

# LABORATÓRIO AOC II



---

## **Lab 00: Ambiente de Desenvolvendo e Primeiro Sistema na BBB**

---

Prof. Thiago Werlley

19 de novembro de 2024

# 1 Carregando a Primeira Aplicação Bare Metal

Nesta primeira prática de laboratório tem-se o objetivo de preparar o desenvolvedor a configurar o ambiente de desenvolvimento para programação em sistema embarcado, utilizando o bootloader U-Boot para carregar o primeiro sistema desenvolvido para a placa Bealgebone, definida pelo nome de bare metal, sendo carregada na placa via TFTP.

Em uma observação mais simples, a programação bare-metal significa escrever uma aplicação diretamente em seu hardware sem usar uma interface de programação de aplicativo externa, ou seja, sem nenhum sistema operacional. Escrevemos aplicações embarcadas acessando diretamente os registradores de hardware do mapa de memória dos microcontroladores.

## 1.1 Instalando o cross-compilador

Um cross-compilador nada mais é do que um compilador para uma plataforma diferente do computador usado no desenvolvimento. Como por exemplo um computador com a arquitetura x86 compilando para uma placa com plataforma ARM cortex-A8. O cross-compilador que será usado será o **arm-none-eabi**, que é o cross-compilador específico para essa plataforma, e que tem característica de programação em bare metal.

### 1.1.1 Baixando o arm-none-eabi

O pacote com o cross-compilador **arm-none-eabi** pode ser encontrado no site próprio da ARM e algumas outras fontes como no site do Linaro. Acesse esse link <https://www.linaro.org> ou <https://developer.arm.com> e baixe a opção de cross-compilar bare metal que rode na máquina x86, algo parecido com (**gcc-arm-none-eabi-10.3-2021.10-x86\_64-linux**). Após baixar o cross-compilador, será preciso criar uma pasta no diretório da disciplina, em que o desenvolvedor irá realizar suas práticas, nesse caso siga os seguintes passos no terminal:

```
$ mkdir lab
$ cd lab/
```

Após baixado o cross-compilador, será preciso criar um diretório nesse ambiente de desenvolvimento, nesse caso siga os seguintes passos no terminal:

```
$ mkdir toolchain
```

Após a criação do diretório é preciso descompactar o arquivo dentro de **toolchain**, digitando o seguinte comando:

```
$ tar jxvf gcc-arm-none-eabi-10.3-2021.10-x86_64-linux.tar.bz2 -C
  toolchain/
```

**Lembre-se de verificar o nome do arquivo antes de copiar e descompactar, pois obviamente se tiver um nome diferente, o linux não vai encontrar!!!**

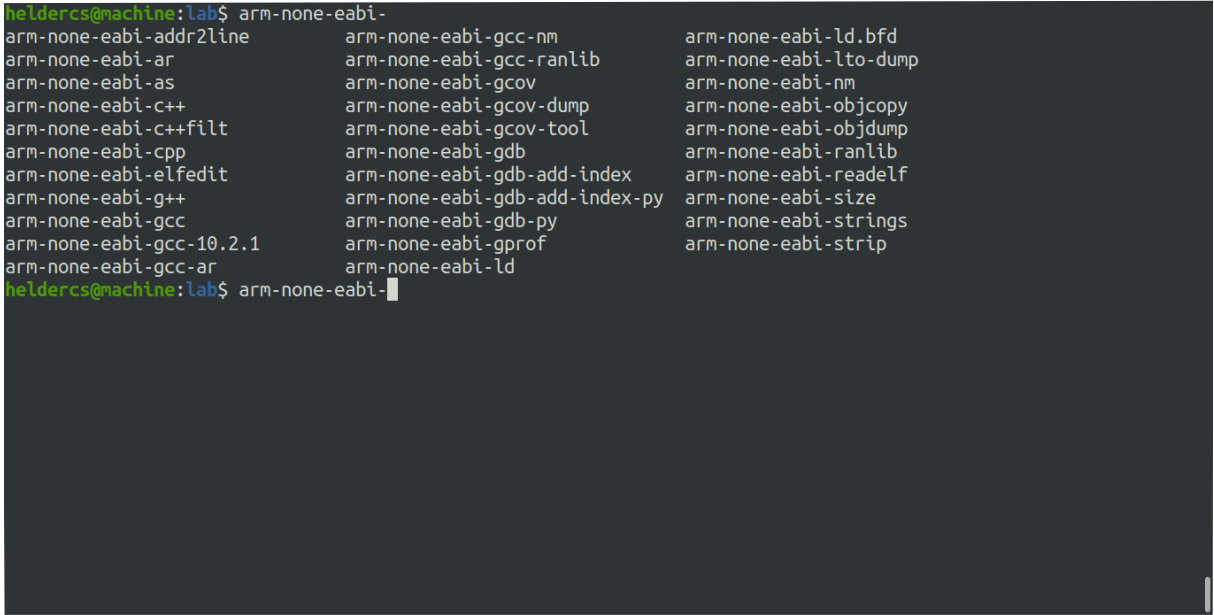
Com isso, já temos o ambiente de cross-compilação, porém para que ele seja acessado de qualquer parte do computador é preciso que seja mudada uma variável de ambiente chamada **\$PATH**, para isso é preciso modificar o arquivo **.bashrc** que está no diretório do usuário, siga os seguintes passos para a modificação da variável.

```
$ cd ~
$ gedit .bashrc
```

Em que a pasta “~” é o diretório principal do usuário e **gedit** é um editor de textos (pode ser usado qualquer outro). Após abrir esse arquivo é preciso adicionar a seguinte linha no final do arquivo:

```
PATH=$PATH:~/lab/toolchain/gcc-arm-none-eabi-10.3-2021.10/bin
```

Veja que `~/lab/toolchain/gcc-arm-none-eabi-10.3-2021.10/bin` é o caminho da pasta que foi criado, portanto adicione o caminho da sua pasta, que pode ser visto no terminal com o comando “`pwd`”. Caso o caminho contenha espaços, como por exemplo **Área de Trabalho**, é necessário uma barra invertida(\), portanto, ficaria **Área\de\Trabalho**. Verifique seu caminho e cole corretamente no arquivo `~bashrc`. Após configurar a variável `$PATH`, o cross-compilador já pode ser usado normalmente.



```
helder@machine:lab$ arm-none-eabi-
arm-none-eabi-addr2line      arm-none-eabi-gcc-nm        arm-none-eabi-ld.bfd
arm-none-eabi-ar            arm-none-eabi-gcc-ranlib    arm-none-eabi-lto-dump
arm-none-eabi-as            arm-none-eabi-gcov          arm-none-eabi-nm
arm-none-eabi-c++           arm-none-eabi-gcov-dump     arm-none-eabi-objcopy
arm-none-eabi-c++filt       arm-none-eabi-gcov-tool     arm-none-eabi-objdump
arm-none-eabi-cpp           arm-none-eabi-gdb           arm-none-eabi-ranlib
arm-none-eabi-elfedit        arm-none-eabi-gdb-add-index arm-none-eabi-readelf
arm-none-eabi-g++           arm-none-eabi-gdb-add-index-py arm-none-eabi-size
arm-none-eabi-gcc           arm-none-eabi-gdb-py        arm-none-eabi-strings
arm-none-eabi-gcc-10.2.1    arm-none-eabi-gprof         arm-none-eabi-strip
arm-none-eabi-gcc-ar        arm-none-eabi-ld
helder@machine:lab$ arm-none-eabi-
```

Figura 1: Ambiente de cross-compilação instalado.

Após a instalação do ambiente de compilação, compile o programa distribuído no início e tente rodar no computador. O computador não conseguirá rodar a aplicação, pois foi compilado para outra plataforma.

```
#include <stdio.h>

int main() {
    printf("Hello World");
    return 0;
}
```

Para compilar programas como esse, é preciso alguns parâmetros de compilação. Após a criação do arquivo, temos que adicionar esses parâmetros no makefile, usando o seguinte comando:

```
all: app

CC= arm-none-eabi-gcc

app: main.o
$(CC) obj/main.o -lc -lrdimon -o bin/app
```

```
main.o: src/main.c
$(CC) -c src/main.c -Iinc -o obj/main.o

clean:
rm obj/*.o bin/app
```

Uma vez gerado o executável, ao tentar executar, o computador não vai conseguir executar. Para saber para qual arquitetura o executável tá compilado, simplesmente use o comando **file** no terminal.

```
helder@machine:pratica_01$ make
arm-none-eabi-gcc -c src/main.c -Iinc -o obj/main.o
arm-none-eabi-gcc obj/main.o -lc -ldimon -o bin/app
helder@machine:pratica_01$ ./bin/app
bash: ./bin/app: cannot execute binary file: Exec format error
helder@machine:pratica_01$ file ./bin/app
./bin/app: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), statically linked, not stripped
helder@machine:pratica_01$
```

Figura 2: Tentando rodar o executável para o ARM.

## 1.2 Configurar rede Ethernet para acessar com o servidor

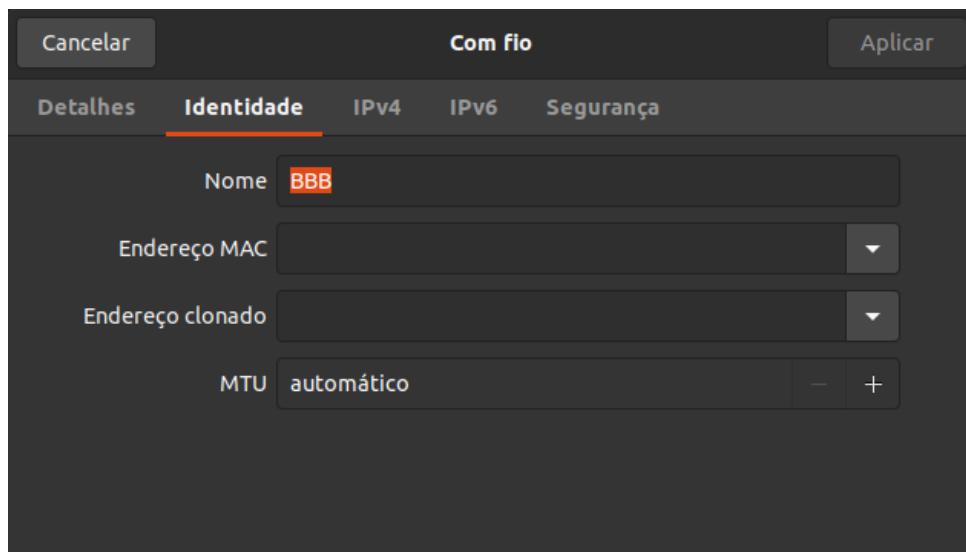


Figura 3: Nome da rede.

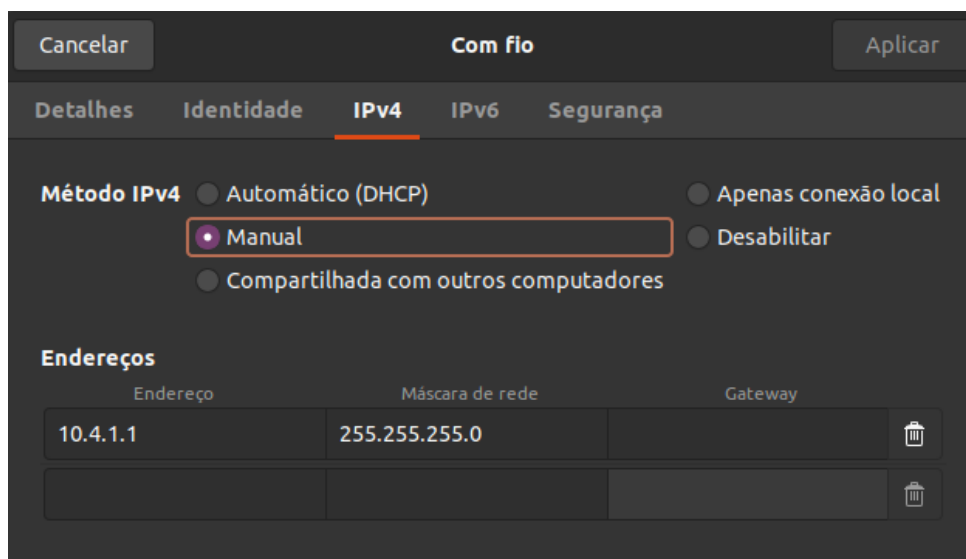


Figura 4: Modo manual, Endereço e Máscara de rede.

### 1.3 Configurando minicom para acessar a BBB

```
$ sudo apt-get install minicom
```

Conferindo a conexão TTL

```
$ dmesg
```

Precisa o Cabo TTL esta conectado para conseguir configurar o minicom

```
$ sudo minicom -s
```

```
/*****
```

Opção Configuração da porta serial

```
*****/
```

```
: /dev/ttyUSB0
```

```
: /var/lock
```

```
:
```

```
:
```

```
:115200 8N1
```

```
: Nao
```

```
: Nao
```

Salvar as configurações como dfi

Sair do Minicom

```
$ sudo minicom
```

## 1.4 O PC deve ser configurado com um servidor TFTP

O *Trivial File Transfer Protocol* (TFTP) fornece uma forma minimalista para transferir arquivos. É geralmente usado como uma parte da inicialização do PXE ou para atualizar configuração ou firmware em dispositivos que possuem memória limitada, tal como roteadores, telefones IP e sistema embarcado como um todo.

Para Instalar o serviço de tftpd realize os seguintes passos:

- 1 instale os seguintes pacotes

```
$ sudo apt-get install xinetd tftpd tftp
```

- 2 Crie o arquivo “tftp” no caminho /etc/xinetd.d, e então cole o seguinte conteúdo

```
service tftp
{
protocol      = udp
port          = 69
socket_type   = dgram
wait         = yes
user          = nobody
server        = /usr/sbin/in.tftpd
server_args   = /tftpboot
disable       = no
}
```

- 3 Crie e configure o diretório “tftpboot” no /

```
$ sudo chmod -R 777 /etc/xinetd.d/tftp
$ sudo mkdir /tftpboot
$ sudo chmod -R 777 /tftpboot
$ sudo chown -R nobody /tftpboot
```

- 4 Inicie o tftpd através do xinetd

```
$ sudo /etc/init.d/xinetd start
```

- 5 Realize um teste de validação do seu serviço tftp. Crie um arquivo “hda.txt” no diretório (tftp) e transfira o arquivo hda.txt para o diretório qualquer (ex: Downloads).

```
$ cd Downloads
$ touch /tftpboot/hda.txt
$ echo "somente um teste..." > /tftpboot/hda.txt
$ chmod 777 /tftpboot/hda.txt
$ ls -l /tftpboot/
-rwxrwxrwx 1 hederics hederics 0 2010-08-31 15:34 hda.txt
$ tftp 127.0.0.1
tftp> get hda.txt
Sent 722 bytes in 0.0 seconds
tftp> quit
$ ls -l
-rwxrwxrwx 1 hederics hederics 707 2010-08-31 15:34 hda.txt
```

