1 Criação da distribuição *Ångström Linux*

1.1 Preparação da plataforma *host*

Vai-se utilizar o sistema operativo *Linux Ubuntu* (64-bits) numa plataforma x86. É necessário instalar algumas aplicações que são necessárias para o *OpenEmbedded* e *Bitbake*. Também se pode utilizar um sistema operativo *Linux Ubuntu* 32-bits.

```
$sudo apt-get install git-core quilt help2man gawk wget texi2html
texinfo build-essential unzip diffstat subversion texinfo cvs chrpath
libsdl1.2-dev xterm
```

Obtenção da estrutura de diretorias necessária para a criação da distribuição *Ångström Linux*. Esta estrutura encontra-se no repositório github em http://github.com/angstrom.

Antes de clonar a diretoria **`setup-scripts`** é necessário executar o seguinte comando:

```
$sudo dpkg-reconfigure dash
```

Apontar para /bin/dash e responder **NO** no menu. À partida já se estará a utilizar a *shell bash* (*bourne again shell*).

dpkg é o sistema de gestão de pacotes da distribuição de *Linux Debian*. Este conjunto de ferramentas de software permite tornar automático o processo de instalação, atualização e remoção de aplicações. Através de uma base de dados que contém informações acerca dos pré-requisitos e dependências é possível instalar de forma transparente pacotes de software num determinado sistema operativo. No entanto dpkg ainda é uma ferramenta de baixo nível. A sua ação é ao nível das dependências em pacotes individuais. Um exemplo de gestor de alto nível é o **APT** (*Advanced Package Tool*), o qual utiliza o dpkg mas age a um nível mais elevado, mantendo relações de interdependência entre diferentes pacotes de software. Foi usado anteriormente para instalar aplicações na distribuição de *Linux Ubuntu* (*apt-get install*).

Todo o processo daqui em diante é baseado primordialmente nos recursos do sítio http://www.angstrom-distribution.org/ em conjunto com recursos de outros sítios da WWW.

Para criar a estrutura de diretorias efetuar o seguinte comando:

```
$git clone git://github.com/Angstrom-distribution/setup-scripts.git
```

Na diretoria atual irá ser criada uma diretoria com a seguinte estrutura:

+ setup-scripts

+ conf

bblayers.conf

Contém os nomes das diretorias com os *recipes* das várias camadas (*layers*) base, que não fazem parte do núcleo do *OpenEmbedded*, das camadas relativas à máquina ou placa (*BSPLAYERS*), no caso, *beagleboard*, e das camadas extra (*EXTRALAYERS*). A variável *BBLAYERS* contém todas estas camadas.

local.conf

Contém parâmetros de configurações globais tais como o tipo de ficheiros gerados, por exemplo tar.gz, e o nome da distribuição.

+ scripts

+ sources

layers.txt

Neste momento ainda não contém ficheiros *recipe*, mas irá conter, em função da plataforma de hardware selecionada. Irá conter os *recipes* presentes no ficheiro bblayers.conf.

As ligações de acesso aos repositórios com os ficheiros *recipe* encontram-se no ficheiro layers.txt, o único, nesta fase, presente nesta diretoria. Este ficheiro contém ainda a ligação ao repositório do *Bitbake*.

oebb.sh

Ficheiro *batch* utilizado para automatizar o processo de criação da distribuição de Linux embebido. Através de comandos passados como argumentos na chamada à execução deste ficheiro é possível invocar o *Bitbake*.

README

1.2 Download dos ficheiros *recipe* para a plataforma *beagleboard*

Através do ficheiro *batch* oebb.sh extraem-se os ficheiros *recipe* os quais permitirão, numa fase posterior, criar a distribuição de Linux para a *beagleboard-xM*. O comando seguinte vai colocar na diretoria *sources* todos os ficheiros *recipe* necessários para compilar o *kernel* do sistema operativo Linux para a plataforma de hardware *beagleboard-xM*:

```
$./oebb.sh config beagleboard
```

De seguida faz-se uma atualização ao *OpenEmbedded* com o seguinte comando:

```
$./oebb.sh update
```

1.3 Compilação do *kernel*

1.3.1 Introdução

Compilar o *kernel* é um processo muito simples, de facto. Extrai-se o código fonte do *kernel* a partir dos repositórios oficiais em www.kernel.org. De seguida entra-se na diretoria raiz do código do *kernel* e executa-se os seguintes comandos:

- **`$make menuconfig`**
  - Permite editar o ficheiro .config que contém a descrição das características do *kernel*, tais como os *drivers* que serão compilados e se o serão de forma estática ou dinâmica. Se forem compilados de forma estática estarão no ficheiro binário da imagem do *kernel* e serão sempre carregados para a memória quando o *kernel* iniciar. Se forem considerados módulos serão carregados caso haja necessidade e essa tarefa pode ser efetuada a qualquer momento, estando já o *kernel* iniciado. Por exemplo, os *drivers* do chip de

ethernet poderiam ser carregados apenas se fossem necessários. Este tipo de procedimento permite poupar memória.
    - A edição do ficheiro .config pode ser efetuada no modo gráfico, o que é uma mais valia, já que se torna mais simples. Para isso é necessário ter a biblioteca ***ncurses*** instalada.
- **`$make`**
    - Compila o código do *kernel*.
- **`$make modules`**
    - Compila os módulos do *kernel*.
- **`$make modules_install`**
- **`$make install`**

A compilação do *kernel* da distribuição *Ångström* Linux é muito semelhante, sendo que, neste caso, é o *Bitbake* que executa estas funções de forma completamente transparente.

1.3.2 Configurar o *kernel*

Para configurar o *kernel* basta editar o ficheiro **`defconfig`** que se encontra em **/setup-scripts/sources/meta-ti/recipes-kernel/Linux/Linux-mainline-3.2/beagleboard/**. Este ficheiro permite configurar o *kernel*, isto é, quais os *drivers* que se pretende inserir na imagem **uImage**. Estes *drivers* vão depender do hardware que se terá. Por exemplo, para adicionar o *driver* do *timer watchdog* basta colocar as seguintes linhas:

```
CONFIG_WATCHDOG=y
CONFIG_WATCHDOG_NOWAYOUT=y
#
# Watchdog Device Drivers
#
# CONFIG_SOFT_WATCHDOG is not set
#CONFIG_OMAP_WATCHDOG is not set
CONFIG_TWL4030_WATCHDOG=y
```

Mas editar o ficheiro desta forma torna-se perigoso e é necessário saber com exatidão os nomes dos diferentes *drivers* e a sua relação com os componentes de hardware. Assim faz-se a configuração por intermédio de um menu gráfico. Mas para isso é necessário instalar na máquina *host* a biblioteca **`libncurses5-dev`**.

Para isso basta fazer:

```
$apt-get install libncurses5-dev
```

É necessário agora voltar para a diretoria **setup-scripts/** e efetuar o *source* do *environment* para se poder utilizar finalmente o *Bitbake*:

```
$ . ~/.oe/environment-angstromv2012.12
```

Invoca-se agora o comando *menuconfig* através do *bitbake*. Este comando vai permitir entrar no modo de configuração gráfico do *kernel*, Figura 1:

```
$bitbake –c menuconfig virtual/kernel
```

Esta operação pode demorar bastante tempo (dezenas de minutos), em função das características da máquina que está a efetuar o processo. Esta operação vai criar mais uma diretoria em ***setup-scripts*** denominada ***build*** com a seguinte estrutura:

+build
- **+sstate-cache**
- **+tmp-angstrom_v2012_12-eglibc**
  - **…**
  - **+work**
    - **+beagleboard-angstrom-linux-gnueabi**
      - **+Linux-mainline-3.2.28-r122a**
        - **…** (várias diretorias com *patchs* do *kernel*)
        - **defconfig** (ficheiro de configuração do *kernel*)
        - **+git** (o código do *kernel* está aqui)
        - **…** (várias diretorias com *patches* do *kernel*)
  - **…**

No final do processo aparece o menu da Figura 1.

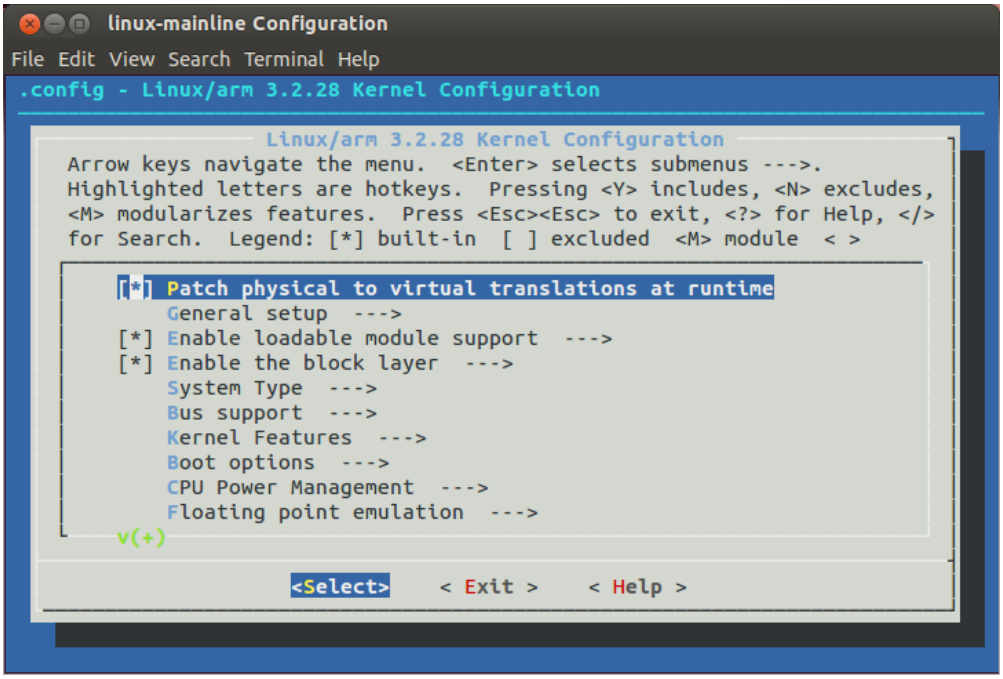

Figura 1 - Configuração do *kernel* do Linux em modo gráfico.

Pode-se sair sem efetuar qualquer configuração. Isto porque se for necessário aplicar um *patch*, uma alteração ao código, este é o momento. Na próxima secção é explicado o processo de aplicação de *patches* ao *kernel*.

Depois de aplicados os *patches* pode-se repetir o processo de configuração, o qual irá gerar o ficheiro ***.config*** na diretoria ***git*** (está oculto). Pode-se sempre colocar este ficheiro na diretoria setup-scripts/sources/meta-ti/recipes-kernel/Linux/Linux-mainline-3.2/beagleboard/

com o nome ***defconfig***. Desta forma pode-se repetir todo o processo e passar a fase de configuração, ou configurar a partir desta configuração, já que a diretoria ***build*** é temporária.

1.3.3 Aplicação de *patches*

Pode agora abrir-se o ficheiro **`beagleboard.conf`** com o editor **gedit** ou **vim** na diretoria dos *recipes* relacionados com o hardware, no caso, a diretoria **`meta-ti`** e subdiretorias **`conf`** e **`machine`**.

```
$gedit /setup-scripts/sources/meta-ti/conf/machine/beagleboard.conf
```

Existe uma linha denominada **`PREFERRED_PROVIDER_virtual/kernel = "linux-mainline"`**. Esta linha define a versão do *kernel* Linux que se pretende compilar.

Para efetuar alterações ao *kernel* é necessário incluir *patches* no ficheiro *recipe* do *kernel*, no caso, o **`linux-mainline_3.2.bb`**. Esse ficheiro pode ser acedido através do seguinte comando:

```
$gedit /setup-scripts/sources/meta-ti/recipes-kernel/linux/linux-mainline_3.2.bb
```

Ao abrir este ficheiro *recipe* com o editor de texto *gedit* pode ver-se que inclui o ficheiro *recipe* linux.inc, que o tipo de ficheiro que se vai criar nesta compilação designa-se por uImage e é um ficheiro binário com o código do *kernel* compilado. O ficheiro **uImage** também pode ser chamado de imagem do *kernel*, neste caso, para processadores da *Texas Instruments*, ou não se estaria na diretoria *meta-ti*.

Este ficheiro *recipe* contém ainda o endereço do código fonte do *kernel* Linux acessível através do repositório kernel.org. Pode-se fazer o download do código fonte do *kernel* através do seguinte comando:

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

Este último é para matar alguma curiosidade em relação ao código fonte do *kernel*. O código é muito extenso e bastante complexo, com C e assembly à mistura, pelo que se fica apenas pela curiosidade de inspecionar alguns ficheiros como o escalonador de tarefas em **core.c** e **sched.h** ou o código específico do processador ARM, em assembly, para iniciar o *kernel* em **arch/arm/boot/bootp/init.S**. Os ficheiros do escalonador de tarefas encontram-se em **kernel/sched**.

O *kernel* lida diretamente com o hardware, fazendo a comunicação entre este e as aplicações do utilizador. Desta forma, o *kernel* depende do tipo de hardware, pelo que a mudança de arquitetura implica nova compilação com código fonte ligeiramente diferente no que diz respeito ao acesso direto a recursos de hardware. Além disso, é claro, o compilador também é diferente, de acordo com a arquitetura do processador. O *kernel* comunica com o hardware por intermédio de *drivers*. Esses *drivers* são específicos de cada hardware. Por exemplo, se existe um chip para controlar as portas USB terá que existir código para aceder a esse chip. Esse código está diretamente relacionado com as funções presentes no datasheet disponibilizado pelo fabricante desse chip. Os *drivers* podem ser compilados com o *kernel* e

nesse caso são colocados na memória logo que o *kernel* arranca pois estão no binário **uImage**. Esses *drivers* são escolhidos na configuração do *kernel*.

Um *patch* é um ficheiro que contém alterações de um ou mais ficheiros de código fonte. Em softwares desenvolvidos por múltiplas equipas a única forma de gerir as modificações do código é por intermédio de um gestor de versões que utilizam *patches* relativos a um determinado ficheiro. Por exemplo, se se pretender modificar uma linha de código basta criar um *patch* que contém a nova linha. Esse *patch* é um ficheiro de texto que vai mencionar qual a linha que desaparece no ficheiro original e qual é que é colocada de novo. Desta forma modifica-se o ficheiro original ao aplicar o *patch*.

Para aplicar um *patch* ao código do *kernel* é necessário adicionar o a localização do mesmo na variável SRC_URI do ficheiro Linux-mainline_3.2.bb. Segue-se o exemplo da aplicação do *patch static-mac-address.patch*, cujo objetivo é o de colocar um endereço *mac* constante:

```
SRC_URI+= "git://git.kernel.org/pub… \
            file://distro/0001-kbuild... \
            … \
            … \
            .
            .
            .
            …\
            file://mypatches/static-mac-address.patch \
            file://defconfig"
```

Esta é a forma utilizada pelo *OpenEmbedded* para alterar o código fonte. No entanto, tendo o código fonte pode-se compilar manualmente, isto é, sem o *OpenEmbedded*, recorrendo mesmo a um IDE. Em capítulos posteriores fala-se sobre esta forma de compilar, muito mais rápida mas também muito mais propícia a erro.

Para criar um *patch* consultar a Secção **Erro! A origem da referência não foi encontrada.**.

Colocar o *patch* na diretoria do *kernel* em **setup-scripts/sources/meta-ti/recipes-kernel/linux/linux-mainline-3.2/mypatches**. É necessário criar a diretoria **mypatches**.

1.3.4 Compilação do código do *kernel*

Depois de se efetuar a configuração do *kernel* é possível efetuar a compilação do mesmo. Para isso basta executar o seguinte comando:

```
$./oebb.sh bitbake virtual/kernel
```

Ao fim de alguns minutos, e se tudo correr bem, serão gerados na diretoria **/setup-scripts/build/tmp-angstrom_v2012_12-eglibc/deploy/images/beagleboard/** dois ficheiros, um dos quais está comprimido e contém os módulos do *kernel* e o outro é binário e designa-se **uImage-3.2.28-r122a-beagleboard-(…).bin**. Este último é a imagem do *kernel* que é carregada após o *boot* e executará indefinidamente enquanto o hardware estiver ligado.

uImage.bin – imagem do *kernel*

1.4 Compilação dos *boot loaders* e criação do sistema de ficheiros

A distribuição de Linux contém elementos para além do *kernel*, tais como, sistema de ficheiros (*root file system*), aplicações, serviços, etc. Torna-se assim necessário construir o sistema de ficheiros e os *boot loaders*, os quais permitem carregar o *kernel* para a memória de dados. Além disso, nesta fase pode-se indicar quais as aplicações ou bibliotecas de software que se pretende instalar no sistema de ficheiros. Isso também poderia ser feito a posteriori, com o uso do **`dpkg`** (semelhante ao ***apt-get install***). Contudo, fazê-lo nesta fase é benéfico pois a distribuição não vai ser alvo de alterações a este nível por se tratar de um sistema embebido.

Começa-se por abrir o ficheiro **`console-image.bb`** na diretoria **/setup-scripts/sources/meta-angstrom/recipes-images/angstrom**. Acrescenta-se de seguida as seguintes aplicações e bibliotecas da lista que se segue em ***IMAGE_INSTALL***.

```
IMAGE_INSTALL += "      … \
                        … \
                        gdbserver \
                        lighttpd \
                        lighttpd-module-cgi \
                        lighttpd-module-alias \
                        cronie \
                        libcrypto \
                        libxext \
                        libsm \
                        libstdc++ \
                "
```

De seguida executa-se o comando que permite criar o sistema de ficheiros e os *boot-loaders*, os quais serão depois colocados no cartão de memória SD, devidamente formatado.

```
$bitbake console-image
```

Esta última operação permite criar vários ficheiros na diretoria **/setup-scripts/build/tmp-angstrom_v2012_12-eglibc/deploy/images/beagleboard/**.

Lista de ficheiros:

- **MLO-beagleboard-2011.12**
  - *X-Loader*
- **u-boot-beagleboard-2011.12-r8.img**
  - *Boot loader*
- **Angstrom-console-image-eglibc-ipk-v2012.12-beagleaboard.rootfs.tar.bz2**
  - Sistema de ficheiros do Ångström Linux (*root file system*)