LABORATÓRIO TPSE I

Laboratório 02: Sistema de Arquivo Embarcado com BusyBox

Prof. Francisco Helder

20 de setembro de 2022

O objetivo deste laboratório é criaremos um rootfs simples com o Busybox e depois montaremos este rootfs via NFS. Veremos como o NFS facilita bastante o processo de desenvolvimento em Linux embarcado. Após este laboratório, você será capaz de:

- Configurar e construir um kernel Linux que inicialize em um diretório em sua estação de trabalho, compartilhado através da rede por NFS.
- Criar e configurar um rootFilesystem minimalista a partir do zero para uma placa específica.
- Entender como um sistema Linux embarcado pode ser pequeno e simples.
- Instalar o BusyBox neste sistema de arquivos.
- Criar um script de inicialização simples baseado em /sbin/init.
- Configurar uma interface web simples para o alvo.

A gênese do BusyBox não teve nada a ver com Linux embarcado. O projeto foi instigado em 1996 por Bruce Perens para o instalador do Debian para que ele pudesse inicializar o Linux a partir de um disquete de 1,44 MB. Coincidentemente, isso era mais ou menos o tamanho do armazenamento em dispositivos contemporâneos e, portanto, a comunidade Linux embarcada rapidamente o adotou. BusyBox tem estado no coração do Linux embarcado desde então.

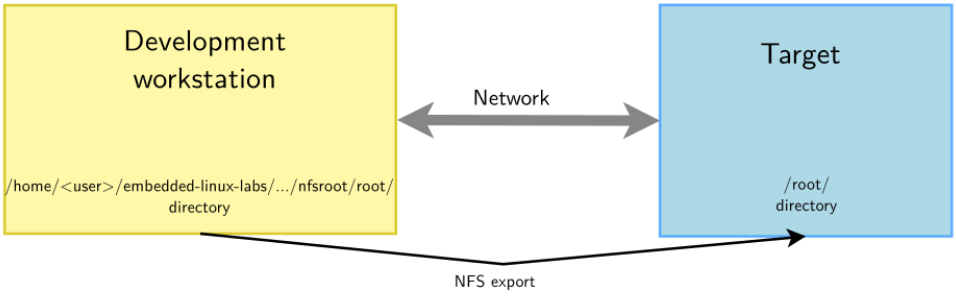
BusyBox foi escrito do zero para executar as funções essenciais desses utilitários essenciais do Linux. Os desenvolvedores aproveitaram a regra 80:20. Os 80% mais úteis de um programa são implementados em 20% do código. Portanto, as ferramentas BusyBox implementam um subconjunto das funções dos equivalentes de desktop, mas fazem o suficiente para serem úteis na maioria dos casos.

1 Implementação do laboratório

Enquanto desenvolve um rootFilesystem para um dispositivo, um desenvolvedor precisa fazer alterações frequentes no conteúdo do sistema de arquivos, como modificar scripts ou adicionar programas recém-compilados.

Não é nada prático atualizar o rootFilesystem na placa toda vez que uma alteração é feita. Felizmente, é possível configurar a rede entre a estação de trabalho de desenvolvimento e a placa. Em seguida, os arquivos da estação de trabalho podem ser acessados pela placa através da rede, usando NFS.

A menos que você teste uma sequência de inicialização, não será mais necessário reinicializar a placa para testar o impacto da atualizações de scripts ou aplicativos.



2 Kernel Configuração

Vamos reutilizar as fontes do kernel do nosso laboratório anterior, em **$HOME/lab/linux/**. Na configuração do kernel criada no laboratório anterior, verifique se você tem todas as opções necessárias para inicializar o sistema usando um rootFilesystem montado em NFS. Verifique também se **CONFIG_DEVTMPFS_MOUNT** está habilitado. Se necessário, reconstrua seu kernel.

3 Construindo BusyBox

BusyBox usa o mesmo sistema Kconfig e Kbuild que o kernel, então a compilação cruzada é direta. Você pode obter os fontes clonando o repositório BusyBox Git da versão desejada (1_35_0 é a mais recente no momento da redação), da seguinte maneira:

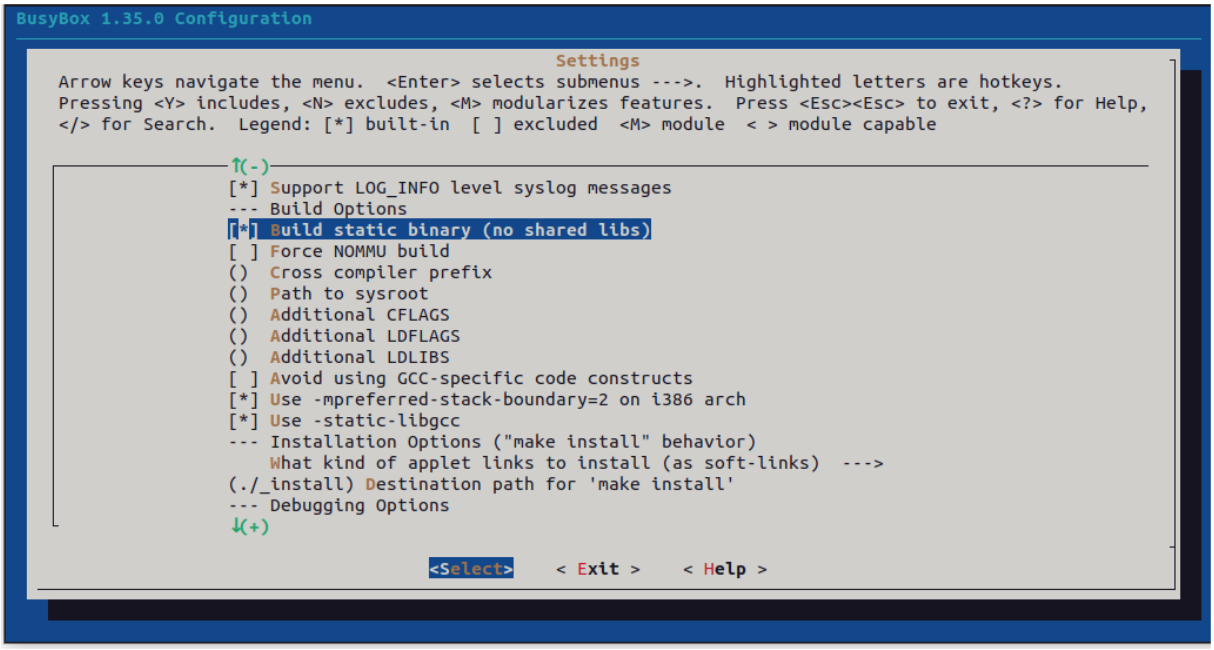
```
$ git clone git://busybox.net/busybox.git
$ cd busybox
$ git checkout 1_35_0
```

Você também pode baixar o arquivo TAR correspondente em https://busybox.net/downloads/.

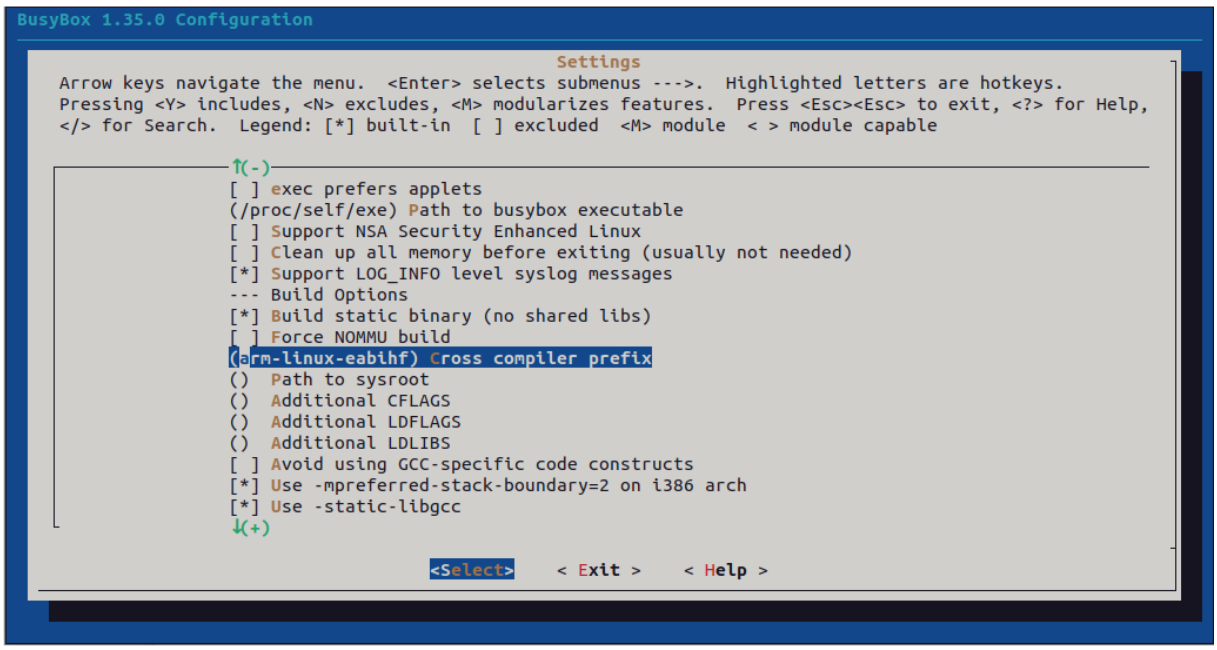
Em seguida, configure o BusyBox começando com a configuração padrão, que habilita praticamente todos os recursos do BusyBox:

```
$ make distclean
$ make defconfig
```

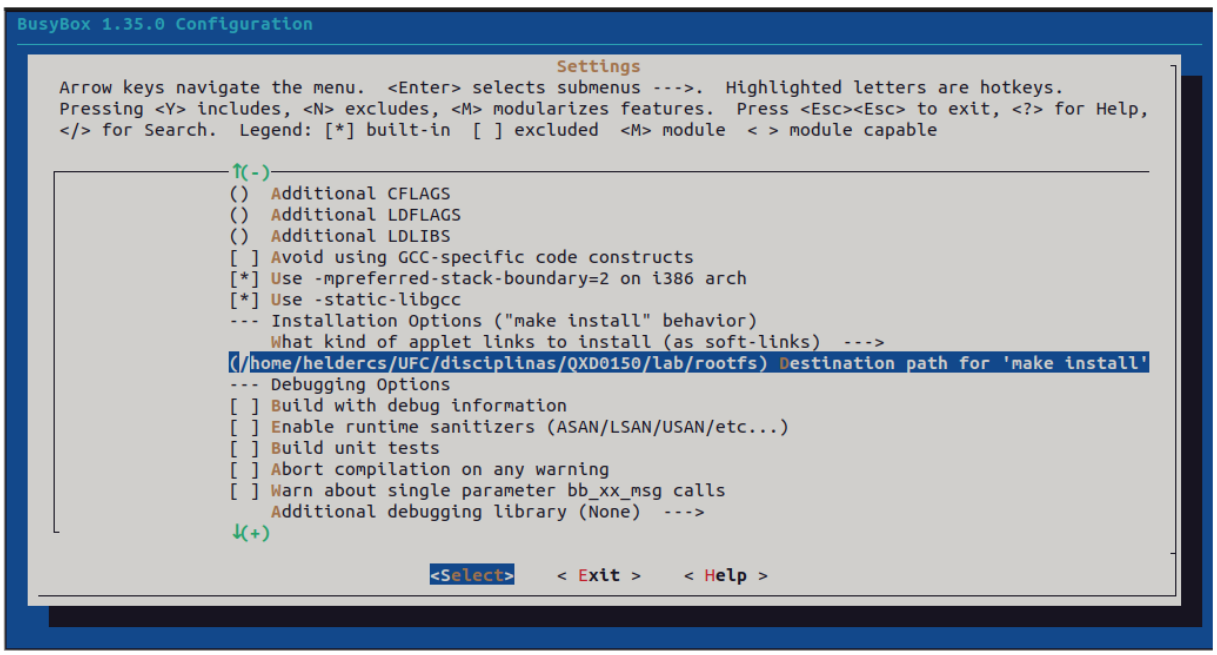
Abra o menuconfig para ajustar a configuração. Por exemplo, você pode mudar a configuração para construir o busybox como biblioteca estática. Selecione o menu “Busybox Settings -> Build Options -> Build static binary (no shared libs)”. Compilar o BusyBox estaticamente em primeiro lugar facilita a configuração do sistema, porque não há dependências de bibliotecas, podendo ser configurado mais tarde com bibliotecas compartilhadas e então recompilar um novo BusyBox.



você pode definir o caminho de instalação em **Busybox Settings | Installation Options (CONFIG_PREFIX)** para apontar para o diretório de teste. Então, você pode fazer a compilação cruzada da maneira usual. Acesse o menu "Busybox Settings -> Build Options -> (arm-linux-eabihf) Cross compiler prefix" e configure o prefixo do toolchain. Para a placa BeagleBone Black, use este comando



Outro configuração importante é definir o caminho de instalação do BosyBox "Busybox Settings -> Installation Options ("make install"behavior -> (./_install) Destination path for 'make install" e configure o caminho onde será instalado o rootFilesystem:



O Busybox já vem configurado por padrão com algumas ferramentas mínimas para a geração do rootfs. Sinta-se livre para navegar entre as opções de configuração e habilitar qualquer

ferramenta adicional. Salve e saia do menu de configuração. Agora compile o busybox:

```
$ make busybox -j8
```

Em ambos os casos, o resultado é o busybox executável. Para uma configuração padrão como esta, o tamanho é de cerca de 900 KiB. Se isso for muito grande para você, você pode reduzi-lo alterando a configuração para deixar de fora os utilitários que você não precisa. Após a compilação, verifique se o binário do Busybox foi gerado corretamente:

```
$ file busybox
busybox: ELF 32-bit LSB executable, ARM, EABI5 version 1 (GNU/Linux),
    statically linked, for GNU/Linux 3.2.0, BuildID[sha1]=
   dd45374ea9cd09a7cf750d5dc4c1bd332b740d7c, stripped
```

4 Criando o rootFilesystem

Agora instale o Busybox com o comando abaixo (o diretório do rootfs será criado com todas as aplicações e ferramentas habilitadas no menu de configuração).

```
$ make install
```

Estude o conteúdo deste diretório:

```
$ cd /home/XXX/lab_05/rootfs
$ ls -l
```

4.1 Nós do Dispositivos

A maioria dos dispositivos no Linux são representados por nós de dispositivos, de acordo com a filosofia Unix de que tudo é um arquivo (exceto interfaces de rede, que são sockets). Um nó de dispositivo pode se referir a um dispositivo de bloco ou a um dispositivo de caractere. Dispositivos de bloco são dispositivos de armazenamento em massa, como cartões SD ou discos rígidos. Um dispositivo de caractere é praticamente qualquer outra coisa, mais uma vez, com exceção de interfaces de rede. A localização convencional para nós de dispositivo é o diretório chamado **/dev**. Por exemplo, uma porta serial pode ser representada pelo nó do dispositivo chamado **/dev/ttyS0**. Os nós de dispositivo são criados usando o programa chamado mknod (abreviação de make node):

```
$ mknod <name> <type> <major> <minor>
```

Os parâmetros para o comando mknod são os seguintes:

- **name** é o nome do nó do dispositivo que você deseja criar.
- **type** é c para dispositivos de caractere ou b para um dispositivo de bloco.
- **major** e **minor** são um par de números que são usados pelo kernel para rotear solicitações de arquivo para o código de driver de dispositivo apropriado. Há uma lista de números principais e secundários padrão na fonte do kernel no arquivo Documentation/devices.txt.

Você precisará criar nós de dispositivo para todos os dispositivos que deseja acessar em seu sistema. Você pode fazer isso manualmente usando o comando mknod como ilustrado. Em um rootFilesystem realmente mínimo, você precisa de apenas dois nós para inicializar com o BusyBox: console e null. O console só precisa ser acessível ao root, o proprietário do nó do

dispositivo, então as permissões de acesso são 600 (rw————-). O dispositivo nulo deve ser legível e gravável por todos, então o modo é 666 (rw-rw-rw-). Você pode usar a opção -m para mknod para definir o modo ao criar o nó. Você precisa ser root para criar nós de dispositivo, conforme mostrado aqui:

```
$ cd ~/rootfs
$ mkdir dev
$ sudo mknod -m 666 dev/null c 1 3
$ sudo mknod -m 600 dev/console c 5 1
```

4.2 O Sistema de Arquivos sys e proc

proc e sysfs são dois pseudo sistemas de arquivos que fornecem uma janela para o funcionamento interno do kernel. Ambos representam os dados do kernel como arquivos em uma hierarquia de diretórios: quando você lê um dos arquivos, o conteúdo que você vê não vem do armazenamento em disco; ele foi formatado em tempo real por uma função no kernel. Alguns arquivos também são graváveis, o que significa que uma função do kernel é chamada com os novos dados que você escreveu e, se estiver no formato correto e você tiver permissões suficientes, ela modificará o valor armazenado na memória do kernel. Em outras palavras, proc e sysfs fornecem outra maneira de interagir com drivers de dispositivo e outros códigos do kernel. Os sistemas de arquivos proc e sysfs devem ser montados nos diretórios chamados **/proc** e **/sys**:

```
$ mkdir proc
$ mkdir sys
```

Crie diretórios de kernel e de biblioteca de espaço do usuário. Copie todas as bibliotecas do diretório de compilação cruzada para os diretórios /lib e /usr /lib na partição nfs.

```
$ mkdir lib usr/lib
$ rsync -a ~/arm-linux-gnueabihf/arm-linux-gnueabihf/libc/lib/ ./lib/
```

Agora nosso sistema de arquivo rootfs está bem completo (faltam apenas os scripts de boot):

4.3 Adicionando os Scripts de Boot

O primeiro processo que o kernel executará será o /sbin/init. Este processo é disponibilizado pelo Busybox e segue o padrão System V Init. O /sbin/init procura e executa os comandos descritos no arquivo de configuração /etc/inittab. Vamos então criar este arquivo. Crie o diretório /etc e, em seguida, crie arquivos adicionais dentro deste diretório:

```
$ mkdir etc
cat >> etc/inittab
null::sysinit:/bin/mount -a
null::sysinit:/bin/hostname -F /etc/hostname
null::respawn:/bin/cttyhack /bin/login root
null::restart:/sbin/reboot
[Ctrl-D]
```

Crie outro arquivo chamado fstab e preencha-o. Este arquivo montará os sistemas de arquivos virtuais.

```
$cat >> etc/fstab
proc  /proc proc  defaults  0 0
sysfs /sys  sysfs defaults  0 0
[ctrl-D]
```

Além disso, crie arquivos chamados hostname e passwd.

```
$ cat >> etc/hostname
myhostemedded
[ctrl-D]
cat >> etc/passwd
root::0:0:root:/root:/bin/sh
[ctrl-D]
```

Perceba que no inittab estamos iniciando a aplicação getty, que irá executa a aplicação de terminal (/bin/sh) na conexão serial ttyS0. Ou seja, quando você iniciar o target, terá acesso ao terminal para a execução de comandos. Agora que já temos um rootfs básico, vamos testá-lo na placa com o NFS.

5 Configurando o servidor NFS

NFS, ou Network File System, é um protocolo de sistema distribuído de arquivos que permite a montagem de diretórios remotos no seu servidor. Isso permite que você gerencie o espaço de armazenamento em um local diferente e grave nesse espaço a partir de vários clientes. O NFS fornece uma maneira relativamente padronizada e de bom desempenho para acessar sistemas remotos através de uma rede e funciona bem em situações onde os recursos compartilhados precisam ser acessados regularmente.

Instale o pacote nfs-kernel-server, o qual permitirá que você compartilhe seus diretórios. Como esta é a primeira operação que você está executando com o apt nesta sessão, atualize seu índice de pacotes local antes da instalação:

```
$ sudo apt install nfs-kernel-server
sudo service nfs-kernel-server start
```

Crie um diretório rootfs no diretório de laboratório atual. Este diretório rootfs será usado para armazenar o conteúdo do nosso novo sistema de arquivos raiz. Instale o servidor NFS instalando o pacote nfs-kernel-server caso ainda não o tenha. Uma vez instalado, edite o arquivo /etc/exports como root para adicionar a seguinte linha, assumindo que o endereço IP da sua placa será 192.168.0.100:

```
/home/<user>/<path>/rootfs 10.4.1.2(rw,no_root_squash,
   no_subtree_check)
```

Claro, substitua <user> pelo seu nome de usuário real e o <path> pelo caminho onde está seu rootFilesystem. Certifique-se de que o caminho e as opções estejam na mesma linha. Certifique-se também de que não há espaço entre o endereço IP e as opções NFS, caso contrário, as opções padrão serão usadas para este endereço IP, fazendo com que seu root filesystem seja somente leitura. Em seguida, faça do servidor NFS a nova configuração:

```
$ sudo service nfs-kernel-server restart
```

6 Configurando o boot por NFS no target

Para configurar a placa para acessar o rootfs via NFS, precisamos alterar a variável de ambiente **bootargs** do U-Boot, passando os parâmetros de sistema de arquivo por NFS. Acesse então o terminal de comandos do U-Boot. Vamos primeiro criar uma variável chamada bootnfs para

armazenar os parâmetros de boot por NFS. Para isso, execute o comando abaixo, substituindo <rootfs_dir> pelo diretório onde se encontra o rootfs no host. Exemplo: /home/<path>/rootfs:

```
Uboot$ setenv argsnfs root=/dev/nfs rw nfsroot=${serverip}:<
   rootfs_dir> rootwait
```

Agora configure a variável bootargs, passando os parâmetros de boot por NFS, conforme abaixo:

```
Uboot$ setenv consolecfg console=ttyS0,115200,n8
Uboot$ setenv ipcfg ip=X.X.X.X
Uboot$ setenv bootargs ${consolecfg} ${ipcfg} ${argsnfs}
```

Salve a configuração:

```
Uboot$ saveenv
```

Conecte o host ao target com o cabo de rede e inicie o processo de boot:

```
Uboot$ boot
```

Ao reiniciar a placa, o kernel irá montar o rootfs com NFS e no fim do boot você deverá ter acesso ao terminal de comandos na console serial:

7 Atividades Práticas

pratica 1

Gere um sistema de arquivo personalizado seu

pratica 2

Implemente um código em C ou em shell script para acessar os LEDs da placa

pratica 3

Implemente uma interface web para a sua placa e acesse via seu celular