Como o sistema de arquivos raiz é o primeiro sistema de arquivos montado, ele não pode ser montado com o comando normal mount
- ❖ Ele é montado diretamente pelo kernel, de acordo com a opção **root=** kernel
- ❖ Quando nenhum sistema de arquivos raiz está disponível, o kernel lança um Kernel Panic:



```
Please append a correct "root=" boot option
```

```
Kernel panic - not syncing: VFS: Unable to mount root fs on unknown block(0,0)
```



- ❖ Ele pode ser montado de diferentes locais
 - Da partição de um disco rígido
 - Da partição de uma pendrive
 - Da partição de um cartão SD
 - Da partição de uma flash NAND
 - Da rede , usando o protocolo NFS(Network File System)
- ❖ O desenvolvedor deverá escolher o dispositivo correto do sistema e configurar no parâmetro **root=** do Kernel



Sistema de Arquivo Root - Montando RootFS

❖ Disco Rígido ou Pendrive

```
root=/dev/sdXY
```

X Número Identificação do Dispositivo
Y Número de Partição

❖ MicroSD ou eMMC

```
root=/dev/mmcblkXY
```

X Número Identificação do Dispositivo
Y Número de Partição

❖ Flash NAND (exemplo com UBIFS)

```
ubi.mtd=3 root=ubi0:rootfs
```



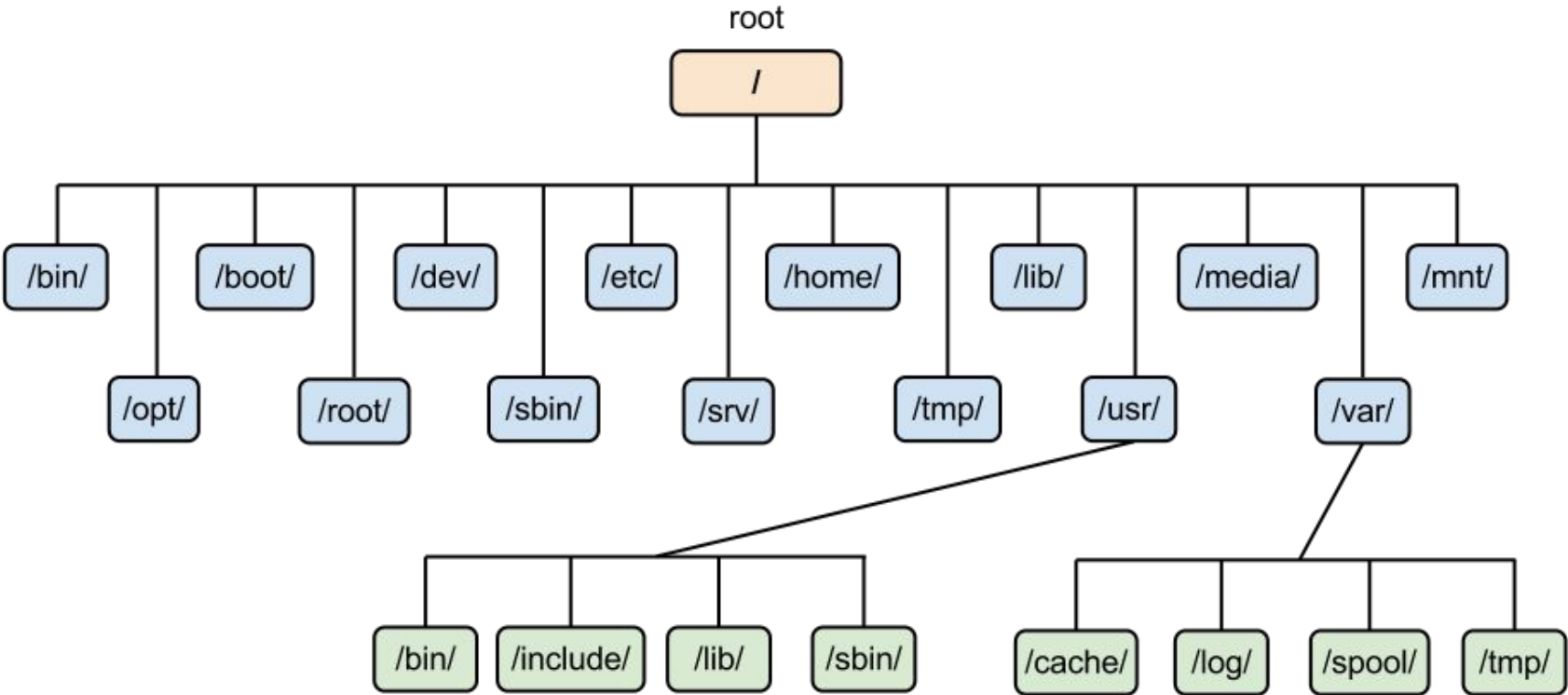
- ❖ Também é possível inicializar o sistema com um sistema de arquivos na memória: **initramfs**
 - A partir de um arquivo CPIO compactado integrado à imagem do kernel
 - No momento da inicialização, este arquivo é extraído para o cache do Linux
 - É útil para dois casos:
 - **FastBoot** para um sistema muito pequeno
 - **OTA(OverTheAir)** - Uma etapa intermediária antes de mudar para um sistema de arquivos raiz real



- ❖ A organização de um RootFS do Linux em termos de diretórios é bem definida pelo **FHS**(Filesystem Hierarchy Standard)
 - <https://wiki.linuxfoundation.org/lsb/fhs>
- ❖ A maioria dos sistemas Linux está em conformidade com esta especificação
- ❖ Os aplicativos esperam esta organização
- ❖ Isso torna mais fácil para desenvolvedores e usuários, pois a organização do sistema de arquivos é semelhante em todos os sistemas



Sistema de Arquivo Root - Organização



Sistema de Arquivo Root - Organização

/bin

Programas Básicos do Sistema

/sbin

Programas de Administração do Sistema

/boot

Arquivos de Bootloader, Kernel e Device-Tree

/etc

Arquivos de Configuração do Sistema

/home

Diretórios dos usuários comuns do Sistema

/root

Diretório do usuário root

/lib

Bibliotecas Essenciais do Sistema e os Módulos do Kernel



Sistema de Arquivo Root - Organização

/mnt

Diretório de Montagem dos Dispositivos

/proc

Diretório Virtual Informações Processos e Sistema - procfs

/sys

Diretório Virtual Controlado pelo Kernel e Módulos - sysfs

/tmp

Arquivos Temporários

/usr/bin
/usr/lib
/usr/sbin

Aplicações Básicas dos Usuários
Bibliotecas dos Usuários
Aplicações de Administração dos Usuários
Unix System Resources

/var

Arquivos Variáveis (Temporários, Logs, Banco de Dados, ...)



/dev

Onde todos os Dispositivos do Sistema são criados

- ❖ Uma função interessante do kernel é permitir que os programas aplicativos acessem dispositivos de hardware como arquivos
- ❖ Internamente, o kernel identifica cada dispositivo por:
 - Type (Dispositivo de Caracteres ou Dispositivo de Bloco)
 - Major Number (normalmente a categoria do dispositivo)
 - Minor Number (normalmente o identificador do dispositivo)



Sistema de Arquivo Root - Organização

/dev

Onde todos os Dispositivos do Sistema são criados

❖ Exemplo de Dispositivos em um Sistema Linux

```
$ ls -l /dev/ttyS0 /dev/tty1 /dev/sda /dev/sda1 /dev/sda2 /dev/sdc1 /dev/zero
brw-rw---- 1 root disk      8, 0  2011-05-27 08:56 /dev/sda
brw-rw---- 1 root disk      8, 1  2011-05-27 08:56 /dev/sda1
brw-rw---- 1 root disk      8, 2  2011-05-27 08:56 /dev/sda2
brw-rw---- 1 root disk      8, 32 2011-05-27 08:56 /dev/sdc
crw----- 1 root root       4, 1  2011-05-27 08:57 /dev/tty1
crw-rw---- 1 root dialout   4, 64 2011-05-27 08:56 /dev/ttyS0
crw-rw-rw- 1 root root       1, 5  2011-05-27 08:56 /dev/zero
```



Sistema de Arquivo Root - Organização

/dev

Onde todos os Dispositivos do Sistema são criados

- ❖ Exemplo de código C que usa a API de arquivo usual para gravar dados em uma porta serial

```
int fd;  
fd = open("/dev/ttyS0", O_RDWR);  
write(fd, "Hello", 5);  
close(fd);
```



Sistema de Arquivo Root - Criando Arquivos de Dispositivos

- ❖ Nas versões de Kernel < 2.6 e sistemas mais simples, os arquivos de dispositivo podem ser criados manualmente com o comando **mknod** (é necessário privilégios de root):

```
$ mknod /dev/<device> [c|b] MAJOR MINOR
```

- ❖ Para sistemas mais complexos, existem mecanismos para adicionar e remover arquivos de dispositivo dinamicamente:
 - udev
 - mdev
 - devtmpfs



Sistema de Arquivo Root - Criando Arquivos de Dispositivos

devtmpfs

O Sistema de Arquivos Virtual devtmpfs é montado em `/dev` e contém todos os dispositivos registrados nas estruturas do kernel. A opção de configuração do kernel `CONFIG_DEVTMPFS_MOUNT` faz com que o kernel o monte automaticamente no momento da inicialização.

mdev

Uma versão minimalista do udev, disponibilizada pelo **Busybox**, mais adequado para uso em sistemas embarcados. Pode ser executado em userspace e analisar `/sys` por novos dispositivos, necessário configurar `echo /sbin/mdev > /proc/sys/kernel/hotplug`

udev

O **udev** é iniciado, ele “escuta” uevents do Kernel, que são enviados sempre que dispositivos são inseridos ou removidos. O **udev** lê e analisa todas as regras encontradas em `/etc/udev/rules.d/` e cria/modifica/remove o dispositivo de acordo com as regras, API para integrar direto em aplicações. Utilitário **udevadm** para monitorar eventos e regras



Sistema de Arquivo Virtual - ProcFS

- ❖ O sistema de arquivos virtual **procfs** existe desde o início do Linux
- ❖ Ele permite:
 - O kernel expor estatísticas sobre os processos em execução no sistema
 - O usuário pode ajustar em tempo de execução diversos parâmetros do sistema, gerenciamento de processos, gerenciamento de memória: Cada programa em execução possui uma entrada em **/proc/<PID>**
- ❖ Programas como **ps** e **top** não funciona sem o sistema de arquivos **procfs**
- ❖ Comando para montar /proc: **mount -t proc nodev /proc**
- ❖ Mais detalhes do procfs em [Documentação ProcFS](#) ou **man proc**



Sistema de Arquivo Virtual - ProcFS

❖ Alguns exemplos e informações úteis do **procfs**:

`/proc/cmdline`

Informações dos parâmetros passado para o Kernel no Boot

`/proc/devices`

Informações de todos os dispositivos reconhecidos pelo Kernel durante o boot

`/proc/interrupts`

Relaciona o número de interrupções por CPU por dispositivo de I/O, ele exibe o número de IRQ

`/proc/filesystems`

Todos os sistemas de arquivos atualmente suportados pelo kernel

`/proc/<PID>/*`

Diversas informações sobre o processo em execução, baseado no ID do Processo <PID>

`/proc/uptime`

Tempo em que o sistema está ligado



Sistema de Arquivo Virtual - SysFS

- ❖ Permite representar no userspace as informações que o kernel tem dos barramentos, dispositivos e drivers do sistema.
- ❖ É útil para vários aplicativos de userspace que precisam listar e consultar o hardware disponível, por exemplo `udev` ou `mdev`.
- ❖ Todos os aplicativos que usam `sysfs` esperam que ele seja montado no diretório `/sys`.
- ❖ Comando para montar `/sys`: `mount -t sysfs nodev /sys`



Sistema de Arquivo Virtual - SysFS

❖ Alguns exemplos e informações úteis do **sysfs**:

/sys/block

Contém subdiretórios para cada sistema de arquivos de bloco (discos rígidos, etc) com diversas informações.

/sys/bus

Contém subdiretórios para cada tipo de barramento suportado pelo Kernel e se expande em diretórios de devices e drivers.

/sys/class

Contém representações de cada classe de dispositivo que está registrado no Kernel: **pwm, gpio, net, i2c, etc.**

/sys/kernel

Contém vários arquivos e subdiretórios que fornecem informações sobre o kernel em execução.

/sys/module

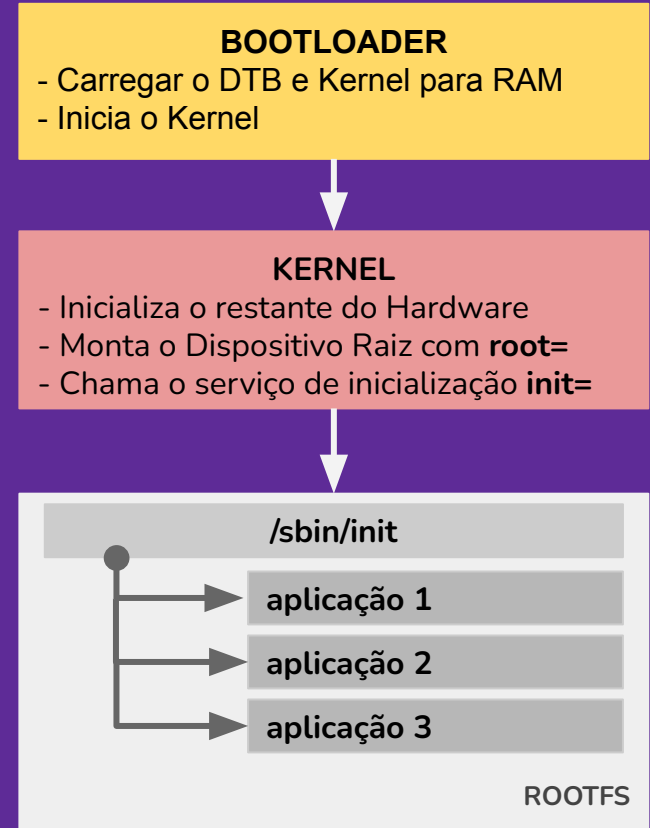
Contém subdiretórios com todos os módulos carregador do Kernel em memória, permite configurações e obter detalhes.

/sys/fs

Contém subdiretórios para alguns sistemas de arquivos do SO, permite obter diversas informações.

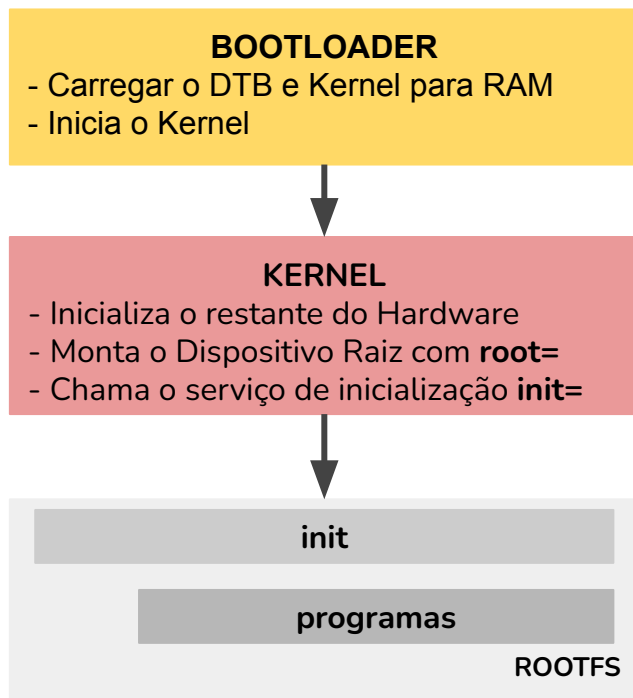


Inicialização Básica no Linux



Inicialização Linux - **init**

- ❖ Assim que o Kernel monta o dispositivo especificado em **root=**, é executado o processo de inicialização **init**.
- ❖ O init é pesquisado na seguinte ordem:
 - /sbin/init
 - /bin/init
 - /etc/init
 - /bin/sh
- ❖ Pode ser especificado manualmente para o kernel com: **init=/bin/startup.sh**

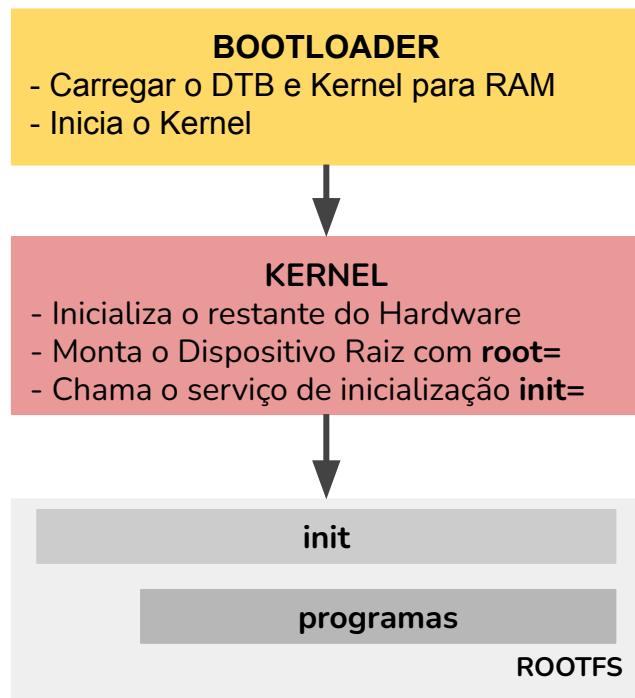


Inicialização Linux - **init**

- ❖ Caso nenhuma das alternativas de init seja encontrada a seguinte mensagem de pânico é lançada:

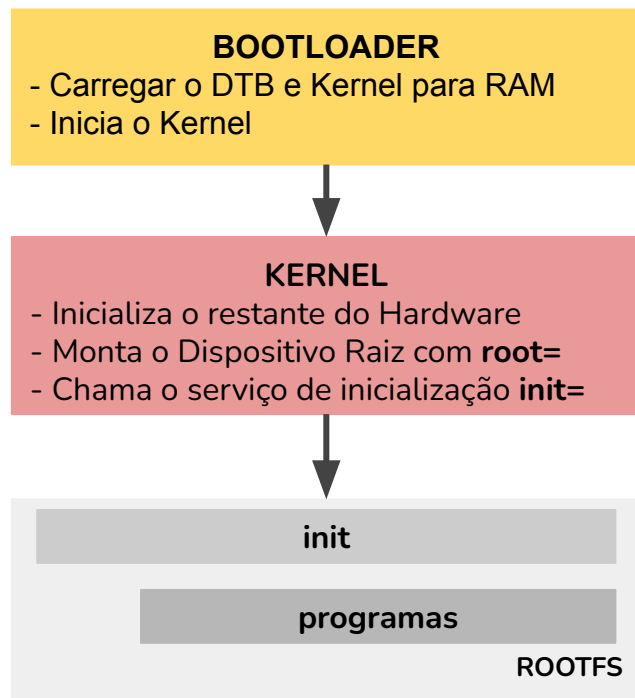
Kernel panic - not syncing: No init found. Try passing init= opt on to kernel

- ❖ Se executado com sucesso, é dado início a inicialização de diversos serviços em sequência ou paralelo (Rede, Aplicações, Servidor Gráfico, ...), responsabilidade está dos **Mecanismos de Inicialização**



Inicialização Linux - Mecanismos de Inicialização

- ❖ **systemd** - Desenvolvido por Engenheiros da RedHat em meados de 2009, logo adotado pelo Ubuntu e demais Distribuições, vale destacar inicialização paralela e diversas ferramentas
- ❖ **sysvinit** - O mais simples e prático, puramente baseado em Script Shell, carrega as configurações em **/etc/inittab** e inicializa os script em **/etc/init.d**



Inicialização Linux - **sysvinit**

❖ /etc/inittab

```
root@b2 # cat /etc/inittab
# When the system starts
::sysinit:/etc/init.d/rcS

# When pressing Ctrl+Alt+Del
::ctrlaltdel:/sbin/reboot

::respawn:-/bin/sh

# When the system is shut down
::shutdown:/bin/umount -a -r

# When the system restarts
::restart:/sbin/init
```



Inicialização Linux - **sysvinit**

❖ /etc/init.d/rcS

```
root@b2 # cat /etc/init.d/rcS
#!/bin/sh

/bin/mount -a

/bin/mkdir -p /dev/pts
/bin/mount -t devpts devpts /dev/pts
/bin/mount -t sysfs sysfs /sys

/bin/echo /sbin/mdev > /proc/sys/kernel/hotplug
/sbin/mdev -s

/bin/mount -o remount, rw /
```



Inicialização Linux - **sysvinit**

❖ /etc/init.d/rcS

```
root@b2 # cat /etc/init.d/rcS
...
for i in /etc/rc$runlevel.d/S*
do
    case "$i" in
        *.sh)
            . $scriptname
            ;;
        *)
            "$@"
            ;;
    esac
done
```



Inicialização Linux - SystemD

❖ /etc/systemd/system/aplicacaoGPS.service

```
root@b2 # cat /etc/systemd/system/aplicacaoGPS.service
```

```
[Unit]
```

```
Description=Aplicacao Demo GPS
```

```
After=multi-user.target
```

```
[Service]
```

```
Type=simple
```

```
Environment=DEBUG=0
```

```
ExecStart=/usr/bin/aplicacaoGPS
```

```
[Install]
```

```
WantedBy=multi-user.target
```



Inicialização Linux - **SystemD**

- ❖ Iniciando o serviço **aplicacaoGPS.service**

```
root@b2 # systemctl start aplicacaoGPS
```

- ❖ Parando o serviço **aplicacaoGPS.service**

```
root@b2 # systemctl stop aplicacaoGPS
```

- ❖ Status do serviço **aplicacaoGPS.service**

```
root@b2 # systemctl status aplicacaoGPS
```



Inicialização Linux - **SystemD**

- ❖ Desabilitando o serviço **aplicacaoGPS.service**

```
root@b2 # systemctl disable aplicacaoGPS
```

- ❖ Habilitando o serviço **aplicacaoGPS.service**

```
root@b2 # systemctl enable aplicacaoGPS
```

- ❖ Verifica toda configuração do serviço **aplicacaoGPS.service**

```
root@b2 # systemctl show aplicacaoGPS
```



Inicialização Linux - SystemD



Obtendo a configuração e localização do serviço:

```
root@b2 # systemctl cat aplicacaoGPS
```



BusyBox

O básico para rootfs em um único binário!





Por que BusyBox?

- ❖ Um sistema Linux precisa de poucos programas para funcionar:
 - O programa init
 - Um Interpretador Shell
 - Alguns utilitários básicos para manipulação de arquivos e configuração do sistema(ps, top, cat, echo, grep, ...)
- ❖ Em sistemas GNU/Linux normais, esses programas são fornecidos por projetos diferentes: coreutils, bash, grep, etc.
 - Muitos componentes diferentes para integrar
 - Componentes não projetados com restrições de sistemas embarcados em mente



Por que BusyBox?

- ❖ **BusyBox** é uma solução alternativa, extremamente comum em sistemas embarcados, e facilita **todos projetos em um único binário!**
- ❖ Completamente configurável os projetos que devem compor
- ❖ Licença: GNU GPLv2(mesma do Kernel Linux)
- ❖ Conhecido como *Swiss Army Knife of Embedded Linux*
- ❖ Busybox: <https://www.busybox.net>
- ❖ Alternativa: [Toybox](#)(Android >6) - Licença BSD



Por que BusyBox?

- ❖ Todos os utilitários são compilados em um único executável,

/bin/busybox

- Links simbólicos para **/bin/busybox** são criados para cada aplicativo integrado ao BusyBox

- ❖ Uma excelente ferramenta para Sistemas Embarcados:

/bin/busybox (compilado com linkagem estática) - 1.2M



Por que BusyBox?

```
_install/
├── bin
│   ├── ash -> busybox
│   ├── busybox
│   ├── cat -> busybox
│   ├── date -> busybox
│   └── ps -> busybox
├── sbin
│   ├── init -> ../bin/busybox
│   ├── poweroff -> ../bin/busybox
│   └── reboot -> ../bin/busybox
└── usr
    ├── bin
    │   ├── top -> ../../bin/busybox
    │   ├── uptime -> ../../bin/busybox
    │   └── wget -> ../../bin/busybox
    └── sbin
        └── udhcpd -> ../../bin/busybox
```



Por que BusyBox?

```
root@tdxBoard # busybox
```

```
BusyBox v1.36.1 (2024-01-13 15:57:21 -03) multi-call binary.
```

Currently defined functions:

[, [[, adduser, arp, ash, awk, base64, basename, bc, blkid, blockdev, cat, chattr, chgrp, chmod, chown, chroot, chvt, clear, cp, crond, crontab, cut, date, dd, deluser, depmod, devmem, df, dmesg, du, echo, env, envdir, false, fdisk, find, free, fsck, fsck.minix, getty, grep, groups, gzip, halt, head, hexdump, hostname, httpd, hwclock, id, ifconfig, ifdown, ifup, inetd, init, insmod, kill, killall, klogd, last, linux64, linuxrc, ln, logger, login, ls, lsattr, lsmod, lsof, lsusb, makedevs, man, md5sum, mdev, mesg, mkdir, mke2fs, mkfifo, mkfs.ext2, mkfs.minix, mkfs.vfat, mknod, modinfo, modprobe, more, mount, mv, nc, netstat, nice, nohup, nologin, nproc, ntpd, od, openvt, passwd, patch, pidof, ping, pkill, pmap, poweroff, printf, ps, pscan, pstree, pwd, reboot, renice, reset, rm, route, run-init, runlevel, sed, seq, setserial, sh, sha256sum, sleep, sort, start-stop-daemon, stat, strings, stty, su, sulogin, swapoff, swapon, switch_root, sync, sysctl, syslogd, tail, tar, taskset, tcpsvd, telnet, telnetd, tftp, tftpd, time, top, touch, traceroute, tree, true, tty, ttysize, udhcpc, udhcpd, uevent, umount, uname, unzip, uptime, users, vi, w, wc, wget, who, whoami, xz, yes, zcat



Por que BusyBox?

- ❖ Configurando: **make menuconfig**

```
BusyBox 1.33.1 Configuration

BusyBox Configuration
Arrow keys navigate the menu. <Enter> selects submenus --->. Highlighted
letters are hotkeys. Pressing <Y> includes, <N> excludes, <M> modularizes
features. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*]
built-in [ ] excluded <M> module < > module capable

  Settings --->
--- Applets
  Archival Utilities --->
  Coreutils --->
  Console Utilities --->
  Debian Utilities --->
  klibc-utils --->
  Editors --->
  Finding Utilities --->
  Init Utilities --->
  Login/Password Management Utilities --->

!(+)
```

<Select> < Exit > < Help >



Por que BusyBox?

❖ Compilando:

```
root@b2 # make ARCH=arm CROSS_COMPILE=arm-unknown-linux-uclibcgnueabi-
```

❖ Instalando:

```
root@b2 # make ARCH=arm CROSS_COMPILE=arm-unknown-linux-uclibcgnueabi-  
install
```

- Um diretório chamado `_install` será criado e utilizado para instalar o binário e os links-simbólicos.



BusyBox Init



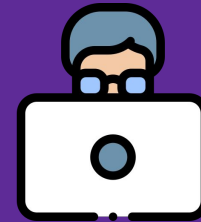


- ❖ BusyBox fornece uma implementação de um programa init
- ❖ Mais simples do que a implementação init encontrada em sistemas como (SysVinit ou systemd)
- ❖ Um único arquivo de configuração: **/etc/inittab**
 - Cada linha tem o formato **<id> :: <action>: <process>**
 - BusyBox init não suporta runlevels, se você precisa de runlevels, use sysvinit
- ❖ Permite executar serviços na inicialização e garantir que determinados serviços estejam sempre em execução no sistema
- ❖ Exemplo de [inittab](#) - [Bootlin inittab](#)



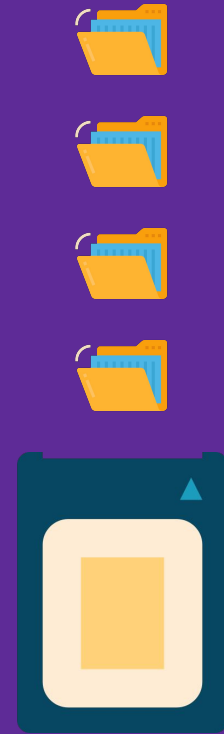
Laboratório

- ❖ Baixando, Configurando e compilando o BusyBox e criando o próprio RootFS





Unidade de Armazenamento e Sistemas de Arquivos





Unidades de Armazenamento

- ❖ Os dispositivos de armazenamento são classificados em dois tipos principais: **Dispositivos de Bloco** e **Dispositivos Flash**
 - Eles são tratados por diferentes subsistemas e diferentes sistemas de arquivos
- ❖ **Dispositivos de Bloco** - Podem ser lidos e gravados em uma base por bloco, em ordem aleatória, sem apagar.
 - Discos rígidos
 - Pendrive, SSD, cartões SD, eMMC: Baseiam-se no armazenamento Flash, mas possuem um controlador integrado que emula um dispositivo de bloco..
- ❖ **Dispositivos Flash RAW** - São gerenciados por um controlador no SoC. Eles podem ser lidos, mas a escrita requer limpar(apagar).
 - Flash NOR, flash NAND



Listagem de Dispositivos e Partições

- ❖ A lista de todos os dispositivos de bloco disponíveis no sistema pode ser encontrada em **/proc/partitions**

```
root@b2 # cat /proc/partitions
major      minor      #blocks      name
  8          0      976762584    sda
  8          1      48827392     sda1
  8          2     244141056    sda2
  8          3     195312640    sda3
  8          4     488479744    sda4
179          0       7761920     mmcblk0
179          1        85196     mmcblk0p1
179          2     3670016     mmcblk0p2
```

- ❖ **/sys/block/** também armazena informações sobre cada dispositivo de bloco, por exemplo, se é um armazenamento removível ou não.



Particionamento

- ❖ Dispositivos de bloco podem ser particionados para armazenar diferentes partes de um sistema, exemplo:
 - `/dev/mmcblk0` - O dispositivo
 - `/dev/mmcblk0p1` - Primeira Partição do Dispositivo - **Bootloader e Kernel**
 - `/dev/mmcblk0p2` - Segunda Partição do Dispositivo - **RootFS**
 - `/dev/mmcblk0p3` - Terceira Partição do Dispositivo - **Banco de Dados/Configurações**
- ❖ Algumas ferramentas para criar e modificar as partições em um dispositivo de bloco: `fdisk`, `cfdisk`, `sfdisk`, `parted`, `gparted`, etc.



Transferindo Dados

- ❖ Para transferir dados para um Dispositivo de Bloco requer um procedimento antes chamado de montar o dispositivo, comando **mount**
- ❖ Com o comando mount o Dispositivo(**/dev/mmcblk0p2**) passa ser acessando como, por exemplo (**/mnt/rootfs**)

```
root@b2 # mkdir /mnt/rootfs
root@b2 # mount -t ext4 /dev/mmcblk0p2 /mnt/rootfs
```

- ❖ Após o ponto de montagem pode ser utilizada ferramentas como cp e mv para transferir arquivos para o Dispositivo
- ❖ Após seu uso, utilizar **umount** para desmontar o Dispositivo



Transferindo Dados

- ❖ Ao executar somente o comando `mount`, irá exibir todos pontos de montagem do sistema

```
root@b2 # mount
```

```
...
```

```
sysfs on /sys type sysfs (rw,nosuid,nodev,noexec,relatime)
```

```
proc on /proc type proc (rw,nosuid,nodev,noexec,relatime)
```

```
udev on /dev type devtmpfs (rw,nosuid,relatime,size=4016184k,mode=755)
```

```
tmpfs on /run type tmpfs (rw,nosuid,noexec,relatime,size=807844k,mode=755)
```

```
tmpfs on /run type tmpfs (rw,nosuid,relatime,size=262144k,mode=755)
```

```
/dev/sda1 on / type ext4 (rw,relatime,errors=remount-ro,data=ordered)
```

```
/dev/mmcblk0p2 on /mnt/rootfs type ext4 (rw,relatime, data=ordered)
```

```
/dev/mmcblk0p1 on /mnt/boot type vfat (rw,relatime,errors=remount-ro)
```

```
...
```



Transferindo Dados

- ❖ Muitas vezes é necessário transferir dados de ou para um dispositivo de bloco de forma RAW
 - Especialmente para gravar uma imagem do sistema de arquivos(com partições) em um dispositivo de bloco
- ❖ Os dispositivos de bloco em `/dev/` permitem tal acesso bruto
- ❖ Uma das principais e mais conhecida ferramenta é o `dd` (*disk duplicate*):

```
root@b2 # dd if=/dev/mmcbk0p1 of=test.raw bs=1M count=32
```

- Copia os primeiros 32 blocos de 1 MB de `/dev/mmcbk0p1` para o arquivo de `test.raw`

```
root@b2 # dd if=test.raw of=/dev/sda2 bs=1M seek=4
```

- Copia o conteúdo completo do `test.raw` para `/dev/sda2`, por blocos de 1 MB, mas começando com deslocamento de 4MB em `/dev/sda2`



Transferindo Dados

- ❖ Exemplo com um arquivo de imagem:

```
$ fdisk -l raspios-buster-armhf-lite.img
Disk raspios-buster-armhf-lite.img: 1,8 GiB, 1874853888 bytes, Sector size
...

Dispositivo          Inicializar   Start        Fim Setores  Size  ID   Tipo
raspios-buster-armhf-lite.img1  8192         532479       524288      256M  c    W95 FAT32 (LBA)
raspios-buster-armhf-lite.img2  532480       3661823      3129344     1,5G  83   Linux

$ sudo dd if=raspios-buster-armhf-lite.img of=/dev/mmcblk0 bs=1M
```

- ❖ bmap-tools é uma alternativa ao dd mais eficiente e com mais recursos



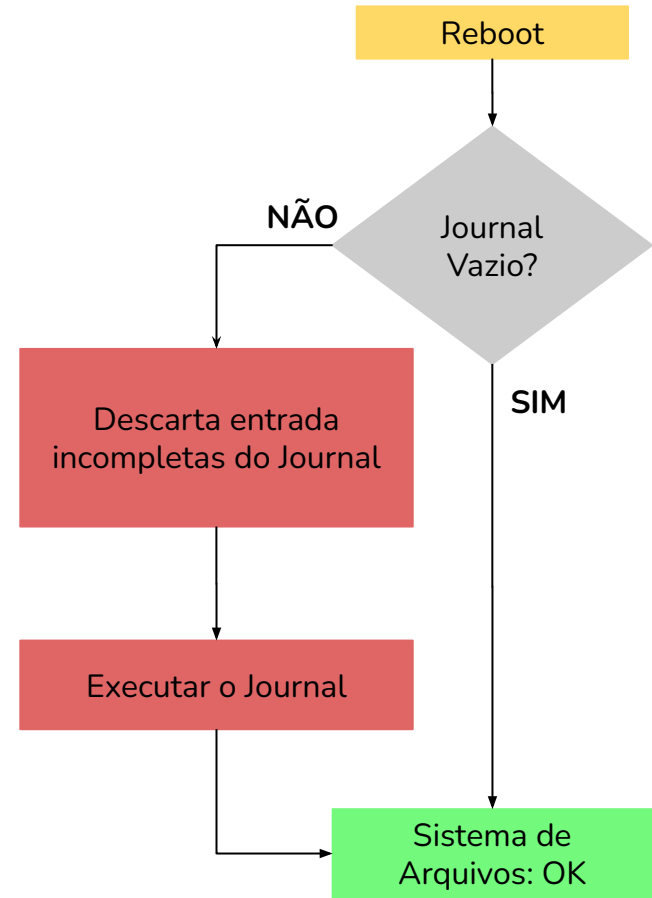
- ❖ O sistema de arquivos padrão usado em Sistemas Linux é:
 - ext2 (utilizado por mais de 10 anos, ainda em uso em Sistemas Embarcados)
 - ext3 (trouxe o journaling em comparação com o ext2, agora obsoleto com o ext4)
 - ext4 (melhorias de desempenho e suporte para partições muito grandes)
- ❖ Suporta todos os recursos que o Linux precisa em um sistema de arquivos raiz: permissões, propriedade, arquivos de dispositivo, links simbólicos, etc.



Sistemas de Arquivos - Journaling

❖ Alguns Sistemas de Arquivos com suporte a journal:

- ext3/ext4
- btrfs
- XFS
- ReiserFS
- ZFS



Sistemas de Arquivos com Compressão

- ❖ Há casos que um Sistema de Arquivos com compressão seja requisitado, por questão de espaço por exemplo:
 - SquashFS
 - EROFS
 - tmpFS



Sistemas de Arquivos com Compressão - SquashFS

- ❖ Sistema de Arquivos compactado somente leitura(read-only) para dispositivos de bloco. Bom para partes de um sistema de arquivos que podem ser somente leitura (kernel, binários ...)
- ❖ Ótima taxa de compressão, o que geralmente traz um melhor desempenho de leitura
- ❖ Suporta vários algoritmos de compressão (LZO, XZ, etc.)
- ❖ Benchmarks:
 - https://elinux.org/Squash_Fs_Comparisons
- ❖ Mais informações em [SquashFS 4.0 FileSystem](#)



Sistemas de Arquivos com Compressão - **EROFS**

- ❖ Uma futura alternativa ao SquashFS
- ❖ Desenvolvido pela Huawei - **EROFS** (Enhanced Read-Only File System)
- ❖ Huawei introduziu em todos os seus produtos com Android[1]
- ❖ Introduzido no Kernel Linux 5.4
- ❖ Algoritmo de Compressão LZ4
- ❖ Tamanho Máximo de Arquivo Comprimido 4GB
- ❖ Mais informações em [Enhanced Read-Only File System - EROFS](#)



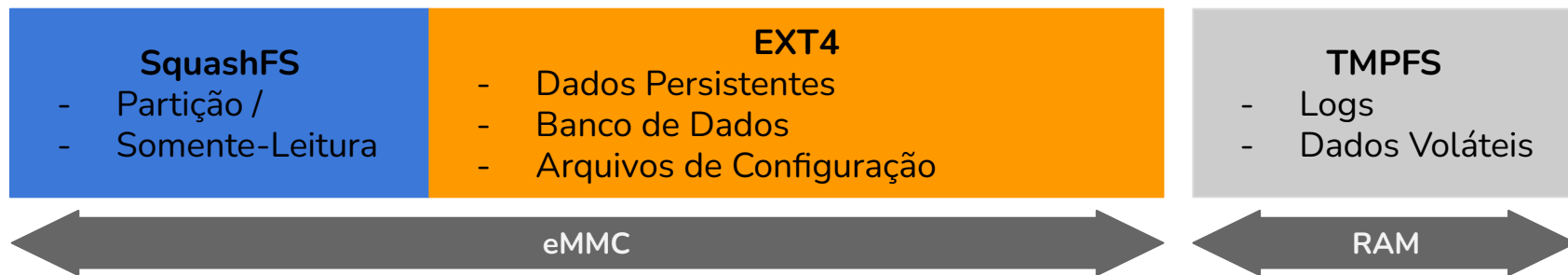
Sistemas de Arquivos com Compressão - TMPFS

- ❖ Não é um sistema de arquivos em bloco, é claro!
- ❖ Perfeito para armazenar dados temporários na RAM: arquivos de log do sistema, dados de conexão, arquivos temporários ...
- ❖ Mais eficiente em termos de espaço do que discos, os arquivos estão diretamente no cache de arquivos, aumenta e diminui dinamicamente
- ❖ Exemplos:
 - `mount -t tmpfs run /var/run`
 - `mount -t tmpfs shm /dev/shm`
- ❖ Mais informações em [TmpFS](#)



Sistemas de Arquivos - Exemplo em Projeto

- ❖ É aconselhável e boa prática e mistura de Sistemas de Arquivos



Memória Flash

- ❖ Flash NAND
 - SLC
 - MLC
- ❖ MTD
- ❖ UBI/UBIFS



- ❖ Permite acesso aleatório aos dados
- ❖ As células Flash NAND são categorizadas dependendo do número de bits que podem armazenar:
 - **SLC(Single Level Cell)** - Pode armazenar um bit por célula, mais rápido, tempo de vida maior, o mais caro de todos, cerca de 100 mil ciclos.
 - **MLC(Multi Layer Cell)** - Pode armazenar dois bits por célula, mais estável que TLC, menos durável e estável como que SLC, de 3 mil a 15 mil ciclos.
 - **TLC(Triple Layer Cell)** - Pode armazenar três bits por célula, custo de produção mais barato, processo de escrita mais lento, cerca de 500 a 1000 ciclos de gravação.



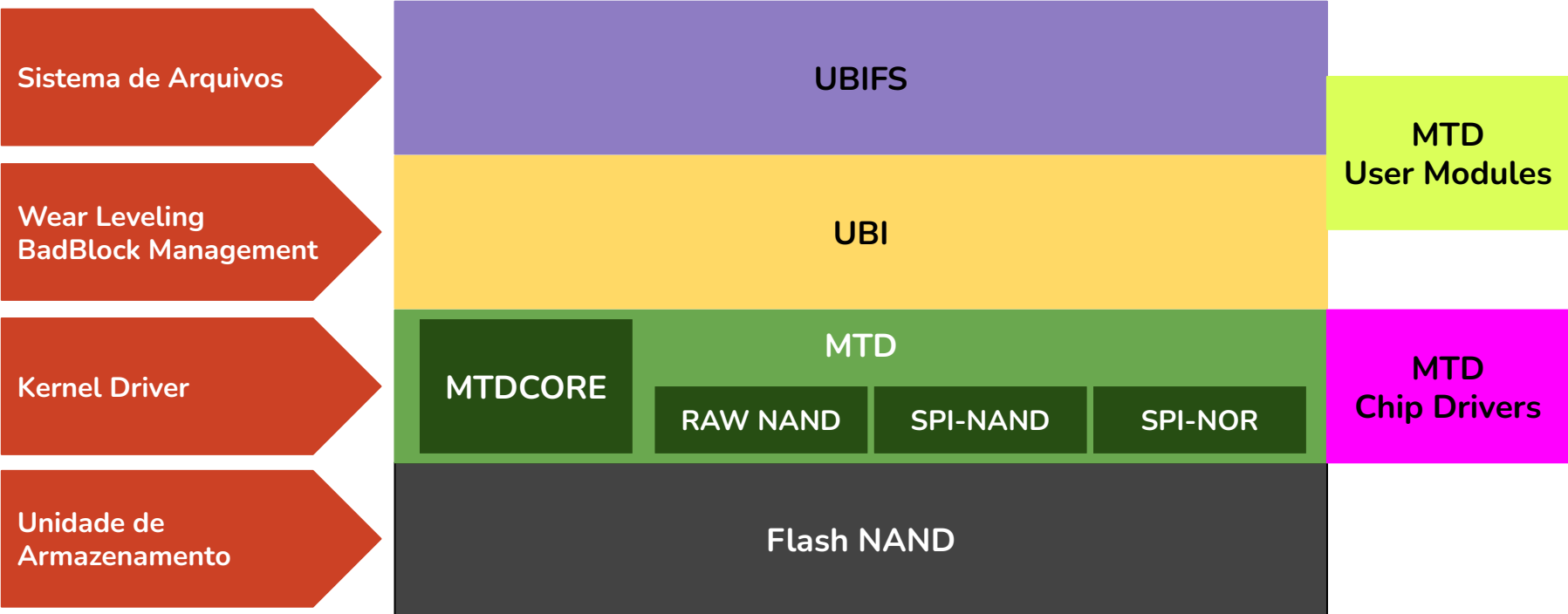
- ❖ Vida útil curta em comparação com outras formas de armazenamento
- ❖ A vida útil depende da tecnologia Flash NAND (SLC, MLC): entre 1.000.000 e 1.000 ciclos de apagamento por bloco
- ❖ **Wear Leveling**(Mecanismos de nivelamento de desgaste) são necessários para apagar blocos uniformemente
- ❖ **Bad Block Management**(Detecção/Gerenciamento de blocos defeituosos)
- ❖ 1 Ciclo para leitura e 2 Ciclos para escrita(apagar e escrever)



- ❖ MTD significa Dispositivos de Tecnologia de Memória
- ❖ Subsistema genérico no Linux lidando com todos os tipos de dispositivos de armazenamento que não se sejam de Subsistema de Bloco
- ❖ Tipos de dispositivos suportados: RAM, ROM, Flash NOR, Flash NAND
- ❖ Características da mídia de armazenamento abstrata e fornece uma API simples para acessar dispositivos MTD



Memória Flash NAND - Subsistema MTD



- ❖ O driver **mtdchar** cria um dispositivo de caractere para cada device/partição MTD do sistema
 - /dev/mtdX
 - /dev/mtdXro
 - Permite comando ioctl() para apagar e manipular a Flash
- ❖ O driver **mtdblock** implementa o módulo "block device" da flash.
 - Normalmente /dev/mtdblockXY
 - Permite acesso de leitura/gravação igual em disco, no entanto, não faz BadBlock Management e Wear Leveling
 - Para isso utiliza um Sistema de Arquivos que faça isso: JFFS2, YAFFS2 e UBIFS



- ❖ Suporta compressão em tempo real
- ❖ O Sistema de Arquivos Flash NAND mais antigo e ainda em uso
- ❖ Suporte Wear Leveling, ECC, o tempo de inicialização depende do tamanho do sistema de arquivos: não escala para partições grandes porque precisa verificar todo o armazenamento no momento da inicialização. É necessário ativar **CONFIG_JFFS2_SUMMARY** para resolver este problema.
- ❖ Mais informações em [JFFS2](#)



Memória Flash NAND - YAFFS2

- ❖ Suporta principalmente flash NAND
- ❖ Sem compressão
- ❖ Suporte a Wear Leveling e resistente a falha de energia
- ❖ Tempo de inicialização rápido
- ❖ Não faz parte do kernel oficial do Linux
- ❖ Mais informações em [YAFFS2](#)



- ❖ UBI - Unsorted Block Images
- ❖ Destina-se a substituir JFFS2, com diversas melhorias
- ❖ Camadas de Wear Leveling e Sistemas de Arquivos separadas, Wear Leveling opera em todo armazenamento e não por partição
- ❖ Foco em escalabilidade, desempenho e confiabilidade
- ❖ Desvantagem: Introduce uma sobrecarga de espaço perceptível, especialmente quando usado em pequenos dispositivos ou partições, o JFFS2 ainda faz sentido em pequenas partições MTD.
- ❖ Os volumes podem ser redimensionados como dinâmico ou estático(ro)
- ❖ Mais informações em [UBI](#)

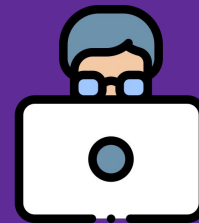


- ❖ Ao se trabalhar com memórias Flash também se deve levar em consideração a vida útil limitada dos dispositivos flash, tomando precauções adicionais:
 - Não use seu armazenamento flash como swap(swapfile)
 - Monte seus sistemas de arquivos como somente leitura(read-only) sempre que possível, ou utiliza um Sistema de Arquivos como SquashFS
 - Mantenha os arquivos voláteis na RAM (tmpfs)
 - Não use a opção **sync** no ponto de montagem (confirma as gravações imediatamente). Use a chamada de sistema **fsync()** para sincronização por arquivo



Laboratório

- ❖ Criando microSD da Instalação com TEZI
- ❖ Prompt Bootloader
- ❖ Primeiro boot!





Desenvolvendo Aplicações em Linux Embarcado

Componentes Open-Source

- ❖ Linguagens Interpretadas
- ❖ WebServers
- ❖ Banco de Dados
- ❖ Bibliotecas Gráficas
- ❖ Bibliotecas Multimédia
- ❖ Utilitários para o Sistema



Escolhendo componentes Open-Source

- ❖ Procure por projetos ativo e com diversos colaboradores
- ❖ Se atente sobre a LICENÇA do projeto - **Atenção com *GPLv3**
- ❖ Analise os releases do projeto e periodicidade de atualizações
- ❖ Estude sobre a qualidade do projeto, plataformas suportadas, sistemas de construção, se possui Testes Unitários, etc



Linguagens Interpretadas

- ❖ Suporte das principais linguagens interpretadas:
 - Shell Script
 - Python
 - Perl
 - Ruby
 - PHP
 - Lua
- ❖ Além de Linguagens Compiladas e Modernas como:
 - C/C++
 - Java
 - Go
 - Rust



- ❖ Alguns dos principais WebServers:
 - Busybox http
 - Apache2
 - Lighttpd
 - Hiawatha
 - Nginx ([lowis](#) versão otimizada para Sistemas Embarcados)
 - Monkey
 - Cherokee
 - Nodejs
 - Python3 http.server
 - Python Flask



- ❖ Alguns dos principais Banco de Dados:
 - SQLite - O mais popular e indicado em Sistemas Embarcados
 - PostgreSQL
 - hsqldb
 - mariadb



- ❖ Para a maioria das aplicações gráficas funcionarem é necessário um “Servidor Gráfico”, digamos que seriam Bibliotecas Gráficas mais baixo nível:
 - Xorg(Kdrive)
 - Wayland



- ❖ Para a maioria das aplicações gráficas funcionarem é necessário um “Servidor Gráfico”, digamos que seriam Bibliotecas Gráficas mais baixo nível:

- Xorg(Kdrive)

- Versão simplificada do servidor X, para sistemas embarcados
- Funciona em cima do Linux framebuffer, graças à variante Xfbdev
- Permite usar qualquer aplicativo ou biblioteca X11 existente
- Licença X11



- ❖ Para a maioria das aplicações gráficas funcionarem é necessário um “Servidor Gráfico”, digamos que seriam Bibliotecas Gráficas mais baixo nível:

- Wayland

- Uma substituição mais simples para o X
- Wayland é um protocolo para um compositor falar com seus clientes.
- Weston: uma implementação de referência mínima e rápida de um compositor Wayland, e é adequado para muitos casos de Sistemas Embarcados



Bibliotecas Gráficas - **Framework Qt**

- ❖ Um dos mais famosos frameworks gráficos para Sistemas Embarcados
- ❖ Multiplataforma: Windows, Linux, Android, MacOS, entre outros
- ❖ Implementado em C++
- ❖ Binding Oficial de Python e de outras linguagens mantido pela comunidade
- ❖ **Qt Widgets**(Desktop Style) e **Qt QML** (QtQuick - Aceleração Grafica)
- ❖ QtCreator IDE Oficial
- ❖ Qt Platform Abstract
 - **XCB**
 - **Wayland**/Qt Wayland
 - **EGLFS** - Sem Servidor Gráfico e com suporte a aceleração gráfica
 - **LinuxFB** - Sem Servidor Gráfico e sem necessidade de GPU



❖ Licença Dual: Comercial e OpenSource



- ❖ LVGL - Light and Versatile Graphics Library
- ❖ Baixo consumo de Memória, surgiu inicialmente para MCU
- ❖ Suporte TouchScreen e Mouse
- ❖ Plataforma: Xorg e FrameBuffer
- ❖ Mais de 30 controles para ser utilizados
- ❖ Linguagem C
- ❖ Licença MIT
- ❖ <https://lvgl.io/>



- ❖ FLTK - Fast Light Toolkit
- ❖ Multiplataforma Plataforma: Linux(Xorg), Windows e MacOS
- ❖ Linguagem C++
- ❖ Licença LGPLv2.1
- ❖ <https://www.fltk.org/>



- ❖ Google Flutter
- ❖ É o SDK do Google para criar experiências de usuário bonitas e rápidas para dispositivos móveis, web e desktop
- ❖ Suporte para Embarcados
 - <https://github.com/sony/flutter-embedded-linux>
- ❖ Linguagem **Dart**
- ❖ Licença BSD-3
- ❖ <https://flutter.dev/>



Flutter



Bibliotecas Gráficas - Kivy

- ❖ Framework Kivy
- ❖ Suporte X11 e Wayland, além de SDL2 e OpenGL
- ❖ Multiplataforma Windows, Linux, Android, MacOS
- ❖ Linguagem **Python**
- ❖ Licença MIT
- ❖ <https://kivy.org/>
- ❖ <https://github.com/kivy/kivy>



- ❖ **Gstreamer** - Um framework para multimídia
 - Suporte encode/decode de diversos codecs
 - Diversas empresas adicional suporte de seus produtos com Gstreamer, ex: Câmeras
- ❖ **ALSA** - Principal biblioteca de manipulação de áudio do Linux, utiliza o subsistema ALSA do Kernel
- ❖ **Pulseaudio** - Um servidor de áudio com diversos plugins
- ❖ **libavcodec** - Projeto de FFMpeg para encode/decode de áudio e vídeo



❖ Rede

- openSSH - Acesso Remoto Seguro via Shell)
- dropbear - SSH mais leve
- dnsmasq - DNS e DHCP
- networkmanager - Gestão de Interfaces e Conexões
- iptables - Linux Firmware e Netfilter
- iw - Ferramenta para gerenciamento de Wireless
- vsftpd e proftpd - Servidores FTP
- openvpn - Solução para VPN segura
- net-tools - Ferramentas como arp, ifconfig, route, hostname, netstat, etc



❖ Sistema

- gpsd - Serviço para processar dados de GPS e entregar em JSON
- dbus - IPC utilizado pelo Linux e diversas aplicações como systemd
- libusb - Biblioteca completa para acessar diversas informações de dispositivos USB via userspace
- libsoc - Biblioteca que abstrai o acesso e uso de GPIO, I2C, SPI e PWM em C e Python*
- i2c-tools - Manipulação de I2C via userspace
- can-utils - Ferramentas para configuração e enviar/receber dados da rede CAN(depends de SocketCAN)
- libgpiod - Novo suporte para manipular GPIO mais eficiente, kernel ≥ 5.4
- bluez5 - Pilha completa para manipulação de dispositivos Bluetooth e Protocolos
- **Busybox** - O canivete suíço de ferramentas para Sistemas Embarcados



Navegadores

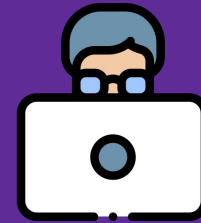
- ❖ **WebKit** - Web Browser Engine, Licença LGPL e BSD
- ❖ **Qt WebEngine** - Web Browser baseado em Chromium, Licença LGPLv3/GPLv3
- ❖ **COG** - Web Browser de baixo consumo e poucos recursos, baseado em WebKit, ideal para Sistemas Embarcados, Licença MIT
- ❖ **Surf** - Web Browser baseado em WebKit2 e GTK, Licença MIT



Laboratório

Prática 02

- ❖ Configurando um DHCP Server com Busybox udhcpd



—



Building System

- ❖ O processo de baixar, configurar, compilar e instalar manualmente pode ser incitar a erros
- ❖ Ao compilar um software com dependências, as mesmas devem ser configuradas, compiladas antes
- ❖ Algumas ferramentas surgiram para facilitar e automatizar estas tarefas
- ❖ No entanto, há uma curva de aprendizado para cada ferramenta de build



❖ Make

- Irá ler e processar o clássico Makefile: `make`

❖ Autotools

- Um sistema mais completo, o famoso: `./configure; make; make install`

❖ CMake

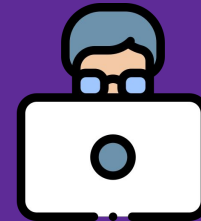
- Pode-se dizer o sucessor de autotools, uma versão melhorada, utilizada amplamente por diversos projetos open-source: `mkdir build; cd build; cmake ../ ; make`



Laboratório

Prática 03

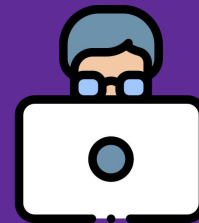
❖ Compilando Dropbear e instalando no Target



Laboratório

Prática 06 - Device-Tree

- ❖ Analisando a estrutura via Linux
- ❖ Analisando Periféricos
- ❖ Modificando Periféricos
- ❖ Compilando Novo Device-Tree







Ferramentas para
BuildSystem:

Buildroot
Yocto Project



Building System

- ❖ Construir um Sistema Linux Embarcado do Zero (Linux-From-Scratch) não é uma tarefa fácil
- ❖ Deverá preparar as ferramentas necessárias para compilação-cruzada(toolchain)
- ❖ Baixar os componentes básicos para o Sistema: Bootloader, Kernel e BusyBox
- ❖ Montar o RootFS manualmente
- ❖ Criar uma imagem (.img) com todas estes componentes



Building System

- ❖ Cada novo software, nova compilação e copiar para o RootFS
- ❖ Realizar configuração e compilação de diversos Buildsystems: Autotools, Makefile, CMake, Scons, QMake, etc
- ❖ Compilar as dependências de cada programa
- ❖ Uma mudança de configuração no Toolchain ou LibC, trabalho de refazer tudo novamente!



Building System

- ❖ Ferramentas foram criadas para resolver estes problemas
- ❖ Image uma ferramenta que baixe e instale o toolchain, Bootloader, Kernel, configura e compila para a placa alvo, cria o RootFS e gera a imagem para ser gravada na placa alvo
- ❖ Estas ferramentas existem e são open-source!
- ❖ Ferramenta que irá economizar dias de trabalho



❖ Buildroot

- Desenvolvido pela comunidade
- Suporta Toolchain Externo e Interno
- Praticamente configuração utilizando arquivos .mk
- Mais simples de usar

❖ Yocto Project

- Desenvolvido pela comunidade e colaboração de grandes empresas
- Configuração baseada em receitas (.bb)
- Construção baseada na ferramenta bitbake
- Curva de aprendizagem maior
- Ideal para projetos grandes
- Principal ferramenta de buildsystem Linux adotada pela Indústria



Ferramentas de BuildSystem Linux - Yocto Project

- ❖ A ferramenta adotada neste treinamento é o Yocto Project
- ❖ Poucas dependências, mas é recomendado utilizar Host Linux
- ❖ Com poucas linhas de configuração pode-se gerar uma imagem bootavel
- ❖ Suporta geração de diferentes SDK's
- ❖ Suporte a gerenciamento de pacotes



BuildSystem Yocto Project



Sobre o Yocto Project

O Yocto Project surgiu em meados de 2010 na iniciativa de diversos fabricantes de hardware, com intuito de reduzir a duplicidade de trabalho entre hardwares.

É uma solução open-source com diversas ferramentas, templates e rotinas para ajudar o desenvolvedor a criar sua própria Distribuição Linux Customizada, sendo suporte na geração da imagem, na geração de SDK e Toolchain e ferramentas de investigação ao Sistema gerado.

O Yocto Project é o resultado de um outro projeto conhecido como OpenEmbedded Project, tanto que, o Yocto Project seus principais componentes é base do OpenEmbedded-Core.

Hoje o projeto é gerenciado pela The Linux Foundation.



Sobre o Yocto Project

Um buildsystem como Yocto Project possui todos recursos necessários para gerar um ambiente **HOST** para o desenvolvedor e preparar o **TARGET** de acordo com suas necessidades e escopo do projeto.

O Yocto Project não é uma Distribuição Linux, mas possui todos recursos necessários para criar uma para você!

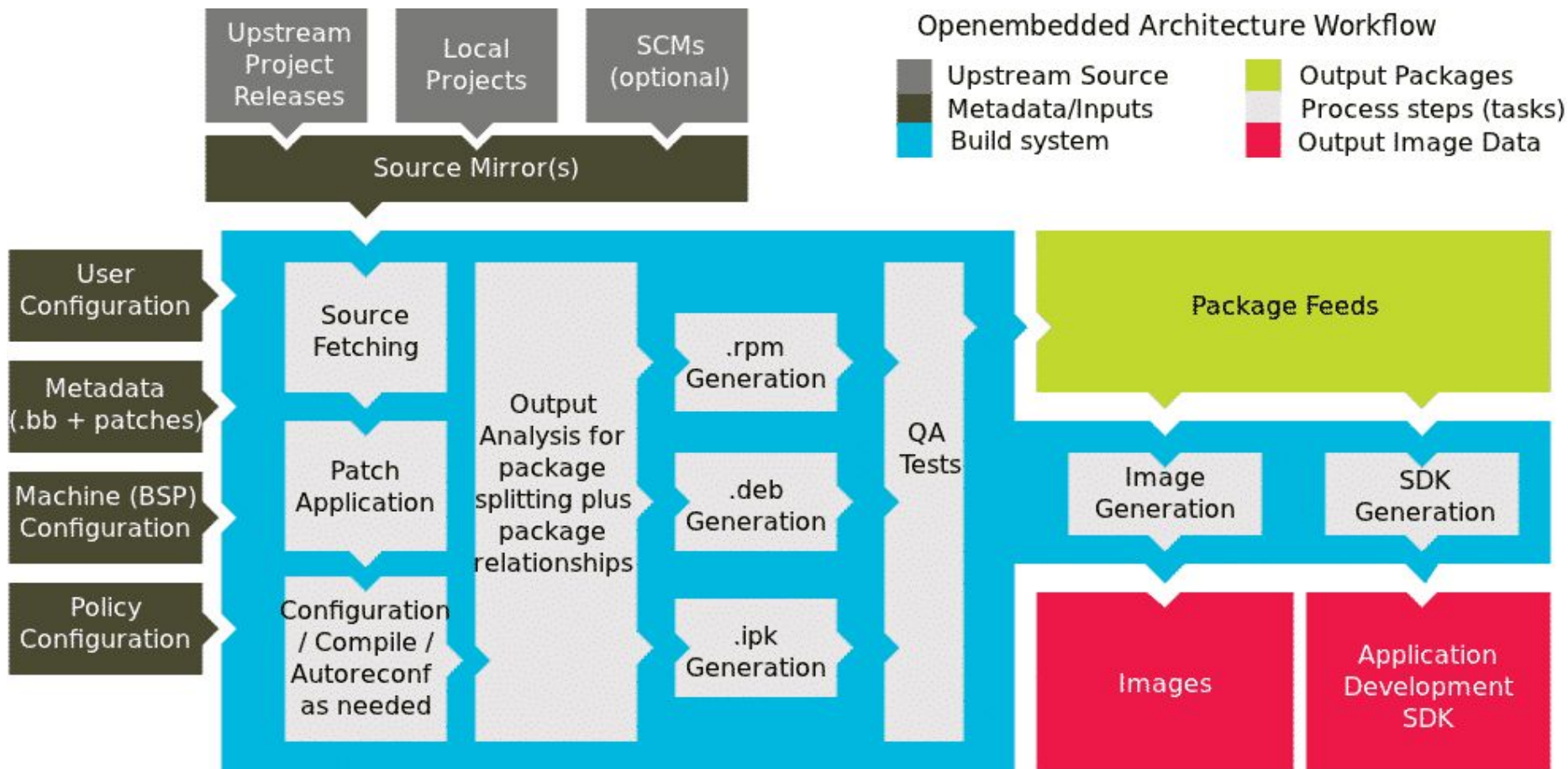


Projetos dentro do Yocto Project

Lista de projetos e ferramentas que contemplam e fornecem os devidos poderes ao Buildsystem Yocto Project.

- ❖ **Metadados** - O Bitbake processa metadados, estes são divididos em:
 - **Configuração** (.conf) - Configurações globais que deve fornecer informações para as Classes e Receitas, como informações do Hardware e Recursos.
 - **Classes** (.bbclass) - Classes são herdadas por todo sistema e aplicam uma específica tarefa para todas receitas que o herdarem, evitando duplicidade de código/tarefas.
 - **Receitas** (.bb e .bbappend) - Receita irá descrever uma aplicação com rotinas de como e de onde baixar a aplicação, descompactar, configurar, compilar, “testar”, empacotar e instalar na imagem, pode ser feito com mistura de rotinas em ShellScript e Python.





Yocto Project

Obtendo o código-fonte e os primeiros
passos

Yocto Project - Releases

CODINOME	YP Version	Release Date	Estado Atual	Bitbake Version
Scarthgap	5.0	04/2024	LTS	2.8
Nanbield	4.3	11/2023	até 05/2024	2.6
Kirkstone	4.0	05/2022	LTS	2.0
Dunfell	3.1	04/2020	LTS	1.46

Referência: <https://wiki.yoctoproject.org/wiki/Releases>



Obtendo o Projeto

```
$ mkdir ~/yp
$ cd ~/yp
~/yp $ git clone -b kirkstone git://git.yoctoproject.org/poky.git
Cloning into 'poky'...
remote: Enumerating objects: 625992, done.
remote: Counting objects: 100% (10556/10556), done.
remote: Compressing objects: 100% (4495/4495), done.
remote: Total 625992 (delta 8047), reused 7536 (delta 5985), pack-reused 615436
Receiving objects: 100% (625992/625992), 200.36 MiB | 356.00 KiB/s, done.
Resolving deltas: 100% (455382/455382), done.
~/yp $
```



Obtendo o Projeto

```
~/yp $ cd poky
~/yp/poky $ git branch
• kirkstone
~/yp/poky $
~/yp/poky $ ls
bitbake          Makefile          oe-init-build-env SECURITY.md
contrib          MEMORIAM          README.hardware.md
documentation    meta              README.md
LICENSE          meta-poky         README.OE-Core.md
LICENSE.GPL-2.0-only meta-selftest     README.poky.md
LICENSE.MIT      meta-skeleton     README.qemu.md
MAINTAINERS.md   meta-yocto-bsp   scripts
~/yp/poky $
```



Criando um Projeto com Yocto Project

Para iniciar um projeto no Yocto Project e preparar o ambiente, deve-se utilizar o scripts **oe-init-build-env**.

Exemplo:

```
source oe-init-build-env <nome-diretorio-build>
```

A primeira vez que executar, irá carregar diversas informações e dicas das primeiras images a utilizar, nas demais vezes irá aparecer uma tela resumida.



Primeiro Projeto

```
~/yp/poky $ source oe-init-build-env build
```

```
### Shell environment set up for builds. ###
```

You can now run 'bitbake <target>'

Common targets are:

- core-image-minimal
- core-image-sato
- meta-toolchain
- meta-ide-support

You can also run generated qemu images with a command like 'runqemu qemux86'

Other commonly useful commands are:

- 'devtool' and 'recipetool' handle common recipe tasks
- 'bitbake-layers' handles common layer tasks
- 'oe-pkgdata-util' handles common target package tasks



Primeiro Projeto

Após preparar o ambiente pela primeira vez, a seguinte estrutura será criada:

```
~/yp/poky/build $ tree
.
├── conf
│   ├── bblayers.conf
│   ├── local.conf
│   └── templateconf.cfg
```

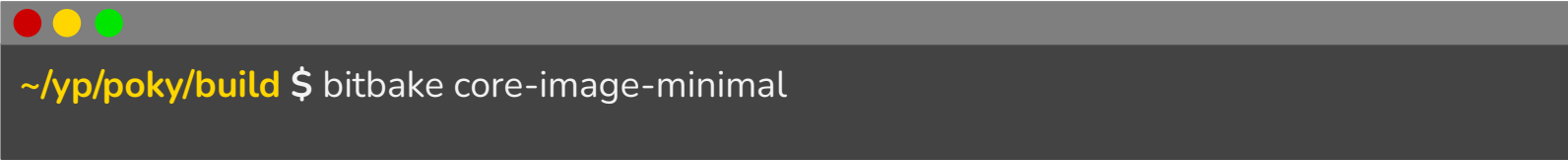


- ❖ **local.conf** - Arquivo onde pode-se configurar o hardware alvo, configurar Distribuição, Hardware, Tipo de Empacotamento e até programas a serem instalados - em um projeto real deverá possuir poucas alterações, pois uma camada deverá ser criada para toda customização.
- ❖ **bblayers.conf** - Arquivo onde serão adicionadas as camadas para estender suporte e recursos, por padrão, já inclui **meta**, **meta-poky** e **meta-yocto-bsp** - Um breve exemplo, para adicionar suporte ao **Framework Qt5**, deverá ser adicionada a camada **meta-qt5**.
- ❖ **templateconf.cfg** - Arquivo que aponta para meta-poky/conf para exibir conf-notes ao criar ou herdar ambiente



Bitbake

Todo trabalho de gerar imagem, toolchain, e processar receitas é realizado pelo **bitbake**.



```
~/yp/poky/build $ bitbake core-image-minimal
```



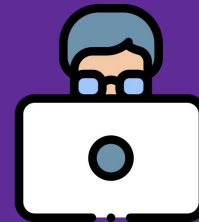
```
~/yp/poky/build $ bitbake picocom
```



Laboratório

Prática 10

- ❖ Obtendo o Yocto Project, clonando as camadas, criando um projeto e gerando uma imagem
- ❖ Comando para gerar o Toolchain

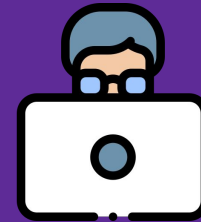




Laboratório

Prática 04

- ❖ Compilar aplicações direto no
TARGET - USB

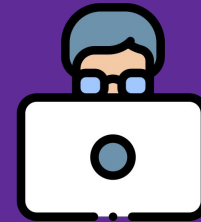




Laboratório

Prática 05

- ❖ Manipulando GPIO
- ❖ Manipulando PWM
- ❖ I2C
- ❖ EEPROM



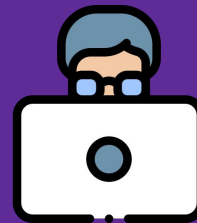


Laboratório

Prática 07



Servidor Web



Licenças

- ❖ Todo software que está sob uma licença de software livre concede quatro liberdades a todos os usuários
 - Liberdade de uso
 - Liberdade para estudar
 - Liberdade para copiar
 - Liberdade para modificar e distribuir cópias modificadas
- ❖ Consulte <https://www.gnu.org/philosophy/free-sw.html>
- ❖ O software de código aberto, de acordo com a definição da Iniciativa de código aberto, é tecnicamente semelhante ao Software Livre em termos de liberdades
 - <https://www.opensource.org/docs/osd> para obter a definição de Open SourceSoftware



- ❖ As licenças de software livre se enquadram em duas categorias principais
 - As licenças copyleft
 - As licenças não copyleft
- ❖ O conceito de copyleft é pedir reciprocidade nas liberdades concedidas ao usuário. O resultado é que quando você recebe um software sob uma licença de software copyleft free e distribuir versões modificadas dele, você deve fazê-lo sob a mesma licença
 - Mesmas liberdades para os novos usuários
 - É um incentivo para contribuir de volta com suas mudanças, em vez de mantê-las em segredo
- ❖ As licenças não copyleft não têm tais requisitos e as versões modificadas podem ser proprietárias, mas ainda requerem atribuição



- ❖ Algumas licenças CopyLeft
 - GPLv2/GPLv3 - General Public License, programas vinculados a uma biblioteca lançada sob a GPL também devem ser lançados sob a GPL, cuidado com GPLv3 em Sistemas Embarcados
 - LGPLv2/LGPLv3 - Lesser General Public License - Requisito linkagem dinamica
- ❖ Algumas licenças Non-CopyLeft
 - Apache License
 - BSD License
 - MIT License
 - X11 License



IDE de Desenvolvimento e Depuração

❖ VisualCode

- Telemetry: O Visual Studio Code coleta dados de uso e os envia para a Microsoft. A Declaração de Privacidade da Microsoft refere-se a todos os produtos da Microsoft e não revela detalhes sobre essa coleta de dados.
- Mas pode ser desabilitado nas configurações do usuário

❖ VsCodium

- Uma compilação de código aberto 100% do Visual Studio Code, com a telemetria desativada

❖ Eclipse

❖ Atom



Depuração com GDB

- ❖ GDB - GNU Project Debugger, veja mais <https://www.gnu.org/software/gdb/>
- ❖ O GDB está disponível para a maioria das arquiteturas embarcadas
- ❖ Linguagens suportadas: C, C ++, Pascal, Objective-C, Fortran, Ada ...
- ❖ Interface do console (útil para depuração remota)
- ❖ Também pode ser usado por meio de IDEs gráficos

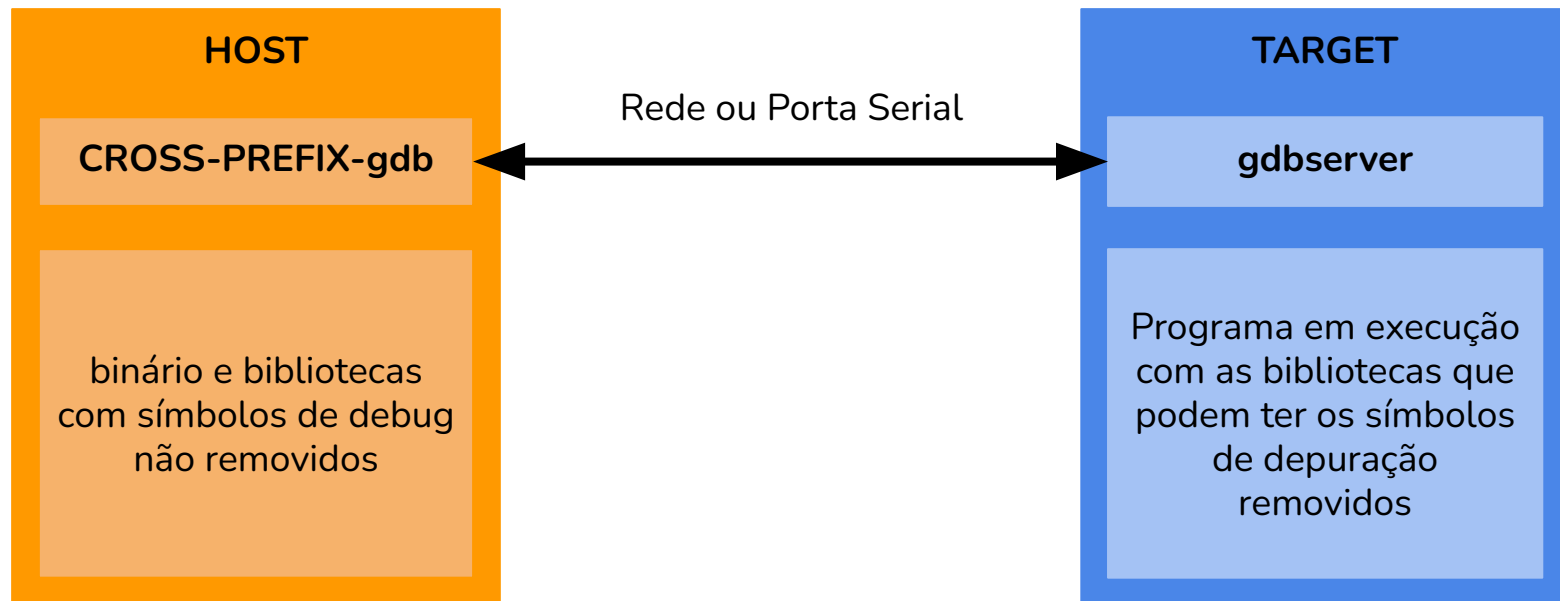


Depuração com GDB

- ❖ Pode ser usado para controlar a execução de um programa, definir pontos de interrupção ou alterar variáveis internas. Você também pode usá-lo para ver o que um programa estava fazendo quando travou (carregando sua imagem de memória, despejada em um arquivo central)
- ❖ GDBGui - Versão Gráfica desenvolvida em Python, repositório [gdbgui](https://github.com/0x00sec/gdbgui)
- ❖ Nova alternativa: lldb (<https://lldb.llvm.org/>) do projeto LLVM



Depuração com GDB - **Debug Remoto**



Depuração com GDB

❖ Opções de uso com GDB no TARGET

➤ Serial



```
# gdbserver /dev/ttyS0 <programa> <parametros>
```

➤ Rede



```
# gdbserver 0.0.0.0:<porta> <programa> <parametros>
```

➤ Lançando em um processo em execução



```
# gdbserver --attach 0.0.0.0:<porta> <pid_programa>
```



Depuração com GDB

❖ Opções de uso com GDB no HOST

➤ Serial

```
$ arm-unknown-linux-uclibcgnueabi-gdb  
gdb> target remote /dev/ttyS0
```

➤ Rede

```
$ arm-unknown-linux-uclibcgnueabi-gdb  
(gdb) target remote <ip-addr>:<port>
```



Depuração com GDB

❖ Executando gdbserver no TARGET

```
# gdbserver 0.0.0.0:<porta> <programa> <parametros>  
Process <programa> created; pid = 251  
Listening on port <porta>  
Remote debugging from host 192.168.254.153, port 56156
```

❖ Executando gdbserver no HOST

```
# arm-unknown-linux-uclibcgnueabi-gdb  
(gdb) target remote 192.168.254.159:6677  
Remote debugging using 192.168.254.159:6677  
(gdb)
```



Outras ferramentas

- ❖ **valgrind** - Ferramenta para detectar BUGS de implementação com alocação de memória e **Helgrind** para erros e bugs com Threads
- ❖ **strace** - System Call Tracer, permite ver o que qualquer um de seus processos está fazendo: acessando arquivos, alocando memória, etc
- ❖ **ltrace** - Library Trace, uma ferramenta para rastrear chamadas de biblioteca usadas por um programa e todos os sinais que ele recebe

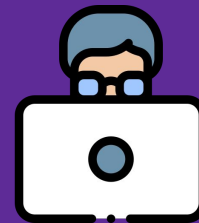


Laboratório

Prática 08



Depurando Aplicações com
GDB





Cleiton Bueno - cleiton.bueno@b2open.com