1、**.Linux中“一切皆文件”的含义是什么？它的优势和不足在哪些方面有所表现。**

Linux系统中，文件具体可分为以下几种类型：

**1) 普通文件**

类似 mp4、pdf、html 这样，可直接拿来使用的文件都属于普通文件，Linux 用户根据访问权限的不同可以对这些文件进行查看、删除以及更改操作。

**2) 目录文件**

对于用惯 Windows 系统的用户来说，目录是文件可能不太好理解。  
  
Linux 系统中，目录文件包含了此目录中各个文件的文件名以及指向这些文件的指针，打开目录等同于打开目录文件，只要你有权限，可以随意访问目录中的任何文件。

注意，目录文件的访问权限，同普通文件的执行权限，是一个意思。

**3) 字符设备文件和块设备文件**

这些文件通常隐藏在 /dev/ 目录下，当进行设备读取或外设交互时才会被使用。  
  
例如，磁盘光驱属于块设备文件，串口设备则属于字符设备文件。

Linux 系统中的所有设备，要么是块设备文件，要么是字符设备文件。

**4) 套接字文件（socket）**

套接字文件一般隐藏在 /var/run/ 目录下，用于进程间的网络通信。

**5) 符号链接文件（symbolic link）**

类似与 Windows 中的快捷方式，是指向另一文件的简介指针（也就是软链接）。

**6) 管道文件（pipe）**

主要用于进程间通信。例如，使用 mkfifo 命令创建一个 FIFO 文件，与此同时，启用进程 A 从 FIFO文件读数据，启用进程 B 从 FIFO文件中写数据，随写随读。

**“一切皆文件”的利弊**

和 Windows 系统不同，Linux 系统没有 C 盘、D 盘、E 盘那么多的盘符，只有一个根目录（/），所有的文件（资源）都存储在以根目录（/）为树根的树形目录结构中。  
  
这样做最明显的好处是，开发者仅需要使用一套 API 和开发工具即可调取 Linux 系统中绝大部分的资源。举个简单的例子，Linux 中几乎所有读（读文件，读系统状态，读 socket，读 PIPE）的操作都可以用 read 函数来进行；几乎所有更改（更改文件，更改系统参数，写 socket，写 PIPE）的操作都可以用 write 函数来进行。  
  
不利之处在于，使用任何硬件设备都必须与根目录下某一目录执行挂载操作，否则无法使用。

2、**Linux系统中，构成文件的数据结构有哪些？各自的作用是什么？它们之间存在哪些逻辑上的联系？**

\* 超级块结构(struct super\_block {...} )  
该结构保存了一个被安装在[linux系统](https://www.baidu.com/s?wd=linux%E7%B3%BB%E7%BB%9F&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "_blank)上的文件系统的信息。对于基于磁盘的文件系统，该结构一般和保存在磁盘上的"文件系统控制块"对应。也就是说如果是磁盘文件系统，该结构保存的磁盘文件系统的控制信息。  
\* inode结构( struct inode {...} )  
该结构中存储的是一个特定文件的一般信息，对于一个基于磁盘的文件系统，该结构对应磁盘上的"文件数据控制块"。每一个inode结构都对应一个inode节点号，这个节点号是唯一的，它也唯一标识一个文件系统中的文件。  
\* file结构( struct file {...} )  
该结构中存储的是一个打开的文件和打开这个文件的进程间的交互信息。该结构保存在kernel的内存区，在打开文件时被创建，关闭文件时被释放。  
\* dentry结构( struct dentry {...} )  
该结构存储的是目录实体和对应的文件的关联信息。

**3.在Linux内核实现内核的过程中，如何使用C语言实现面向对象的基本特点的，请用代码举例并附加文字说明的形式解释。**

1）. Linux内核C语言模拟JAVA的单继承机制，不支持多重继承，遇到同时具备cdev对象与platform\_driver对象两种类型的驱动时，只继承其中一个对象，另一个则作为接口实现，以描述LIKE-A的概念。

2）. 对于类似s3c-fb.c这类驱动，我们认为它是framebuffer 与 cdev的子类，实现struct  platform\_driver这个虚拟总线实例化接口，因为s3c-fb驱动核心的功能特性是framebuffer的显示缓存功能逻辑，而struct  platform\_driver这个接口的相关函数，只是在动态实例化framebuffer设备节点的时候调用一次，并非framebuffer本质的特征(就像机器狗，本质特性是机器的特性)。所以我们说s3c-fb IS-A  framebuffer, s3c-fb LIKE-A struct  platform\_driver。

3). 由于驱动设备的复杂性，并不像自然界的事物容易看出继承关系。所以在研究内核设备驱动单继承关系的时候，不要拘泥于条条框框，要根据自己的研究目的来选择基类与继承关系。例如图6所示的，如果要研究狗的哺乳动物属性，在单继承条件下，我们可以认为狗继承了哺乳动物，实现了食肉动物的接口。

4). 在研究类似s3c-fb.c这类驱动时，如果关注重点在s3c-fb的显示缓冲等framebuffer特性上，我们就认为s3c-fb继承了framebuffer，实现了struct  platform\_driver的虚拟总线实例化接口。如果我们关注点在s3c-fb如何识别fb设备动态识别实例化，如果通过sys文件系统进行电影管理的特性，那么我们可以认为s3c-fb继承了platform\_driver类，实现了framebuffer的接口(尽管这种case比较少见)。

5). 总之，在研究Linux内核设备驱动单继承与接口实现规则时，要主动权衡研究目的，根据需要选择继承的基类与实现的接口。但是字符设备驱动在大多数情况下，我们都认为XX驱动继承了字符设备cdev，实现了platform\_driver的虚拟总线实例化接口。

在Linux内核设备驱动中，我们在用户态open一个字符设备，然后调用字符设备的read/write/ioctl函数，最终也会调用到内核态设备驱动程序相应的read/write/ioctl函数的实现，从而模拟了通过基类与多态函数的特性来访问子类的目的。

6).实际上，Linux内核会维护基类与子类cdev对象实例的链表，当用户态发起read/write/ioctl等字符设备系统调用函数时，read/write/ioctl等字符设备系统调用函数会通过/dev/\*下的字符设备，设备节点号的方式(主节点号major，次节点号minor)从cdev链表的子类中，找到对应子类的cdev对象实例，然后判断是否为空并调用子类cdev->ops->read/write/ioctl等实际子类的多态函数实现，从而最终实现了通过访问基类的多态函数，最终访问到子类实际的多态函数，这个面向对象的特性。

7).在Linux内核中，模拟多态的方法要简单一些，实际上是在基类的函数方法中，通过获取子类对象，再嵌套调用子类对象的同名函数来实现的。图8为framebuffer设备驱动的多态函数read实现。实际上，framebuffer的cdev的struct  file\_operations对象的read函数会调用子类struct  fb\_info对象中同名fb\_read函数(如果子类未实现该函数，则不调用)，从而模拟了继承关系中基类同名函数通过多态的方法调用子类同名函数的行为。