Universidade Federal do Rio Grande do Sul
Escola de Engenharia
Departamento de Sistemas Elétricos de Automação e Energia
ENG10032 Microcontroladores

Roteiro de Laboratório 11
Compilação Cruzada de um *Kernel* para a Galileo

Prof. Walter Fetter Lages

12 de junho de 2019

1 Objetivo

O objetivo deste laboratório é compilar um *kernel* para a Galileo, utilizando o compilador cruzado, independentemente do Yocto.

2 Fundamentação Teórica

Embora com o Yocto se possa gerar toda uma distribuição personalizada para a Galileo, muitas vezes se deseja apenas fazer pequenas alterações ou adições à imagem disponibilizada pela Intel. Compilar todo um sistema Poky (aproximadamente 60GB) para isto não é nem um pouco prático, devido ao tempo e recursos necessários.

A forma de compilar programas no espaço do usuário já foi explorada nos laboratórios anteriores. Neste laboratório será compilado o *kernel* do Linux, o que permitirá fazer alterações sem ter que compilar todo o Poky. Também possibilitará adicionar novos *drivers* ou trocar a versão do *kernel* por uma mais recente.

2.1 Pacotes Necessários e *Patch* do *Kernel*

Será utilizado um *kernel* original[1], disponível em `<https://www.kernel.org/pub/linux/kernel>`.

Embora o *kernel* do Linux original executa na Galileo sem problemas, mesmo nas versões mais recentes ele não inclui os *drivers* para todos os dispositivos existentes na Galileo Gen2 e portanto estes dispositivos não poderiam ser usados.

[1] Algumas distribuições Linux vem com *kernel* modificado. É aconselhável baixar um original.

Os *drivers* para estes dispositivos podem ser incluidos no *kernel* através de uma série de *patches* disponibilizados pela Intel no *Board Support Package* (BSP) da Galileo, disponível em <https://downloadcenter.intel.com/download/23823>. Estes arquivos de *patch* nada mais são do que o resultado de um `diff` entre uma versão do *kernel* modificada e o *kernel* original. Assim, partir de uma copia do *kernel* original e dos arquivos de *patch* o utilitário `patch` pode gerar uma cópia do *kernel* modificado.

2.2 Configuração do *Kernel*

Esta é a etapa mais *tricky* do processo. Configurar o *kernel* significa escolher quais subsistemas, *drivers*, sistemas de arquivos, etc, serão compilados e farão parte do arquivo de *boot*, quais serão compilados como módulos e quais não serão compilados porque nunca serão utilizados.

Geralmente, as distribuições configuram o *kernel* com inúmeras funcionalidades que dificilmente serão utilizadas na maioria dos sistemas. Embora essas funcionalidades sejam, em sua maioria, compiladas como módulos e portanto apenas ocupem espaço em disco, a compilação de todos estes módulos faz com que a compilação do *kernel* torne-se bastante demorada. Assim, o ideal é configurar o *kernel* para que sejam compilados apenas os módulos que serão utilizados, ou que tem alguma possibilidade de virem a ser utilizados.

Para fazer esta configuração de forma adequada é necessário conhecer o *hardware* da máquina onde o *kernel* vai executar e quais serviços do *kernel* o *software* que será executado necessitará. Obviamente, isto requer um bom conhecimento do funcionamento do Linux e do *hardware* para o qual o *kernel* será compilado.

Uma alternativa é copiar a configuração do *kernel* que está executando na Galileo e fazer as alterações necessárias. Esta configuração está disponível na forma compactada no pseudo-arquivo `/proc/config.gz`.

A forma mais robusta de alterar a configuração do *kernel* é através do comando `make menuconfig`. Este comando executa um aplicativo com interface baseada na biblioteca `ncurses` que permite a configuração do *kernel* através de *menus* hierárquicos.

Por *default*, a configuração do *kernel* é salva no arquivo `.config` dentro da árvore do *kernel*. Este arquivo de configuração é utilizado pelas etapas subsequentes de compilação.

Uma boa prática é manter-se uma cópia do arquivo com a configuração do *kernel* em um diretório fora da árvore do *kernel*, de forma que ele não seja destruído acidentalmente ao trocar-se o código-fonte do *kernel* por outra versão.

Em especial, para configurar um *kernel* para uso com a Galileo, deve-se habilitar a opção **Intel Quark platform support** no sub-menu **Processor type and**

features e selecionar **Pentium-Classic** na opção **Processor family** do mesmo sub-menu.

2.3 Compilação do *Kernel* e dos Módulos

Esta etapa é relativamente simples, embora demorada. Basta executar os comandos para compilar o *kernel* e os módulos e esperar. Após a compilação a imagem de *boot* do *kernel* estará no arquivo `arch/<arquitetura>/boot/bzImage`. Note que os módulos podem ser compilados independentemente do arquivo de *boot* do *kernel*. Assim, se for necessário reconfigurar o *kernel* para incluir outros módulos, basta recompilar os módulos. Por outro lado, a reconfiguração de um subsistema que estava compilado na imagem de *boot* do *kernel* para módulo, ou vice-versa requer a compilação tanto da imagem de *boot* do *kernel* quanto dos módulos.

2.4 Instalação do *Kernel* e Módulos

Na Galileo, os arquivos de *boot* do *kernel* ficam em `/media/card`, que é onde é montada a primeira partição do microSD. Pode-se ter vários arquivos de *boot* neste diretório, de forma a poder escolher entre eles através do gerenciador de *boot* (no caso da Galileo, o GRUB). Para diferenciar as diferentes versões do arquivo de *boot* inclui-se no nome do arquivo a versão do *kernel* correspondente, de forma que o nome do arquivo assume a forma `bzImage-<versao>`.

Os módulos do *kernel* são instalados em `/lib/<versao>`.

2.5 Gerenciador de *Boot*

O gerenciador de *boot* utilizado pela Galileo é o GRUB. Para qualquer alteração deve-se editar o arquivo `/media/card/boot/grub/grub.conf`[2].

2.6 Parâmetros do *Kernel*

Em alguns casos é necessários passar parâmetros de configuração para o *kernel* durante a sua carga. A forma de passar tais parâmetros depende do gerenciador de *boot* utilizado. No GRUB, os parâmetros são simplesmente passados após o nome do arquivo com a imagem do *kernel* no arquivo `/media/card/boot/grub/grub.conf`.

[2]Normalmente, o arquivo de configuração é `/boot/grub/menu.lst`, mas na Galileo a localização *default* foi alterada.

3 Experimentos

Nesta seção estão as instruções passo-a-passo para compilação cruzada do *kernel*. Será utilizada a versão 3.8.7 do *kernel*, os *patches* disponíveis no BSP 1.1.0[3] e o compilador cruzado disponível no *kit* da Intel. Versões mais recentes do *Kernel*, (4.x, por exemplo) também podem ser compiladas para a Galileo, mas como já comentado, nem todos os *drivers* já foram portados. A versão do *kernel* suportada pelo BSP 1.1.0 é a 3.8.7.

1. Normalmente o código-fonte do *kernel* deveria ser baixado[4] e descompactado[5]. No entanto, como o código fonte do *kernel* é relativamente grande (555 MB), para evitar congestionamento na rede com todos os alunos baixando a sua cópia ao mesmo tempo, ele já está instalado em `~/src/lab11/linux-3.8.7`.

2. Baixar o arquivo BSP e descompactar os *patches* do *kernel*:

```
wget https://downloadmirror.intel.com/23823/eng/\
bsp_sources_and_docs_for_intel_quark_v1.1.0.zip
unzip bsp_sources_and_docs_for_intel_quark_v1.1.0.zip *.7z
7z e Board_Support_Package_Sources_for_Intel_Quark_v1.1.0.7z quark*.tar.gz
tar -xzf quark_linux_v3.8.7+v1.1.0.tar.gz
```

3. Fazer o patch do kernel:

```
cd ~/src/lab11/linux-3.14
for i in {0001..0026} ; do
        patch -p1 < ../quark_linux_v3.8.7+v1.1.0/$i-*.patch ;
done
```

4. Copiar a configuração do *kernel* que está executando na Galileo e descompactar no diretório onde está o código-fonte do Linux:

```
cd ~/src/lab11/linux-3.8.7
scp <login@galileo>:/proc/config.gz .
zcat config.gz > .config
rm config.gz
```

   onde `<login@galileo>` é o seu *login* e nome da Galileo em uso.

5. Ajustar as variáveis de ambiente para a arquitetura x86 e o `PATH` do sistema para localizar o compilador cruzado:

[3] As versões posteriores do BSP possuem *bugs* e os *patches* não são aplicados corretamente.

[4] O código-fonte do *kernel* pode ser baixado com o comando: `wget https://www.kernel.org/pub/linux/kernel/v3.x/linux-3.8.7.tar.xz`.

[5] Com o comando: `tar -xJf linux-3.8.7.tar.xz`

```
export ARCH=x86
export DEVKIT=/opt/iot-devkit/devkit-x86
export POKYSDK=$DEVKIT/sysroots/x86_64-pokysdk-linux
export PATH=$PATH:$POKYSDK/usr/bin/i586-poky-linux
export CROSS_COMPILE=i586-poky-linux-
```

6. Configurar o *kernel*:
   (a) Executar o *script* de configuração:
   ```
   make menuconfig
   ```
   (b) No sub-menu **General setup**:
      i. Troque o nome da **Local version** para `-eng10032`
      ii. Selecionar **Exit** para voltar ao nível anterior
   (c) No sub-menu **Processor type and features**:
      i. Habilitar **Intel Quark platform support**
      ii. Ajustar **Processor family** para Pentium-classic
      iii. Selecionar **Exit** para voltar ao nível anterior
   (d) No sub-menu **Device Drivers**:
      i. No sub-menu **Misc Devices**, **EEPROM support**:
         A. Habilitar **I2C EEPROMS / RAMs / ROMs from most vendors** como módulo
         B. Selecionar **Exit** 2 vezes para voltar ao nível anterior
      ii. No sub-menu **Multifunction device drivers**:
         A. Habilitar **Intel Quark MFD for High Speed UART with DMA Engine**
         B. Selecionar **Exit** para voltar ao nível anterior
      iii. No sub-menu **Industrial I/O support**, **Analog to digital converters**:
         A. Habilitar **Texas Instruments ADC1x8S102** como módulo
         B. Selecionar **Exit** para voltar ao nível anterior
      iv. Selecionar **Exit** para voltar ao nível anterior
   (e) Salvar a configuração no arquivo `.config`[6], usando a opção **Save**.
   (f) Sair do *script* de configuração.

[6]Algumas versões do *script* de configuração não salvam corretamente o arquivo `.config` na saída. Assim, é aconselhável salvá-lo explicitamente.

7. Compilar o *kernel*

```
make -j <n> bzImage
```

onde `<n>` é o número de *cores* do processador *host*. As máquinas do laboratório tem 2 *cores*, portanto use a opção `-j 2`. A opção `-j <n>` é opcional. Se não for usada, apenas 1 *core* será usado na compilação.

8. Compilar os módulos do *kernel*:

```
make -j <n> modules
```

onde `<n>` é o número de *cores* do processador.

9. Instalar o arquivo de *boot* do *kernel*:

```
scp arch/x86/boot/bzImage root@<galileo>:/media/card/bzImage-3.8.7-eng10032
```

onde `<galileo>` é o nome da Galileo em uso.

10. Instalar os módulos

```
INSTALL_MOD_PATH=~ make modules_install
cd ~
rm lib/modules/3.8.7-eng10032/build
rm lib/modules/3.8.7-eng10032/source
scp -r lib root@<galileo>:/
```

onde `<galileo>` é o nome da Galileo em uso.

11. Atualizar a configuração do gerenciador de *boot*: Editar (como superusuário) o arquivo `/media/card/boot/grub/grub.conf` na Galileo e incluir no final do arquivo:

```
title Kernel de teste
    root (hd0,0)
    kernel /bzImage-3.8.7-eng10032 root=/dev/mmcblk0p2 rootwait \
console=ttyS1,115200n8 earlycon=uart8250,mmio32,0x8010f000,115200n8 \
reboot=efi,warm apic=debug rw LABEL=boot debugshell=5
```

12. Conecte o cabo FTDI à Galileo e abra um emulador de terminal no *host* com o comando:

```
minicom -D <dispositivo> -b 115200 -o
```

onde `<dispositivo>` é o dispositivo tty da porta serial (ou conversor USB/FTDI).

13. Em outro terminal do *host* desative o controle de fluxo por *hardware*, pois o GRUB não usa controle de fluxo, com o comando:

```
stty -F <dispositivo> -crtscts
```

O controle de fluxo também pode ser desativado através do menu do `minicom` (CTRL-a z o). Infelizmente, o `minicom` reabilita o controle de fluxo por *hardware* ao ser inicializado, portanto é necessário desativa-lo após a inicialização do `minicom`.

14. No terminal da Galileo, reinicie o sistema, execute como superusuário:

```
reboot
```

15. Observe no console serial o *boot* da Galileo. Na tela do gerenciador de *boot* selecione a opção Kernel de teste.

16. Após logar-se na Galileo, utilize o comando

```
uname -a
```

para verificar se o *kernel* em execução é o que foi compilado.