

LABORATÓRIO AOC II



Lab 00: Ambiente de Desenvolvendo e Primeiro Sistema na BBB

Prof. Thiago Werlley

18 de outubro de 2025

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-compiler

Um cross-compiler 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-compiler que será usado será o **arm-none-eabi**, que é o cross-compiler 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-compiler **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/downloads/-/arm-gnu-toolchain-downloads> e baixe a opção de cross-compile bare metal que rode na máquina x86, algo parecido com (**arm-gnu-toolchain-14.2.rel1-x86_64-arm-none-eabi.tar.xz**). Após baixar o cross-compiler, 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-compiler, 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:

```
$ wget https://developer.arm.com/-/media/Files/downloads/gnu/14.2.
    rel1/binrel/arm-gnu-toolchain-14.2.rel1-x86_64-arm-none-eabi.tar.
    xz
```

```
$ tar jxvf arm-gnu-toolchain-14.2.rel1-x86_64-arm-none-eabi.tar.xz -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

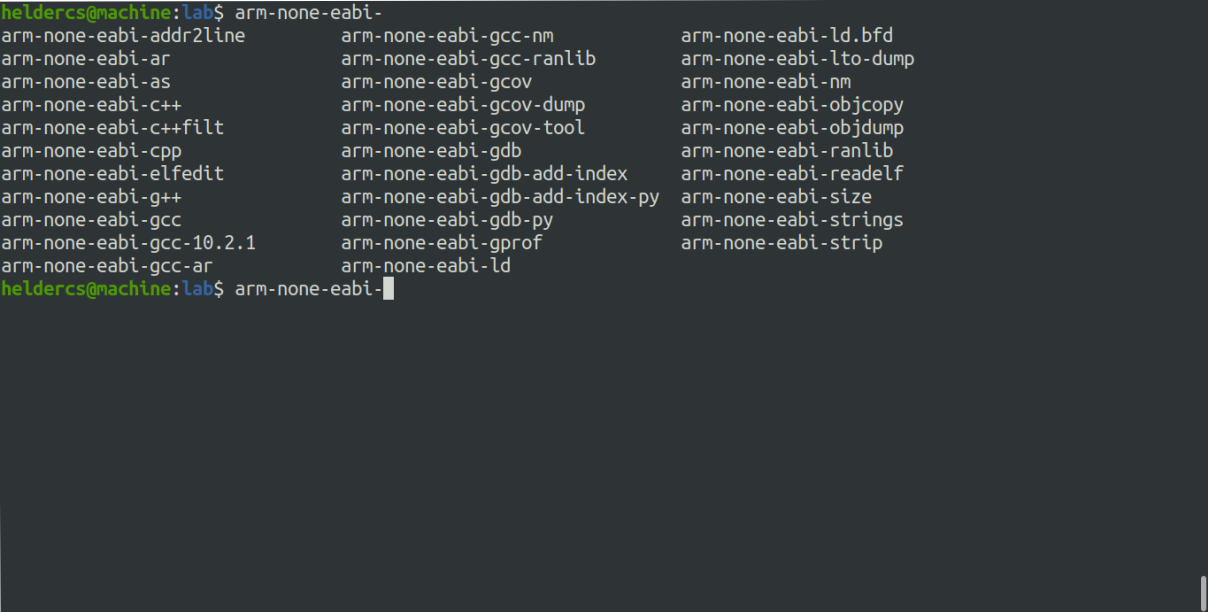
\$PATH, para isso é preciso modificar o arquivo **.bashrc** que está no diretório do usuário, siga os seguintes passos para a modificação da variável.

```
$ 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:~/toolchain/arm-gnu-toolchain-14.2.rel1-x86_64-arm-none-eabi/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.

2 Exemplo para entender o uso do Bare Metal

2.1 Exemplo para rodar no PC com sistema operacional

Após a instalação do ambiente de compilação, compile o programa distribuído no início e tente rodar no computador.

Header protegido contra inclusão múltipla diretivas desta biblioteca. Define a assinatura da função.

hello.h

```
#ifndef _H_TESTE
#define _H_TESTE

void helloWorld(void);

#endif
```

Implementação da função hello.c, usando printf, que chama o sistema operacional para escrever no console.

hello.c

```
#include <stdio.h>

void helloWorld(void) {
    printf("Hello World\n");
}
```

main.c

```
#include <stdio.h>          // Biblioteca padrao do C (stdio.h) - usa
                             sistema operacional
#include "helloWorld.h" // Header da funcao personalizada

int main(void) {
    helloWorld();           // Chama a funcao definida no helloWorld.
    c
    return 0;
}
```

Makefile

```
all: exec

exec: main.o hello.o
    gcc main.o hello.o -o exec

main.o: main.c hello.h
    gcc -c main.c -o main.o

helloWorld.o: hello.c hello.h
    gcc -c hello.c -o hello.o

clean:
    rm *.o exec
```

2.2 Exemplo para rodar na BBB sem sistema operacional

O computador não conseguirá rodar a aplicação, pois foi compilado para outra plataforma. Aqui, está chamando a `printf` — mas no baremetal não existe sistema operacional para tratar `printf`.

main.c

```
#include <stdio.h>

int main(void) {
    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:

Makefile

```
CC=arm-none-eabi-gcc

all: directories app

directories:
mkdir -p bin obj src

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

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

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

- **Compilador arm-none-eabi-gcc:** compila para arquitetura ARM Cortex-A8.
- **Gera um arquivo .elf, mas que:**
 - Não tem suporte a Linux.
 - Não tem suporte a funções padrão (`printf`, `malloc`, etc.), exceto se você implementar tudo (ou usar bibliotecas especiais para baremetal como `newlib-nano`).
- Usa `-lrdimon` para suporte mínimo a entrada/saída, mas a BeagleBone Black baremetal não tem UART configurada automaticamente, então `printf` simplesmente não funciona.

2.3 Verificando o executável

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.

O comando **file** app revela várias informações sobre o arquivo executável app:

- **ELF 32-bit LSB executable:** Indica que app é um executável no formato ELF (Executable and Linkable Format), com 32 bits e codificação Least Significant Byte first, comum em arquiteturas x86 e ARM.
- **ARM, EABI5 version 1 (SYSV):** O executável é compilado para a arquitetura ARM, usando a ABI (Interface Binária de Aplicativo) versão 5, compatível com o padrão System V.
- **statically linked:** Mostra que o executável é estaticamente ligado, o que significa que todas as bibliotecas necessárias são incorporadas diretamente no arquivo, resultando em um tamanho maior, mas eliminando dependências externas.
- **with debug_info:** Contém informações de depuração, útil para desenvolvedores durante a fase de teste para diagnosticar problemas.
- **not stripped:** Indica que o executável ainda possui símbolos de depuração e outras informações que, frequentemente, são removidas (stripped) para reduzir o tamanho do arquivo para produção.

- Isso significa:
 - ELF de 32 bits para ARM.
 - Linkado estaticamente.
 - Com debug info
- Mas não é um binário para Linux ARM, nem para um bootloader — é um programa baremetal cru, esperando ser carregado diretamente na memória e executado.

2.4 Configurar rede Ethernet para acessar com o servidor

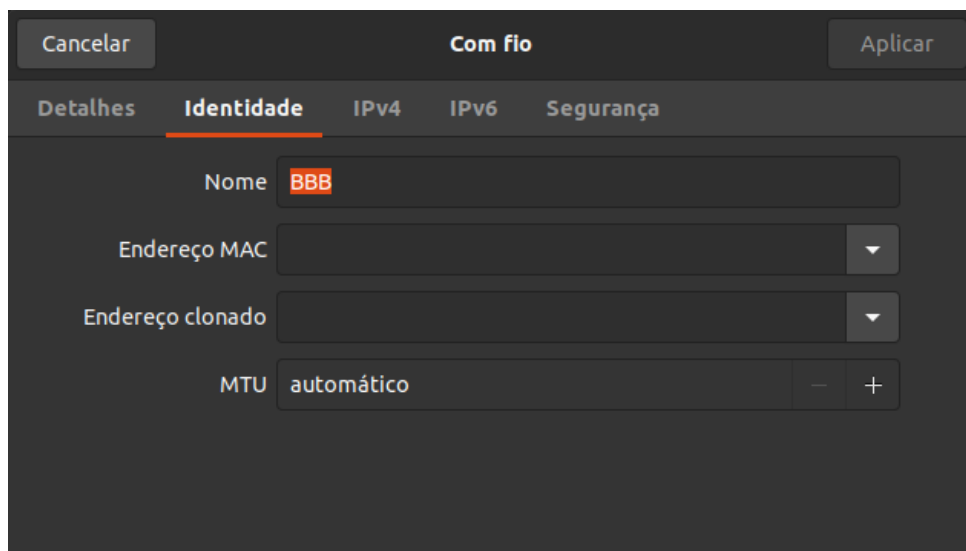


Figura 3: Nome da rede.

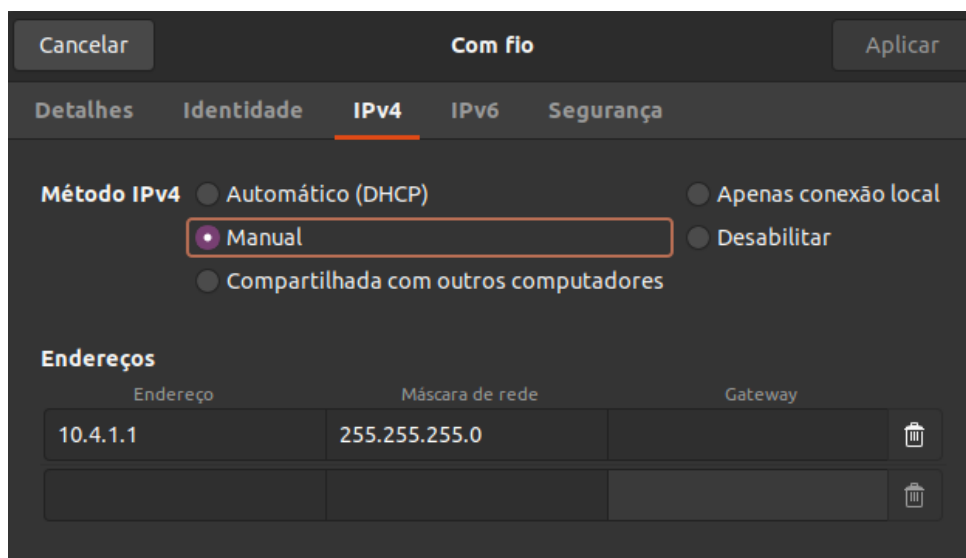


Figura 4: Modo manual, Endereço e Máscara de rede.

2.5 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 dfl

Sair do Minicom

```
$ sudo minicom
```

Como sair do Minicom via comando

Ctrl+a

Depois aperta:

x

2.6 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).

```
$ touch /tftpboot/hda.txt
$ echo "somente um teste..." > /tftpboot/hda.txt
$ chmod 777 /tftpboot/hda.txt
$ ls -l /tftpboot/
-rwxrwxrwx 1 werlley werlley 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 werlley werlley 707 2010-08-31 15:34 hda.txt
```

6 Copie a aplicação exemplo para o diretório TFTP:

```
$ cp bin/app /tftpboot/
$ cd /tftpboot
```

7 Crie o link simbólico

```
$
```

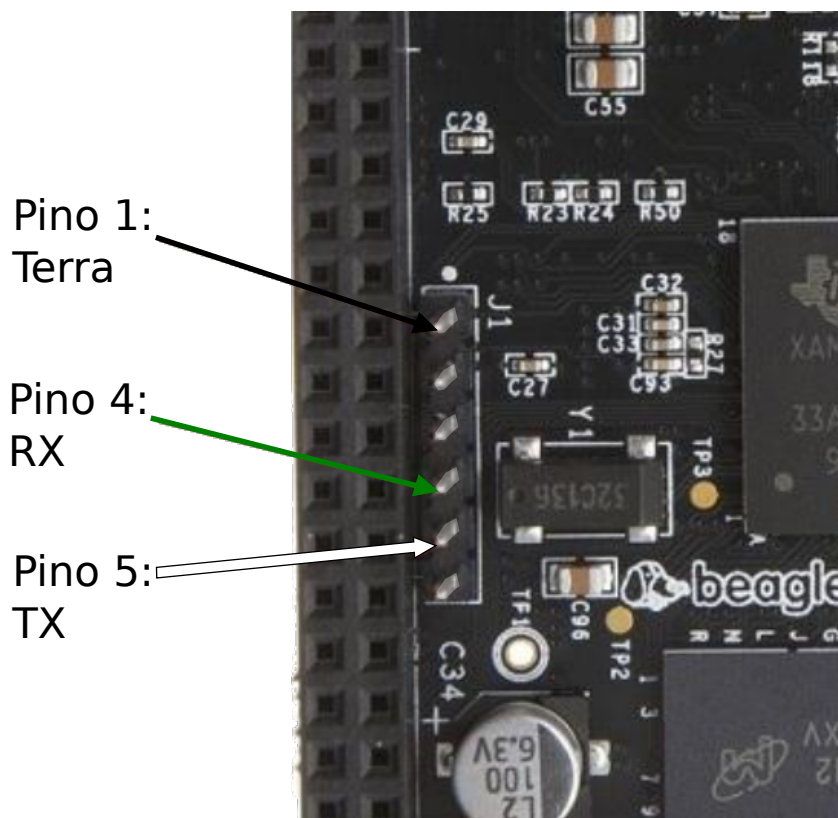


Figura 5: Conexões com o cabo TTL.

2.7 Conexões corretas na BBB

- A Beaglebone Black possui uma porta serial configurada como console que pode ser acessada através do barramento de pinos J1, conforme figura abaixo:

A pinagem é a seguinte:

- Pino 1 : **Terra (PRETO)**
- Pino 2 : não tem nada
- Pino 3 : não tem nada
- Pino 4 : **RX (VERDE)**
- Pino 5 : **TX (BRANCO)**
- Pino 6 : não tem nada

Para a conexão, você vai precisar de um cabo conversor FTDI para TTL de 3,3V.

2.8 Carregando uma Aplicação Bare Metal na BBB

Para carregar um sistema na BBB é necessário seguir os seguintes passos:

- Reboot a placa e entre no U-boot apenas pressionando qualquer tecla. Você deve está conectado via cabo serial.

- Configure a variável de ambiente do boot para carregar uma imagem via tftp.

configure o seguinte script na CLI (*Command Line Interface*) do bootloader na BBB:

```
U-Boot# setenv app "setenv autoload no;setenv ipaddr 10.4.1.2; setenv
serverip 10.4.1.1; tftp 0x80000000 /tftpboot/download.bin;echo
***Booting to BareMetal ***;go 0x80000000"
```

em seguida execute o script:

```
U-Boot# run app
```

...e temos o primeiro sistema rodando na placa BBB...

```
heldercs@Machine: ~
File Edit View Search Terminal Help

Net: <ethaddr> not set. Validating first E-fuse MAC
cpsw
Hit any key to stop autoboot: 0
U-Boot# 
U-Boot# 
U-Boot# 
U-Boot# 
U-Boot# set uenvcmd "setenv autoload no;dhcp; setenv serverip 192.168.0.181;;tftp 0x80000000 /tftpboot/download.bin;echo *** Booting to BareMetal ***"
U-Boot# run uenvcmd
link up on port 0, speed 100, full duplex
BOOTP broadcast 1
BOOTP broadcast 2
BOOTP broadcast 3
DHCP client bound to address 192.168.0.144 (3001 ms)
link up on port 0, speed 100, full duplex
Using cpsw device
TFTP from server 192.168.0.181; our IP address is 192.168.0.144
Filename '/tftpboot/download.bin'.
Load address: 0x80000000
Loading: ###
          989.3 KiB/s
done
Bytes transferred = 11144 (2b88 hex)
*** Booting to BareMetal ***
## Starting application at 0x80000000 ...

##### 
# # # # # # # # # # 
# # # # # # # # # # 
##### 
# # # # # # # # # # 
# # # # # # # # # # 
##### 
# # # # # # # # # # 
Application running: 0

CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.7 | VT102 | Offline | ttyUSB0
```

Figura 6: Primeiro sistema rodando na BBB.

3 Erros que podem acontecer

3.1 Erros no caso do TFTP não execute

Teste para ver se tem algum erro de TFTP

```
$ systemctl status xinetd.service
```

Possível erro que pode aparecer é no arquivo `/etc/xinetd.d/tftp` precisando concertar problemas de espaço e de chaves que acontecem:

Apr 11 08:23:43 werlley-M2 xinetd[1697]: Attribute protocol needs a space before operator [file=/etc/xinetd.d/tftp] [line=3]

- Depois que consertar confira com os seguintes comandos:

```
$ systemctl stop xinetd.service
$ sudo /etc/init.d/xinetd start
$ systemctl status xinetd.service
```

3.2 Esta parte é se acontecer erros de conexão – Solução de Conflitos de Porta no Serviço TFTP Gerenciado pelo xinetd

Problema: O serviço TFTP no Ubuntu, gerenciado pelo xinetd, apresentou um erro de vinculação à porta 69, pois estava em uso por outro processo do TFTP já ativo, resultando em falhas repetidas de início do serviço.

Detalhes Técnicos: A mensagem de erro "bind failed (Address already in use (errno = 98)). service = tftp" indicava que a porta 69 UDP já estava ocupada pelo processo in.tftpd (PID 2126), impedindo que o xinetd iniciasse o serviço TFTP.

Solução Implementada:

```
sudo netstat -ltn | grep :69
```

Verificação do uso da porta com o comando:

udp	0	0	0.0.0.0:69	0.0.0.0:*	2126/in.tftpd
udp6	0	0	:::69	:::*	2126/in.tftpd

que confirmou que o processo in.tftpd estava utilizando a porta.

Interrupção forçada do processo que estava usando a porta:

```
sudo kill -9 2126
```

Reinício do serviço xinetd para tentar iniciar o TFTP novamente:

```
$ sudo systemctl restart xinetd  
$ systemctl status xinetd
```

Resultado: Após matar o processo que estava bloqueando a porta e reiniciar o xinetd, o serviço TFTP foi iniciado corretamente, sem erros de vinculação, e começou a funcionar com 1 serviço disponível, como indicado pela mensagem "Started working: 1 available service".

Conclusão: O problema foi resolvido com sucesso através da liberação da porta e subsequente reinício do serviço. Este procedimento assegurou que o xinetd pudesse gerenciar o TFTP sem conflitos de porta.