2018年10月19日

首先接着NILFS2的1部分。开始真正的源码阅读。

会涉及到每个使用到的linux代码或者glibc的c相关源代码，遇到的都分析下。

首先看mkfs.c代码，因为一个命令，就是一个main函数开头，因此，看mkfs.nilfs2就主要看这个文件，而如果看lscp命令，就专门看lscp.c文件。当然这些命令相关文件，还都是在bin文件夹下面。

先从mkfs.c的main函数开始。

1、命令为mkfs.nilfs  devname

因此第一步一定是解析设备的路径名字。

解析步骤为：作者封装的parse\_options(argc, argv);   device = argv[optind];

而陌生的代码为：optind这个变量，是根据c标准库来的？？

其中cl说的对，由于glibc太复杂，下载musllibc这种轻量级别的。其实#include<unistd.h>这个可以从内核的源码中看到unistd.h因为这个头文件不是说是c标准的，而是unix版本的，所以windows里面就没有。而在linux环境下，就多考虑glibc这个标准，其实c标准库于glibc库两者是两个库，glibc依照c标准库为模板，而自己又增加了好多很方便的库，例如此处显示的getopt（）函数，就是gnu的标准glibc库内容，并不是c的标准库。因此调用的时候，要包含unistd.h这个gnu相关的头文件。而为什么能够直接用getopt函数里面的optind 这种变量？？假如对getopt函数不熟悉，还已下载不知道如上，argv[optind]中的optind的来源的。。。

为什么能直接使用？？两个点缺一不可，就是1、gitopt里面的extern int optind；定义。但是仅仅包含头文件是根本没用的，必须实际包含头文件的时候，optind就已经有个地方存在了。那么就是必不可少的第二步，2、gitopt.c这个被生成的链接库。本来链接库一直存在，然后当包含了gitopt.h头文件，那么就是明确指明，使用gitopt.c中已经定义的optind那么变量了。所以使用成功。

而glibc与系统调用函数，与c标准库之间的关系。可以这么说整个glibc才是对编程者的接口，而glibc中有一部分就是系统调用函数。**所以可以说glibc是按照c的标准规定，，实现了与linux密切相关的库，而glibc里面包含了系统调用，当然也有不是系统调用的函数。所以glibc中的read就真的是系统调用的read。而与c标准中的fread也很类似。但是glibc当然支持c标准库，所以fread这个标准函数同样能够实现读操作的完成**。

既然glibc是包含的c标准规定以及自己扩展的与系统调用相关的api也有。那么怎么区分呢？

<http://blog.chinaunix.net/uid-30128330-id-4992628.html>这个链接进行了说明，怎么说了，本来这两部分都有很大的重合部分，看编程人员站的立场，假如想使用系统调用，那么考虑更多系统调用，假如一个编程人员只站在c标准时候，对文件操作等，那就多调用标准库相关的函数。最终，功能都是一样的可以实现。

ubuntu中查看glibc的版本，使用ldd --version即可。

而getopt()函数对于main传入的argc argv的使用方法，看musllibc中的gitopt（）函数源码也可以。源码中显示传入了三个参数，最后一个是const char \* optstring.

而这最后一个optstring就是为了识别linux中常用的短参数命令。-c -cjvf这种，等等。而规定假如-c后面不需要加值，那么就在optstring中仅仅显示一个c，假如-a后面要加值，那么就在optstring里面是a：即多一个冒号显示，而假如是可选值的参数d，那么在optstring里面是两个冒号d::

getopt()使用optstring所指的字串作为短参数列表，象"1ac:d::"就是一个短参数列表。短参数的定义是一个'-'后面跟一个字母或数字，象-a, -b就是一个短参数。每个数字或字母定义一个参数。

其中短参数在getopt定义里分为三种：

1. 不带值的参数，它的定义即是参数本身。

2. 必须带值的参数，它的定义是在参数本身后面再加一个冒号。

3. 可选值的参数，它的定义是在参数本身后面加两个冒号 。

在这里拿上面的"1ac:d::"作为样例进行说明，其中的1,a就是不带值的参数，c是必须带值的参数，d是可选值的参数。

在实际调用中，'-1 -a -c cvalue -d', '-1 -a -c cvalue -ddvalue', '-1a -ddvalue -c cvalue'都是合法的。这里需要注意三点：

1. 不带值的参数可以连写，象1和a是不带值的参数，它们可以-1 -a分开写，也可以-1a或-a1连写。

2. 参数不分先后顺序，'-1a -c cvalue -ddvalue'和'-d -c cvalue -a1'的解析结果是一样的。

3. 要注意可选值的参数的值与参数之间不能有空格，必须写成-ddvalue这样的格式，如果写成-d dvalue这样的格式就会解析错误。

注意，这个可选值的竟然需要值与参数之间没有空格也是没有想到。而且不带值的参数可以连着写。

而四个内部变量，分别用于，optarg是当短参数真正有值时候，用于自动被赋予这个值的   字符串内容的。因此可以在switch中直接**操作optarg变量**。**所以这个optarg是使用最多的**。

optind是agrv当前的索引值。其实假如test  ./a.out -t 123 -n

上述其实根本不显示optstring.optstring是编程人员自己考虑自己输入的所有参数类型而已。上述只是标识argv有4个值了。目前。

当发现无效选项字符时候，optopt会包含那个无效选项字符，让编程者使用。

<https://blog.csdn.net/Mr_JJ_Lian/article/details/6835137?utm_source=blogxgwz0>

这个链接讲的也很好。

device = argv[optind];  这个意思就是截取到真实的device内容。

2、首先自己打印%s，在长时间不用c的情况下竟然不认为%s就是指针头打印，还去使用了\*device来打印%s.这明显不对，自己打印了device这个%s,确实是自己设置的disk.img设备名字。

3、不知不觉又多出来一个stat结构体以及stat函数。

if (stat(device, &statbuf) != 0)  
        perr("Error: cannot find %s: %s", device, strerror(errno));

首先如果device名字没有取到，那么，就用strerror这个标准c库来将errno信息报出来。注意这个函数返回的就是char\*字符串，但是为什么不直接用errno？？？还要加一个streeror??因为errno是一个number而其真实信息是在errno.h中定义的。所以想打印出字符串就用strerror。

而stat(device , &statbuf)这个函数是用于，将**字符串内容 当做的是文件名字**！！！！！，device给statbuf这个stat结构体！！赋值。不仅仅是字符串了，所以给结构体赋值成功了，那么就返回0，失败返回-1.所以上述，失败的话是！=0.

而这个device是真的设备，或者说设备也是文件，就是文件特征，所有的特征都放置在stat结构体中。

此时stat，是

struct stat {  
    dev\_t         st\_dev;       //文件的设备编号  
    ino\_t         st\_ino;       //节点  
    mode\_t        st\_mode;      //文件的类型和存取的权限  
    nlink\_t       st\_nlink;     //连到该文件的硬连接数目，刚建立的文件值为1  
    uid\_t         st\_uid;       //用户ID  
    gid\_t         st\_gid;       //组ID  
    dev\_t         st\_rdev;      //(设备类型)若此文件为设备文件，则为其设备编号  
    off\_t         st\_size;      //文件字节数(文件大小)  
    unsigned long st\_blksize;   //块大小(文件系统的I/O 缓冲区大小)  
    unsigned long st\_blocks;    //块数  
    time\_t        st\_atime;     //最后一次访问时间  
    time\_t        st\_mtime;     //最后一次修改时间  
    time\_t        st\_ctime;     //最后一次改变时间(指属性)  
};

这么详细的结构，如果自己想打印，那么完全可以打印一遍。代码中的statbuf结构体。

4、得到结构体以后，就是else if (!S\_ISREG(statbuf.st\_mode) && !S\_ISBLK(statbuf.st\_mode))  
        perr("Error: device must be a block device or a file");

这种直接利用，这两个大写的是宏定义，分别判断文件类型是否是常规文件，或者一个块设备BLK。类似的还有，

S\_ISREG是否是一个常规文件.

S\_ISDIR是否是一个目录

S\_ISCHR是否是一个字符设备.

S\_ISBLK是否是一个块设备

S\_ISFIFO是否是一个FIFO文件.

S\_ISSOCK是否是一个SOCKET文件.

这些宏定义对结构体成员的内容判断。

5、    if (cflag)  
        disk\_scan(device);  /\* check the block device \*/

这个是cflag，那么，从man mkfs.nilfs2上可以看到，c的意思是在build filesystem之前，检测有没有badblocks。做个检测。因此如下，在分析输入命令时候，如果出现短参数-c，那么就进行给全局变量cflag加上值，为1.所以当有-c就是做disk\_scan函数，**我们保留着一会分析disk\_scan()函数**。

case 'c':  
            cflag++;  
            break;

6、fd = open(device, O\_RDWR);

这个就是标准的打开了，并且O\_RDWR是实际的打开，能够读，也能够写入。

if (fd < 0)就是失败了怎么做。

7、**check\_mount**(fd, device);这个是，检测mount挂载的。但是对这个确实不熟悉，先分析，

8、**check\_safety\_of\_device\_overwrite(fd, device);**

9、nilfs\_disk\_info

第二部分：

就是实际的vfs理解。