Desenvolvimento de Sistemas Linux Embarcados

Exercícios Práticos

Guilherme Brandão da Silva

brandaogbs.github.io

2 de março de 2020

Sobre o Documento

1. Este documento é baseado noa materiais de treinamentos disponibilizados pela Embedded Labworks e Free Electrons em:
   https://e-labworks.com/treinamentos/linux/slides e
   https://bootlin.com/doc/training/embedded-linux/
2.
3. Os fontes serão liberados, ao final do curso, sob mesma licença, no link:
   http://brandaogbs.github.io/linux/slides

Sumário

Lista de Abreviações

RPi	Raspberry Pi;

Lista de comandos mais utilizados no shell[1]

cat	*concatenate* - exibir conteúdo de arquivos;
cd	*change directory* - navegar entre diretórios;
cp	*copy* - copiar arquivos;
cp -r	*copy recursively* - copiar pastas;
echo	*"echo" string* - utilizado para exibir/enviar strings de texto;
ls	*list* - lista o conteúdo de diretórios;
mkdir	*make directory* - criar um diretório;
mv	*move* - mover ou renomear arquivos e diretórios;
rm	*remove* - remover arquivos;
rm -r	*remove recursively* - remover diretórios;

Auxiliares de navegação[2][3]

/	Representa o diretório raiz;
~/	Representa o diretório *home* do usuário logado;
.	Representa o diretório atual;
..	Representa o diretório anterior ou um nível acima;

[1] A lista acima apresenta as funções mais básicas ou mais utilizadas de cada comando. Muitos deles possuem diversas funções diferentes e, para uma melhor compreensão, recomenda-se a leitura de seu manual. Ex: *man cat*.

[2] Experimente testar os auxiliares seguidos do comando ls. Ex: ls ~/.

[3] Adicionalmente, teste o efeito da tecla TAB (função de autocompletar) enquanto navega entre os diretórios.

1 Conhecendo o Terminal

Durante a primeira prática iremos estudar alguns comandos e conceitos básicos da linha de comandos do Linux, necessários para um melhor aproveitamento e entendimento do treinamento.

1.1 Linha de Comando Básico

O **shell** é uma interface de linha de comandos entre o usuário e o sistema operacional. No Ubuntu, ele pode ser acessado pressionando a tecla Super e digitando "Terminal" ou pelas teclas de atalho *CTRL+ALT+T*.

Sua aparência inicial é uma tela preta com um cursor piscando para a entrada de comandos. Dentro desta janela, você pode abrir outras abas e trabalhar nelas simultaneamente, acessando o menu "*File-> Open Tab*" ou pelas teclas de atalho *CTRL+SHIFT+T*.

1.2 Diretórios e Arquivos

Windows e Linux são sistemas operacionais com conceitos totalmente diferentes. Enquanto que no Windows temos diversas árvores de diretório, cada uma associada à um dispositivo (*C:*, *D:* e etc), no Linux temos apenas uma árvore de diretórios "/", chamada de diretório root ou diretório principal. Perceba que inclusive a orientação da barra é diferente.

Outra diferença importante é que o Linux é "case sensitive", ou seja, diferencia maiúsculas de minúsculas. Portanto, "*Texto.txt*" é diferente de "*texto.txt*".

Alguns comandos básicos para manipulação de arquivos:

- *cd*: Mudar de diretório
- *mkdir*: Criar diretório
- *rmdir*: Remover diretório vazio
- *rm*: Remover arquivo
- *rm -R*: Remover diretóros e subdiretórios recursivamente
- *ls*: Listar arquivos e diretórios
- *cp*: Copiar arquivos
- *mv*: Mover/renomear arquivo
- *cat*: Listar conteúdo do arquivo

No Linux, enquanto que o diretório "/" representa o diretório principal (raiz), um ponto "." representa o diretório atual, e dois pontos ".." representam o diretório anterior. Exemplos:

Listando o diretório principal

```
$ ls /
```

Listando o diretório atual:

```
$ ls .
```

Listando o diretório anterior:

```
$ ls ..
```

Exercício 1: Entre na pasta de arquivos temporários "*/tmp*". Crie um diretório "*testedir*" e entre nele. Faça uma cópia do arquivo "*/etc/resolv.conf*" para este diretório. Liste o diretório e o conteúdo deste arquivo. Renomeie este arquivo para "*resolv.txt*". Liste novamente o conteúdo do arquivo. Apague o arquivo, volte ao diretório "*/tmp*" e apague o diretório "*testedir*".

Obs: Você pode usar as teclas de seta para cima e para baixo para navegar entre os comandos já digitados. O comandos "history" também exibe os últimos comandos executados. A tecla *TAB* ajuda a preencher de forma inteligente enquanto você esta digitando um comando.

1.3 Variáveis de Ambiente

Todo **shell** do Linux possui um ambiente de execução com algumas variáveis configuradas. Você pode imprimir estas variáveis com o comando *printenv*.

Um variável importante é o *PATH*, que possui todos os caminhos por onde o shell irá procurar um binário para execução. Aqui é o mesmo conceito usado em máquinas Windows para variáveis de ambiente.

Você pode usar o comando *echo* para imprimir uma variável ambiente:

```
$ echo $PATH

/home/gbs/bin:/home/gbs/.local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/
    local/games:/snap/bin
```

Os valores são separados por dois pontos "*:*". Neste exemplo, quando você tentar executar uma aplicação na linha de comandos, o shell irá procurar esta aplicação nos diretórios */usr/sbin*, */usr/bin*, */bin* e */sbin*.

Se sua aplicação não estiver em um dos diretórios do PATH, você pode executá-la digitando o caminho completo:

```
/home/user/bin/app
```

Ou então entrar no diretório da aplicação e executar usando "*./*" conforme abaixo:

```
cd /home/user/bin/
./app
```

Para alterar ou criar uma nova variável de ambiente, você precisa usar o comando *export*:

```
export VAR=value
```

E para remover você pode usar o comando *unset*, conforme abaixo:

```
unset VAR
```

Exercício 2: Crie, imprima e remova uma variável de ambiente.

1.4 Editor de Texto

Existem dezenas de editores de texto disponíveis no Linux. Um dos mais usados é o **vi**, ou sua versão mais avançada, o **vim**.

Apesar de altamente produtivo, o **vi** possui uma curva de aprendizado relativamente grande, o que pode assustar um pouco quem esta começando a aprender Linux. Uma outra solução menos agressiva é o **nano**, que também possui diversas funcionalidade e permite a edição de arquivos dentro do terminal. Por fim, uma escolha gráfica pode ser o **gedit**.

1.5 Sheel Script

Shell script tem o mesmo conceito de arquivos *.bat* no Windows. Possui uma linguagem de programação própria, e permite automatizar diversas tarefas que de outra forma seriam repetitivas.

Um shell script normalmente tem na primeira linha o nome do programa interpretador, seguido pelos comandos a serem executados. Exemplo:

```
#!/bin/bash

mkdir -p /tmp/testedir
cp /etc/resolv.conf /tmp/testedir/
```

Um shell script precisa de permissão para execução. Portanto, após criar um shell script, você precisa dar esta permissão com o comando abaixo:

```
chmod u+x script.sh
```

Exercício 4: Crie um shell script dentro do diretório */tmp* para printar uma mensagem no terminal. Execute-o e depois remova-o.

1.6 Dispositivos

No Linux, boa parte dos dispositivos do sistema são representados através de arquivos, incluindo o hardware, cujo acesso é abstraído através de arquivos de dispositivo.

Ou seja, se uma aplicação quer enviar um byte pela porta serial, ela deve escrever este byte no arquivo correspondente à porta serial.

Os arquivos de dispositivo no Linux ficam no diretório "*/dev*", que tem a seguinte correspondência com relação aos dispositivos no Windows:

Tipo	Windows	Linux
Portal Serial	COM1	*/dev/ttyS0*
Porta Paralela	LPT1	*/dev/lp0*
HD	C:	*/dev/sda1*
CDROM	D:	*/dev/sr0*

1.7 Ponto de Montagem

Dispositivos de armazenamento, para serem utilizados, devem ser montados em um diretório do sistema operacional. Este diretório é chamado de ponto de montagem.

Por exemplo, ao inserir o pendrive em uma porta USB, será criado um arquivo de dispositivo (Ex*: /dev/sdc1*). Este dispositivo deve ser montado com o comando mount em um diretório do sistema para que se possa acessar os arquivos do pendrive:

```
sudo mount /dev/sdc1 /mnt/pendrive
```

No exemplo acima, o pendrive estará acessível através do diretório */mnt/pendrive*. Perceba a necessidade do comao entrar no diretório da ao entrar no diretório dao entrar no diretório da ao entrar no diretório dando sudo antes do mount, já que o mount requer permissões de root para execução.

Após o uso, o pendrive deve ser desmontado com o comando:

```
sudo umount /dev/sdc1
```

Obs: Cuidado! Se você não desmontar, corre o risco de perder as alterações realizadas no pendrive!

O nome atribuído ao seu dispositivo é normalmente dinâmico. Existem algumas técnicas e ferramentas disponíveis que podem nos ajudar a descobrir o nome do arquivo associado ao dispositivo.

Uma delas é usando a ferramenta *fdisk*, conforme abaixo:

```
sudo fdisk -l

Disk /dev/sda: 447,1 GiB, 480103981056 bytes, 937703088 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: gpt
Disk identifier: 0E18D5E1-5210-47B3-A1FB-73F4C8699FEF

Device          Start       End   Sectors   Size Type
/dev/sda1        2048   1050623   1048576   512M EFI System
/dev/sda2     1050624 935700479 934649856 445,7G Linux filesystem
/dev/sda3  935700480 937701375   2000896   977M Linux swap


Disk /dev/sdb: 7,5 GiB, 8086618112 bytes, 15794176 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0xe0e0a313

Device     Boot Start      End  Sectors  Size Id Type
/dev/sdb1        2048 15794175 15792128  7,5G  c W95 FAT32 (LBA)
```

Você pode também listar o arquivo */proc/partitions*:

```
cat /proc/partitions

major minor  #blocks  name

   8        0  468851544 sda
   8        1     524288 sda1
   8        2  467324928 sda2
   8        3    1000448 sda3
   8       16    7897088 sdb
   8       17    7896064 sdb1
```

Outra forma é usando olhando as mensagens do kernel com o comando *dmesg*. Para isso, conecte o dispositivo, execute o comando *dmesg* e olhe as últimas linhas da saída do comando.

Estas foram atividades básicas para conhecer o shell e o ambiente Linux. É extremamente aconselhável um estudo mais completo usando o Guia Foca Linux ou outro livro qualquer que explique bem o funcionamento de sistemas operacionais baseados em Linux.

- **Guia Foca Linux** – documento "*GuiaFocaLinux.pdf*" disponível em "*docs/guides/*".
- **Canivete suíço do Shell** – documento "*canivete-shell.pdf*" disponível em "*docs/guides/*".
- **Guia de referência do vi** – documento "*vi.pdf*" disponível em "*docs/guides/*".
- **Guia de referência do Shell** – documento "*shell.pdf*" disponível em "*docs/guides/*".

2 Configuração Inicial

Devido ao limitado tempo de aula em função da quantidade de conteúdo a ser visto, seria inapropriado realizar o download das diversas ferramentas, códigos fontes, documentos, dentre outros, necessários para realização das atividades práticas deste treinamento. Assim, todo o material julgado necessário para a realização das atividades foi previamente baixado e compilado em um único tarball.

O objetivo desta atividade é extrair o conteúdo desse tarball, analisa-lo e familiarizar-se com o ambiente de trabalho que será utilizado ao longo de todo o treinamento.

2.1 Configurando o ambiente de trabalho

Faça o download e extraia o arquivo *dsle20.tar.xz*:

```
$ cd
$ wget http :// brandaogbs.github.io/linux/dsle20.tar.xz
$ tar  xvf  dsle20.tar.xz
```

Ao finalizar o procedimento de extração, todo o conteúdo do tarball estará disponível na recém criada pasta *dsle20*, dentro do diretório *home* do usuário logado *(~/dsle20)*. A pasta *dsle20* possui a seguinte estrutura:

...

```
dsle20
├── dl
├── docs
│   ├── guides
│   ├── hardware
│   └── training
└── exs
```

O diretório ***dl*** contém todos os componentes que foram baixados. Durante a realização dos exercícios, os comandos comuns de download (*wget*) e clonagem de repositório (*git clone*) serão substituídos pelo processo de extrair esses componentes. Cada atividade conterá as informações necessárias para realização de tal procedimento.

O diretório ***docs*** contém uma documentação de apoio para consulta durante o treinamento. Neste diretório, estão contidos pdfs relacionados ao hardware da RaspberryPi, tabelas de comandos do shell, vi, guia introdutório do Ubuntu e kernel Linux, por exemplo.

O diretório ***training*** contém a Agenda, Slides e Atividades de Laboratório (este documento).

O diretório ***exs*** será dedicado a realização dos exercícios. A fim de manter uma estrutura organizada, cada exercício será realizado dentro de seu próprio diretório .

3 Cross-Compiling Toolchain

Os objetivos desta atividade são:

1. Aprender a utilizar um toolchain pronto, isto é, fornecido pelo fabricante do SoC ou repositório padrão por exemplo;
2. Compilar seu próprio toolchain utilizando a ferramenta crosstool-ng.

3.1 Utilizando um toolchain pré-compilado

A Raspberry Pi Foundation fornece gratuitamente em seu repositório oficial https://github.com/raspberrypi, diversos componentes open-source para a Raspberry Pi. Dentre eles, estão inclusos os fontes do kernel Linux, drivers, bibliotecas e firmwares específicos, além de seu próprio toolchain, o qual será utilizado nesta atividade.

3.1.1 Instação e Configuração

Inicialmente, abra um novo shell e crie a seguinte estrutura de diretórios dentro do diretório *dsle20*:

```
$ cd  ~/dsle20
$ mkdir toolchains
$ cd toolchains
```

Em seguida, o correto seria clonar a pasta *tools* do repositório oficial da Raspberry Pi, no intuito de obter a última versão disponível do toolchain:

```
$ git clone https :// github.com/raspberrypi/tools.git
```

No entanto, a fim de garantir uma maior eficiência na execução das atividades deste treinamento, todos os arquivos necessários foram previamente baixados e encontram-se na pasta *~/dsle20/dl*. Portanto, substituiremos o processo de clonar o repositório *tools* pela extração do seguinte arquivo:

```
$ unzip ~/dsle20/dl/toolchains/tools.zip
```

O comando acima extrai o arquivo **tools.zip** no diretório atual. Ao final do processo, verifique se o diretório *tools* foi criado, por meio do comando *ls*.

Como o toolchain fornecido pela RPi Foundation é pré-compilado, para instalá-lo basta adicionar o caminho das ferramentas (binários) do toolchain à variável de ambiente $PATH:

```
$ export PATH=$PATH:~/dsle20/toolchains/tools/arm-bcm2708/gcc-linaro-arm-linux-gnueabihf-raspbian/bin
```

Note que tal alteração é momentânea, isto é, ao fechar o shell atual ela será perdida. Um dos meios de fazer essa alteração permanente (opcional) é por meio da inserção do mesmo comando no final do arquivo *~/***.bashrc**, que é um script shell que é executado sempre que um novo shell é iniciado.

Simples assim, toolchain instalado! Uma maneira de verificar se o *export* do PATH funcionou é digitar **arm** no shell e pressionar a tecla TAB duas vezes. Ao fazer isso, o shell tentará auto-completar o comando **arm** de acordo com as opções de executáveis disponíveis na variável PATH. O resultado deverá ser uma lista semelhante à:

```
gbs@dsle20:~/dsle20/toolchains$ arm

arm2hpdl                        arm-linux-gnueabihf-gfortran
arm-linux-gnueabihf-addr2line   arm-linux-gnueabihf-gprof
arm-linux-gnueabihf-ar          arm-linux-gnueabihf-ld
arm-linux-gnueabihf-as          arm-linux-gnueabihf-ld.bfd
arm-linux-gnueabihf-c++         arm-linux-gnueabihf-ldd
arm-linux-gnueabihf-c++filt     arm-linux-gnueabihf-ld.gold
arm-linux-gnueabihf-cpp         arm-linux-gnueabihf-nm
arm-linux-gnueabihf-dwp         arm-linux-gnueabihf-objcopy
arm-linux-gnueabihf-elfedit     arm-linux-gnueabihf-objdump
arm-linux-gnueabihf-g++         arm-linux-gnueabihf-pkg-config
arm-linux-gnueabihf-gcc         arm-linux-gnueabihf-pkg-config-real
arm-linux-gnueabihf-gcc-4.8.3   arm-linux-gnueabihf-ranlib
arm-linux-gnueabihf-gcc-ar      arm-linux-gnueabihf-readelf
arm-linux-gnueabihf-gcc-nm      arm-linux-gnueabihf-size
arm-linux-gnueabihf-gcc-ranlib  arm-linux-gnueabihf-strings
arm-linux-gnueabihf-gcov        arm-linux-gnueabihf-strip
arm-linux-gnueabihf-gdb
```

3.1.2 Configurando uma conexão de rede com a RPi

3.1.3 Testando o toolchain

É possível testar o toolchain através da compilação de um programa bem simples, como o **helloworld.c**, em C. Inicialmente, crie a pasta *ex01* dentro do diretório de exercícios *exs* e, em seguida, crie/edite o arquivo **helloworld.c** com seu editor favorito:

```
$ cd ~/dsle20/exs
$ mkdir ex01 && cd ex01
$ gedit helloworld.c
```

```c
#include <stdio.h>

int main()
{
  printf("Hello cross-compiled world!\n");
  return 0;
}
```

Em seguida, compile o arquivo **helloworld.c** nativamente, isto é, utilizando o *gcc*:

```
$ gcc helloworld.c -o hellox86
```

Utilize o comando *file* e verifique algumas informações sobre o binário gerado:

```
$ file hellox86
```

```
hellox86: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked, interpreter /lib/ld-linux.
    so.2, for GNU/Linux 2.6.32, BuildID[sha1]=4d1dbc14e02c87dbf1578335b77febef80b524a4, not stripped
```

Note que a segunda informação, Intel 80386 indica que o programa foi compilado para a mesma arquitetura de sua máquina, *x86*. Em seguida, execute-o:

```
$ ./hellox86

Hello cross-compiled world!
```

Agora repita o processo e utilize o recém-instalado toolchain para cross-compilar o mesmo programa para a Raspberry PI:

```
$ arm-linux-gnueabihf-gcc helloworld.c -o helloARM
```

Novamente, verifique o programa gerado através do comando *file* e veja que desta vez a arquitetura indicada será ARM:

```
$ helloARM: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynamically linked, interpreter /lib/ld-linux-
    armhf.so.3, for GNU/Linux 2.6.26, BuildID[sha1]=68bc6061bdbb1b80e5190760e91b687a1caebf97, not stripped
```

Não será possível executar este segundo programa na sua máquina pois o programa e a máquina possuem arquiteturas diferentes. Faça o teste. No entanto, é possível copiá-lo para a Raspberry Pi e executá-lo nela. Copie-o para a RPi através do comando *scp* (secure copy):

```
$ scp helloARM pi@10.1.1.100:/home/pi/helloARM

pi@10.1.1.100's password:
helloARM                                      100% 6018     5.9KB/s   00:00
```

Acesse-a via *ssh* e tente executar o programa:

```
$ ssh pi@10.1.1.100
pi@10.1.1.100's password:       // digite a senha padrao -> raspberry

# cd
# ./helloARM
Hello cross-compiled world!
```

3.2 Compilando seu próprio toolchain

A Crosstool-ng é provavelmente a principal ferramenta open-source para customização de toolchains. Suporta diversas arquiteturas diferentes, é capaz de gerar código bare-metal e possui uma interface baseada no menuconfig.

Nesta atividade, a ferramenta Crosstool-ng será utilizada para criar um toolchain customizado para a RPi. Este toolchain será utilizado para todas as atividades posteriores como compilação do bootloader, kernel, bibliotecas e aplicações.

3.2.1 Instalação e Configuração

Abra um novo shell e navegue até a pasta *~/dsle20/toolchains*. Em seguida, descompacte o arquivo *~/dsle20/dl/toolchains/crosstool-ng-1.23.0.tar.xz* no diretório atual e navegue até o novo diretório:

```
$ cd ~/dsle20/toolchains/
$ tar xvf ~/dsle20/dl/toolchains/crosstool-ng-1.23.0.tar.xz
$ cd crosstool-ng-1.23.0/
```

Após a descompactação, configure o crosstool-ng para ser instalado localmente, isto é, no mesmo diretório em que se encontra o script de configuração. Então, execute os comandos make e make install para realizar a instalação:

```
$ ./configure --enable-local
$ make
$ make install
```

Se tudo correu bem e nenhuma mensagem de erro foi exibida, o crosstool-ng pode ser executado a partir do diretório atual. O seguinte comando exibe as informações de ajuda da ferramenta:

```
$ ./ct-ng help
```

O Crosstool-ng permite criar um ou mais toolchains para arquiteturas diferentes e, por isso, fornece um conjunto de configurações pré-definidas para arquiteturas mais comuns. É possível listá-las por meio do seguinte comando:

```
$ ./ct-ng list-samples
```

Note que, dentre as configurações, existe uma pré-definida para a RPi 3: *armv8-rpi3-linux-gnueabihf*. Neste treinamento, essa configuração será usada como base para a criação do toolchain. Para tal, carregue-a por meio do comando abaixo:

```
$ ./ct-ng armv8-rpi3-linux-gnueabihf
```

Uma vez feito o *loading* da configuração, abra o menu de configurações do *crosstool-nt* e faça alguns ajustes:

```
$ ./ct-ng menuconfig
```

Na opção “Path and misc options” :

- **Number of parallel jobs:** 2 - Essa opção define a quantidade de threads de execução, que por sua vez diminui o tempo de compilação do toolchain.
- **Maximum log level to see:** DEBUG - Essa opção exibe informações a respeito do procedimento de *build* em tempo real. Assim, caso ocorra algum erro durante a compilação do toolchain, esse erro será exibido no shell.
- **CT_PREFIX**: Este é o diretório de instalação do toolchain. Altere-o **de:** ${CT_PREFIX:-${HOME}/x-tools}... **para:** ${CT_PREFIX:-${HOME}/dsle20/toolchains/x-tools}...

Na opção “Toolchain options” :

- **Tuple's alias:** Defina o alias como: arm-linux. Desta maneira, o crosstool-ng irá gerar o compilador como *arm-linux-gcc*, por exemplo. Caso contrário ele gera um nome extenso como: *armv8-rpi3-linux-gnueabihf-gcc*.

Na opção "Debug facilities" :

- **gdb:** Ative o gdb, pois será o debugger utilizado no restante do treinamento. Além disso, assegure-se de que as seguintes opções foram selecionadas:
  - Cross-gdb;
  - Build a static cross gdb;
  - gdbserver;

Salve suas alterações, saia do *menuconfig* e inicie o processo de compilação do toolchain:

```
$ ./ct-ng build
```

Ao final do processo, conforme configurado em CT_PREFIX, os binários do toolchain gerado estarão disponíveis no diretório: *~/dsle20/toolchains/x-tools/armv8-rpi3-linux-gnueabihf/bin/*. Dê um comando *ls* no diretório e verifique as ferramentas geradas.

3.2.2 Configuração do PATH

Como este toolchain será utilizado nas próximas atividades do treinamento, é necessário adicionar seus executáveis à variável de ambiente $PATH. Abra o arquivo ~/**.bashrc** com seu editor favorito:

```
$ gedit ~/.bashrc
```

Em seguida, adicione a seguinte linha no final do arquivo:

```
export PATH=$PATH:/home/gbs/dsle20/toolchains/x-tools/armv8-rpi3-linux-gnueabihf/bin/
```

Salve e feche o arquivo. Então basta carregar a configuração e, em seguida, realizar um pequeno teste para verificar se os binários foram adicionados corretamente à variável $PATH:

```
$ source ~/.bashrc
$ arm-linux-gcc --version

arm-linux-gcc (crosstool-NG crosstool-ng-1.23.0) 6.3.0
Copyright (C) 2016 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

3.2.3 Testando o toolchain

Recompile o programa da atividade 01, **helloworld.c**, em C com o novo toolchain gerado. Inicialmente, crie a pasta *ex02* dentro do diretório de exercícios *exs* e, em seguida, copie o arquivo **helloworld.c** da pasta *ex01* para a pasta *ex02*:

```
$ cd ~/dsle20/exs
$ mkdir ex02
$ cp ex01/helloworld.c ex02/
```

Navegue até a pasta *ex02* e compile o programa *helloworld.c* utilizando o *gcc* do toolchain recém-criado:

```
$ cd ex02
$ arm-linux-gcc helloworld.c -o helloARMctng
```

Novamente, verifique o programa gerado através do comando *file* e veja que, as informações exibidas como arquitetura e versão do kernel, estarão de acordo com as opções selecionadas no crosstool-ng:

```
$ file helloARMctng

helloARMctng: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynamically linked, interpreter /lib/ld-linux
    -armhf.so.3, for GNU/Linux 4.10.8, not stripped
```

Se desejar, repita o teste de rodar o executável na Raspberry Pi 3:

```
$ scp helloARMctng pi@10.1.1.100:/home/pi/helloARMctng

pi@10.1.1.100's password: #digite a senha padrao do raspbian-> raspberry
helloARMctng                                  100% 6018     5.9KB/s   00:00

# ssh pi@10.1.1.100
pi@10.1.1.100's password: #digite a senha padrao do raspbian-> raspberry

# cd
# ./helloARMctng

Hello cross-compiled world!
```

4 Bootloader - U-Boot

4.1 Configurando a comunicação serial com a RPi 3

Por padrão, a Raspberry Pi possui duas *built-in* UARTs, uma comum (PL011) e outra mini UART. Essa última, é limitada e carece de alguns recursos básicos como *break detection*, bit de paridade, dentre outros. Alem disso, a mini UART possui uma FIFO menor quando comparada à PL011 e sua *baudrate* é linkada ao clock da GPU, ou seja, a taxa de transmissão da mini UART varia de acordo com a oscilação do clock da GPU.

O módulo BT (bluetooth) da RPi 3 necessita de uma UART para funcionar e, por padrão, o BT da RPi 3 é conectado à UART PL011, tornando-a inutilizável para o usuário. No entanto, o firmware da GPU permite que o usuário escolha uma dentre as três opções:

- Desabilitar o módulo BT;
- Utilizar o módulo BT com a mini UART;
- Utilizar o módulo BT com a UART PL011;

A princípio, a utilização da mini UART para redirecionamento de console é suficiente, pois além da interação entre usuário e target via shell possuir um fluxo de dados baixo, existe a possibilidade de realizar a comunicação via rede (ssh). Portanto, será necessário apenas ativar o redirecionamento do console para a mini UART. Para tal, acesse a RPi via *ssh* e adicione as seguintes linhas no final do arquivo **config.txt**:

```
$ ssh pi@10.1.1.100
pi@10.1.1.100's password:  //digite  a senha padrao do raspbian-> raspberry
# sudo nano /boot/config.txt

 [..]
dtparam=audio=on
enable_uart=1
```

As opções *enable_uart*, *dtoverlay=pi3-miniuart-bt* e *core_freq=250*, ativam o redirecionamento do console do Linux via porta serial, remapeia o módulo BT com a mini UART e fixa a frequência do clock da GPU em 250 Mhz, respectivamente.

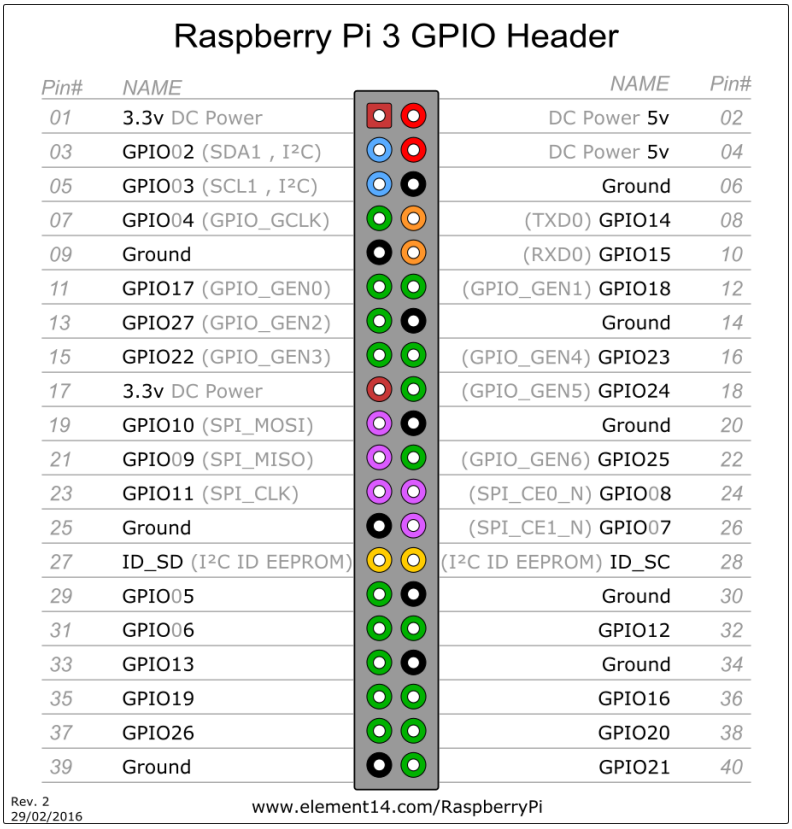
Obs: *para salvar as alterações feitas em um arquivo com o nano, utilize as teclas CTRL+X, Y, Enter*.

Para maiores informações, consulte a documentação da UART no site da RPi.

Após realizar a configuração acima, desligue a Raspberry Pi,

```
# sudo poweroff
```

e faça a conexão entre o adaptador USB-Serial (TTL) e a placa. Certifique-se de escolher a opção/jumper **3.3v** caso o adaptador permita. Por padrão, a RPi roteia os pinos GPIO14 e GPIO15 como TX e RX da serial principal (aquela que não está sendo utilizada pelo módulo BT), respectivamente. Esses GPIOS estão mapeados nos pinos 08 e 10, respectivamente, do conector geral de 40 pinos. Os pinos 01 e 02 de tal conector são aqueles fisicamente próximos da borda oposta às entradas USB. O pino 01 é o pino mais interno e o pino 02 é o mais externo, próximo da borda lateral. As figuras a seguir exibem a descrição completa do conector de 40 pinos e indicação dos pinos fisicamente. Após analisar as figuras, faça as três conexões, TX, RX e GND, entre seu adaptador e a RPi3.



Raspberry Pi 3 GPIO Header

Pin#	NAME		NAME	Pin#
01	3.3v DC Power		DC Power 5v	02
03	GPIO02 (SDA1 , I²C)		DC Power 5v	04
05	GPIO03 (SCL1 , I²C)		Ground	06
07	GPIO04 (GPIO_GCLK)		(TXD0) GPIO14	08
09	Ground		(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)		(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)		Ground	14
15	GPIO22 (GPIO_GEN3)		(GPIO_GEN4) GPIO23	16
17	3.3v DC Power		(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)		Ground	20
21	GPIO09 (SPI_MISO)		(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)		(SPI_CE0_N) GPIO08	24
25	Ground		(SPI_CE1_N) GPIO07	26
27	ID_SD (I²C ID EEPROM)		(I²C ID EEPROM) ID_SC	28
29	GPIO05		Ground	30
31	GPIO06		GPIO12	32
33	GPIO13		Ground	34
35	GPIO19		GPIO16	36
37	GPIO26		GPIO20	38
39	Ground		GPIO21	40

Rev. 2
29/02/2016

www.element14.com/RaspberryPi

Figura 1: RPi3 Pinout - Conector de 40 pinos. Fonte: [?]

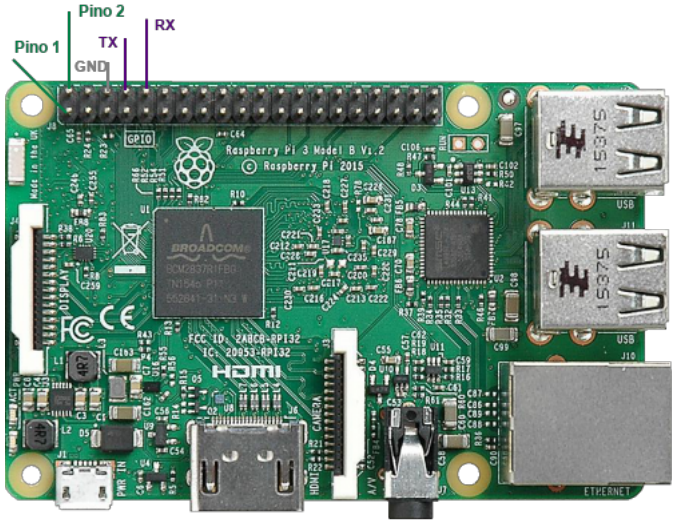


Figura 2: RPi3 - Conexões

4.1.1 Configurando Acesso ao USB no Ubuntu VM

Para acessar a Serial no Ubuntu com o usuário padrão (você), dois procedimentos são necessários: Primeiramente deve-se criar o grupo dialout, pois por padrão a serial pertencerá a este e grupo e, o grupo ainda não foi criado e nem você pertence a ele:

```
$ sudo groupadd dialout
```

O comando acima cria o grupo **dialout** no sistema. Então, é preciso adicionar você como um usuário deste grupo, ou seja, para que você tenha acesso à qualquer arquivo/dispositivo pertencente ao grupo:

```
$ sudo gpasswd -a nome_do_seu_usuario dialout
```

O comando acima adiciona o usuário *nome_de_usuario* ao grupo **dialout**. Para maiores informações sobre grupos e usuários: Users and Groups

Normalmente, existe uma opção para chavear um dispositivo de hardware entre o *host* (seu computador) e *guest* (máquina virtual). Verifique na sua aplicação da VirtualBox as opções ficam no menu *Devices>USB*. Encontre o dispositivo que você deseja colocar na máquina virtual e selecione.

4.1.2 Testando a comunicação serial

Conecte o adaptador serial na máquina de desenvolvimento e rode o comando *dmesg*. A saída será algo parecido com:

```
$ dmesg

[ 5331.599292] usb 1-3: new full-speed USB device number 9 using xhci_hcd
[ 5331.740103] usb 1-3: New USB device found, idVendor=1a86, idProduct=7523
[ 5331.740106] usb 1-3: New USB device strings: Mfr=0, Product=2, SerialNumber=0
[ 5331.740107] usb 1-3: Product: USB2.0-Serial
[ 5331.740607] ch341 1-3:1.0: ch341-uart converter detected
[ 5331.740888] usb 1-3: ch341-uart converter now attached to ttyUSB0
```

Procure pelo nome do arquivo criado pelo driver para manipular o dispositivo conectado. No caso acima, o nome do arquivo é **ttyUSB0**. Normalmente dispositivos seriais são mapeados como ttyUSB0, ttyS0, ou algo similar, e são criados dentro do diretório de dispositivos do Linux */dev*. Não se preocupe com detalhes a respeito de drivers e arquivos de dispositivo neste momento, essas informações serão vistas mais adiantes no treinamento.

Após identificar seu dispositivo, abra o *putty* e o configure da seguinte maneira:

- **Connection type:** Serial;
- **Serial line:** Nome do seu dispositivo. No exemplo acima: /dev/ttyUSB0;
- **Speed:** Este é o baudrate da conexão serial. Configure-o para: 115200;

Dica: clique em Default Settings e em seguida Save, de modo a salvar as configurações feitas e não precisar repeti-las sempre que iniciar o *putty*.

Clique em *Open* para iniciar uma nova conexão e, se tudo foi configurado corretamente, ao ligar a RPi 3, um console deverá ser exibido na janela do *putty*. Caso tenha ocorrido algum erro, peça ajuda ao instrutor.

4.2 Baixando o U-Boot

Assim como nas atividades anteriores, os fontes do U-Boot foram previamente baixados e encontram-se na pasta *~/dsle20/dl/bootloader*. Crie um diretório chamado *bootloader* dentro da pasta *~/dsle20* e, em seguida, extraia o U-boot no diretório criado:

```
$ cd ~/dsle20/
$ mkdir bootloader && cd bootloader
$ unzip ~/dsle20/dl/bootloader/u-boot-master.zip
```

Acesse o diretório do U-Boot e liste as configurações pré-definidas para as diversas plataformas suportadas pelo U-Boot. Devido à extensão da lista de configurações, é recomendado a utilização do comando *less*. Este comando permite a visualização do conteúdo de um arquivo ou *output* de algum comando, por meio de páginas do tamanho da tela :

```
$ cd u-boot-master
$ ls  configs/  |  less

10m50_defconfig
3c120_defconfig
A10-OLinuXino-Lime_defconfig
A10s-OLinuXino-M_defconfig
A13-OLinuXino_defconfig
A13-OLinuXinoM_defconfig
A20-Olimex-SOM204-EVB_defconfig
A20-Olimex-SOM204-EVB-eMMC_defconfig
A20-Olimex-SOM-EVB_defconfig
A20-OLinuXino-Lime2_defconfig
A20-OLinuXino-Lime2-eMMC_defconfig
A20-OLinuXino-Lime_defconfig
A20-OLinuXino_MICRO_defconfig
A20-OLinuXino_MICRO-eMMC_defconfig
A33-OLinuXino_defconfig
a64-olinuxino_defconfig
adp-ae3xx_defconfig
adp-ag101p_defconfig
Ainol_AW1_defconfig
alt_defconfig
am335x_baltos_defconfig
am335x_boneblack_defconfig
am335x_boneblack_vboot_defconfig
:
```

Use as teclas *espaço*, *b* e *q* para exibir a próxima página, a página anterior ou encerrar a visualização, respectivamente. Uma vez que o U-Boot suporta diversas arquiteturas de hardware diferentes, é necessário selecionar uma delas antes de compilar o programa. Selecione a configuração *rpi_3_32b_defconfig* e compile o U-Boot, indicando o prefixo do cross-compilador conforme configurado anteriormente:

```
$ make rpi_3_32b_defconfig
$ make -j2 CROSS_COMPILE=arm-linux-
```

Verifique o arquivo criado, **u-boot.bin**, através do comando:

```
$ ls -l u-boot.bin
-rw-rw-r-- 1 gbs gbs 429848 Jan 21 03:17 u-boot.bin
```

Note que a saída do comando *ls -l* exibirá informações como permissões sobre o arquivo, número de links, proprietário, tamanho do arquivo em bytes, data de modificação (criação neste caso), etc.

4.3 Gravando o U-Boot

O processo de gravação do bootloader é dependente de hardware, específico para cada plataforma. Conforme visto em aula, a RPi 3 busca o bootloader e firmwares relacionados ao processo de boot na partição *boot* do cartão micro SD. Assim, dois procedimentos são necessários para gravar e ativar o U-Boot no processo de boot da RPi:

1. Copiar o binário **u-boot.bin** para a partição *boot* do cartão micro SD;
2. Configurar o arquivo **config.txt** para que o firmware da GPU, **start.elf**, execute o U-Boot no lugar do kernel Linux;

É possível copiar o arquivo **u-boot.bin** para o cartão micro SD de várias maneiras. Visto que já existe uma configuração de rede entre a máquina de desenvolvimento e a RPi, realizada em atividades anteriores, o arquivo será copiado via comando *scp*:

```
$ scp u-boot.bin pi@10.1.1.100:/home/pi/

pi@10.1.1.100's password:
u-boot.bin                                    100%  418KB 417.5KB/s   00:00
```

Logue na RPi, via ssh, e copie o arquivo da pasta *home* para a pasta *boot* com privilégios de administrador:

```
$ ssh pi@10.1.1.100
pi@10.1.1.100's password:

# sudo cp u-boot.bin /boot/
# ls /boot

bcm2708-rpi-0-w.dtb       bcm2710-rpi-cm3.dtb  fixup_db.dat         overlays
bcm2708-rpi-b.dtb         bootcode.bin         fixup_x.dat          start_cd.elf
bcm2708-rpi-b-plus.dtb    cmdline.txt          issue.txt            start_db.elf
bcm2708-rpi-cm.dtb        config.txt           kernel7.img          start.elf
bcm2709-rpi-2-b.dtb       COPYING.linux        kernel.img           start_x.elf
bcm2710-rpi-3-b.dtb       fixup_cd.dat         LICENCE.broadcom     u-boot.bin
bcm2710-rpi-3-b-plus.dtb  fixup.dat
```

Feito isso, adicione a seguinte linha no final do arquivo **config.txt**:

```
# sudo nano /boot/config.txt

 [..]
```

```
dtparam=audio=on
enable_uart=1
kernel=u-boot.bin
```

O parâmetro acima especifica qual imagem do kernel, o firmware **start.elf** deve carregar. Neste caso, a imagem é o próprio U-Boot. Após salvar as alterações no arquivo, renicie a RPi (*sudo reboot*). **Atenção**, só reinicie a RPi se você possui uma conexão serial funcional entre ela e a máquina de desenvolvimento, pois a única interação com o menu do U-Boot, por enquanto, é via serial.

Por último, interrompa o processo de boot do U-Boot pressionando uma tecla para entrar no shell:

```
U-Boot 2018.07-rc1 (Oct 3 2019 - 19:50:41 -0300)

DRAM:  948 MiB
RPI 3 Model B (0xa02082)
MMC:   mmc@7e202000: 0, sdhci@7e300000: 1
Loading Environment from FAT... *** Warning - bad CRC, using default environment

Failed (-5)
In:    serial
Out:   vidconsole
Err:   vidconsole
Net:   No ethernet found.
starting USB...
USB0:  scanning bus 0 for devices... 3 USB Device(s) found
scanning usb for storage devices... 0 Storage Device(s) found
Hit any key to stop autoboot:  0
U-Boot>
```

4.4 Configurando variáveis de ambiente no U-Boot

Após obter acesso ao shell do U-Boot, configure algumas variáveis de ambiente básicas para manter o sistema da RPi bootável, pois por padrão, o U-Boot não identifica a imagem do kernel automaticamente. Inicialmente, desative o processo de autoboot:

```
U-Boot> setenv bootdelay -1
```

Futuramente, o U-Boot será configurado para carregar o kernel pela rede, de uma pasta de sua máquina de desenvolvimento. Portanto, configure o endereço IP da RPi e máquina de desenvolvimento:

```
U-Boot> setenv ipaddr 10.1.1.100
U-Boot> setenv serverip 10.1.1.1
```

Por fim, salve as alterações realizados no disco, de tal forma que elas estejam disponíveis na próxima inicialização, e reinicie o RPi:

```
U-Boot> saveenv
U-Boot> reset
```

Após enviar o comando *saveenv*, o U-Boot cria um arquivo de configuração chamado **u-boot.env** que contém todas as variáveis de ambiente definidas. Após reiniciar novamente a RPi, o processo de autoboot não deve ser inicializado e o controle do shell exibido logo em seguida.

Para “bootar” o Raspbian novamente, pelo U-Boot, algumas configurações adicionais são necessárias. O primeiro passo, é carregar um arquivo (imagem do kernel) do cartão micro SD para a RAM em um endereço específico. Utilize o seguinte comando:

```
U-Boot> fatload mmc 0:1 ${kernel_addr_r} kernel7.img
```

O comando acima carrega um arquivo binário de um sistema de arquivos FAT (fatload). Os parâmetros são:

- **mmc:** o dispositivo de armazenamento onde se encontra o binário (cartão SD);
- **0:** o número do dispositivo, pois muitas vezes o seu target pode possuir mais de um dispositivo de armazenamento;
- **1:** o número da partição no dispositivo (partições são enumeradas a partir de 1);
- **${kernel_addr_r}:** o endereço inicial na RAM onde deseja-se carregar o binário, neste caso a configuração do U-Boot já fornece uma variável de ambiente com o endereço inicial da RAM, para carregar o kernel (${kernel_addr_r});
- **kernel7.img:** o binário que se deseja carregar. Lembrando que por padrão, a partição boot do Raspbian fornece duas imagens: kernel.img e kernel7.img. A primeira é para as versões de hw anteriores (RPi e RPi 2) e a última (kernel7.img) é a imagem para as placas da RPi 3 em diante;

Algumas plataformas possuem dispositivos de hardware que não podem ser identificados dinamicamente pelo kernel. Em tais casos, é necessário um mecanismo para informar o kernel sobre dispositivos presentes no sistema. Atualmente, o mecanismo utilizado para tal função é chamado de Device Tree, que é uma estrutura de dados que descreve a topologia e configuração de hardware no sistema. Analisaremos esse mecanismo em breve durante o curso.

A RPI 3 é um tipo de plataforma que utiliza esse recurso e bootloader é o responsável por passar o Device Tree para o kernel e, portanto, é necessário fazer o loading desse arquivo no U-Boot. O processo é o mesmo anterior, porém com algumas alterações nos parâmetros:

```
U-Boot> fatload mmc 0:1 ${fdt_addr_r} bcm2710-rpi-3-b.dtb
```

- **mmc:** o dispositivo de armazenamento onde se encontra o binário (cartão SD);
- **0:** o número do dispositivo, pois muitas vezes o seu target pode possuir mais de um dispositivo de armazenamento;
- **1:** o número da partição no dispositivo (partições são enumeradas a partir de 1);
- **${fdt_addr_r}:** o endereço inicial na RAM onde deseja-se carregar o binário, neste caso a configuração do U-Boot já fornece uma variável de ambiente com o endereço inicial da RAM, para carregar o *Flattened Device Tree* (${fdt_addr_r});
- **bcm2710-rpi-3-b.dtb:** arquivo compilado do Device Tree (dtb) referente à RPi 3 B;

Obs: O **bcm2710-rpi-3-b-pus.dtb** é arquivo compilado do Device Tree (dtb) referente à Raspberry Pi Model 3 B+, caso esteja utilizando este modelo de placa, utilize este Device Tree ao longo do treinamento.

Estes são basicamente os únicos arquivos necessários para realizar o boot do Raspbian via U-Boot. Entretanto, além dos arquivos, ainda é necessário configurar alguns parâmetros do kernel. Configure a variável de ambiente *bootargs*, responsável por armazenar parâmetros para serem passados ao kernel:

```
U-Boot> setenv bootargs 8250.nr_uarts=1 root=/dev/mmcblk0p2 rootwait console=ttyS0,115200
```

onde:

- **8250.nr_uarts=1** número de portas seriais para serem registradas (utilizadas);
- **root=/dev/mmcblk0p2** path para a localização do RootFS (Root FileSystem);
- **rootwait** espera (indefinidamente) a inicialização dispositivo onde o RootFS está localizado. mmcblk0 refere-se ao micro SD e p2 à partição número 2, visto que a primeira é a partição de boot;
- **console=ttyS0,115200** redireciona um console para a porta serial ttyS0 (mini UART da RPi 3), e seu respectivo baudrate;

Finalmente, com as configurações acima realizadas, "basta" dar o comando de boot:

```
U-Boot> bootz ${kernel_addr_r} - ${fdt_addr_r}
```

onde:

- **kernel_addr_r** é o endereço na RAM onde está carregada a imagem do kernel;
- **-** seria o endereço na RAM da imagem initrd, que é uma imagem do RootFS. É possível carrega-lo na RAM, assim como kernel e Device Tree. No entanto, no momento a RPi não possui nenhuma imagem desse tipo. Ela será gerada em atividades futuras. Este parâmetro é opcional;
- **fdt_addr_r** é o endereço na RAM onde está carregado o compilado do Device Tree;

Se os comandos acima foram executados corretamente, você deverá enxergar o loading do kernel pela serial.

4.5 Criando um script de inicialização no U-Boot

O U-Boot possui uma funcionalidade que permite que o usuário crie scripts contendo sequência de comandos, como os executados acima. Na sua máquina de desenvolvimento, navegue até o diretório do U-Boot e crie o seguinte arquivo:

```
$ cd ~/dsle20/bootloader/u-boot-master
$ gedit rpi3scr.txt
```

Digite (copie do PDF) todos os comandos novamente no arquivo aberto e em seguida salve-o:

```
fatload mmc 0:1 ${kernel_addr_r} kernel7.img
fatload mmc 0:1 ${fdt_addr_r} bcm2710-rpi-3-b.dtb
setenv bootargs 8250.nr_uarts=1 root=/dev/mmcblk0p2 rootwait console=ttyS0,115200
bootz ${kernel_addr_r} - ${fdt_addr_r}
```

Em seguida, dentro da pasta *tools* no diretório do U-Boot, existe uma ferramenta chamada *mkimage* capaz de criar imagens do U-Boot, kernel, RootFS e inclusive scripts para o U-Boot. Digite o seguinte comando:

```
$ tools/mkimage -A arm -O linux -T script -d rpi3scr.txt boot.scr
```

onde os argumentos representam: arquitetura do executável (-A), sistema operacional (-O), tipo da imagem (-T), arquivo de entrada com os comandos digitados no *gedit* (-d) e arquivo de saída (boot.src), que é uma imagem do script.

Após a execução do comando, copie o arquivo **boot.scr** para o diretório *boot* da RPi via *scp*. Se tudo foi realizado corretamente, reinicie a Rpi e digite o comando *boot* no shell do U-Boot. É possível ativar o autoboot através da modificação da variável de ambiente *bootdelay*, que representa o delay em segundos antes de iniciar o boot pelo script boot.scr. Se desejar, altera-a, salve as variáveis de ambiente no cartão SD (*saveen*) e reinicie o U-Boot.

5 Kernel Linux

Nesta atividade, os fontes do Kernel Linux serão baixados e a partir deles, será realizada a configuração para compilar uma imagem para a RPi3, a compilação da imagem, a gravação na RPi3 e a inicialização do kernel através do U-Boot.

5.1 Baixando os fontes do kernel

A Raspberry Pi Foundation fornece os fontes do kernel para as placas RPi em seu repositório oficial https://github.com/raspberrypi/linux e estes foram os fontes baixados para esta atividade. Assim como outras ferramentas, eles se encontram no diretório *~/dsle20/dl/kernel/linux-rpi-4.14.y.zip*. Crie uma pasta chamada *kernel* dentro de *dsle20* e em seguida, extraia os fonte do kernel nela e entre no diretório extraído:

```
$ cd ~/dsle20
$ mkdir kernel && cd kernel
$ unzip ~/dsle20/dl/kernel/linux-rpi-4.14.y.zip
$ cd linux-rpi-4.14.y
```

5.2 Configurando o kernel

Conforme apresentado em aula, o kernel possui suporte para muitas arquiteturas diferentes. É possível visualizá-las através do diretório *arch*:

```
$ ls arch/
alpha  arm64    cris    hexagon  m32r  microblaze  nios2     powerpc  sh    um         xtensa
arc    blackfin  frv     ia64     m68k  mips        openrisc  s390     sparc  unicore32
arm    c6x      h8300   Kconfig  metag  mn10300     parisc    score    tile   x86
```

Além disso, o kernel também possui suporte especifico para a mesma arquitetura, porém de fabricantes diferentes. Por exemplo, mach-bcm refere-se SoCs da Broadcom, mach-exynos a SoCs da Samsung, mach-sti da ST, e assim por diante:

```
$ ls arch/arm
boot            mach-at91        mach-imx         mach-netx       mach-sa1100    mach-zynq
common          mach-axxia       mach-integrator  mach-nomadik    mach-shmobile  Makefile
configs         mach-bcm         mach-iop13xx     mach-nspire     mach-socfpga   mm
crypto          mach-berlin      mach-iop32x      mach-omap1      mach-spear     net
firmware        mach-clps711x    mach-iop33x      mach-omap2      mach-sti       nwfpe
include         mach-cns3xxx     mach-ixp4xx      mach-orion5x    mach-stm32     oprofile
Kconfig         mach-davinci     mach-keystone    mach-oxnas      mach-sunxi     plat-iop
Kconfig.debug   mach-digicolor    mach-ks8695      mach-picoxcell  mach-tango     plat-omap
Kconfig-nommu   mach-dove        mach-lpc18xx     mach-prima2     mach-tegra     plat-orion
kernel          mach-ebsa110     mach-lpc32xx     mach-pxa        mach-u300      plat-pxa
kvm             mach-efm32       mach-mediatek    mach-qcom       mach-uniphier  plat-samsung
lib             mach-ep93xx      mach-meson       mach-realview   mach-ux500     plat-versatile
mach-actions    mach-exynos      mach-mmp         mach-rockchip   mach-versatile probes
mach-alpine     mach-footbridge  mach-moxart      mach-rpc        mach-vexpress  tools
```

```
mach-artpec    mach-gemini      mach-mv78xx0    mach-s3c24xx    mach-vt8500    vdso
mach-asm9260   mach-highbank    mach-mvebu      mach-s3c64xx    mach-w90x900   vfp
mach-aspeed    mach-hisi        mach-mxs        mach-s5pv210    mach-zx        xen
```

Como pode-se perceber, o suporte do kernel Linux às mais variadas plataformas é bem grande. Além dos exemplos acima, existem ainda muitos arquivos de configurações prévias, relacionados a diferentes placas para cada arquitetura. Observe:

```
$ ls arch/arm/configs/
acs5k_defconfig             eseries_pxa_defconfig  mps2_defconfig        rpc_defconfig
acs5k_tiny_defconfig        exynos_defconfig       multi_v4t_defconfig   s3c2410_defconfig
am200epdkit_defconfig       ezx_defconfig          multi_v5_defconfig    s3c6400_defconfig
aspeed_g4_defconfig         footbridge_defconfig   multi_v7_defconfig    s5pv210_defconfig
aspeed_g5_defconfig         gemini_defconfig       mv78xx0_defconfig     sama5_defconfig
assabet_defconfig           h3600_defconfig        mvebu_v5_defconfig    shannon_defconfig
at91_dt_defconfig           h5000_defconfig        mvebu_v7_defconfig    shmobile_defconfig
axm55xx_defconfig           hackkit_defconfig      mxs_defconfig         simpad_defconfig
badge4_defconfig            hisi_defconfig         neponset_defconfig    socfpga_defconfig
bcm2709_defconfig           imote2_defconfig       netwinder_defconfig   spear13xx_defconfig
bcm2835_defconfig           imx_v4_v5_defconfig    netx_defconfig        spear3xx_defconfig
bcmrpi_defconfig            imx_v6_v7_defconfig    nhk8815_defconfig     spear6xx_defconfig
cerfcube_defconfig          integrator_defconfig   nuc910_defconfig      spitz_defconfig
clps711x_defconfig          iop13xx_defconfig      nuc950_defconfig      stm32_defconfig
cm_x2xx_defconfig           iop32x_defconfig       nuc960_defconfig      sunxi_defconfig
cm_x300_defconfig           iop33x_defconfig       omap1_defconfig       tango4_defconfig
cns3420vb_defconfig         ixp4xx_defconfig       omap2plus_defconfig   tct_hammer_defconfig
colibri_pxa270_defconfig    jornada720_defconfig   orion5x_defconfig     tegra_defconfig
colibri_pxa300_defconfig    keystone_defconfig     palmz72_defconfig     trizeps4_defconfig
collie_defconfig            ks8695_defconfig       pcm027_defconfig      u300_defconfig
corgi_defconfig             lart_defconfig         pleb_defconfig        u8500_defconfig
davinci_all_defconfig       lpc18xx_defconfig      prima2_defconfig      versatile_defconfig
```

Perceba que a maioria das ferramentas de desenvolvimento para Sistemas Linux Embarcado seguem um padrão de configuração no intuito de facilitar o processo para o desenvolvedor. O processo para carregar um arquivo de config prévia do kernel, é o mesmo como no crosstool e U-Boot. No entanto, por padrão o kernel considera a mesma arquitetura da maquina de desenvolvimento e, portanto, é necessário configurar a arquitetura para ARM. O arquivo de config da RPi3 é o *bcm2709_defconfig*:

```
$ export ARCH=arm
$ make bcm2709_defconfig
#
# configuration written to .config
#
```

A partir deste momento, as configurações básicas para compilar um kernel funcional para a RPi 3 foram carregadas e salvas em um arquivo chamado **.config** no diretório raiz dos fontes do kernel. Antes de compilar, acesse o *menuconfig* e verifique a quantidade de funcionalidades, drivers, protocolos de comunicação que o kernel oferece suporte:

```
$ make menuconfig
```

Antes de compilar o kernel é necessário definir também o toolchain, pois por padrão o processo de build do kernel irá utilizar as ferramentas nativas. Assim, defina a variável *CROSS_COMPILE* e compile o kernel:

```
$ export CROSS_COMPILE=arm-linux-
$ make -j4
```

Ao final do processo de compilação, as imagens geradas se encontrarão no diretório boot da arquitetura utilizada:

```
$ ls -l arch/arm/boot
```

5.3 Compilando o Device Tree

A RPi 3 faz uso de Device Tree para disponibilizar as informações de hardware ao kernel. Os fontes de Device Tree fornecidos encontram-se na pasta *arch/<arch>/boot/dts*:

```
$ ls arch/arm/boot/dts/

aks-cdu.dts                          kirkwood-dns320.dts
alphascale-asm9260-devkit.dts        kirkwood-dns325.dts
alphascale-asm9260.dtsi              kirkwood-dnskw.dtsi
alpine-db.dts                        kirkwood-dockstar.dts
alpine.dtsi                          kirkwood-dreamplug.dts
am335x-baltos.dtsi                   kirkwood-ds109.dts
am335x-baltos-ir2110.dts             kirkwood-ds110jv10.dts
am335x-baltos-ir3220.dts             kirkwood-ds111.dts
am335x-baltos-ir5221.dts             kirkwood-ds112.dts
am335x-baltos-leds.dtsi              kirkwood-ds209.dts
am335x-base0033.dts                  kirkwood-ds210.dts
am335x-boneblack-common.dtsi         kirkwood-ds212.dts
am335x-boneblack.dts                 kirkwood-ds212j.dts
am335x-boneblack-wireless.dts        kirkwood-ds409.dts
am335x-boneblue.dts                  kirkwood-ds409slim.dts
am335x-bone-common.dtsi              kirkwood-ds411.dts
am335x-bone.dts                      kirkwood-ds411j.dts
am335x-bonegreen-common.dtsi         kirkwood-ds411slim.dts
 [...]
```

A lista é bem longa. Os fontes para a RPi 3 e RPi 3 Plus são *bcm2710-rpi-3-b.dts* e *bcm2710-rpi-3-b-plus.dts* respectivamente. Compile de acordo com sua placa:

```
$ make bcm2710-rpi-3-b.dtb
```

ou

```
$ make bcm2710-rpi-3-b-plus.dtb
```

Após a compilação, o objeto final especificado em um dos comandos acima, estará disponível na pasta *arch/arm/boot/dts*.

5.4 Gravando as novas imagens e bootando a RPi3

Agora é com você. Com as explicações dadas em aula em conjunto com as atividades anteriores, você deverá ser capaz de realizar esta etapa. Dicas:

- Copie os arquivos recém-compilados para a RPi via *scp*;
- Faça um backup do script do U-Boot (*boot.scr*) com o comando *mv* ou *cp*;
- Faça também um backup do Device Tree fornecido pelo Raspbian (bcm2710-rpi-3-b.dtb);
- Todos esses arquivos encontram-se na partição /boot do seu cartão SD;
- Após realizar os backups, gere um novo boot.scr com a ferramenta mkimage do U-Boot alterando a imagem do kernel para zImage;
- Copie os seguintes arquivos na sua partição /boot: **zImage, bcm2710-rpi-3-b.dtb** e **boot.scr**.

Obs: Lembre-se de utilizar o arquivo de Device Tree correto para sua versão de placa.

6 RootFS

Nesta atividade será gerado um sistema de arquivos simples (RootFS) baseado no BusyBox, porém utilizaremos a ferramenta Buildroot. O Sistema será testado via cartão SD e NFS.

6.1 Baixando o Buildroot

A página oficial do projeto é https://buildroot.org/ e seu repositório oficial é https://github.com/buildroot/buildroot. É possível baixá-lo dos dois lugares. A versão utilizada no curso é a última estável, e encontra-se no diretório *~/dsle20/dl/rootfs/buildroot-2019.02.8.tar.gz*. Crie o diretório *~/dsle20/rootfs* e extraia o Buildroot neste diretório:

```
$ cd
$ cd dsle20
$ mkdir rootfs
$ cd rootfs
$ tar xvf ~/dsle20/dl/rootfs/buildroot-2019.02.8.tar.gz
```

6.2 Configurando e Compilando o RootFS

Novamente, o sistema de configuração do Buildroot é idêntico às ferramentas anteriores. Todas as configurações realizadas são salvas em um arquivo **.config** no diretório principal. Para listar as opções de placas / plataformas disponíveis, execute o seguinte comando:

```
$ cd buildroot-2019.02.8
$ make list-defconfigs

Built-in configs:
acmesystems_aria_g25_128mb_defconfig - Build for acmesystems_aria_g25_128mb
acmesystems_aria_g25_256mb_defconfig - Build for acmesystems_aria_g25_256mb
acmesystems_arietta_g25_128mb_defconfig - Build for acmesystems_arietta_g25_128mb
acmesystems_arietta_g25_256mb_defconfig - Build for acmesystems_arietta_g25_256mb
amarula_vyasa_rk3288_defconfig - Build for amarula_vyasa_rk3288
arcturus_ucls1012a_defconfig - Build for arcturus_ucls1012a
arcturus_ucp1020_defconfig - Build for arcturus_ucp1020
armadeus_apf27_defconfig - Build for armadeus_apf27
armadeus_apf28_defconfig - Build for armadeus_apf28
armadeus_apf51_defconfig - Build for armadeus_apf51
arm_foundationv8_defconfig - Build for arm_foundationv8
arm_juno_defconfig - Build for arm_juno
 [...]
```

Após analisar com calma os arquivos, perceba que o da Raspberry Pi 3 B é o *raspberrypi3_defconfig*. Faça o load deste arquivo da mesma maneira como anteriormente:

```
$ make raspberrypi3_defconfig
```

Como já possuímos: Toolchain, Bootloader e Kernel configurados, precisamos passar tais informações ao buildroot para que essas ferramentas não sejam baixadas novamente. Inicie o aplicativo *menuconfig* e faça as seguintes alterações, cautelosamente:

```
$ make menuconfig
```

No menu Toolchain options, iremos realizar praticamente as mesmas configurações do nosso toolchain:

- **Toolchain Type:** External toolchain;
- **Toolchain:** Custom Toolchain;
- **Toolchain origin:** Pre-isntalled toolchain;
- **Toolchain path (ATENÇÃO):** DIgite o path completo do seu toolchain, não utilize o path relativo ∼/. Ex: */home/gbs/dsle20/toolchains/x-tools/armv8-rpi3-linux-gnueabihf/*
- **External toolchain gcc version:** 6.x;
- **External toolchain kernel headers series:** 4.10.x;
- **External toolchain C library:** glibc/eglibc;
- **Toolchain has C++ Support:** * selecionar;
- **Copy gdb server to the Target:** * selecionar;

No menu System Configuration:

- **System hostname:** Pode colocar o nome que desejar;
- **System banner:** Idem. Será a mensagem de boas-vindas assim que o terminal é exibido;
- **Init system:** BusyBox;
- /**dev management:** Dynamic using devtmpfs only;
- **Enable root login with password:** * selecione. Nota: ao criar este RootFs, somente o usuário root estará disponível;

No menu Kernel:

- **Kernel Version:** Custom version;
- **Kernel version (2a opção logo abaixo):** 4.14.y. Esta é a mesma versão dos fontes baixados do repositório da RPi;
- **Linux Kernel Tools:**
  - cpupower: algumas ferramentas que permitem gerenciar o desempenho / consumo da cpu no kernel;
  - gpio: ferramentas para manipulação de gpio;
  - perf: ferramenta utilizada para profilling do sistema;

O menu Target packages é responsável por fornecer a maioria das aplicações de usuário. Por enquanto, selecionaremos algumas básicas. No entanto, caso necessitemos durante o desenvolvimento de softwares, podemos adicionar alguma aplicação específica no futuro:

- **Audio and Video Applications:** bluez-alsa, hcitop;
- **Compressors and Decompressors:** bzip2, zip;
- **Debugging, profilling and benchmark:** gdb->full debuger, memstat, rt-tests;
- **Development Tools:** binutils, git, make, tree;
- **Filesystems and Flash Utilities:** sshfs, ntfs-3g;
- **Hardware handling-> Firmware:** rpi-bt-firmware, rpi-firmware, Install DTB overlays, rpi-wifi-firmware; item **Interpreter languages and scripting:** php, python;
- **Libraries->Hardware Handling:** bcm2835, WiringPi;
- **Netwok Applications:** bluez-utils 5.x, dhcpcd, lighttpd, openssh, wpa_supplicant, wpa_cli binary;
- **Shell and utilities:** sudo;
- **System Tools:** htop;
- **Text edtior and viewers:** nano;

Perceba o quanto de personalização, entre drivers, ferramentas, softwares, é possível selecionar ou não, no intuito de customizar um sistema Linux embarcado.

No menu Filesystem Images:

- **exact size:** 1g (opcional o tamanho). Infelizmente o buildroot não estima o menor tamanho possível da imagem final do Rootfs. Até porque, considerando sistemas embarcados, a ideia é sempre utilizar o máximo do espaço disponível que a placa oferece. Assim, sinta-se à vontade para selecionar o tamanho final da sua imagem do RootFS. Se der algum erro no final da compilação, dizendo que a imagem não cabe no tamanho selecionado, não se preocupe. Apenas abra o menuconfig novamente, aumente o tamanho e recompile. A menos que você dê um clean, o buildroot não irá apagar tudo que já foi compilado;

Certifique-se de que não haja nenhum Bootloader selecionado no menu Bootloader. Pois já compilamos o U-Boot. Saia e SALVE as configurações. **ATENÇÃO:** antes de compilar, ainda é necessário definir o local dos fontes do kernel (que você já baixou). Crie e edite o arquivo **local.mk**:

```
$ nano local.mk
LINUX_OVERRIDE_SRCDIR = /home/gbs/dsle20/kernel/linux-rpi-4.14.y
```

Insira a linha *LINUX_OVERRIDE_SRCDIR = /home/gbs/dsle20/kernel/linux-rpi-4.14.y* indicando o path para os fontes do kernel. **Note** que é necessário utilizar o caminho completo e altere o nome de usuário (**gbs**) de acordo com seu usuário.Por fim, compile as imagens:

```
$ make -j4
```

Assim que a compilação terminar, verifique os arquivos gerados dentro da pasta *output/image*:

- **bcm2710-rpi-3-b.dtb:** Device Tree compilado para a RPi 3 Model B;
- **bcm2710-rpi-3-b-plus.dtb:** Device Tree compilado para a RPi 3 Model B Plus;
- **bcm2710-rpi-cm3.dtb:** Device Tree compilado para a RPi Compute Module (ComputeModule);
- **boot.vfat:** Partição de boot com todos os arquivos necessários para bootar o sistema recém-criado;

- **rootfs.ext2:** Partição de montagem do RootFS; Contém todos os arquivos, programas, bibliotecas que o kernel Linux precisa para executar, bem como as aplicações e ferramentas extras selecionadas;
- **rootfs.ext4:** Cópia da partição acima, no entanto esta utiliza o sistema de arquivos ext4;
- **rpi-firmware:** Pasta com todos os binários e arquivos de configuração necessários para bootar a RPi;
- **sdcard.img:** Imagem pronta com todos arquivos gerados pelo Buildroot. Contém ambas partições, boot e rootfs. Pronta para gravar num cartão SD ou pendrive ou HD;
- **zImage:** Imagem compilada do kernel Linux;

Obs: Caso o U-Boot fosse selecionado antes do processo de build, nessa mesma pasta existiria um arquivo **u-boot.bin**, imagem do U-Boot.

6.3 Bootando seu sistema pela 1ª vez - Configurações básicas

Após gravar a imagem completa do SD card, abra o arquivo **config.txt** presente na partição boot e faça as seguintes alterações:

```
# fixes rpi3 ttyAMA0 serial console
#dtoverlay=pi3-miniuart-bt
enable_uart=1
```

isso habilitará o boot via serial (conforme explicado nas seções anteriores). Ao carregar seu sistema pela primeira vez, será necessário realizar algumas configurações básicas para ativar o servidor SSH e criar uma nova conta de usuário, por exemplo.

6.3.1 Criando uma senha de root e nova conta de usuário

De acordo com a configuração realizada no Buildroot, somente o usuário root foi criado no sistema. Na tela de login, entre como root e senha em branco. Note que o conteúdo da mensagem de boas vindas será exibido de acordo com sua configuração no Buildroot:

```
Welcome to DSLE20
gbs login: root

#
```

Após logar no sistema, crie os diretórios */home* e */boot*. Em seguida adicione um segundo usuário no sistema:

```
# mkdir /home
# mkdir /boot
# adduser -h /home/gbs gbs

Changing password for gbs
New password:
Bad password: too weak
```

```
Retype password:
passwd: password for gbs changed by root
```

Nota: substitua o nome *gbs* pelo nome/nick de seu usuário desejado. Será necessário definir uma nova senha para o usuário. Aproveite e defina também uma senha para o root:

```
# passwd

Changing password for root
New password:
Bad password: too weak
Retype password:
passwd: password for root changed by root
```

Feito isso, altere o arquivo /**etc**/**sudoers** para dar permissões de root ao seu usuário por meio do comando *sudo*. Encontre as linhas referentes a permissões de usuário dos grupos *wheel* e *sudo* e remova seus respectivos comentários:

```
# nano /etc/sudoers

 [...]
## Uncomment to allow members of group wheel to execute any command
%wheel ALL=(ALL) ALL

## Uncomment to allow members of group sudo to execute any command
%sudo ALL=(ALL) ALL
 [...]
```

Em seguida, crie o grupo *sudo*e adicione seu usuário nos grupos *dialout, root, sudo e wheel*:

```
# addgroup sudo
# nano /etc/group
root:x:0:gbs

 [...]

wheel:x:10:root, gbs

 [...]

dialout:x:18:gbs

 [...]

sudo:x:1003:gbs
```

Nota: é possível utilizar esse arquivo para adicionar usuários em qualquer grupo do sistema. Basta encontrar o grupo desejado e adicionar o usuário no final da linha. Caso já exista algum usuário definido naquele grupo (como no caso do grupo *wheel* acima, adicione uma vírgula, espaço e o novo usuário).

```
# nano /etc/ssh/sshd_config

 [...]
#LoginGraceTime 2m
PermitRootLogin yes
#StrictModes yes
#MaxAuthTries 6
#MaxSessions 10
 [...]
# To disable tunneled clear text passwords, change to no here!
PasswordAuthentication yes
PermitEmptyPasswords yes
 [...]
```

6.3.2 Configurando um endereço IP estático

Para configurar um IP estático no novo sistema, edite o arquivo /**etc**/**network**/**interfaces** da seguinte maneira:

```
# nano /etc/network/interfaces

 [...]
auto eth0
iface eth0 inet static
    address 10.1.1.100
    netmask 255.255.255.0
    gateway 10.1.1.1
    pre-up /etc/network/nfs_check
    wait-delay 15

 [...]
```

6.3.3 Habilitando o SSH

Inicialmente, altere as configurações do arquivo /**etc**/**ssh**/**sshd_config**:

As configurações acima permitem 1- Logar como root via SSH; 2- Possibilita logar através de login e senha (caso contrário somente com chaves públicas); 3- Permite logar somente com login, sem a necessidade de digitar a senha. Este último será útil para copiar automaticamente o binário do seu programa pelo comando *scp* sem precisar digitar a senha.

Em seguida, altere o arquivo /**etc**/**shadow**. Este arquivo é responsável por armazenar informações seguras de contas de usuário. Os campos são separados por "**:**". O primeiro campo representa o nome de usuário, o segundo representa a senha de usuário encriptada e o terceiro, que devemos alterar, representa o número de dias que se passaram, desde a última alteração de senha para a respectiva conta de usuário. É necessário alterar esse campo de 0 para um número qualquer (10 por exemplo) para que tal usuário possa logar na RPi via SSH:

```
# nano /etc/shadow
```

```
root:$1$8N/irElt$uJ8THhDt.c2plt8cU336j/:10:0:99999:7:::
 [...]
gbs:$1$cmF9.YCp$ATeyqkHK..4sDEq9jkbq71:10:0:99999:7:::
```

Para maiores informações sobre o arquivo shadow: https://www.cyberciti.biz/faq/understanding-etcshadow-file/.

Reinicie o sistema e teste o SSH, tanto com root quanto seu usuário. Caso apareça alguma mensagem de warning relacionada à mudança de identificação de Host, apague o arquivo ~/**.ssh**/**known_hosts**:

```
$ ssh root@10.1.1.100
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!    @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.

 [...]

$ rm ~/.ssh/known_hosts
```

Após remover o arquivo, será pedido a autorização para gerar uma nova *key* para o ip 10.1.1.100. Digite *yes* e prossiga com o login.

7 Desenvolvimento de aplicações para Linux Embarcado

7.1 Instalando e iniciando o Eclipse

Instale o Eclipse CDT (C/C++ Development Tooling) através dos repositórios oficiais do Ubuntu:

```
$ sudo apt install eclipse-cdt*
```

Ao terminar o processo de instalação, abra o Eclipse (via terminal ou pelo launcher) e configure um Workspace de sua preferência. Ex: */home/gbs/dsle20/workspaceuel*. Em seguida, clique no ícone superior direito **Workbench**.

7.2 Configurando a conexão entre Eclipse e Target

A conexão entre o Eclipse e a RPi será realizada via SSH, por meio do plugin RSE (Remote System Explorer). Se o comando anterior, de instalação do Eclipse, foi executado corretamente, este plugin já estará instalado.

Antes de prosseguir nas configurações do Eclipse, faça a liberação de login sem senha de seu usuário/-máquina no servidor SSH da RPi. Para isso, primeiro gere uma chave rsa para seu host (Ubuntu):

```
$ ssh-keygen
 Generating public/private rsa key pair.
Enter file in which to save the key (/home/gbs/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/gbs/.ssh/id_rsa.
Your public key has been saved in /home/gbs/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:e8ARAAHCdxCXRGpFdd655Kvq875lRkhO4BO5/2igiy0 gbs@ubuntuvm
The key's randomart image is:
+---[RSA 2048]----+
|o .+OO+=o.       |
| o .+o ..=..  .  |
|  .o.  oo+ +     |
|  .    o=.+ .    |
|        So +     |
|         . +. .  |
|        . o +=   |
|   Eo. .  +=.    |
|    .. ooo*=.    |
+----[SHA256]-----+
```

A ferramenta irá perguntar sobre local para salvar a chave criada e senha, deixe ambos como padrão, apenas tecle ENTER. Em seguida, copie sua chave recém criada para a RPi, utilizando o usuário que você deseja vincular a key:

```
$ ssh-copy-id gbs@10.1.1.100
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
```

```
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new
    keys
gbs@10.1.1.100's password:

Number of key(s) added: 1
Now try logging into the machine, with: ssh 'gbs@10.1.1.100'
and check to make sure that only the key(s) you wanted were added.
```

Por fim, adicione sua key recém-criada ao agente de autenticação SSH:

```
$ ssh-add
```

Este passo final vinculará a key com o usuário utilizado no passo anterior. Assim, sempre que o comando *ssh gbs@10.1.1.100* for chamado, o agente fornece tal key e remove a necessidade de digitar uma senha. Faça o teste:

```
$ ssh gbs@10.1.1.100
$
```

Pronto. Agora retorne ao Eclipse. Abra a perspectiva do RSE em: "Window > Open Perspective > Other...". Selecione Remote System Explorer e clique em OK. Na janela mais a esquerda, chamada Remote Systems, clique com o botão direto no espaço em branco e selecione: "New > Connection...".

Na janela *New Connection* selecione o tipo Linux e clique em Next. Preencha com as seguintes informações:

- **Host name:** 10.1.1.100 - O endereço de IP da Raspberry PI;
- **Connection name:** RPi3 - O nome para esta conexão (pode colocar o nome que achar melhor);
- **Description:** Alguma descrição sobre essa conexão (opcional);

Clique em Next. Na janela superior esquerda, Configuration, marque a opção *ssh.files* e Next. Novamente, na janela superior esquerda, Configuration, marque a opção *processes.shell.linux* e Next. Em seguida, na mesma janela, marque *ssh.shells*, Next e por último marque *ssh.terminals* e Finish.

A partir deste momento, será possível explorar os arquivos da RPi através do Eclipse. Expanda o menu Sftp Files e perceba que todas as pastas do RootFS estão listadas ali. É possível copiar arquivos, editar, apagar, tudo de forma integrada com o Eclipse. Provavelmente será pedida a senha do seu usuário da RPi, apenas digite-a e o acesso aos arquivos será liberado.

Logo abaixo de Sftp Files, no Menu Shell Processes, você possui acesso à todos os processos (incluindo threads) em execução no target. É possível finalizar um processo pelo Eclipse e escolher o tipo de sinal que deseja enviar (botão direito no processo -> kill -> signal type).

No canto superior direito do Eclipse, existe um ícone de atalho para a opção "Open Perspective", ele estará do lado do atalho para Remote System Explorer perspective. Clique nele e selecione a perspectiva C/C++.

Adicione também a View do Remote Systems na perspectiva C/C++. Dessa forma você pode explorar os arquivos do target na mesma perspectiva de programação. Selecione o menu "Window > Show View > Other...". Em seguida, selecione "Remote Systems > Remote Systems". A View Remote System Details também é interessante. Ela exibe informações de permissão de arquivos e diretórios.

Na aba recém-adicionada Remote Systems, Clique no último ícone da lista com o botão direito, Ssh Terminals, e escolha a opção Launch Terminal. Note que um novo terminal SSH será aberto numa nova aba. A partir de agora é possível também acessar o terminal da RPi3 via SSH, pelo Eclipse.

7.3 Cross-compilando sua primeira aplicação

Na perspectiva C/C++, clique no menu "File > New > C Project". Em Project type, selecione Hello World ANSI C Project. Em Toolchains, selecione Cross GCC. Dê um nome para seu projeto em Project name: "MyFirstApp". Clique em Next e, se desejar, adicione informações de autor (suas iniciais, por exemplo). Next novamente, Next e, em **Cross compiler prefix:** *arm-linux-* e no campo **Cross compiler path:** selecione o caminho do toolchain gerado na primeira aula: */home/gbs/dsle20/toolchains/x-tools/armv8-rpi3-linux-gnueabihf/bin*. Finalmente, clique em Finish.

Para compilar o projeto, acesse o menu "Project > Build All"ou "CTRL+B". Se tudo foi configurado corretamente, a seguinte mensagem será exibida na aba Console: *"15:46:14 Build Finished (took 183ms)"*.

Para testar o binário, copie-o da pasta *Binaries*, logo abaixo da pasta do seu projeto para sua pasta Home, na RPi, através da janela Remote Systems. Note que ao tentar executar o arquivo pelo terminal SSH, será exibida uma mensagem de erro, dizendo que não é possível executar o binário:

```
$ ./MyFirstApp
-sh: ./MyFirstApp: Permission denied
```

Para corrigir este problema, dê permissões de execução para o arquivo:

```
$ chmod +x MyFirstApp
$ ./MyFirstApp
!!! Hello World !!!
```

7.4 Configurando o GDB no Eclipse

Acesse o menu "Run > Debug Configurations":

- Clique duas vezes em C/C++ Remote Application;
- Na aba "Main", no campo "Connection", selecione o nome ("RPi3") da conexão que você criou anteriormente no Remote System Explorer;
- Em "Remote Absolute File Path for C/C++ Application", digite o caminho do executável na RaspberryPi . Por exemplo: "/home/gbs/MyFirstApp";
- Na aba "Debugger", digite "arm-linux-gdb";
- Clique em "Apply"e depois em "Debug";
- Ao abrir a janela perguntando se você deseja abrir a perspectiva de "Debug", apenas clique em Yes;

7.5 Configurando o Eclipse para copiar seu executável no Target automaticamente

Clique com o botão direito no seu projeto, na janela da esquerda chamada Project Explorer, e acesse propriedades. Na janela exibida, navegue em "C/C++ Build > Settings". Clique na aba "Build Steps"e digite o seguinte comando no campo "Command"do bloco "Post-build steps":

scp MyFirstApp gbs@10.1.1.100:/home/gbs.

O comando acima faz com que o Eclipse copie seu binário para o Target, na pasta "/home/gbs", a cada novo build. Apague o arquivo via terminal ssh:

rm /home/gbs/MyFirstApp

e em seguida, compile o projeto novamente e verifique que o novo binário estará disponível na pasta e pronto para execução: *./MyFirstApp*.

7.6 Configurando e linkando bibliotecas externas no Eclipse

Três configurações devem ser realizadas no intuito de linkar uma biblioteca a um projeto do Eclipse:

- **Definir o diretório de headers da biblioteca:** A adição deste diretório no projeto possibilita que o *indexer* do Eclipse reconheça as funções, definições de constantes, declarações em geral da biblioteca. Uma vez reconhecida a biblioteca, é possível utilizar o recurso *autocompletar* durante a programação. Para adicionar um diretório de headers a um projeto do eclipse:

  Clique com o botão direito no projeto desejado e selecione Properperties > C/C++ Build > Settings. Em seguida, no Menu da direita, selecione as opções: *Cross GCC Compiler > Includes* e, finalmente, adicione o *path* onde se encontra os headers da biblioteca a ser incluída.

- **Definir o nome da biblioteca para linkagem durante a compilação:** Além de definir o diretório de headers, é necessário especificar ao compilador quais são as bibliotecas que devem ser linkadas com o executável do seu projeto. Para isso:

  Clique com o botão direito no projeto desejado e selecione Properperties > C/C++ Build > Settings. Em seguida, no Menu da direita, selecione as opções: *Cross GCC Linker > Libraries* e, finalmente, adicione o nome das bibliotecas que deseja linkar com seu projeto. Note que, em C, apesar de os executáveis de bibliotecas normalmente serem nomeados como *libNOMEDABIBLIOTECA.so*, deve-se adicionar apenas o NOMEDABIBLITOECA, sem o lib e o .so. Ex: se o nome do executável da biblioteca é *libpigpio.so*, adiciona-se somente *pigpio*.

- **Definir o diretório onde se encontra o executável da biblioteca:** Além de definir quais bibliotecas deseja-se linkar com o executável, é necessário indicar no projeto, o diretório onde tal executável se encontra, caso a biblioteca seja externa ou não esteja incluída na GLibC. Navegue até o mesmo ponto descrito no passo anterior:

  textitClique com o botão direito no projeto desejado e selecione Properperties > C/C++ Build > Settings. Em seguida, no Menu da direita, selecione as opções: *Cross GCC Linker > Libraries*.

  Note que existe uma barra de rolagem lateral e, ao rolá-la para baixo, uma nova opção será exibida: *Library Search Path:*. Este é o local onde deve ser indicado o diretório contendo o executável da biblioteca a ser adicionada.

7.7 Configurando o path de bibliotecas externas no GDB

Normalmente, para debugar uma aplicação remotamente que foi compilada com alguma biblioteca externa (não incluída na GLIBC), é necessário definir o path de tal biblioteca em um arquivo de configuração do GDB, chamado **.gdbinit**. Esse arquivo, na verdade, é utilizado para passar parâmetros ao gdb, que normalmente seriam passados via linha de comando. Crie um arquivo chamado **.gdbinit** e adicione a seguinte linha no arquivo:

set solib-search-path /PATH/DA//BIBLITOECA

Salve o arquivo e adicione-o nas configurações de Debug do seu projeto. Acesse o menu "Run > Debug Configurations":

- Selecione a configuração já realizada no passo descrito na seção 7.4;
- Na aba "Debugger", no campo "GDB command file", clique no botão browse e selecione o path do arquivo **.gdbinit**;
- Ps: a) É possível selecionar um path qualquer para o arquivo ou; b) deixar essa opção inalterada e criar o arquivo na raiz do seu workspace atual;