山东大学 · 泰山学堂

操作系统课程设计

VLFO: Linux 文件操作可视化工具

韩　卓 (201300302014)

冯宗浩 (201300130013)

指导老师　杨兴强

2016 年 1 月 23 日

目录

1. 概述

在本次操作系统课程设计中，我们构建了一个完整的 linux 文件操作可视化工具 VLFO (**V**isualizer for **L**inux **F**ile **O**perations)。该工具主要包含以下两个组件：

- 一个修改版的 linux 0.11 操作系统，与原系统相比，增加了输出系统的内核级调试信息的功能；
- 一个交互式网站，详尽地解释了 linux 文件操作的原理，并且能够智能地分析系统的真实调试信息，以动画的形式展示文件操作的流程。

2. 系统修改

2.1. Linux 0.11 简介

Linux 0.11 发行于 1991 年，是一个较为原始的 linux 版本，其代码量约为两万行。经过二十余年的发展，目前的 linux 内核日趋成熟，加入了许多复杂的特性，代码量已经增长至数百万行，一个直观的增长图线如图. 1 所示。相比之下，linux 0.11 的源代码更易于阅读和修改，同时其功能虽然相对简单，但设计思想与今日大致相同，从学习的角度来讲并没有太大差别。另外值得一提的是 linux 0.11 已经实现对硬盘和软驱设备的支持，这对我们研究文件操作十分有利。综合以上考虑，我们选择以 linux 0.11 为基础修改代码。

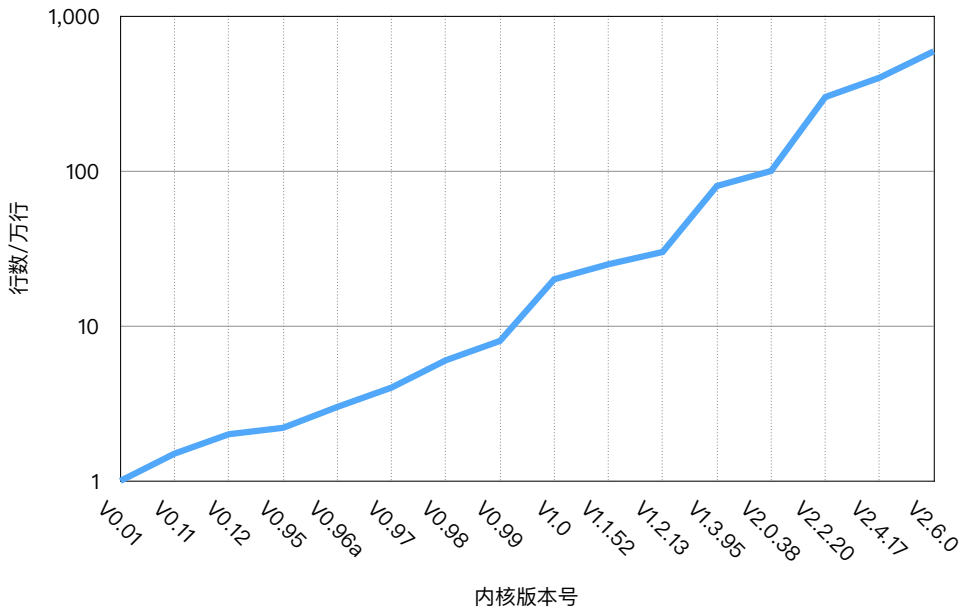

图. 1. Linux 内核各版本源代码行数

2.2. 工作环境

我们在 Bochs[1] 虚拟机上运行与调试 linux 0.11。Bochs 是一款开源的 PC 机仿真器，它仿真了 x86 的硬件环境（CPU 指令）及其外围设备，具有易于移植、便于调试等优点。

在 OldLinux 网站可以获取包含有 Bochs 模拟器、linux 0.11 根文件系统和 linux 0.11 内核启动镜像文件的集成软件包[2]。解压后执行安装程序，再执行相应的配置文件，即可在 Bochs 中启动 linux 0.11 系统，效果如图. 2 所示。

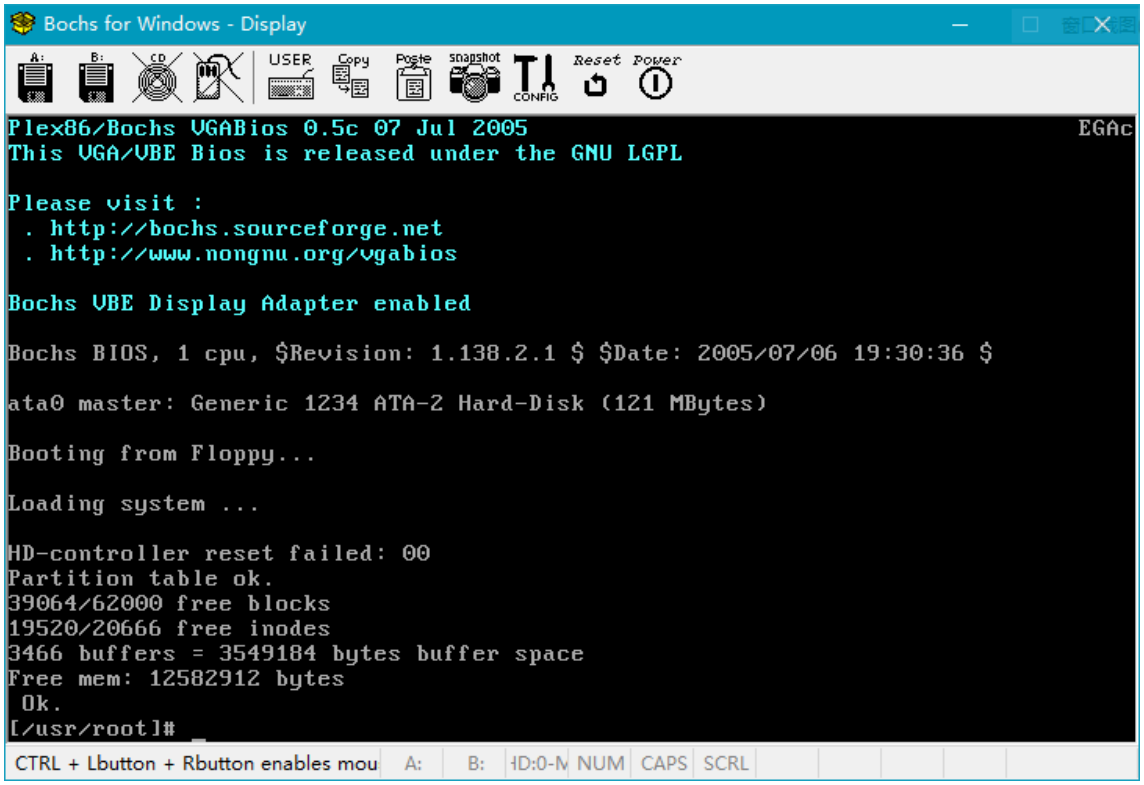

图. 2. 在 Bochs 中运行 linux 0.11

2.3. 内核编译

由于我们使用的系统中已经包含了 linux 0.11 源代码和 gcc 1.40 编译环境，因此我们只需利用自举的思想，修改源代码并编译得到新的引导启动映像文件，再替换原有的引导文件即可。

Linux 0.11 源代码位于 `/usr/src/linux` 目录。移动到该目录，并对代码进行修改之后，执行 `make` 命令即可进行编译。如果代码没有错误，便可得到名为 `Image` 的引导启动映像文件。为了使虚拟机从新的引导文件启动，执行

```
dd bs=8192 if=Image of=/dev/fd0
```

[1] http://sourceforge.net/projects/bochs/
[2] http://oldlinux.org/Linux.old/bochs/linux-0.11-devel-060625.zip

命令，以完成引导启动文件的替换。之后重新启动 Bochs，便可进入修改后的 linux 0.11 系统。[2]

2.4. 输出调试信息

考虑到需要输出的调试信息可能较多，我们决定先将调试信息输出至文件，再从虚拟机中取出。

2.4.1. 创建日志文件

首先要在操作系统启动时建立日志文件。我们需要修改 main 函数，在建立进程 1 之前提前初始化文件系统。这是通过移动 init 函数中的相关代码实现的。同时我们还需要添加一个文件描述符，由于描述符 0、1、2 已经分别关联到 stdin、stdout、stderr，我们选择把日志文件的描述符关联到 3。修改后的 main.c 的部分代码如下所示：

```
    move_to_user_mode();

    /* modify begin */
    setup((void *) &drive_info);
    (void) open("/dev/tty0",O_RDWR,0);
    (void) dup(0);
    (void) dup(0);
    (void) open("/var/linux.log",O_CREAT|O_TRUNC|O_WRONLY,0666);
    /* modify end */

    if (!fork()) {      /* we count on this going ok */
        init();
    }
```

可以看到，通过添加上述代码，我们能够在进程 0 中创建日志文件 /var/linux.log，并将其与文件描述符 3 关联。

2.4.2. 写入日志文件

由于我们需要记录的操作都在内核态内进行，因此无法调用 printf 与 fprintf 进行输出，只有 printk 可用。考虑到我们需要输出至文件，而 printk 无法完成这项操作，因此我们通过自己编写 fprintk 函数来实现内核态下的文件输出。我们向 kernel/printk.c 加入如下的代码：

```
#include <linux/sched.h>
#include <sys/stat.h>

static char logbuf[1024];
int fprintk(int fd, const char *fmt, ...)
{
    va_list args;
    int count;
    struct file * file;
```

```
    struct m_inode * inode;

    va_start(args, fmt);
    count=vsprintf(logbuf, fmt, args);
    va_end(args);

    if (fd < 3)
    {
        __asm__("push %%fs\n\t"
            "push %%ds\n\t"
            "pop %%fs\n\t"
            "pushl %0\n\t"
            "pushl $_logbuf\n\t"
            "pushl %1\n\t"
            "call _sys_write\n\t"
            "addl $8,%%esp\n\t"
            "popl %0\n\t"
            "pop %%fs"
            ::"r" (count),"r" (fd):"ax","cx","dx");
    }
    else
    {
        if (!(file=task[0]->filp[fd]))
            return 0;
        inode=file->f_inode;

        __asm__("push %%fs\n\t"
            "push %%ds\n\t"
            "pop %%fs\n\t"
            "pushl %0\n\t"
            "pushl $_logbuf\n\t"
            "pushl %1\n\t"
            "pushl %2\n\t"
            "call _file_write\n\t"
            "addl $12,%%esp\n\t"
            "popl %0\n\t"
            "pop %%fs"
            ::"r" (count),"r" (file),"r" (inode):"ax","cx","dx");
    }
    return count;
}
```

实现 fprintk 函数之后，在需要输出调试信息的位置调用该函数即可。它的使用方式与 fprintf 类似，不过需要注意第一个参数要传入文件描述符而并非文件指针。之前我们已经将日志文件的描述符关联到 3，因此一个简单的调试信息输出实例如下所示：

```
int sys_creat(const char * pathname, int mode)
{
    fprintk(3, "-sys_creat at %ld\n", jiffies);

    return sys_open(pathname, O_CREAT | O_TRUNC | O_DEBUG, mode);
}
```

上述代码修改了 fs/open.c 中的 sys_creat 函数，在函数开始增加了一句 fprintk 的调用，所以每当 sys_creat 被执行时，都会向日志文件中输出相应的信息。

2.5. 文件交换

尽管 linux 0.11 中已经配置了 vi 文本编辑器，但 vi 的使用方法较为复杂，学习曲线较为陡峭，在需要修改大量代码时可能并不方便；而且，我们输出的日志文件保存在了 linux 0.11 的虚拟机中，但我们需要取出对其进行进一步处理。因此我们需要实现 bochs 虚拟机与外部的文件交换，以方便传入修改后的代码和取出日志文件。

文件交换过程主要借助 WinImage[3] 软件来实现。通过 WinImage，我们可以读写虚拟机加载的软盘镜像文件，以它作为中介实现少量的文件交换。

而如果想在虚拟机中对软盘镜像文件进行操作，可以通过 mtools 来实现。在 linux 0.11 中已经配置好了 mtools 软件，它的功能是为类 UNIX 系统提供读写 MSDOS 文件系统的支持。我们使用的 linux 0.11 虚拟机的配置文件已经将 diskb.img 软盘镜像文件加载至虚拟机的第二个软驱中，因此可在 linux 0.11 里使用 mtools 对其进行操作。mtools 的命令与 MSDOS 的命令相对应，只不过命令开头加上了字母 m 进行区分，例如 mdir、mcopy 等。我们主要用到的是 mcopy 命令，该命令实现了文件的复制功能，第一个参数为待复制文件，第二个参数为目标地址。例如下面这条命令，完成了从软盘复制 main.c 到虚拟机中相应位置的操作：

```
mcopy b:main.c /usr/src/linux/init
```

再比如下面这条命令，作用是将日志文件复制到软盘上：

```
mcopy /var/linux.log b:
```

2.6. 调试信息格式

掌握上述的几种方法后我们便可以按照自己的需求修改 linux 0.11 系统了。在本设计中，我们需要跟踪 linux 文件操作的调用流程，因此我们修改了 open.c、namei.c、inode.c 等文件，改动涉及数十个函数，以获取函数调用的完整轨迹信息。

对每个函数，我们在其开始部分输出函数名称和调用时间。linux 0.11 中定义了全局变量 jiffies（滴答），它记录了从开机到当前时间的时钟中断发生次数，更直观地来讲，它是一个值每 10 ms 增加 1 的长整型数。我们可将 jiffies 输出来实现计时效果。在函数内部，我们适当地输出了一些函数的执行细节。

需要注意的是，我们必须区分调试信息输出的时机。由于 linux 0.11 在系统开机初始化时也会进行文件操作，这些操作的时间早于我们建立日志文件，因此有可能导致开机时内核崩溃。一种方法是将信息先行暂存于内存中，之后再集中输出；另一个可行的解决方法是设置额外的

[3]http://www.winimage.com/download.htm

全局变量，来标记系统是否处于调试模式。我们选择了后一种方法，因为它不仅易于实现，而且有助于筛去无用的调试信息。为了标记调试模式的开始，我们可以自行编写一个系统调用来开启调试模式。但是经过仔细的观察与研究，我们发现创建文件的系统调用，即 sys_creat 函数不会在开机时被调用，而且其传进的 flag 参数是根据二进制位进行标记，因此我们定义了自己的 flag 标记位，用于标记调试模式的开启。

下面的测试程序进行了创建文件、写入文件、打开文件、读取文件、关闭文件以及删除文件的操作。

```
#include <fcntl.h>
#include <stdio.h>
#include <string.h>

int main() {
    int fd;
    int siz;
    char buf[128];
    char str[] = "Hello world";

    fd = creat("/usr/root/hello.txt", 0644);
    siz = write(fd, str, strlen(str));

    fd = open("/usr/root/hello.txt", O_RDWR|O_DEBUG|O_DEBUG, 0644);
    siz = read(fd, buf, sizeof(buf));

    close(fd);
    unlink("/usr/root/hello.txt");
    return 0;
}
```

执行上面的程序后，输出了如下的调试信息。其中短横线标识了函数名称和调用时间信息，点标识了函数内部的细节信息，它们的个数反应了函数的调用深度。

```
—sys_creat at 4168
—sys_open at 4168
.find free filp
.free filp found at filp[3]
.find free file_table
.free file_table found at file_table[59]
.f_count is 1
.filp and file_table connected
——open_namei at 4168
..check file mode
..find inode
———dir_namei at 4169
————get_dir at 4169
..../
....usr
—————find_entry at 4169
.....find entry in cache
—————get_empty_inode at 4169
.....free inode found at inode_table[6]
....root
```

```
------find_entry at 4169
.....find entry in cache
------get_empty_inode at 4169
.....free inode found at inode_table[7]
....hello.txt
.../usr/root/hello.txt
---find_entry at 4170
...find entry in cache
..check permission
---get_empty_inode at 4170
...free inode found at inode_table[8]
..set inode info
..i_uid = 0
..i_mode = 33152
..i_dirt = 1
---add_entry at 4170
.init file struct
.f_mode = 33152
.f_flags = 4672
.f_count = 1
.f_inode = 107548
.f_pos = 0

-sys_write at 4171

-sys_open at 4171
.find free filp
.free filp found at filp[4]
.find free file_table
.free file_table found at file_table[58]
.f_count is 1
.filp and file_table connected
--open_namei at 4171
..check file mode
..find inode
---dir_namei at 4171
-----get_dir at 4171
..../
....usr
------find_entry at 4171
.....find entry in cache
------get_empty_inode at 4171
.....free inode found at inode_table[9]
....root
------find_entry at 4172
.....find entry in cache
------get_empty_inode at 4172
.....free inode found at inode_table[12]
....hello.txt
.../usr/root/hello.txt
---find_entry at 4172
...find entry in cache
..read inode content
---get_empty_inode at 4173
...free inode found at inode_table[13]
```

```
..update inode visit time
.init file struct
.f_mode = 33152
.f_flags = 4098
.f_count = 1
.f_inode = 107548
.f_pos = 0

—sys_read at 4173

—sys_close at 4173
.filp—>f_count = 1
.f_count is 0, free inode
—iput at 4173

—sys_unlink at 4173
——dir_namei at 4173
———get_dir at 4173
.../
...usr
————find_entry at 4173
....find entry in cache
————get_empty_inode at 4173
....free inode found at inode_table[14]
...root
————find_entry at 4174
....find entry in cache
————get_empty_inode at 4174
....free inode found at inode_table[15]
...hello.txt
../usr/root/hello.txt
——find_entry at 4174
..find entry in cache
——get_empty_inode at 4174
..free inode found at inode_table[16]
```

输出了调试信息之后，我们便可以用之前描述的方法取出日志文件，并将其传入下面将要描述的可视化网站中，生成示意动画。

3. 可视化设计

我们搭建的可视化网站主要包含四个部分。

3.1. 相关操作

这一部分展示了 linux 中主要的文件操作，外观如图. 3 所示。将鼠标移至相应的操作卡片上方，即可查看该操作的示例调试信息，这些信息可直接用于后面动画演示。

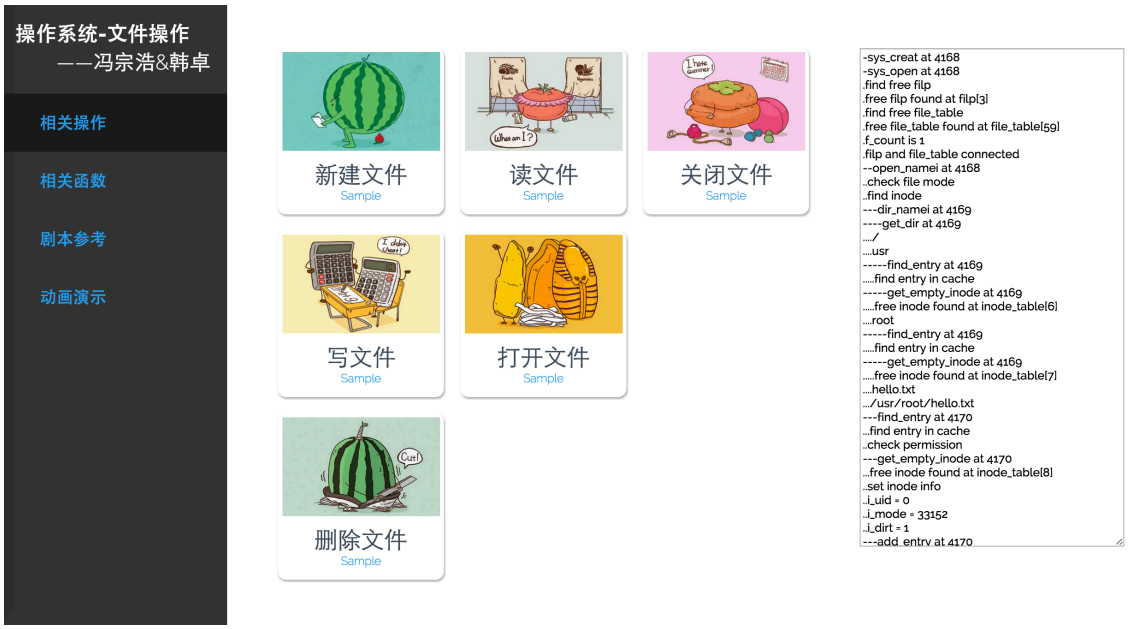

图. 3. 相关操作

3.2. 相关函数

这一部分展示了 linux 0.11 中与文件操作紧密关联的一些函数的定义与实现，外观如图. 4 所示。这些函数都能在修改过的 linux 0.11 系统中产生相对应的调试信息，以方便我们观察其调用流程。

图. 4. 相关函数

3.3. 剧本参考

这一部分展示了我们编写的《闻剑的一生》剧本，外观如图. 5 所示。该部分旨在以剧本的形式，通过讲述 linux 王国内发生的各种故事，用拟人化的手段生动地表现文件操作的流程。

图. 5. 剧本参考

3.4. 动画演示

这一部分是可视化设计的核心，主要功能是对调试文件进行解析，用动画一步步展示文件操作的内部机理，外观如图. 6 所示。下面将结合实际操作详解动画演示的工作流程。

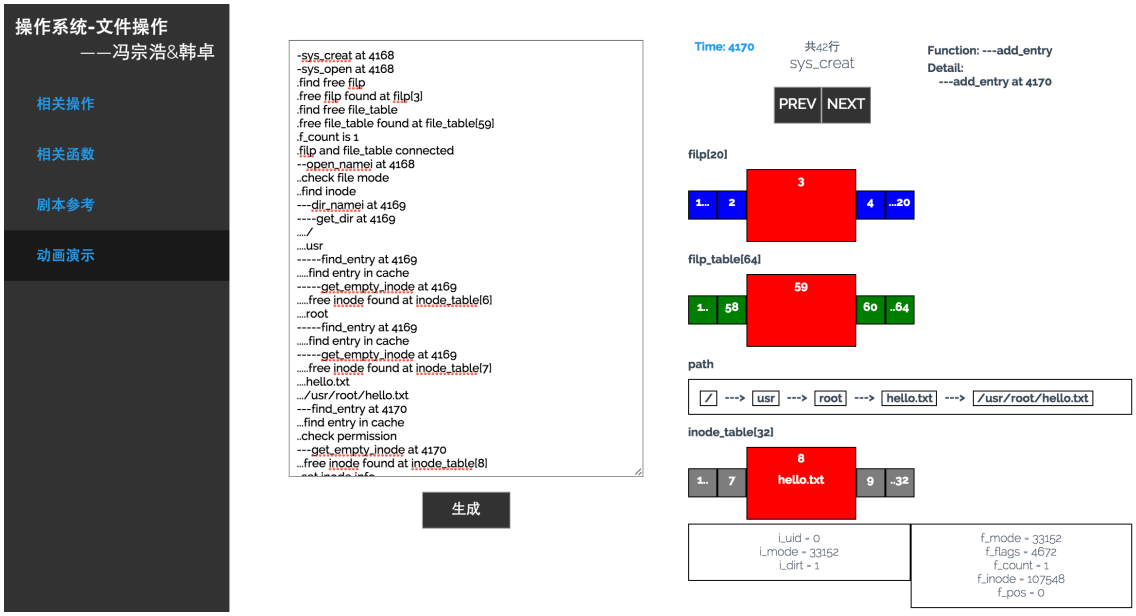

图. 6. 动画演示

图. 6 展示了创建文件操作的可视化效果。可以看到，动画演示区的左部设置了文本框，在此处输入调试信息，并点击生成，即开始了调试文件的解析。在这里既可使用预先提供的调试文件，也可使用自己实际操作中生成的调试文件。与此同时，在界面右部会出现动画演示操作台，用户可以自由地点击上一步、下一步来浏览可视化效果，具有极强的交互性。对于每一步操作，将会展示操作的详情、操作从属的函数以及操作执行的时间。

对于涉及到数组的操作，例如寻找进程空闲的文件指针、寻找系统文件结点表的空闲项、文件指针表与文件结点表的挂接 [1] 等，将特别展示出数组的结构，可参见图. 6 中 flip、filp_table 以及 inode_table 数组的可视化，如图. 7 所示。

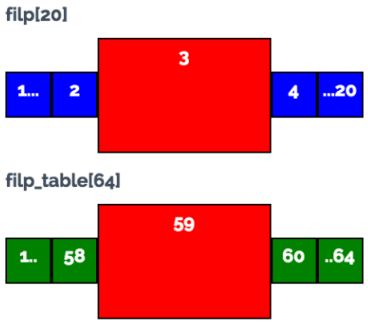

图. 7. 数组可视化

特别地，对于路径解析过程，我们会展示出路径的完整链式结构，如图. 8 所示。

图. 8. 路径可视化

4. 总结

通过本次操作系统课程设计，我们深入探索了 linux 文件操作的原理，并在理解知识的基础上加以运用，最终设计出了 linux 文件操作可视化工具 VLFO。

VLFO 是依照化抽象为具体的思想来设计的，它能够挖掘出文件操作背后的系统调用流程，并以动画的形式清晰而细致地加以展现。它的最大特点是能够捕获和处理系统的真实调试信息来进行可视化。我们认为，VLFO 对理解 linux 文件操作的原理有极大的帮助，富有实际意义。

引用

1. 新设计团队. *Linux* 内核设计的艺术（第 *2* 版）. 机械工业出版社, 2013.

2. 赵炯. *Linux* 内核完全注释. http://oldlinux.org/download/clk011c-3.0-toc.pdf, 2007.