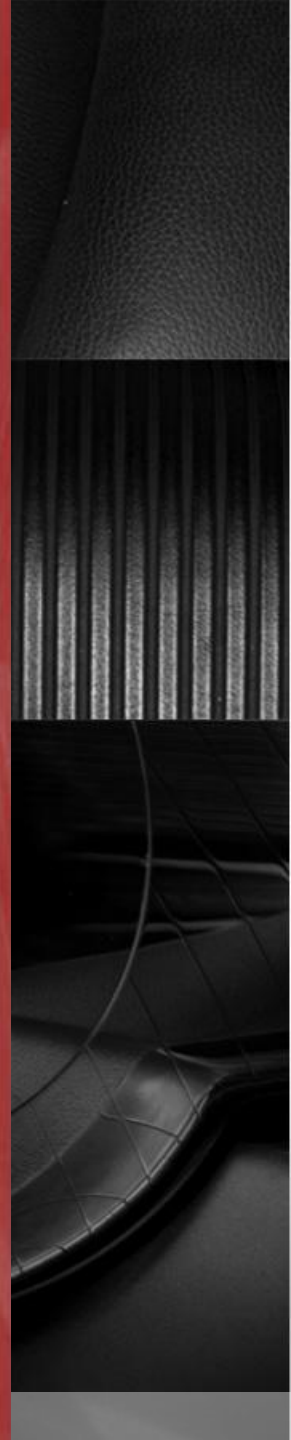
Linux Embarcado

Definição, Recursos e Ferramentas



O que é Linux Embarcado?

- É um *kernel* de Linux modificado para funcionamento em dispositivos embarcados.
- Amplo crescimento e uso no mercado
- Exposto às restrições de sistemas embarcados
  - Memória
  - Energia
  - Processamento

Por que é tão útil?

- Suporte total à padrões e protocolos de rede
- Estabilidade
- Modularidade
- *Multicore*
- Ampla gama de recursos e suporte, disponíveis pela internet.

Um sistema Linux Embarcado

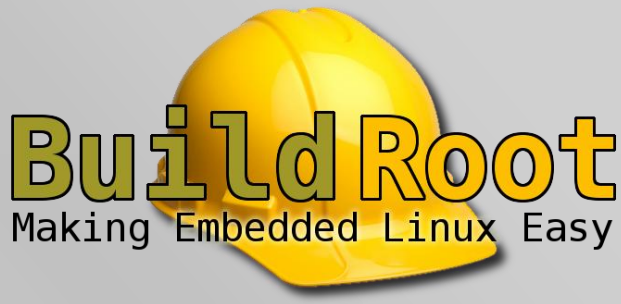
- Composto por:
  - *Bootloader*
  - *Kernel*
  - *Root Filesystem*

  - Normalmente, armazenamento em memória *flash*.
  - Domínio de arquitetura ARM de 32 *bits*
  - Existem outras arquiteturas compatíveis, como MIPS e AVR32.

Ferramentas de Uso

- Sistema de Build (*buildsystem*)
  - Compilar e gerar o *toolchain*
  - Compilar e gerar a imagem do *bootloader*
  - Compilar e gerar a imagem do *kernel*
  - Compilar bibliotecas e aplicações, resolver dependências e gerar o sistema de arquivos (*rootfs*).

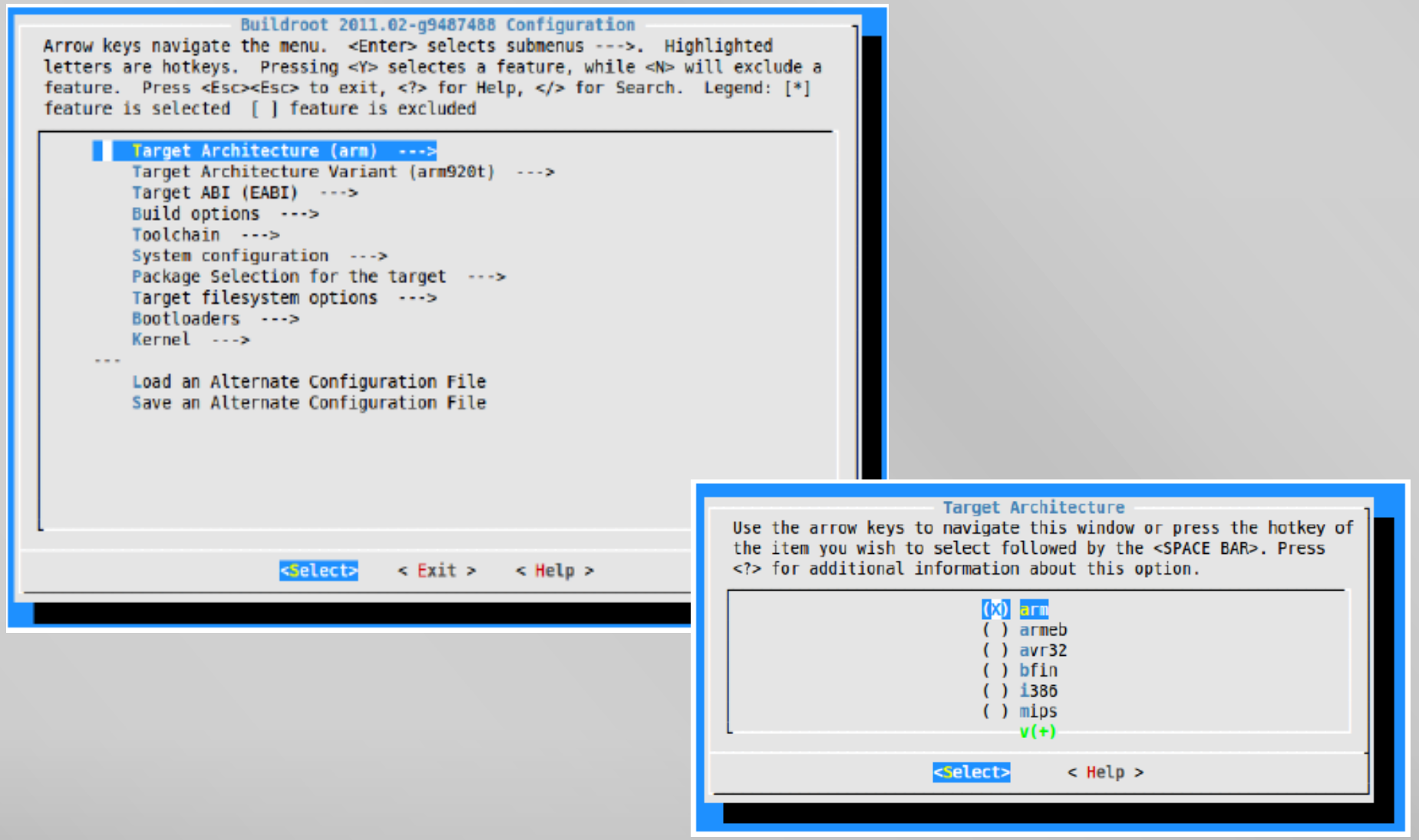
Sistema de *Build*



- *Buildroot*
  - www.buildroot.net
  - Desenvolvido pelos mantenedores da *uClibc*
  - Implementa um sistema automático de build através de um conjunto de *makefiles*
  - Gera o *toolchain*, *bootloader*, *kernel*, *rootfs* com bibliotecas e aplicações

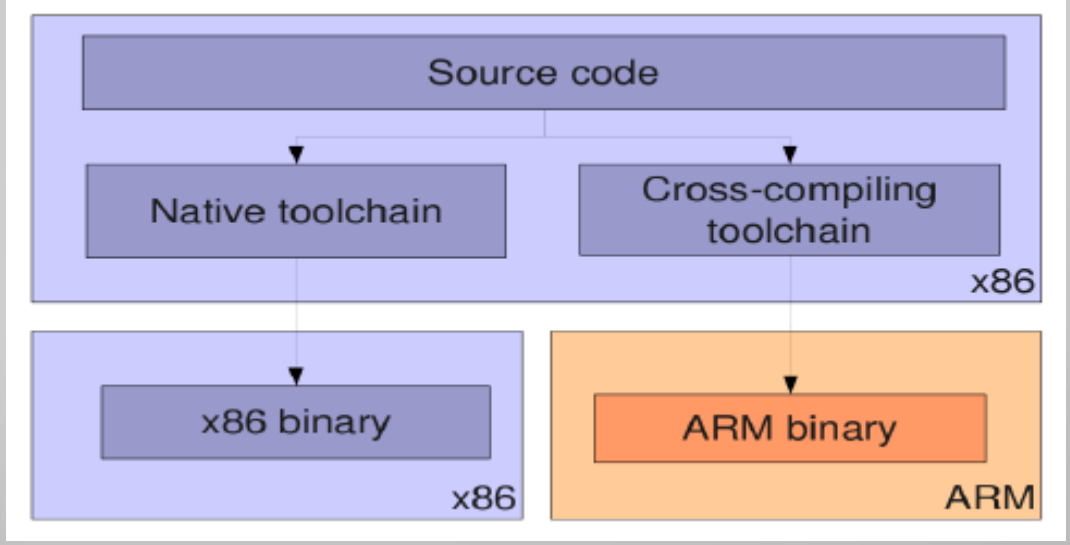
Buildroot

- Configuração parecida com a do menu do *kernel* Linux para PCs
- Configuração salva em um arquivo *'.config.'* no diretório principal da aplicação
- Seleção de suporte a demais linguagens de programação e bibliotecas de sistema
- Adição de pacotes para serem inseridos no dispositivo alvo


Buildroot 2011.02-g9487488 Configuration
Arrow keys navigate the menu. <Enter> selects submenus --->. Highlighted letters are hotkeys. Pressing <Y> selectes a feature, while <N> will exclude a feature. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*] feature is selected [ ] feature is excluded
Target Architecture (arm) --->
Target Architecture Variant (arm920t) --->
Target ABI (EABI) --->
Build options --->
Toolchain --->
System configuration --->
Package Selection for the target --->
Target filesystem options --->
Bootloaders --->
Kernel --->
Load an Alternate Configuration File
Save an Alternate Configuration File
<Select>
< Exit >
< Help >
Target Architecture
Use the arrow keys to navigate this window or press the hotkey of the item you wish to select followed by the <SPACE BAR>. Press <?> for additional information about this option.
(X) arm
( ) armeb
( ) avr32
( ) bfin
( ) i386
( ) mips
v(+)
<Select>
< Help >

Toolchain

- "Cadeia de ferramentas"
  - Uma sequência de operações executadas uma após a outra.
  - Ex.: Numa linguagem de programação: Compilador -> *Assembler* -> *Linker*
- 2 Tipos de *Toolchain:*
  - *Native toolchain*
  - *Cross-compiling toolchain*
- Na maioria dos casos não existe um *native toolchain* para Linux embarcado – o *toolchain* é sempre executado em outra máquina.



Toolchain – 5 Componentes

1. Compilador GCC
2. *Binutils* – utilizado para converter *assembly* em código objeto e *linkar* arquivos objeto.
3. Biblioteca C padrão – faz a interface com o *kernel* através das *system calls*.
   - *Linkagem* estática ou dinâmica – pode ocorrer problemas de compatibilidade quando é feita a *linkagem* dinâmica em algum determinado dispositivo. A biblioteca C padrão deve ser a mesma utilizada pelo *toolchain*.
   - Várias implementações. Entre elas, *glibc* (padrão em *desktops*, maior performance), e *uClibc* (mais utilizada em Linux embarcado por gerar códigos menores).

C program	*Compiled with shared libraries*		*Compiled statically*	
	glibc	*uClibc*	*glibc*	*uClibc*
Plain "hello world" (stripped)	5.6 K (glibc 2.9)	5.4 K (uClibc 0.9.30.1)	472 K (glibc 2.9)	18 K (uClibc 0.9.30.1)
Busybox (stripped)	245 K (older glibc)	231 K (older uClibc)	843 K (older glibc)	311 K (older uClibc)

Toolchain – 5 Componentes

4. Kernel Headers – local onde estão as referências às *system calls*
   - *Defines, structs,* constantes...
   - Há muito pouca diferença dos *kernel headers* para cada versão do *kernel* Linux

```
...
#define __NR_SYSCALL_BASE       0

#define __NR_exit               (__NR_SYSCALL_BASE+  1)
#define __NR_fork               (__NR_SYSCALL_BASE+  2)
#define __NR_read               (__NR_SYSCALL_BASE+  3)
#define __NR_write              (__NR_SYSCALL_BASE+  4)
#define __NR_open               (__NR_SYSCALL_BASE+  5)
#define __NR_close              (__NR_SYSCALL_BASE+  6)
...
```

5. GDB – *debugger* padrão em sistemas Linux. Opcional.

Bootloader

- Responsável por inicializar o *hardware*, como memória RAM e demais periféricos.
- É quem prepara o sistema para o carregamento do *kernel*.
- Assim como temos diversos *bootloaders* em *desktops*, como GRUB e LILO, também temos diversos *bootloaders* para dispositivos embarcados
- Um dos mais comuns é o *U-boot*.

Configurações recomendadas

Configurar Kernel

Configurar uClibc

Configurar Busybox

Kernel

- Gerencia execução do processador e controla acesso à memória e *I/O*
- *Kernel Space vs. User Space*
- Interface de *User Space* com *Kernel Space* via chamadas do sistema (*System Calls*)
- Acesso ao *hardware* via arquivos de dispositivo
- Gerenciamento dinâmico dos módulos do *kernel*.

Rootfs

- Sistema de arquivos do dispositivo
- Contém os componentes básicos
  - Bibliotecas do sistema, como *uClibc, glibc, dietlibc*, etc.
  - *Scripts* de inicialização
  - Bibliotecas e aplicações
- Exemplos de aplicações:
  - *Dropbear*: Cliente e servidor ssh – 110kb de memória
  - *Thttpd*: Servidor *web* – 88kb de memória
  - *SQLite*: Banco de Dados – 250kb
  - *Busybox*: Conjunto de ferramentas UNIX – 1mb

Etapas Finais

- Após todo o processo de montar igual *Lego*:
  - Compilar todos os elementos (processo demorado)
  - Carregar as imagens na memória do dispositivo

    Pode envolver uso de dispositivos *JTAG* ou outras formas, mais fáceis ou não, de inserção e *debug*.

Dúvidas?