Tutorial HPS Kernel

e executando o kernel do Linux

Nesse tutorial iremos compilar o kernel do Linux para o ARM na nossa FPGA, usando o toolchain que já temos configurado.

Kernel

Clone o kernel do linux fornecido pela altera/intel para a nossa pasta `work`:

```
$ git clone https://git.kernel.org
$ cd
```

Kernel 4.4

Vamos trabalhar com a versão `4.4` do kernel que é uma versão com: `Long Time Suport` (LTS), ou seja, será mantida por muito mais tempo que as outras versões. A versão 4.4 foi lançada em 10 de Janeiro e será mantida oficialmente até 2021, ela é também a versão Super LTS, com suporte estendido até 2036.

> **Linux**
>
> |

Pense que um desenvolvedor de um sistema embarcado, que vai criar toda uma infra dedicada não quer ficar ter que ajustando e validando tudo novamente só para ter a versão mais nova do kenrel. A ideia de usar uma com maior suporte é minimizar esforços.

O kernel utiliza o sistema de `tag` do git:

```
$ git tag
...
v2.6.11
```

```
v2.6.11-tree
v2.6.12
v2.6.12-rc2
v2.6.12-rc3
v2.6.12-rc4
v2.6.12-rc5
v2.6.12-rc6
v2.6.13-rc2
v2.6.14-rc2
v2.6.14-rc3
...
```

Note que revisões ímpares são para Karnel em estágio de desenvolvimento, e números pares para versão de produção, exemplo :

- **Linux 2.4.x** - Produção
- **Linux 2.5.x** - Desenvolvimento
- **Linux 2.6.x** - Produção
- ....

Vamos criar um branch da versão v4.4, para isso execute o comando a seguir:

```
$ git checkout v4.4
$ git checkout -b 4.4-SoC
```

Uma vez no branch `4.4-SoC` precisamos configurar o kernel para nosso processador (ARM) e fazer as configurações necessárias no kernel. Primeiramente iremos gerar um arquivo de configuração `.config` padrão para SoCs ARM Altera:

```
$ export ARCH=arm        # indica a arquitetura do Kernel
$ make socfpga_defconfig # gera o arquivo padrão de configuração
para soc
```

Agora vamos configurar o Kernel do Linux para a nossa aplicação :

```
$ make ARCH=arm menuconfig
```

> **Note**
>
> Talvez seja necessário instalar o pacote **libncurses5-dev**

Esse comando irá abrir a interface de configuração do Kernel do Linux (existem outras opções: `make xconfig`; `make config`; `make gconfig`, ...). Essa interface permite selecionarmos várias configurações do Kernel. Agora iremos seguir o roteiro proposto no tutorial a seguir, traduzido de maneira reduzida nesse tutorial.

- https://rocketboards.org/foswiki/Documentation/EmbeddedLinuxBeginnerSGuide

Configurando

1. Automatically append version information to the version string
2. General Setup ➡️
    - **Desabilite** : *Automatically append version information to the version string*

> - ref : https://rocketboards.org/foswiki/Documentation/EmbeddedLinuxBeginnerSGuide#8 Go into the “General Setup” menu. Uncheck “Automatically append version information to the version string”. This will prevent the kernel from adding extra “version” information to the kernel. Whenever we try to dynamically load a driver (also called kernel modules, as discussed in a later section) the kernel will check to see if the driver was built with the same version of the source code as itself. If it isn’t, it will reject to load that driver. For development, it’s useful to disable these options to make it easier to test out different versions of drivers. In a production system however, it’s recommend to keep this option enabled and only use drivers that were compiled with the correct version of the kernel. I encourage you to peruse the options in the General Setup menu and see what’s available to you (hitting “?” to view the help info for the highlighted option). Of particular importance to us is the “Embedded System” option (turns on advanced features) and the type of SLAB allocator used (determines how memory will be dynamically allocated in the kernel). If you want to use an initial ram disk or ram filesystem that would be enabled here as well (these will be explained in the next section). (texto extraído da referência)

---

1. *Enable loadable module support*

2. Volte para o menu principal (ESC ESC) ➡️
    - Note que o *Enable loadable module support* está ativado.

> Isso permite que o kernel seja modificado (pelo carregamento de drivers) após a sua execução. Isso será útil quando formos desenvolver nosso próprio device driver, sem a necessidade de recompilarmos o kernel toda vez que desejamos testar uma modificação no código. É essa configuração que permite utilizarmos USBs, SSDs, placas de rede via a possibilidade do carregamento de drivers de forma dinâmica pelo sistema operacional.

---

1. Support for large (2TB+) block devices and files**
2. No menu principal ➡️ Enable the block layer
    - **Ative** : Support for large (2TB+) block devices and files

> Essa opção irá permitir a utilização de partições do tipo EXT4. Se esquecer essa opção e o kernel tiver em uma partição EXT4 a mesma será montada como READ-ONLY.

---

1. The Extended 4 (ext4) filesystem
2. Menu principal ➡️ File systems
    - **Ative** : The Extended 4 (ext4) filesystem

> Essa opção irá possibilitar que o kernel monte dispositivos formatados em EXT4. Pretendemos usar isso no SDCARD.

---

1. Altera SOCFPGA family
2. Menu principal ➡️ System Type
    - **Ative** : Altera SOCFGPA family

> Isso indica para o kernel qual será o dispositivo que o mesmo será executado, note que esssa opção possui um novo menu onde podemos ativar ou não a suspensão para RAM.

1. Symmetric Multi-Processing
2. Menu principal ➡️ Kernel Features
    - **Ative**: Symmmetric Multi-Processing

> Essa opção indica para o kernel que ele deve utilizar os dois cores presente no ARM HPS da FPGA.

---

1. Device Drivers
2. Menu principal ➡️ Device Drivers

> Indica quais drivers serão compilados junto com o kernel, note que já temos configurado drivers de rede (Network device support); GPIO (GPIO Support); RTC; DMA; ... . Lembre que já inicializamos o `.config` com uma configuração padrão para SoCs Altera.

Salvando

Aperte ESC duas vezes (ESC ESC) e salve as configurações no arquivo `.config`

Compilando

O makefile utiliza a variável `CROSS_COMPILE` para definir o toolchain que irá fazer a compilação do kernel, vamos definir como sendo o GCC do Linaro baixado recentemente:

```
$ export CROSS_COMPILE=$GCC_Linaro/arm-linux-gnueabihf-
```

Para compilarmos o kernel :

```
make ARCH=arm LOCALVERSION= zImage -j 4
```

Esse comando faz com que o kernel do linux seja compilado em uma versão compactada que é auto-extraída. Outras opções seriam :

- Image : Binário do kernel
- zImage: versão compactada que possui *self-extracting*
- uImage: uma versão que já possui o bootloader uboot

1^: https://stackoverflow.com/questions/22322304/image-vs-zimage-vs-uimage

o zImage é salvo em :

- `arch/arm/boot/zImage`

Testando

Agora devemos atualizar o kernel que está no SDCard, para isso segue o tutorial no `/info-SDcard.md/` : Atualizando o kernel

Executando

Para verificar se tudo está certo, basta colocar o cartão de memória no kit e verificar a versão do kernel em execução:

```
$ uname -a
Linux buildroot 4.14.0 #1 SMP Mon Jul 16 21:22:58 -03 2018 armv7l
GNU/Linux
```