Linux Root Filesystem

Técnicas de Programação para Sistemas Embarcados II

Prof. Francisco Helder

Universidade Federal do Ceará

September 19, 2022

UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS QUIXADÁ

Filesystems

- Os sistemas de arquivos são usados para organizar dados em diretórios e arquivos em dispositivos de armazenamento ou na rede. Os diretórios e arquivos são organizados como uma hierarquia
- Em sistemas UNIX, aplicativos e usuários veem uma única hierarquia global de arquivos e diretórios, que pode ser composta por vários sistemas de arquivos
- Os sistemas de arquivos são **montados** em um local específico nesta hierarquia de diretórios
  - Quando um sistema de arquivos é montado em um diretório (chamado ponto de montagem), o conteúdo desse diretório reflete o conteúdo desse sistema de arquivos
  - Quando o sistema de arquivos é desmontado, o ponto de montagem fica vazio novamente
- Isso permite que os aplicativos acessem arquivos e diretórios facilmente, independentemente do local exato de armazenamento

Filesystems: Exemplo

- Crie um ponto de montagem, que é apenas um diretório

```
$ sudo mkdir /mnt/usbkey
```

- Ele está vazio

```
$ ls /mnt/usbkey
$
```

- Monte um dispositivo de armazenamento neste ponto de montagem

```
$ sudo mount -t vfat /dev/sda1 /mnt/usbkey
```

- Agora você pode acessar o conteúdo do dispositivo USB

```
$ ls /mnt/usbkey
docs prog.c picture.png movie.avi
```

mount/umount

mount permite mountar sistemas de arquivos

```
1
2
    $ mount -t type device mountpoint
```

type é o tipo de sistema de arquivo (opcional para sistema de arquivo no-virtual)
device é o dispositivo de armazenamento, ou rede para montagem
mountpoint é o diretório onde o dispositivo de armazenamento ou local de rede estarão acessíveis
mount sem argumentos mostra os sistemas de arquivos montados atualmente

umount permite desmontar sistemas de arquivos

Isso é necessário antes de reinicializar ou antes de desconectar um dispositivo USB, porque o kernel armazena em cache as gravações na memória para aumentar o desempenho, **umount** garante que essas gravações sejam confirmadas no armazenamento.

Root Filesystem

- Um sistema de arquivos específico é montado no root da hierarquia, identificado por /
- Este sistema de arquivos é chamado de root filesystem
- Como **mount** e **umount** são programas, são arquivos dentro de um sistema de arquivos.
  - Eles não são acessíveis antes de montar pelo menos um sistema de arquivos.
- Como o root filesystem é o primeiro sistema de arquivos montado, ele não pode ser montado com o comando **mount** normal
- É montado diretamente pelo kernel, de acordo com a opção **root=**
- Quando nenhum sistema de arquivos raiz está disponível, o kernel entra em pânico:

```
1    Please append a correct ``root='' boot option
2    Kernel panic - not syncing: VFS: Unable to mount root fs on unknown block(0,0)
3
```

Montando rootfs de Dispositivos de Armazenamento

Partições de um disco rígido ou USB

root=/dev/sdXY, onde X é uma letra que indica o dispositivo e Y a partição
/dev/sdb2 é a segunda partição da segunda unidade de disco (USB ou disco rígido ATA)

Partições de um cartão SD

root=/dev/mmcblkXpY, onde X é um número que indica o dispositivo e Y a partição
/dev/mmcblk0p2 é a segunda partição do primeiro dispositivo

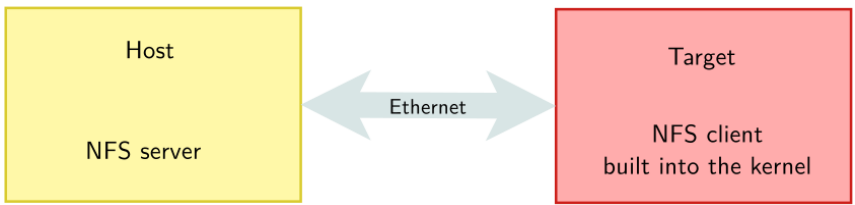
Partições de armazenamento flash

root=/dev/mtdblockX, onde X é o número da partição
/dev/mtdblock3 é a quarta partição flash no sistema (pode haver vários chips flash)

Montando rootfs na Rede

Uma vez que a rede funciona, o root filesystem pode ser um diretório em seu host de desenvolvimento GNU/Linux, exportado por NFS (Network File System). Isso é muito conveniente para o desenvolvimento do sistema:

- Facilita muito a atualização de arquivos no root filesystem, sem reinicializar
- Pode ter um grande root filesystem, mesmo que você ainda não tenha suporte para armazenamento interno ou externo
- O root filesystem pode ser enorme. Você pode até construir ferramentas de compilador nativas e construir todas as ferramentas que você precisa no próprio target (embora seja melhor fazer a compilação cruzada)



Montando rootfs na Rede

No lado da estação de trabalho de desenvolvimento, é necessário um servidor NFS

- Instale um servidor NFS (exemplo Debian e Ubuntu)

```
1
2
        sudo apt-get install nfs-kernel-server
```

- adicione um diretório a ser exportado no arquivo **/etc/exports**:

```
1
2
        /home/user/rootfs 10.4.1.2(rw,no_root_squash,no_subtree_check)
```

  - **10.4.1.2** é o IP do cliente
  - **rw,no_root_squash,no_subtree_check** são as opções do servidor NFS

- Peça ao seu servidor NFS para recarregar este arquivo:

```
1
2
    $ sudo service nfs-kernel-server restart
```

Montando rootfs na Rede

- O kernel deve ser compilado com:

CONFIG_NFS_FS=y (suporte a cliente NFS)
CONFIG_IP_PNP=y (configuração do IP em tempop de boot)
CONFIG_ROOT_NFS=y (suporte a NFS como rootfs)

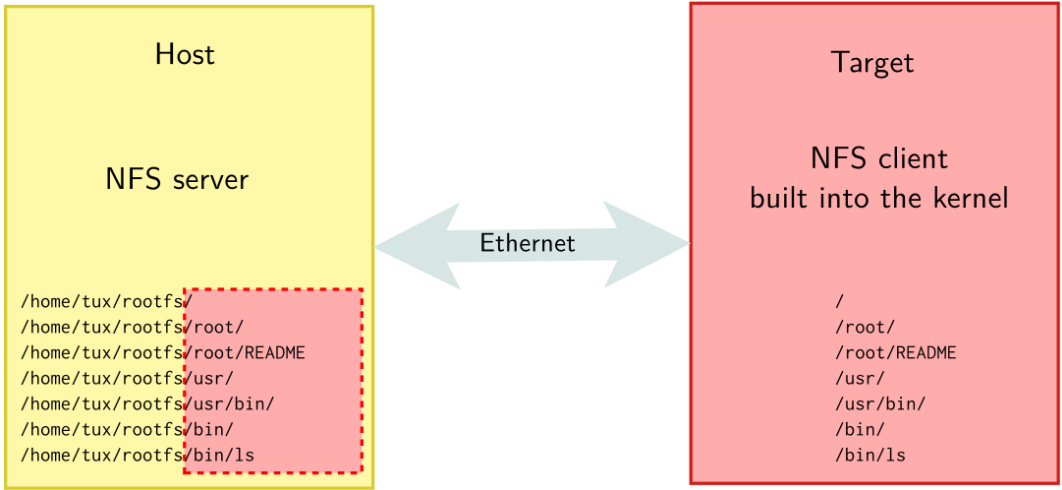
- O kernel deve ser inicializado com os seguintes parâmetros:

root=/dev/nfs (we want rootfs over NFS)
ip=10.4.1.2 (endereço IP address)
nfsroot=10.4.1.1:/home/heldercs/rootfs/ (servidor NFS)

Atenção

Você pode precisar adicionar **,nfsvers=3,tcp** à configuração nfsroot, pois um cliente NFS versão 2 e UDP podem ser rejeitados pelo servidor NFS em distribuições GNU/Linux recentes.

Montando rootfs na Rede



Filesystem Virtual proc

- O sistema de arquivos virtual **proc** existe desde o início do Linux
- Permite:
  - O kernel expor estatísticas sobre processos em execução no sistema
  - O usuário deve ajustar em tempo de execução vários parâmetros do sistema
- O sistema de arquivos **proc** é usado por muitos aplicativos padrão, e eles esperam que ele seja montado em **/proc**
- Aplicativos como ps ou top não funcionariam sem o sistema de arquivos **proc**
- Comando para montar **proc**:

```
1
2
```

```
$ mount -t proc nodev /proc
```

- Veja **filesystems/proc** na documentação do kernel ou **man proc**

Filesystem sysfs

- Permite representar no espaço do usuário a visão que o kernel tem dos barramentos, dispositivos e drivers do sistema
- É útil para vários aplicativos que precisam listar e consultar o hardware disponível, por exemplo, udev ou mdev
- Todos os aplicativos que usam sysfs esperam que ele seja montado no diretório /sys
- comando para mountar o **sys**:

```
1  $ mount -t sysfs nodev /sys
2  $ ls /sys
3  block bus class dev devices firmware fs kernel module power
4
```

Filesystem Minimo: Aplicações Básicas

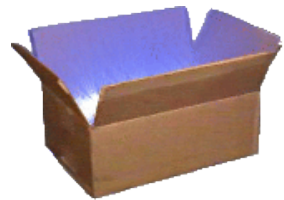
- Para funcionar, um sistema Linux precisa de pelo menos alguns aplicativos
- A aplicatição **init**, que é o primeiro aplicativo iniciado pelo kernel após a montagem do rootfilesystem:
  - O kernel tenta executar o comando especificado pelo parâmetro de linha de comando **init**, se disponível
  - Caso contrário, ele tenta executar **/sbin/init**, **/bin/init**, **/etc/init** e **/bin/sh**
  - No caso de um initramfs, ele procurará apenas por **/init**
  - Se nada disso funcionar, o kernel entra em pânico e o processo de inicialização é interrompido
  - A aplicação **init** é responsável por iniciar todos os outros aplicativos e serviços e por atuar como um pai universal para processos
- Um **shell**, para implementar scripts, automatizar tarefas e permitir que um usuário interaja com o sistema
- Executáveis básicos do UNIX, para uso em scripts de sistema ou em shells interativos: mv, cp, mkdir, cat, modprobe, mount, ip, etc
- Esses componentes básicos devem ser integrados ao rootfilesystem para torná-lo utilizável

Por que o BusyBox?

- Um sistema Linux precisa de um conjunto básico de programas para funcionar
  - Um programa de inicialização
  - Como o **shell**
  - Vários utilitários básicos para manipulação de arquivos e configuração do sistema
- Em sistemas GNU/Linux normais, esses programas são fornecidos por diferentes projetos
  - coreutils, bash, grep, sed, tar, wget, modutils, etc. são todos projetos diferentes
  - Muitos componentes diferentes para integrar
  - Componentes não projetados com restrições de sistemas embarcados em mente: eles não são muito configuráveis e possuem uma ampla gama de recursos
- BusyBox é uma solução alternativa, extremamente comum em sistemas embarcados

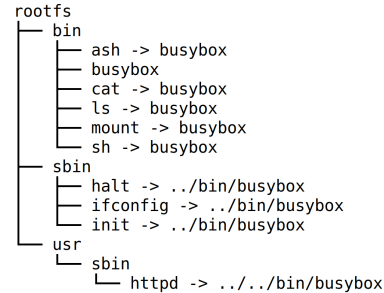
ToolBox de Uso Geral: BusyBox

- Reescrever muitos utilitários de linha de comando UNIX úteis
  - Criado em 1995 para implementar um sistema de resgate
  - Integrado em um único projeto, o que facilita o trabalho
  - Projetado com sistemas embarcados em mente: altamente configurável, sem recursos desnecessários
  - Chamado de canivete suíço do Linux Embarcado
- Licença: GNU GPLv2
- Alternativa: Toybox, licenciado BSD



ToolBox de Uso Geral: BusyBox

- Todos os utilitários são compilados em um único executável, **/bin/busybox**
  - Links simbólicos para **/bin/busybox** são criados para cada aplicativo integrado ao BusyBox
- Para configuração bastante funcional, menos de 500 KB (compilado estaticamente com uClibc) ou menos de 1 MB (compilado estaticamente com glibc).



Configurando BusyBox

- Obtenha as fontes estáveis mais recentes em https://busybox.net
- Configure o BusyBox (cria um arquivo .config):

```
$ make defconfig
Bom para comecar com BusyBox.
Configura BusyBox com todas as opcoes para usuarios regulares.

```

```
$ make allnoconfig
Desmarca todas as opcoes. Bom para configurar apenas o que voce precisa.

```

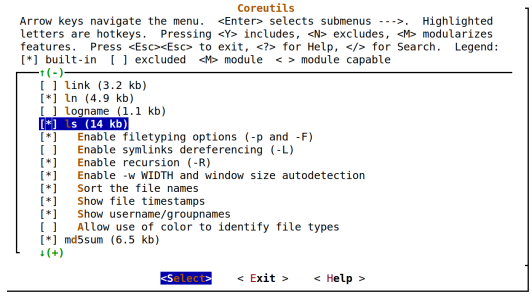
```
$ make menuconfig

```

Mesmas interfaces de configuração que as usadas pelo kernel do Linux (embora versões mais antigas sejam usadas, fazendo com que o **make xconfig** seja quebrado em distribuições recentes).

BusyBox make menuconfig

- Você pode escolher:
  - os comandos para compilar,
  - e até as opções de comando e recursos que você precisa!



Compilando BosyBox

- Defina o prefixo do compilador cruzado na interface de configuração:

```
1 Settings->Build Options->Cross Compiler prefix
2 Exemplo: arm-linux-
3
```

- Defina o diretório de instalação na interface de configuração:

```
1 Settings->Installation Options->BusyBox installation prefix
2
```

- Adicione o caminho do compilador cruzado à variável de ambiente PATH:

```
1 $ export PATH=$HOME/x-tools/arm-unknown-linux-gnueabi/bin:$PATH
2
```

- Compilar BusyBox:

```
1 $ make
2
```

- Instalar (cria uma estrutura de diretório com links simbólicos para o executável):

```
1 $ maake install
2
```

Prática de lab - Um Pequeno FileSystem Embarcado

Hora de começar o laboratório prático!

1. Faça o Linux inicializar em um diretório em sua estação de trabalho, compartilhado pelo NFS
2. Crie e configure um sistema embarcado Linux minimalista
3. Instalar e usar BusyBox
4. Inicialização do sistema com /sbin/init
5. Configure uma interface web simples
6. Use bibliotecas compartilhadas