Desenvolvimento embarcado com OpenEmbedded e Mamona

Ricardo Salveti

24 de Junho, 2009

Fórum Internacional de Software Livre - FISL 10

openBossa
Software in a creative way

Tópicos

- O que é o OpenEmbedded
- Por que da existência
- Como o OE funciona
- Problemas gerais em manter uma plataforma para diferentes máquinas e distribuições
- Problemas gerais de cross-compilação e como OE resolve boa parte deles
- Demonstração rápida do Mamona
- Perguntas

OpenEmbedded

O que é o OE?

- Conjunto de meta-dados e descrições
- Exemplos:
    - Máquinas
    - Distros (Mamona!)
    - Classes
    - Pacotes
- Junto com o Bitbake, se torna um build framework responsável por cuidar de todos problemas de cross-compilação para embedded Linux
- Arquitetura similar ao Portage do Gentoo

Por que do OpenEmbedded?

- Cross-compilar software na mão não é algo trivial
- Diversas soluções separadas, pouco reuso de código
- Gerenciamento manual de dependências
- Falta de uma comunidade para agregar desenvolvedores GNU/Linux de ambientes embarcados

Como o ambiente do OE funciona

- Bitbake
    - Executa tarefas
    - Parseia e executa todos os recipes (meta-dados)
- OpenEmbedded
    - Meta-dado com as definições dos pacotes, máquinas e distribuições
    - Descreve ao Bitbake como compilar um software e como gerar os pacotes necessários
- Ferramentas externas
    - De acordo com a distribuição
    - Flash do device
    - No caso do Mamona, o mamona-installer e o 0xFFFF

Problemas comuns que o OE ajuda a solucionar

Problemas comuns no desenvolvimento e uso de uma Plataforma

Cross-Compilação e Toolchains

- Staging
- Pacotes nativos
- Escolha do toolchain e respectivas versões:

```
# gcc
PREFERRED_PROVIDER_virtual/${TARGET_PREFIX}gcc-initial = "gcc-
cross-initial"
PREFERRED_PROVIDER_virtual/${TARGET_PREFIX}gcc = "gcc-cross"
PREFERRED_PROVIDER_virtual/${TARGET_PREFIX}g++ = "gcc-cross"

PREFERRED_VERSION_gcc ?= "4.1.2"
PREFERRED_VERSION_gcc-cross ?= "4.1.2"
PREFERRED_VERSION_gcc-cross-sdk ?= "4.1.2"
PREFERRED_VERSION_gcc-cross-initial ?= "4.1.2"
```

Suporte a diferentes máquinas

```
# Beagle board
TARGET_ARCH = "arm”
PACKAGE_EXTRA_ARCHS = "armv4 armv4t armv5te armv6"
PREFERRED_PROVIDER_virtual/xserver = "xserver-kdrive"
XSERVER = "xserver-kdrive-fbdev"
TARGET_FPU = "hard"
include conf/machine/include/tune-arm1136jf-s.inc

IMAGE_FSTYPES += "tar.bz2 jffs2"
EXTRA_IMAGECMD_jffs2 = "-lnp "
SERIAL_CONSOLE = "115200 ttyS2"
PREFERRED_PROVIDER_virtual/kernel = "linux-omap"
KERNEL_IMAGETYPE = "uImage"
UBOOT_ENTRYPOINT = "0x80008000"
UBOOT_LOADADDRESS = "0x80008000"
UBOOT_MACHINE = "omap3_beagle_config"
PREFERRED_VERSION_u-boot = "git"
MACHINE_FEATURES = "kernel26 screen color32 keyboard usbgadget
usbhost vfat videoplay sound alsa x11"
```

Suporte a diferentes distros

- Arquivo de configuração para cada distro:
  - Nome e mantenedores
  - Toolchain
  - Features
  - Versões desejadas de certos pacotes
  - Features e pacotes de acordo com as máquinas suportadas
- Mamona:
  - Plataforma para ARM
  - Foco inicial nos Tablets da Nokia

Problemas comuns ao Cross-Compilar softwares

Diversas ferramentas de build

- Exemplos:
  - Autotools
  - Cmake
  - Make
  - Distutils
  - Setuptools
  - Cpan
  - Qmake
- Classes para cada tipo:
  - Implementa as peculiaridades de cada ferramenta
  - Funções que deixam transparente o uso de diferentes ferramentas de build ao compilar um software

Build Class (cmake.bbclass)

```
DEPENDS += " cmake-native "

inherit autotools

cmake_do_configure() {
    cmake . -DCMAKE_INSTALL_PREFIX:PATH=${prefix}
}

EXPORT_FUNCTIONS do_configure
```

Compilação em duas etapas

- Quando a ferramenta gera primeiro uma ferramenta que será utilizada para compilar o resto do software
- Exemplo: Cmake
    - Primeiro passo, compilar nativamente o cmake
    - Segundo passo, utilizar o cmake gerado para compilar o resto do software
- Problemas:
    - Por ser cross-compilado, o binário da primera etapa não necessáriamente pode rodar no sistema nativo do build
- Solução? Uso de pacotes nativos

Uso de uma ferramenta de build própria

- Geralmente pouco flexível
- Não verifica dependências
- Uso incorreto de bibliotecas e headers
- Premissas incorretas sobre onde o software será compilado
- Solução? Evitar o uso, ou definir o build no próprio OE

Uso incorreto do Autotools

- Uso incorreto das macros
- Definições incorretas de novas macros
- Paths hardcoded
- Falta de verificação de todas as depêndencias
- Solução?
    - Parar de dar copy&paste de outros projetos
    - Patches adicionais no OE com as correções

Hard Coded Stuff! :D

- Erro mais comum e mais chato de ser debugado
- Independe da ferramenta de build utilizada
- Exemplos:
  - Paths para includes
  - Bibliotecas
  - Compiladores

Developers, be kind!

Lembrem-se que ao criar um software, ele não necessariamente vai rodar só no seu PC :-)

Demonstração do Mamona!

More information

- Mamona - http://dev.openbossa.org/trac/mamona
- OpenEmbedded - http://wiki.openembedded.net
- Bitbake Manual - http://bitbake.berlios.de/manual
- Listas de email - mamona-devel, oe-devel
- IRC - Freenode
    - #mamona
    - #oe
    - #beagle

Perguntas?
:-)

Obrigado!

ricardo.salveti@openbossa.org
rsalveti@Freenode