TRƯỜNG ĐẠI HỌC BÁCH KHOA – ĐẠI HỌC QUỐC GIA TP.HCM
TRUNG TÂM KỸ THUẬT ĐIỆN TOÁN
268 Lý Thường Kiệt, Phường 14, Quận 10, TP.Hồ Chí Minh
Điện thoại: 84-8-3864 7256 ext 5371 – Fax: 84-8-3865 8687
Website: **www.cce.hcmut.edu.vn**
E-mail: **dientoan@hcmut.edu.vn**

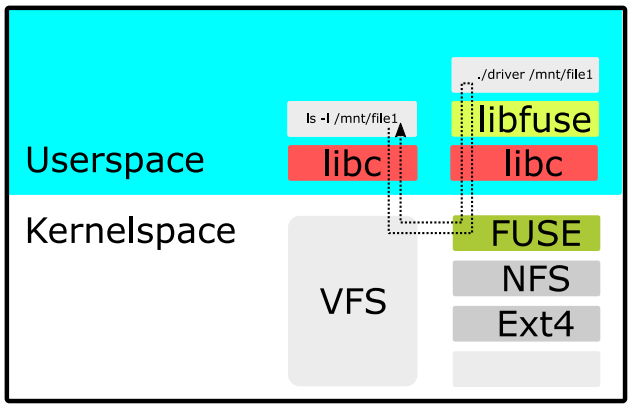
Lab 9 – Virtual File System

1 Introduction

The Virtual File System (VFS) [1] is the software layer in the kernel that provides the filesystem interface to userspace programs.

In computing, filesystem is a method and data structure that the operating system uses to control how the data is stored and retrieved [1]. Filesystem can be used on many storage media, some are used on local data storage devices; others provide file access via a network protocol. Some file system are "virtual", meaning that the files are virtual and computed on request. e.g. procfs and sysfs. To avoid confusion, the purpose of VFS, in general, is to allow communication between user space and kernel space in a uniform way regardless of various types of concrete file system. VFS is implented as an abstract layer.

In our practicing environment using Linux, filesystem can be implemented as kernel modules. But there is also Filesystem in USErspace (aka. FUSE) interface, which can allow user process act as a filesystem driver.



You can see all supported filesystem

```
$ cat /proc/filesystem
```

2 Background

2.1 Linux file system

In UNIX philosophy, everything is a file. The advantage of this approach is that the same set of tools, ultilities and APIs can be used on a wide range of resources. A program can use such a file system to perform simple file operations and do not have to implement complex hardware protocol underneath.

A **device file** is an interface to a device driver that appears in a file system. There is two general kinds of device files in Unix-like operating systems, know as character special files and block special files. The main memory of the computer is a character special file mounted under /dev/mem path.

[1] https://www.kernel.org/doc/Documentation/filesystems/vfs.txt

A range of **pseudo** and **virtual filesystems** exists which exposes information about processes and other system information in a hierarchical file-like structure. ProcFS and SysFS are pseudo filesystem provided in Linux.

2.2 File system in Linux kernel[2]

2.2.1 Directory Entry Cache (dcache)¶

The VFS implements the open, stat, chmod, and similar system calls. The pathname argument that is passed to them is used by the VFS to search through the directory entry cache (also known as the dentry cache or dcache). This provides a very fast look-up mechanism to translate a pathname (filename) into a specific dentry. Dentries live in RAM and are never saved to disc: they exist only for performance.

```
$ cat filepath
```

The dentry cache is meant to be a view into your entire filespace. As most computers cannot fit all dentries in the RAM at the same time, some bits of the cache are missing. In order to resolve your pathname into a dentry, the VFS may have to resort to creating dentries and then loading the inode. This is done by looking up the inode.

2.2.2 The Inode Object

An individual dentry usually has a pointer to an inode. Inodes are filesystem objects such as regular files, directories, FIFOs and other beasts. They live either on the disc (for block device filesystems) or in the memory (for pseudo filesystems). Inodes that live on the disc are copied into the memory when required and changes to the inode are written back to disc.
To look up an inode requires that the VFS calls the lookup() method of the parent directory inode. This method is installed by the specific filesystem implementation that the inode lives in. we can do all those boring things like open the file, or stat it to peek at the inode data. he stat operation is fairly simple: once the VFS has the dentry, it peeks at the inode data and passes some of it back to userspace.

2.2.3 The File Object

Opening a file requires another operation: allocation of a file structure (this is the kernel-side implementation of file descriptors). The freshly allocated file structure is initialized with a pointer to the dentry and a set of file operation member functions. These are taken from the inode data. The open() file method is then called so the specific filesystem implementation can do its work. You can see that this is another switch performed by the VFS. The file structure is placed into the file descriptor table for the process.
Reading, writing and closing files (and other assorted VFS operations) is done by using the userspace file descriptor to grab the appropriate file structure, and then calling the required file structure method to do whatever is required. For as long as the file is open, it keeps the dentry in use, which in turn means that the VFS inode is still in use.

[2] Reference from Linux Virtual File System Document https://www.kernel.org/doc/html/latest/filesystems/vfs.html

2.2.4 Pathname lookup

The most obvious aspect of pathname lookup, which very little exploration is needed to discover. Computer science has long been acquainted with such complexity and has tools to help manage it. One tool that we will make extensive use of is "divide and conquer".

Pathnames (sometimes "file names"), used to identify objects in the filesystem, will be familiar to most readers. They contain two sorts of elements: "slashes" that are sequences of one or more "/" characters, and "components" that are sequences of one or more non-"/" characters. These form two kinds of paths. Those that start with slashes are "absolute" and start from the filesystem root.

These paths can be divided into two sections: the final component and everything else:

- The final component is not so simple. Not only do different system calls interpret it quite differently (e.g. some create it, some do not), but it might not even exist: neither the empty pathname nor the pathname that is just slashes have a final component. If it does exist, it could be "." or ".." which are handled quite differently from other components.

Pathname ends with a slash, such as "`/tmp/foo/`". In particular, mkdir() and rmdir() each create or remove a directory named by the final component, and they are required to work with pathnames ending in "/"

The Linux pathname walking code deals with all of these issues: breaking the path into components, handling the "everything else" quite separately from the final component, and checking that the trailing slash is not used where it isn't permitted.

2.2.5 The symlink

There are only two sorts of filesystem objects that can usefully appear in a path prior to the final component: directories and symlinks. Handling directories is quite straightforward: the new directory simply becomes the starting point at which to interpret the next component on the path. Handling symbolic links requires a bit more work.
Conceptually, symbolic links could be handled by editing the path. If a component name refers to a symbolic link, then that component is replaced by the body of the link and, if that body starts with a '/', then all preceding parts of the path are discarded. This is what the "readlink -f" command does.

```
$ readlink –f /usr/bin/vim
$ readlink –f /usr/lib/libxxx.so.yy
```

Like other filesystem resources, such as inodes and directory entries, symlinks are cached by Linux to avoid repeated costly access to external storage. Following the symlink iterates seamlessly over all components in the path and all of the non-final symlinks. As symlinks are processed, the name pointer is adjusted to point to a new symlink, or is restored from the stack, so that much of the loop doesn't need to notice.

2.3 ProcFS and SysFS

ProcFS[3] presents the information about processes and other system information. It provides a more convenience and standardized method for dynamically accessing process data held in kernel instead of tracing and direct acessing to kernel memory. For example, the GNU version of processing report ultility *ps* used the proc filesystem to obtain data, without using any specialized *system calls*.

- We can *retrieve* various information in read-only part of */proc* file system:
    1. Process-specific subdirectories (/proc/PID)
    2. Kernel info in /proc/
    3. Network info in /proc/net
    4. SCSI info in /proc/scsi
    5. Parallel port info in /proc/parport
    6. TTY info in /proc/tty
    7. Miscellaneous kernel statistic in /proc/stat
    8. Filesystem info in /proc/fs/<FS_ID>
    9. Console info in /proc/console
- We can *modify* system parameters: writing info file found in /proc/sys [4] or using *sysctl* is a utility for interacting with /proc. There is a systemcall "sysctl" which was deprecated a decade ago and is set to finally eliminated the code since Linux 5.5.

abi/	binary emulation relevant aka personality types
debug/	NONE
dev/	device specific information
fs/	tune and monitor
kernel/	global kernel info/tuning
net/	networking parts
vm/	memory management tuning, buffer and cache management
user/	Override the default limits on the number of namespaces

SysFS[5] is a ram-based file system. It provides a mean to export kernel data structures, their attributes, and the linkages between them to userspace.

> *In the beginning (UNIX), programs fetch process info via directly reading process structure from kernel memory via /dev/mem. The /proc filesystem is designed to publish process info and some system attributes, but its methodology of implementing control was ad-hoc then it soon grew into a tangled mess. The sysfs was designed to add structure to this mess by exporting the information present in device tree and is mounted under /sys mount point.*

SysFS is a little bit more structured and restricted in supporting only two methods *show* and *store*. It is created during the 2.6 kernel release cycle to fix the limitation of existing Linux driver model:

- Lack of unified method to represent the device driver relationship.
- No generic hotplug mechanism

[3] https://www.kernel.org/doc/Documentation/filesystems/proc.txt
[4] https://www.kernel.org/doc/Documentation/admin-guide/sysctl/
[5] https://www.kernel.org/doc/Documentation/filesystems/sysfs.txt

- Procfs was cluttered with non-process information

The directory layout of /sys file

- /sys/block : contains known block devices.
- /sys/bus : contains all registered buses.
- /sys/class : contains devices.
- /sys/device : all devices
- /sys/firmware : contains firmware files for some devices.
- /sys/fs : contains files to control filesystems.
- /sys/kernel : various kernel related files.
- /sys/module : loaded kernel modules..
- /sys/power : various files to handle power state of system

3 Practice

3.1 Retrieve read only information

By using the cat, more, or less commands on files within the /proc/ directory, users can immediately access enormous amounts of information about the system

- Process related information: to get the information of the process, all you have to do is read the files in /proc/PID/ directory

```
$ cat /proc/PID/statm
```

The specification of /proc entries

File	Content
clear_refs	Clears page referenced bits shown in smaps output
cmdline	Command line arguments
cpu	Current and last cpu in which it was executed (2.4)(smp)
cwd	Link to the current working directory
environ	Values of environment variables
exe	Link to the executable of this process
fd	Directory, which contains all file descriptors
maps	Memory maps to executables and library files (2.4)
mem	Memory held by this process
root	Link to the root directory of this process
stat	Process status
statm	Process memory status information
status	Process status in human readable form
pagemap	Page table
stack	
…	…

- kernel /hardware information

Check which interrupts are currently in use and what they are used for by looking in the file /proc/interrupts

```
$ cat /proc/interrupts
           CPU0
  0:    8728810          IO-APIC  timer
  1:        895          IO-APIC  keyboard
  2:          0          IO-APIC  cascade
...
```

Get the information about network devices are available in your system

```
$ cat /proc/net/dev
Inter-|Receive                                                |[...
 face |bytes    packets errs drop fifo frame compressed multicast|[...
    lo:  908188   5596     0    0    0     0          0         0 [...
 ppp0:15475140  20721   410    0    0   410          0         0 [...
 eth0:  614530   7085     0    0    0     0          0         1 [...
```

3.2 Filesystem in Linux kernel

3.2.1 Retrieve the file system information

```
$ls /dev/disk/by-
by-id/   by-path/ by-uuid/

$ ls /dev/disk/by-uuid/ -l
total 0
lrwxrwxrwx 1 root root 10 Oct 19 06:48 5d8561df-d5a7-48c1-9327-fd23bfb2cfa3 ->
../../sda5
lrwxrwxrwx 1 root root 10 Oct 19 06:48 6abb8ed8-842e-4b4b-81c4-9b0b5291d54b ->
../../sda1
```

The UUID of the disk drive is used to initial identify where the filesystem is located event before the FS is initialized itself. For those purposes, the boodloader store uses all these information in it configuration. For example.

```
$ vi /boot/grub/grub.cfg

menuentry 'Ubuntu' --class ubuntu --class gnu-linux --class gnu --class os
$menuentry_id_option 'gnulinux-simple-6abb8ed8-842e-4b4b-81c4-9b0b5291d54b' {
        recordfail
        load_video
        gfxmode $linux_gfx_mode
        insmod gzio
        insmod part_msdos
        insmod ext2
        set root='hd0,msdos1'
        if [ x$feature_platform_search_hint = xy ]; then
          search --no-floppy --fs-uuid --set=root --hint-bios=hd0,msdos1 --hint-
efi=hd0,msdos1 --hint-baremetal=ahci0,msdos1  6abb8ed8-842e-4b4b-81c4-9b0b5291d54b
        else
          search --no-floppy --fs-uuid --set=root 6abb8ed8-842e-4b4b-81c4-
9b0b5291d54b
        fi
        linux   /boot/vmlinuz-4.4.0-142-generic root=UUID=6abb8ed8-842e-4b4b-81c4-
9b0b5291d54b ro find_preseed=/preseed.cfg noprompt quiet
        initrd  /boot/initrd.img-4.4.0-142-generic
}
```

3.2.2 Boot file system

During the boot of a Unix system, the kernel does a few things that it doesn't do during normal operation. One of these things is to mount a filesystem on the directory /; this is quite different from normal mounting operations since the mounting is not triggered of a mount system call, and the target directory is not an existing directory. Another thing is to execute a program as PID 1, which is different from normal operation since this creates a process without duplicating an existing process.

The way this "magic" mounting of the root directory is very different in different Unix variants. The kernel chooses what device to mount based on configuration parameters that may be specified in a variety of ways: compile-time configuration, runtime configuration in the kernel image, runtime configuration in some predefined memory location, command line parameters.

- Linux can start with a "special" filesystem attached to the path /, which consists of files stored in RAM. This special filesystem is called initramfs. The initramfs is compiled directly into the kernel image that is loaded into memory by the bootloader.
- Linux can mount a device to / that is part of a restricted (but large) set of volume types that are recognized by the initialization code in the kernel. Such device types include any filesystem on common types of partitions on common types of disks (anything vaguely SCSI-like, including ATA, USB, etc.), as well as RAM disks and NFS mounts.
- Depending on which path was taken, the initial root filesystem may later be shadowed or replaced by another. Shadowing is what happens with an initramfs, and that's how most desktop and server systems operate (embedded systems, on the other hand, often have a hard-coded root filesystem). Replacement is what happens with an initrd, which is a particular kind of RAM disk. The job of the initramfs or initrd is to load the drivers that provide the "real" root filesystem that will get used in normal operation.

```
$ sudo cpio -ivF /boot/initramfs-5.14.14-300.fc35.x86_64.img
.
early_cpio
kernel
kernel/x86
kernel/x86/microcode
kernel/x86/microcode/GenuineIntel.bin
3080 blocks
```

cpio copies files into an archive. It reads a list of filenames, one per line, on the standard input, and writes the archive onto the standard output.

3.2.3 Root filesystem *[6]

Disk-based root file system

This is the normal mode where the root is set to a hdX device.

```
set root='hd0,msdos1'
linux   /boot/vmlinuz-4.4.0-142-generic root=...
initrd /initrd.img-4.4.0-142-generic
```

Ram-based root file system

initrd provides the capability to load a RAM disk by the boot loader. This RAM disk can then be mounted as the root file system and programs can be run from it. Afterwards, a new root file system can be mounted from a different device.

```
linux   /boot/vmlinuz-4.4.0-142-generic root=...
initrd /ramdisk.img.gz root=/dev/ram0
```

[6] Read only, no environment for illustration

NFS-based root file system

NFS based file system can be described to a boot loader using the following kernel command line. An example of GRUB config nfs as root:

```
$ cat /boot/grub/grub.cfg

linux /netboot/vmlinuz-linux add_efi_memmap root=/dev/nfs rootfstype=nfs
nfsroot=192.168.0.101:/srv/[CLIENT OS] nfsrootdebug rw ip=dhcp
initrd /netboot/initramfs-linux.img
```

For more details of the parameters and configuration in Linux kernel source code Documentation/admin-guide/nfs/nfsroot.rst.[7]

```
root=/dev/nfs

nfsroot=[<server-ip>:]<root-dir>[,<nfs-options>]

<nfs-options> Standard NFS options. All options are separated by commas.
port            = as given by server portmap daemon
rsize           = 4096
wsize           = 4096
timeo           = 7
retrans         = 3
acregmin        = 3
acregmax        = 60
acdirmin        = 30
acdirmax        = 60
flags           = hard, nointr, noposix, cto, ac

ip=<client-ip>:<server-ip>:<gw-ip>:<netmask>:<hostname>:<device>:<autoconf>:<dns0-
ip>:<dns1-ip>:<ntp0-ip>


<autoconf> Method to use for autoconfiguration.
off or none: don't use autoconfiguration
              (do static IP assignment instead)
on or any:   use any protocol available in the kernel
             (default)
dhcp:        use DHCP
bootp:       use BOOTP
rarp:        use RARP
both:        use both BOOTP and RARP but not DHCP
             (old option kept for backwards compatibility)

nfsrootdebug
This parameter enables debugging messages to appear in the kernel log at boot time
so that administrators can verify that the correct NFS mount options, server
address, and root path are passed to the NFS client.
```

[7] https://www.kernel.org/doc/Documentation/admin-guide/nfs/nfsroot.rst

```
rdinit=<executable file>
To specify which file contains the program that starts system initialization,
administrators can use this command line parameter. The default value of this
parameter is "/init".
```

3.2.4 Symlink and file path

The usage of symlink in kernel library version management

```
$ ls -l /lib
total 432
drwxr-xr-x  2 root root   4096 Aug 26 10:36 apparmor
lrwxrwxrwx  1 root root     21 Aug 29 06:57 cpp -> /etc/alternatives/cpp
drwxr-xr-x  3 root root   4096 Aug 26 10:32 crda
drwxr-xr-x 71 root root  20480 Aug 26 10:32 firmware
drwxr-xr-x  2 root root   4096 Aug 26 10:36 hdparm
```

To create a symlink

```
ln -s <path to the file/folder to be linked> <the path of the link to be created>

Ex
$ touch print60Hz.cfg
$ touch print120Hz.cfg
$ ln -s print120Hz.cfg print.cfg
$ ls -lltotal 0
-rw-rw-r-- 1 oslab oslab  0 Oct 22 22:00 print120Hz.cfg
-rw-rw-r-- 1 oslab oslab  0 Oct 22 22:00 print60Hz.cfg
lrwxrwxrwx 1 oslab oslab 14 Oct 22 22:00 print.cfg -> print120Hz.cfg
```

How to remove sysmlink

```
# Verify the existence of the link
ls -l <path-to-assumed-symlink>

unlink <path-to-symlink>
```

Meanwhile, a symlink is just another file or folder pointing to an original file or folder. To remove that relationship, you can remove the linked file.

```
$ rm <path-to-symlink>
```

For example:

```
$ ls -l print.cfg
lrwxrwxrwx 1 oslab oslab 14 Oct 22 22:00 print.cfg -> print120Hz.cfg
$ unlink print.cfg

$ ln -s print60Hz.c print.cfg

$ ls -l print.cfg
lrwxrwxrwx 1 oslab oslab 11 Oct 22 22:07 print.cfg -> print60Hz.c
$ rm print.cfg
```

How to delete and find the lost symlink

```
$ ln -s print60Hz.cfg print.cfg
$ ls -l
total 0
-rw-rw-r-- 1 oslab oslab  0 Oct 22 22:00 print120Hz.cfg
-rw-rw-r-- 1 oslab oslab  0 Oct 22 22:00 print60Hz.cfg
lrwxrwxrwx 1 oslab oslab 13 Oct 22 22:15 print.cfg -> print60Hz.cfg
$ rm print60Hz.cfg

$ ls -l
total 0
-rw-rw-r-- 1 oslab oslab  0 Oct 22 22:00 print120Hz.cfg
lrwxrwxrwx 1 oslab oslab 13 Oct 22 22:15 print.cfg -> print60Hz.cfg

$ sudo find . -xtype l -delete

$ ls -l
total 0
-rw-rw-r-- 1 oslab oslab  0 Oct 22 22:00 print120Hz.cf
```

In the end of the day, we can traverse filesystem in a tree-like structure using the special folder "." and "..".

```
$ cd /
$ pwd ..
/
```

The root directory has a '.' and a '..' entry in it, and the inode number for each is the same. In the 1980s, there was a system called Newcastle Connection that treated networked computers as being above the root of your local computer (ref the File Explorer in Windows). Thus, on such a machine, you would type:

```
cd /../othermachine/path/to/interesting/place
```

This behavious is described in POSIX standard for chroot:

The dot-dot entry in the root directory is interpreted to mean the root directory itself. Thus, dot-dot cannot be used to access files outside the subtree rooted at the root directory.[8]

3.2.5 Mount

The mount command serves to attach the filesystem found on some device to the big file tree.

```
sudo mount -l
/dev/sda1 on / type ext4 (rw,errors=remount-ro)
proc on /proc type proc (rw,noexec,nosuid,nodev)
sysfs on /sys type sysfs (rw,noexec,nosuid,nodev)
none on /sys/fs/cgroup type tmpfs (rw)
none on /sys/fs/fuse/connections type fusectl (rw)
none on /sys/kernel/debug type debugfs (rw)
none on /sys/kernel/security type securityfs (rw)
udev on /dev type devtmpfs (rw,mode=0755)
```

[8] https://pubs.opengroup.org/onlinepubs/7908799/xsh/chroot.html

```
devpts on /dev/pts type devpts (rw,noexec,nosuid,gid=5,mode=0620)
tmpfs on /run type tmpfs (rw,noexec,nosuid,size=10%,mode=0755)
none on /run/lock type tmpfs (rw,noexec,nosuid,nodev,size=5242880)
none on /run/shm type tmpfs (rw,nosuid,nodev)
none on /run/user type tmpfs (rw,noexec,nosuid,nodev,size=104857600,mode=0755)
none on /sys/fs/pstore type pstore (rw)
tracefs on /var/lib/ureadahead/debugfs/tracing type tracefs (rw,relatime)
systemd on /sys/fs/cgroup/systemd type cgroup
(rw,noexec,nosuid,nodev,none,name=systemd)
```

The configuration of file system through mount is done in the system initialization phase using the fstab configuration.

```
$ cat /etc/fstab
# /etc/fstab: static file system information.
#
# Use 'blkid' to print the universally unique identifier for a
# device; this may be used with UUID= as a more robust way to name devices
# that works even if disks are added and removed. See fstab(5).
#
# <file system> <mount point>   <type>  <options>       <dump>  <pass>
# / was on /dev/sda1 during installation
UUID=6abb8ed8-842e-4b4b-81c4-9b0b5291d54b /               ext4    errors=remount
-ro 0       1
# swap was on /dev/sda5 during installation
UUID=5d8561df-d5a7-48c1-9327-fd23bfb2cfa3 none            swap    sw
0       0
/dev/fd0        /media/floppy0  auto    rw,user,noauto,exec,utf8 0       0
```

fstab structure
First column <file system> file path point to the device file.
Second column <mount point> file path to the mount point
Third column <type> filesystem type, common types are ext2/3/4, swap, vfat or ntfs, nfs, auto
Fourth columm <options> option using in mounting process, common options are auto, noauto (mount manually), nouser (only administrator can work, exec, ro (read only), rw (read-write), sync (I/O manipulation is synchronized), async (not sync) or default (an equivalent of rw, suid, dev, exec, auto, nouser, async)
Fifth column <dump> an option for automatic saving feature of file system. Use 0 to discard autosave, 1 for autosave
Sixth column <pass> an option for fschk, use 0 for ignore the checking and 1 for enable the checking.

An example for nfs mounting, most trickly usage:

```
10.42.0.1:/netroot /netroot nfs rw,noatime,nolock,noauto 1 1
```

3.2.6 Further boot filesystem

In combination of bootfs and mounting, Linux supports a special block device called the loop device, which maps a normal file onto a virtual block device. This allows for the file to be used as a "virtual file system"

inside another file. With Linux it's possible to create a file-system inside a single file. These storage devices are available as device files such as

Create a file:

```
$ dd if=/dev/zero of=loopbackfile.img bs=1M count=10
10+0 records in
10+0 records out
10485760 bytes (10 MB) copied, 0.0217895 s, 481 MB/s

$ du -sh loopbackfile.img
10M     loopbackfile.img
```

Create loop device

```
$ sudo losetup -find ./loopbackfile.img
/dev/loop0: [0801]:56996 (/home/oslab/page/loopbackfile.img)
```

Create filesystem:

```
$ mkfs.ext4 /home/oslab/page/loopbackfile.img
mke2fs 1.42.9 (4-Feb-2014)
/home/oslab/page/loopbackfile.img is not a block special device.
Proceed anyway? (y,n) y
Discarding device blocks: done
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
Stride=0 blocks, Stripe width=0 blocks
2560 inodes, 10240 blocks
512 blocks (5.00%) reserved for the super user
First data block=1
Maximum filesystem blocks=10485760
2 block groups
8192 blocks per group, 8192 fragments per group
1280 inodes per group
Superblock backups stored on blocks:
        8193

Allocating group tables: done
Writing inode tables: done
Creating journal (1024 blocks): done
Writing superblocks and filesystem accounting information: done
```

Mount and use the filesystem

```
$ sudo mkdir /loopfs

$ df -hP /loopfs/
Filesystem      Size  Used Avail Use% Mounted on
```

```
/dev/loop1       8.7M   92K  7.9M   2% /loopfs


$ mount -o loop /dev/loop0 /loopfs/

$  mount | grep loopfs
/dev/loop0 on /loopfs type ext4 (rw)
```

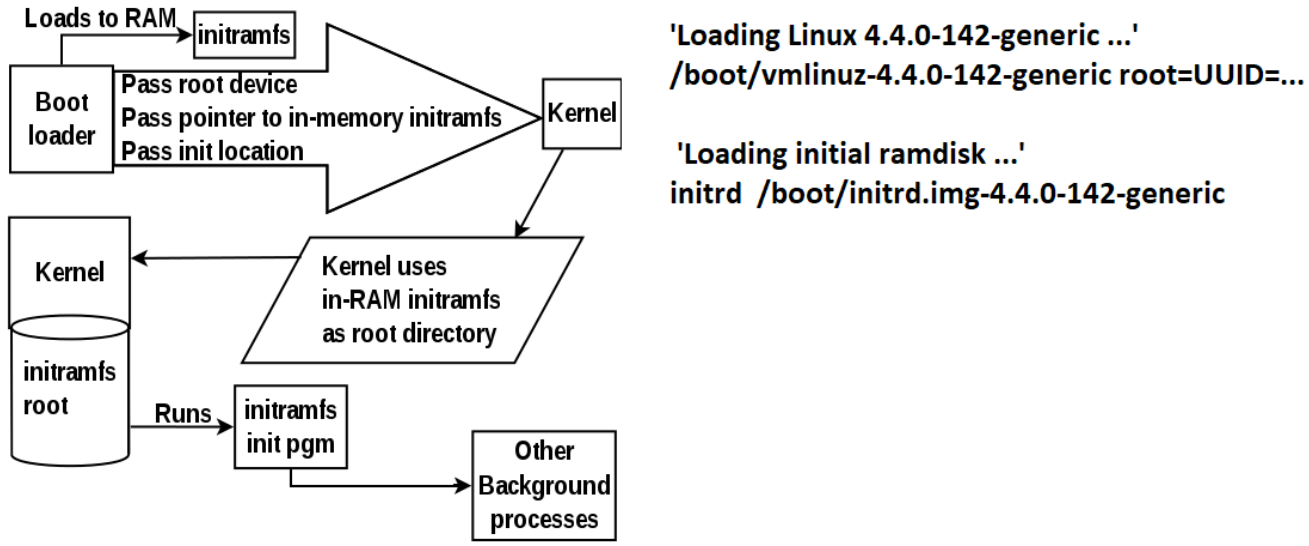


Figure 1 An illustration of Linux booting process

3.2.7 Debugfs in device driver probling and initializing during boot process

The debugfs program is an interactive file system debugger. It can be used to examine and change the state of an ext2 file system. Debugfs is very similar to other virtual filesystems, such as sysfs and procfs, but it has no functional purpose except to provide debugging information. The files in this filesystem are generated by the kernel, and the contents will vary depending on the driver/subsystem. When writing new drivers, debugfs is preferred to printk statements, since it is much easier to enable/disable, and provides a much more standardized interface.

Prerequisites

Kernel build with the following configuration options:
CONFIG_DEBUG_FS=y

Mounting debugfs

```
$ mkdir /sys/kernel/debug
$ mount -t debugfs none /sys/kernel/debug


$ mkdir /home/oslab/debug
$ sudo mount -t debugfs none /home/oslab/debug
```

```
$ sudo cat /home/oslab/debug/usb/devices
T:  Bus=02 Lev=00 Prnt=00 Port=00 Cnt=00 Dev#=  1 Spd=12   MxCh= 2
B:  Alloc=  0/900 us ( 0%), #Int=  1, #Iso=  0
D:  Ver= 1.10 Cls=09(hub  ) Sub=00 Prot=00 MxPS=64 #Cfgs=  1
P:  Vendor=1d6b ProdID=0001 Rev= 4.04
S:  Manufacturer=Linux 4.4.0-142-generic uhci_hcd
...
```

Adding debugfs support to a driver

A developer wishing to use debugfs starts by creating a directory within the filesystem:

```
struct dentry *debugfs_create_dir(const char *name,
                                  struct dentry *parent);
struct dentry *debugfs_create_file(const char *name, mode_t mode,
                                   struct dentry *parent, void *data,
                                   struct file_operations *fops);
void debugfs_remove(struct dentry *dentry);
```

To implement additional helpers to make export a single value as easy as possible

```
struct dentry *debugfs_create_u8(const char *name, mode_t mode,
                                 struct dentry *parent, u8 *value);
struct dentry *debugfs_create_u16(const char *name, mode_t mode,
                                  struct dentry *parent, u16 *value);
struct dentry *debugfs_create_u32(const char *name, mode_t mode,
                                  struct dentry *parent, u32 *value);
struct dentry *debugfs_create_bool(const char *name, mode_t mode,
                                   struct dentry *parent, u32 *value);
```

3.2.8 Debugfs in File System analysis

Prepare file on loopfs we created in the previous section at /loopfs

```
$ cd /loopfs/

$ sudo touch first.rst.lst

$ sudo touch first.1.1.rst.lst
$ sudo ln -s first.1.1.rst.lst first.lst
$ ls
first.1.1.rst.lst  first.lst  first.rst.lst  lost+found
```

Debugfs command

```
$ sudo debugfs /dev/loop1
debugfs 1.42.9 (4-Feb-2014)
debugfs:  ls
  (12) .    2  (12) ..    11  (20) lost+found    12  (24) first.rst.lst
 13  (28) first.1.1.rst.lst    14  (928) first.lst

debugfs:  ls -l

      2   40755 (2)      0      0    1024 22-Oct-2022 23:09 .
```

```
      2   40755 (2)      0      0    1024 22-Oct-2022 23:09 ..
     11   40700 (2)      0      0   12288 22-Oct-2022 22:29 lost+found
     12  100644 (1)      0      0       0 22-Oct-2022 23:09 first.rst.lst
     13  100644 (1)      0      0       0 22-Oct-2022 23:09 first.1.1.rst.lst
     14  120777 (7)      0      0      17 22-Oct-2022 23:09 first.lst
```

Traverse inode (ref Background 2.3)

```
debugfs:  show_inode_info first.lst
Inode: 14   Type: symlink    Mode:  0777   Flags: 0x0
Generation: 1812141506    Version: 0x00000001
User:     0   Group:     0   Size: 17
File ACL: 0    Directory ACL: 0
Links: 1   Blockcount: 0
Fragment:  Address: 0    Number: 0    Size: 0
ctime: 0x6354daaf -- Sat Oct 22 23:09:51 2022
atime: 0x6354dab5 -- Sat Oct 22 23:09:57 2022
mtime: 0x6354daaf -- Sat Oct 22 23:09:51 2022
Fast_link_dest: first.1.1.rst.lst
```

Retrieve file information

```
debugfs:  show_inode_info .

Inode: 2   Type: directory    Mode:  0755   Flags: 0x80000
Generation: 0    Version: 0x00000003
User:     0   Group:     0   Size: 1024
File ACL: 0    Directory ACL: 0
Links: 3   Blockcount: 2
Fragment:  Address: 0    Number: 0    Size: 0
ctime: 0x6354daaf -- Sat Oct 22 23:09:51 2022
atime: 0x6354dab5 -- Sat Oct 22 23:09:57 2022
mtime: 0x6354daaf -- Sat Oct 22 23:09:51 2022
EXTENTS:
(0):44

debugfs:  dump <2> file-debugfs

$ sudo hexdump -C /file-debugfs
00000000  02 00 00 00 0c 00 01 02  2e 00 00 00 02 00 00 00  |................|
00000010  0c 00 02 02 2e 2e 00 00  0b 00 00 00 14 00 0a 02  |................|
00000020  6c 6f 73 74 2b 66 6f 75  6e 64 00 00 0c 00 00 00  |lost+found......|
00000030  18 00 0d 01 66 69 72 73  74 2e 72 73 74 2e 6c 73  |....first.rst.ls|
00000040  74 00 00 00 0d 00 00 00  1c 00 11 01 66 69 72 73  |t...........firs|
00000050  74 2e 31 2e 31 2e 72 73  74 2e 6c 73 74 00 00 00  |t.1.1.rst.lst...|
00000060  0e 00 00 00 a0 03 09 07  66 69 72 73 74 2e 6c 73  |........first.ls|
00000070  74 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |t...............|
00000080  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```

3.3 Further practices with Virtual FS

The Devpts Filesystem

When devices like the keyboard and mouse are directly connected to the computer through serial ports, the connection is called TTY.

Dev/pts is a pseudoterminal stimulated by programs, i.e. ssh, telnet. Dev/pts files store information related to the connected devices through Linux in special directories.

```
# ls /dev/pts/
0  1  2  3  7  ptmx

# Print the file name of the terminal connected to standard input.
$ tty
/dev/pts/3

$ echo 3 > /dev/pts/3
3
```

Every unique terminal window is related to a Linux pts entry in the /dev/pts system. A typical Linux kernel-based operating system provides many PTYs to support text-based interfaces as provided by terminal emulators (such as xterm or gnome-terminal) and remote access interfaces like SSH.

To list all the remote terminal connection

```
$ w
 00:39:59 up 3 days, 15:09,  4 users,  load average: 0.00, 0.00, 0.00
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
oslab    pts/0    192.168.232.1    23:08   31:59   0.48s  0.04s sshd: oslab [pr
oslab    pts/1    192.168.232.1    21:36    1:05m  2.38s  0.04s sshd: oslab [pr
oslab    pts/3    192.168.232.1    23:27    0.00s  0.29s  0.03s sshd: oslab [pr
oslab    tty1                      Sat07    4:31   0.41s  0.40s -bash
```

You can also use command mount to verify whether the dev/pts are mounted in Linux

```
$ mount -l  | grep pts
devpts on /dev/pts type devpts (rw,noexec,nosuid,gid=5,mode=0620)
```

In general PTS is the Pseudo TTY is a pair of pseudo-devices – a slave and a master – that provide a special sort of communication channel.

- The slave pseudo-device emulates a physical computer text terminal
- The master pseudo-device provides the means by which a program providing a text-based user interface acts with and controls its slave.

A pseudoterminal pair is similar to a bidirectional pipe. Anything that is written on the master appears as input on the slave, and anything that is written on the slave appears as input on the master.

The Proc Filesystem

Kernel data

File	Content
buddyinfo	Kernel memory allocator information (see text) (2.5)
bus	Directory containing bus specific information
cmdline	Kernel command line
cpuinfo	Info about the CPU

devices	Available devices (block and character)
dma	Used DMS channels
filesystems	Supported filesystems
driver	Various drivers grouped here, currently rtc (2.4)
devices	Available devices (block and character)
fb	Frame Buffer devices (2.4)
fs	File system parameters, currently nfs/exports (2.4)
ide	Directory containing info about the IDE subsystem
interrupts	Interrupt usage
iomem	Memory map (2.4)
ioports	I/O port usage
irq	Masks for irq to cpu affinity (2.4)(smp?)
kcore	Kernel core image (can be ELF or A.OUT(deprecated in 2.4))
kmsg	Kernel messages
kallsyms	Kernel symbol table
locks	Kernel locks
meminfo	Memory info
misc	Miscellaneous
modules	List of loaded modules
mounts	Mounted filesystems
net	Networking info (see text)
pci	Deprecated info of PCI bus (new way -> /proc/bus/pci/, decoupled by lspci (2.4)
rtc	Real time clock
scsi	SCSI info (see text)
slabinfo	Slab pool info
softirqs	softirq usage
stat	Overall statistics
swaps	Swap space utilization
sys	System config items /proc/sys. The more details are in the next section of Modification.
uptime	Wall clock since boot, combined idle time of all cpus
version	Kernel version
vmallocinfo	Show vmalloced areas

An example of highlighted data:

```
$ sudo cat /proc/devices
Character devices:
  1 mem
  4 /dev/vc/0

$ sudo cat /proc/filesystems
nodev   sysfs
nodev   rootfs
nodev   ramfs
nodev   bdev
nodev   proc
...

$ sudo file /proc/kcore
```

```
/proc/kcore: ELF 64-bit LSB  core file x86-64, version 1 (SYSV), SVR4-style, from
'BOOT_IMAGE=/boot/vmlinuz-4.4.0-142-generic root=UUID=6abb8ed8-842e-4b4b-81c4-9b'

$ sudo cat /proc/kallsyms
0000000000000000 A irq_stack_union
0000000000000000 A __per_cpu_start
0000000000000000 A __per_cpu_user_mapped_start
0000000000004000 A vector_irq
...
```

Some example realated to device driver

```
$ cat /proc/modules  | grep i2c
i2c_piix4 24576 0 - Live 0xffffffffc0148000

$ cat /proc/net/sockstat
sockets: used 465
TCP: inuse 4 orphan 0 tw 0 alloc 5 mem 1
UDP: inuse 2 mem 1
UDPLITE: inuse 0
RAW: inuse 0
FRAG: inuse 0 memory 0
```

Memory related as in the previous lab.

```
$ cat /proc/vmallocinfo
...
0xffffc900003d6000-0xffffc900003d8000   8192 bpf_prog+0x36/0xa0 pages=1 vmalloc N0=1
0xffffc900003d8000-0xffffc900003da000   8192 bpf_prog +0x36/0xa0 pages=1 vmalloc N0=1
0xffffc900003da000-0xffffc900003de000  16384 n_tty +0x19/0xb0 pages=3 vmalloc N0=3
0xffffc900003ea000-0xffffc900003ed000  12288 tt _init+0x69/0xa0 [ttm] pages=2 vmalloc N0=2
...

 cat /proc/interrupts    | grep TLB
 TLB:      18179      21265      21428   TLB shootdowns
```

TLB shootdown the event one processor to ask other processor flush its cached content a page.

Modify system parameter:

Modifying kernel parameters by writing into files found in /proc/sys:

To change a value, simply echo the new value into the file. You need to be root to do this.
The files in /proc/sys can be used to fine tune and monitor miscellaneous and general things in the operation of the Linux kernel
Reading and modify the variable

```
$ sysctl <tunable class>.<tunable>

$ sysctl <tunable class>.<tunable>=<value>

$ sysctl -w <tunable class>.<tunable>=<value> >> /etc/sysctl.conf
```

Exploring the files which modify certain parameters

Kernel configuration

```
$ sysctl kernel.core_pattern
kernel.core_pattern = |/usr/share/apport/apport -p%p -s%s -c%c -d%d -P%P -u%u -g%g
-- %E

$ sudo sysctl -w kernel.core_pattern=core

$ sysctl kernel.core_pattern
kernel.core_pattern = core
```

Filesystem

```
$ sysctl fs
...
fs.inode-nr = 47366     141
fs.inode-state = 47366  141     0       0       0 0    0
...
```

Read and modify network configuration

```
$ sudo sysctl net | grep ip_forward
net.ipv4.ip_forward = 0
net.ipv4.ip_forward_use_pmtu = 0

$ sudo sysctl net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1

$ sudo sysctl -w net.ipv4.ip_forward=1 >> /etc/sysctl.conf

$ tail /etc/sysctl.conf
# Log Martian Packets
#net.ipv4.conf.all.log_martians = 1
#
net.ipv4.ip_forward = 0
```

Review of the /proc/sys file tree

To display all of the configuration value

```
$ sysctl -a
```

The Sys Filesystem

The Sysfs is target device tree with the hierachical relationship.

An example of the relation ship

```
$ ll /sys/class/tty/*/device/driver
lrwxrwxrwx 1 root root 0 Oct 23 01:00 /sys/class/tty/ttyS0/device/driver ->
../../../bus/pnp/drivers/serial/
lrwxrwxrwx 1 root root 0 Oct 23 01:00 /sys/class/tty/ttyS10/device/driver ->
../../../bus/platform/drivers/serial8250/
lrwxrwxrwx 1 root root 0 Oct 23 01:00 /sys/class/tty/ttyS11/device/driver ->
../../../bus/platform/drivers/serial8250/
```

This show the relationship between the specifice device tty driver and the all drivers installed in the system at /sys/bus/platform/drivers/serial8250

All of the bus in the system can be showed in the folder structure

```
$ ls /sys/bus/
...
drwxr-xr-x  5 root root 0 Oct 19 06:48 pci/
drwxr-xr-x  4 root root 0 Oct 19 06:48 pci_express/
...
drwxr-xr-x  4 root root 0 Oct 19 06:48 usb/
...
```

For the nested structure, we are interested in these following bus (usb, pci)

```
$ cat /sys/bus/usb/devices/*/product
VMware Virtual USB Mouse
Virtual Bluetooth Adapter
VMware Virtual USB Hub
EHCI Host Controller
UHCI Host Controller
```

```
$ cat /sys/bus/pci/devices/*/modalias
pci:v00008086d00007191sv00000000sd00000000bc06sc04i00
pci:v00008086d00007110sv000015ADsd00001976bc06sc01i00
pci:v00008086d00007111sv000015ADsd00001976bc01sc01i8A

$ ls -ld /sys/bus/pci_express/devices/*/subsystem/
/sys/bus/pci_express/devices/0000:00:15.0:pcie01/subsystem/
/sys/bus/pci_express/devices/0000:00:15.0:pcie04/subsystem/
/sys/bus/pci_express/devices/0000:00:15.1:pcie01/subsystem/
/sys/bus/pci_express/devices/0000:00:15.1:pcie04/subsystem/
```

PCI-sysfc

```
$ ls -ld /sys/devices/pci*/*/
drwxr-xr-x  3 root root 0 Oct 23 02:45 /sys/devices/pci0000:00/0000:00:00.0/
drwxr-xr-x  4 root root 0 Oct 23 02:45 /sys/devices/pci0000:00/0000:00:01.0/
drwxr-xr-x  5 root root 0 Oct 23 02:45 /sys/devices/pci0000:00/0000:00:07.0/
...
drwxr-xr-x 82 root root 0 Oct 23 02:45 /sys/devices/pci0000:00/firmware_node/
drwxr-xr-x  3 root root 0 Oct 23 02:45 /sys/devices/pci0000:00/pci_bus/
drwxr-xr-x  2 root root 0 Oct 23 02:45 /sys/devices/pci0000:00/power/
```

The interface:

file	function
class	PCI class (ascii, ro)
config	PCI config space (binary, rw)
device	PCI device (ascii, ro)
…	…
remove	remove device from kernel's list (ascii, wo)
revision	PCI revision (ascii, ro)

rom	PCI ROM resource, if present (binary, ro)
subsystem_device	PCI subsystem device (ascii, ro)
subsystem_vendor	PCI subsystem vendor (ascii, ro)
vendor	PCI vendor (ascii, ro)

```
# sudo echo 1 >  /sys/devices/pci0000:00/0000:00:16.3/remove

$ sudo ls  /sys/devices/pci0000:00/0000:00:16.3/
ls: cannot access /sys/devices/pci0000:00/0000:00:16.3/: No such file or directory
```

To list all vendor and device info of PCI devices

```
$ sudo cat /sys/bus/pci/devices/*/vendor
0x8086
…
0x15ad
…
0x1000
…
0x1274

$ sudo cat /sys/bus/pci/devices/*/device
0x7190
0x7191
0x7110
0x7113
0x0740
0x0405
0x0030
0x0790
0x07a0
0x0774
0x100f

$ cat /sys/bus/pci/devices/*/class
0x060000
0x060400
0x060100
0x01018a
0x068000
0x088000
0x030000
```

The class code can be identified as in the document.[9]

[9] https://pcisig.com/sites/default/files/files/PCI_Code-ID_r_1_11__v24_Jan_2019.pdf

4 Exercise

PROBLEM 1

Retrieve and extract the amount of free memory using the file entry /proc/meminfo

PROBLEM 2

Assume you are equipped with a hardware specific device through GPIO, design a software stack to manipulate (retrieve status and control output bit) using the filesystem interface. Student are recommended to provide a GUI i.e. web GUI, a piece of software deamon etc.

PROBLEM 3

Write a script to manipulate plug and unplug USB using Linux hotplug mechanism