山东大学
SHANDONG UNIVERSITY

Linux操作系统运行过程可视化

学　　院: 泰山学堂

专　　业: 计算机取向

学　　号: 201600301291

姓　　名: 王文嵩

同组成员: 杜洪超

指导老师: 杨兴强

2018~2019学年第一学期

使用LaTeX撰写于2019 年 1 月 12 日

目录

1 实验目的与环境

1.1 实验目的

- 该实验以Linux0.11为例探索操作系统的结构、方法和运行过程，理解计算机软件和硬件协同工作的机制。主要需要完成以下4项任务:

  1. 分析Linux0.11系统源代码，了解操作系统的结构和方法。
  2. 通过调试、输出运行过程中关键状态数据等方式，观察、探究Linux系统的运行过程。
  3. 建立合适的数据结构，描述Linux0.11系统运行过程中的关键状态和操作，记录系统中的这些关键运行数据，形成系统运行日志。
  4. 用图形表示计算机系统中的各种软、硬件对象，如内存、CPU、驱动程序、键盘、中断事件等等。根据已经产生的系统运行日志，以动画的动态演示系统的运行过程。

1.2 Linux-0.11 介绍

通过阅读Linux早期内核版本的源代码，的确是学习Linux系统的一种行之有效的途径，并且对研究和应用Linux嵌入式系统也有很大帮助，正如Linux系统的创始人linux所言，要理解一个软件系统的真正运行机制，一定要阅读其源代码(RTFSC-Read The Fucking Source Code)。系统本身是一个完整的整体，具有很多看似不重要的细节存在，但若是忽略这些细节，就会对整个系统的理解带来困难。而目前的Linux内核源代码数量都在几百万行，阅读这些代码不是一个人几年能完成的工作量，而0.11版内核不超过2万行代码量，但它却基本涵盖了一个完整的Linux系统的运行过程，麻雀虽小，五脏俱全，使我们能在尽可能短的时间内深入理解Linux内核的基本工作原理。

1.3 实验安排

1. Linux0.11源代码分成基础模块和选读模块，前四周分析基础模块，然后从选读模块中选择感兴趣的模块文件系统重点分析。
2. 针对文件系统中文件的创建、文件的打开、文件的写入以及文件的关闭，分析在此过程中操作系统的运行状态。
3. 在代码级别上理解系统的运行过程，获取系统运行过程的数据，生成系统运行日志。
4. 演示系统运行的过程。

1.4 实验环境

我们搭建了Linux0.11实验平台，大体基于上一届学长的工作，但针对我们的实验方案做了改进，详细情况见https://github.com/MrDuGitHub/OS/tree/master/the-linux-kernel，其中要点如下：

1. 实验平台基于开源的Linux 0.11 Lab实验平台，包括Linux 0.11及qemu和bochs环境，能在多种环境下运行，且因为支持qemu和bochs能进行多种调试。图1就是我们使用的qemu界面。关于这个基础平台的详情参看https://github.com/tinyclub/linux-0.11-lab

2. 在Linux 0.11 Lab的基础上，学长通过gdb脚本和添加函数实现了log功能，只需要在源代码合适位置添加log函数，使用与C原因类似的格式化输出方式，就能将想获取的数据保存在文件中，另外还实现了一键启动脚本等功能。

3. 经过简单实验后，发现原有实验平台对调试文件系统的支持存在问题，如不支持vi，无法删除文件等，因此对实验平台进行了改进，使用了新的文件系统镜像，添加了vi支持；添加了系统调用以实现文件的正常删除操作。

4. 为了更适合设计方案，使用git技术创建了基于分支的环境框架，包括基础分支和模块分支，每个模块分别实现不同的功能，如提取不同过程的数据等，减小了提取数据的工作量和复杂程度；构建了简单的基础分支，用于扩展出不同的模块分支。

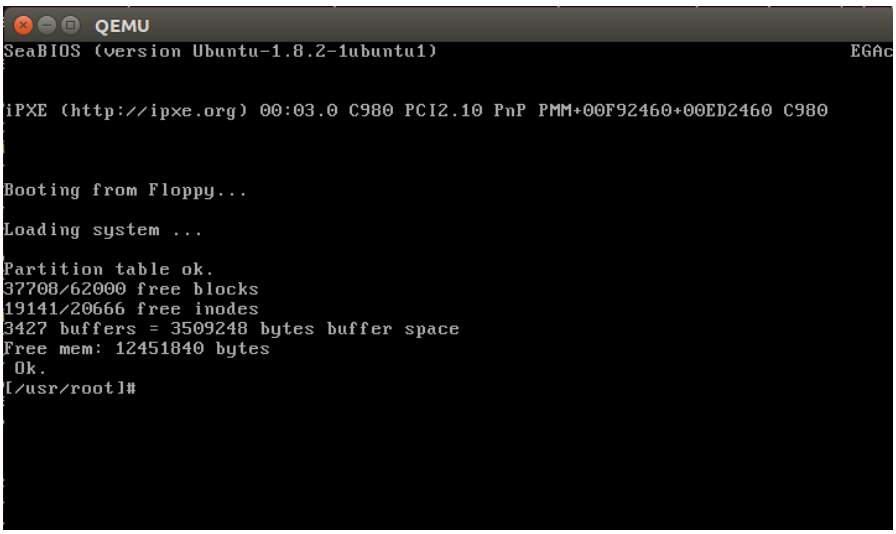


图 1: 实验平台

2 实验流程

2.1 实验分工

将整个系统的运行过程可视化需要付出巨大的工作量，一个学期内难以完成。在全面分析源代码的基础上，我们选择了对文件系统的可视化展示，为了分析文件系统，我们决定从文件系统的基本操作出发，如创建文件，写入文件等，通过追踪这些基本操作中代码的调用执行情况，就能对文件系统有一个清晰的了解。同时，因为文件系统相较内核其它部分较为独立，且涉及到了文件系统镜像rootimage.Z，文件系统源代码部分中使用的大部分数据结构与该镜像有关，熟悉了这个镜像的结构才能能有效的分析代码，因此我们除了对文件系统的动态分析(基本操作的过程)，还对文件系统本身进行了静态分析，这样有利于最终的可视化效果。最终我们的实验方案包括：文件系统的静态分析，创建文件，打开文件，写入文件以及关闭文件的动态分析。其中，我负责写入和关闭文件的方案设计。 linux 0.11中文件系统主要由fs目录实现，基本架构图及引用关系如图2所示。

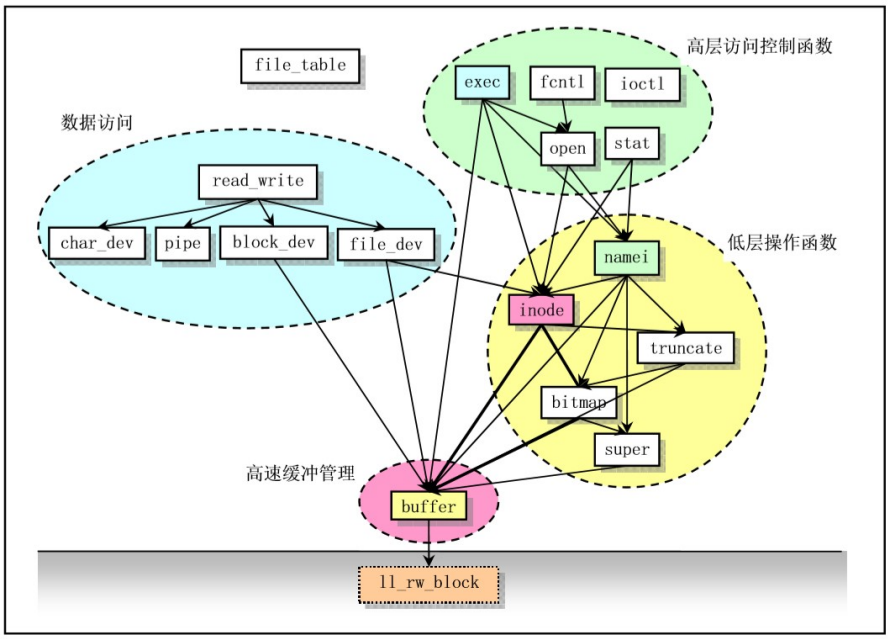


图 2: fs目录下源文件结构及引用关系

2.2 创建文件

1. 创建文件操作会首先调用内核fs目录中open.c里的sys_creat()函数。可以发现这个函数里的实质操作是调用了打开文件函数sys_open()。原因是创建文件和打开文件操作首先都要判断该文件是否已经存在；创建文件的流程是尝试打开文件，如果没有这个文件，才进行真正的创建操作。

2. 对于打开文件的函数，通过传入参数区分是创建操作还是打开操作；同时，在此函数中首先对创建或打开文件做一些准备工作,见图3：

   (1) 对当前要打开文件的进程，查找其文件指针数组，寻找第一个空项

   (2) 设置当前进程的写时复制位图，标记相应的文件指针数组

   (3) 在内存的文件表中找一个空白项，用于存放相应的文件，同时用1中的文件指针指向它

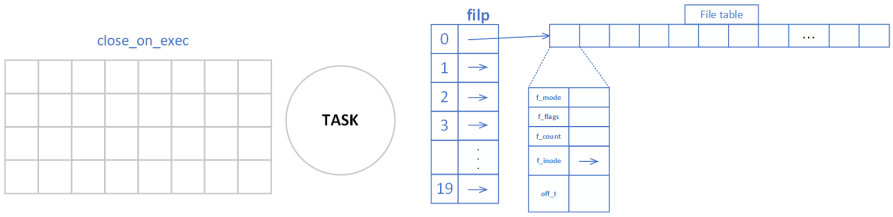


图 3: 创建文件的基础设置

3. 设置完之后，下一步就是调用namei.c中的open_namei()函数，根据文件路径找到相应的文件。根据路径找文件，是通过dir_namei()函数实现的；首先调用get_dir()找到该文件所在的直接文件夹。每个进程有两个inode节点用于寻找文件，一个是根路径节点root，一个是当前目录节点pwd；如果是绝对路径，就从root开始查找，否则就从pwd开始查找。

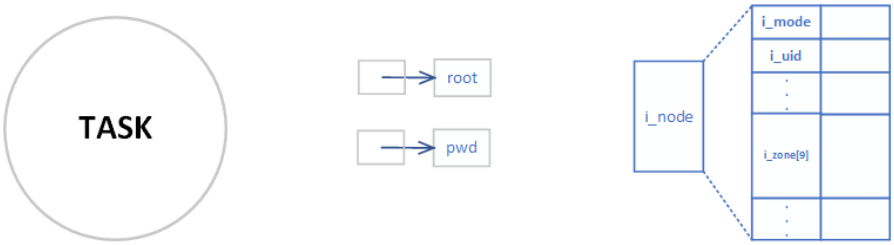


图 4: 当前进程的根inode节点和当前目录inode节点

4. 实验中我们使用的路径是文件名，因此会在pwd所指向的文件夹中寻找这个文件；get_dir()返回pwd后，调用find_entry()函数，在当前目录下查找这个文件。

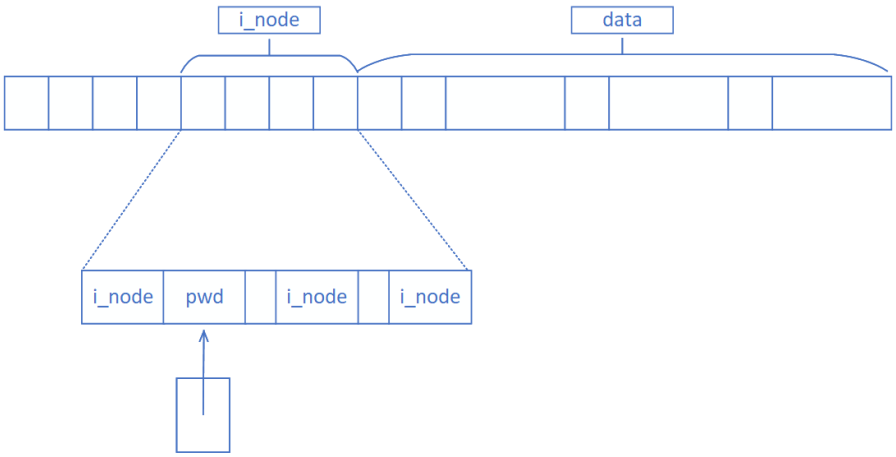


图 5: 找到pwd

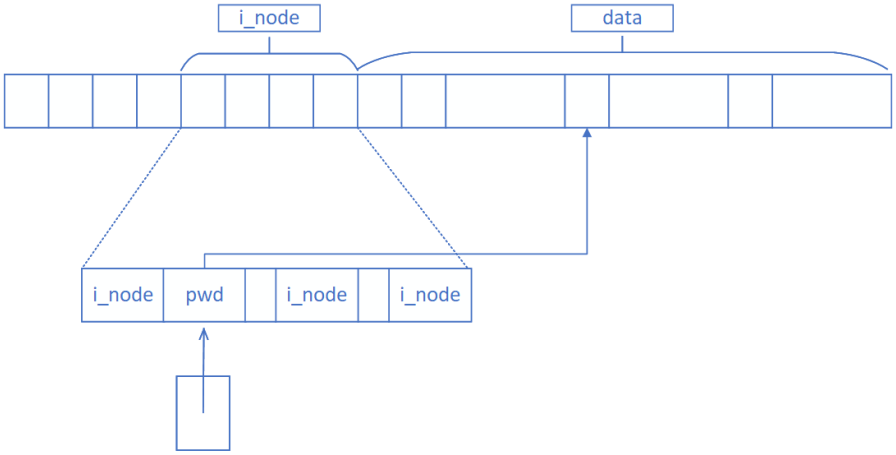


图 6: 找到pwd所指向的数据区

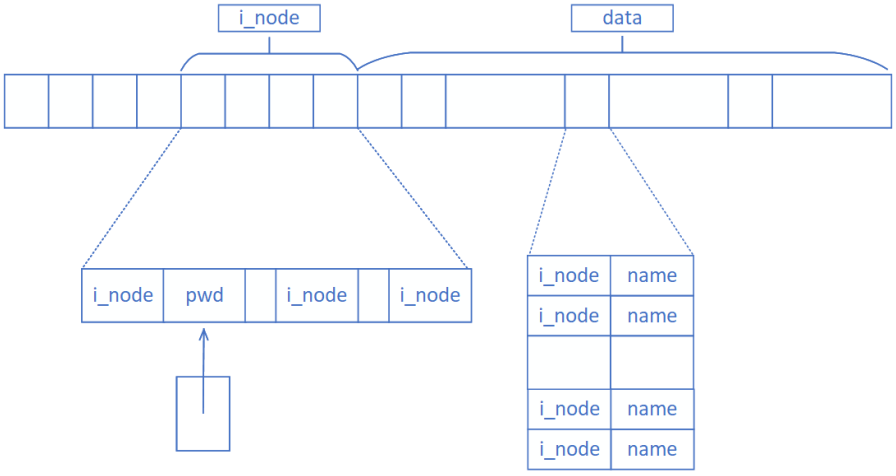


图 7: 从数据区中查找文件

5. 因为是创建文件操作，所以找不到这个文件，下面进入创建文件操作，首先给这个文件分配一个inode节点，调用bitmap.c中的new_inode()函数；在这个函数负责找到一个空白的inode并设置相关标志位，这里主要用到的是inode.c和bitmap.c。最后返回到namei.c中，在当前目录下添加这一项，创建文件操作就完成了。

6. 上述函数调用关系如下:

```
1   //open.c
2   int sys_creat(const char * pathname, int mode)
3   sys_open(pathname, O_CREAT | O_TRUNC, mode);
4
5   /* Open a file */
6   //open.c
7   int sys_open(const char * filename,int flag,int mode)
8   // Find an empty file struction pointer for current progcess, return the index as handle
9   // Set the close_on_exec
10  // Find an empty file structioon in the file table
11  //namei.c
12  open_namei(filename,flag,mode,&inode));
13  // open file,set the inode,return wrong code if failed
14  dir_namei(pathname,&namelen,&basename);
15  // Find the inode of the file specified by filename
16  get_dir(pathname);
17  // Find the inode of the file specified by filename
18  inode = current->pwd;
19  return pwd;
20  // Find the file in this dir
21  find_entry(&dir,basename,namelen,&de);
22  // find the target and return buffer
23  // If this is a create operation, apply for a new inode and add it under the dir
24  new_inode(dir->i_dev); // bitmap.c
25  //bitmap.c
26  struct m_inode * new_inode(int dev)
27  inode=get_empty_inode()
28  //inode.c
29  struct m_inode * get_empty_inode(void)
30  //bitmap.c
31  sb = get_super(dev)
32  //super.c
33  struct super_block * get_super(int dev)
34  //namei.c
35  add_entry(dir,basename,namelen,&de);
36  return;
37  // If not, the file already exists, return the point of inode
```

2.3 打开文件

打开文件与创建文件操作有大部分重叠，从open.c的sys_open()开始，直到扫描文件所

在目录，如果查不到所要打开的文件，就返回打开文件出错；否则返回找到文件的inode节点，将其填入当前文件数组中。

2.4 写入文件

1. 写入文件会调用read_write.c中的sys_write(unsigned int fd,char * buf,int count)函数

   (1) 传入的参数是文件句柄,内容，字节数，先判断文件句柄是否大于文件数或者写入的字节数小于零或者句柄的结构指针为空，都没问题后，得到当前文件结构指针。

   (2) 判断文件结构指针的inode节点是管道类型，还是字符设备或是普通文件，在我们的实验中由于是往普通文件里写入字符串，由于采用echo命令，文件句柄1被重定位，指向要写入的文件，是普通文件类型

   (3) 因此调用普通文件的写入file_wirte(inode,file,buf,count)

   调用file_write(struct m_inode * inode, struct file * filp, char * buf, int count)函数

2. (1) 根据i节点和文件结构信息，将用户数据写入文件中，首先确定文件的位置，判断是向文件后添加数据还是在当前指针处写入

   (2) 在写入字节数小于count时，循环执行：取文件数据块号对应的逻辑号，如果对应的逻辑块不存在就创建一块。

   (3) 取得缓冲块指针，求出文件当前读写指针在该数据块中的偏移值，并将指针指向缓冲块中开始写入数据的位置，并置缓冲块修改位。

   (4) 修改i节点中文件长度字段，并置i节点已修改标志，从用户缓冲区复制字节到高速缓冲区指向的开始处，复制完释放该缓冲区。

   (5) 当数据已经全部写完，修改文件修改时间为当前时间，并调整文件读写指针。

3. 逻辑块不存在则创建调用creat_block(struct m_inode * inode, int block),此函数直接调用_bmap(inode,block,1);1指明是创建一块new标志

   (1) 首先判断文件数据块号block的有效性，如果块号不在0与直接块数加间接块数加二次间接块数之间，则报错

   (2) 由于在我们实验中输入字符比较少，而且是对一个刚刚新建的文件进行写操作，所以block是直接块，向相应的磁盘申请一磁盘块，并将逻辑块号填入inode的izone区域，然后设置i节点改变时间，置i节点已修改标志。

   (3) 向设备申请一个逻辑块，首先根据超级块，在超级块的逻辑位图中寻找第一个0值比特位，置位，接着为该逻辑块在缓冲区中取得一块对应缓冲块,见图8，最后将该缓冲块清零，并设置其更新标志和修改标志

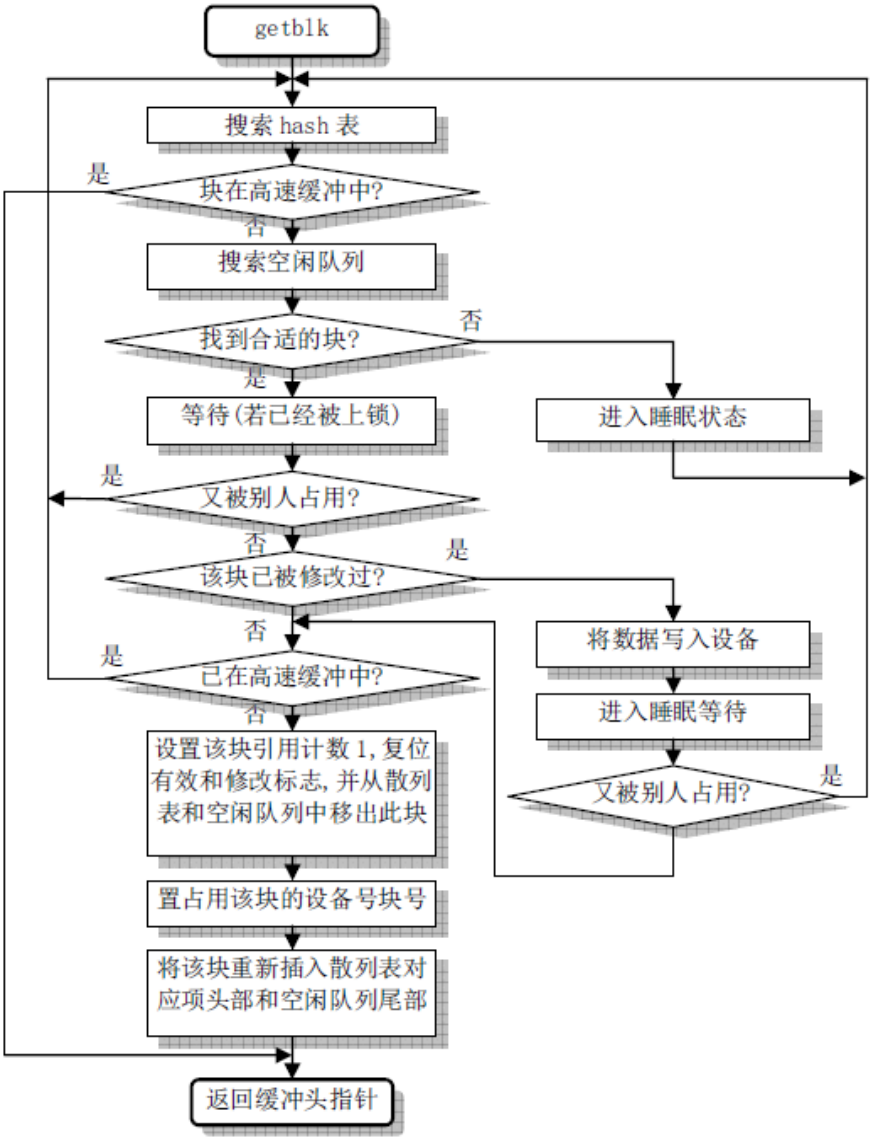


图 8: 获得一块空闲缓冲块

4. 上述函数调用关系如下:

```
//read_write.c
int sys_write(unsigned int fd,char * buf,int count)

file_write(inode,file,buf,count);

/* write buf to the file */
//file_dev.c
int file_write(struct m_inode * inode, struct file * filp, char * buf, int count)
// The task with the file handle which is the std:out and contents and the length of bytes.
// Find the file structure pointer
// Look for the corresponding file table item.
// Judge the inode type
create_block(inode,pos/BLOCK_SIZE)
//inode.c
int create_block(struct m_inode * inode, int block)
    _bmap(inode,block,1);
/*create a new block */
//inode.c
inode->i_zone[block]=new_block(inode->i_dev)
//bitmap.c
int new_block(int dev)
sb = get_super(dev)
j=find_first_zero(bh->b_data))
//super.c
struct super_block * get_super(int dev)
return;
```

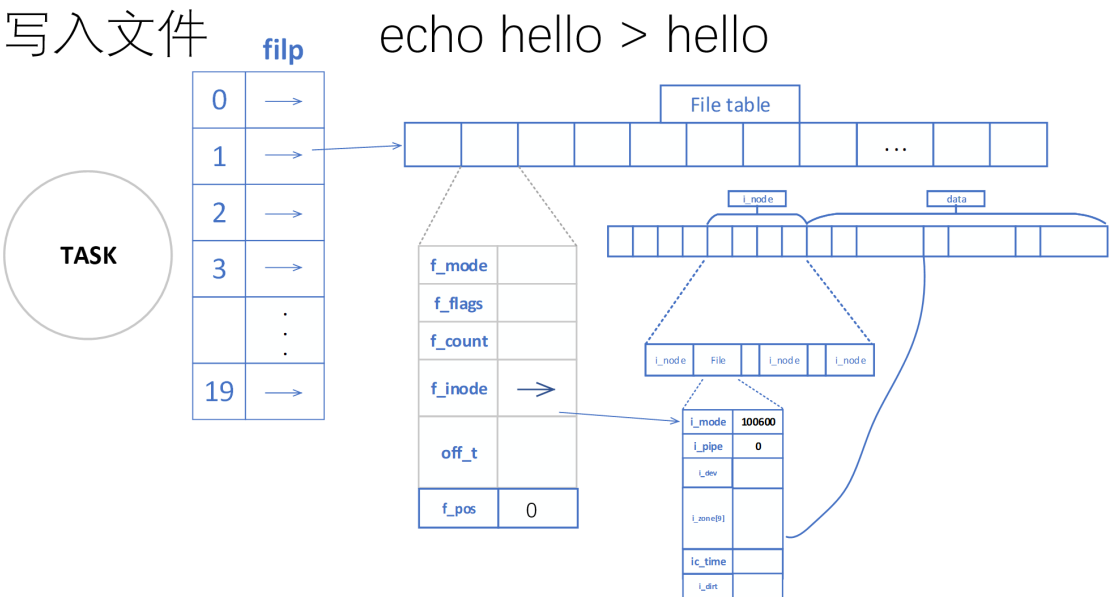


图 9: 写入文件流程

2.5 关闭文件

关闭文件较为简单，调用int sys_close(unsigned int fd)函数,判断传入文件句柄的有效性，复位进程的执行时关闭文件句柄对应位，置该文件句柄的文件结构后指针为null，释放文件i节点。

3 数据提取与筛选

3.1 静态数据提取

静态数据主要根据文件系统镜像提取，我们使用的是改进后的hdc-0.11.img镜像，主要提取手段是使用UltraEdit打开16进制的镜像文件，根据镜像的结构找到相应的数据并进行解释和整理。主要的一些数据如表1：

数据 / 区域	地址空间	大小	大小
引导扇区	0x00000000-0x000003ff	1KB	1KB
分区1	0x00000800-0x03c903ff	62016KB	60MB+576KB
分区2	0x03c90400-0x079207ff	62017KB	60MB+577KB
其它	0x079207ff-0x079b7fff	606KB	606KB
总大小	0x00000000-0x079b7fff	124640KB	121MB+736KB

表 1: 静态数据

我们的Linux 0.11文件系统就挂载在分区1上，数据如表2：

数据 / 分区	地址空间	大小
引导块	0x00000400-0x000007ff	1KB
超级块	0x00000800-0x00000bff	1KB
i节点位图	0x00000c00-0x000017ff	3KB
逻辑块位图	0x00001800-0x000037ff	8KB
i节点	0x00038000-0x000a4fff	645KB
数据区	0x000a5000-0x03c903ff	61357KB

表 2: Linux 0.11文件系统

静态数据的详细内容见https://github.com/MrDuGitHub/OS/blob/master/note.md

3.2 动态数据格式

因为采用了学长的提取数据的方法，所以数据格式为JSON。具体字段如表3：

＼	module	file	function	line	provider	time	data
含义	模块	文件	函数	行号	提供者	时间	数据
举例	”file_system”	”super.c”	”get_super”	63	”Mr.d”	2	””

表 3: 动态数据JSON字段详解

图 10: JSON数据举例

3.3 动态数据

动态数据的具体内容不再具体列出，详见https://github.com/MrDuGitHub/OS/tree/master/data

3.4 统计数据

根据收集到的动态数据，统计信息如下：

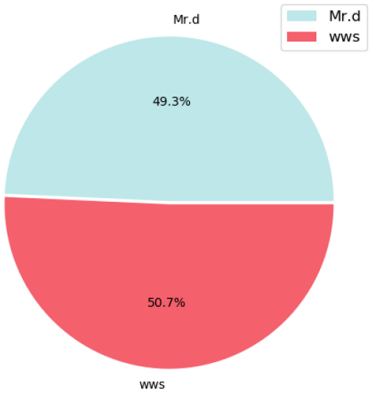


图 11: provider分布

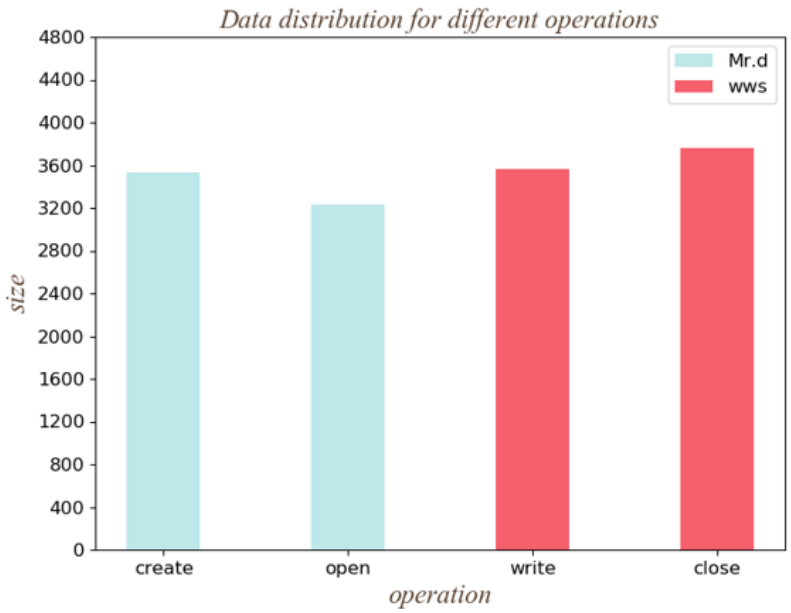


图 12: operation分布

4 可视化与展示

采取了processing作为可视化工具对静态数据的展示和打开文件、创建文件，关闭文件、写入文件进行了可视化，我主要负责写入文件和关闭文件。

4.1 可视化工具

在考察了几种可视化方案后，最终选择了Processing作为可视化工具.关于Processing的介绍可以参见https://processing.org/

4.2 可视化架构

为了更整洁简单的实现可视化，我们基于Processing设计了一个编程架构：

- 将一个完整的动画用若干个场景划分开，在每一个场景中设计若干帧
- 一个帧只包含一个或少数简单的动画
- 若干个基本元素不变的帧就构成了一个场景

有了这两个基本单位后，我们就能把复杂的动画分解开来，用分而治之的方法去设计实现。这样的设计还有其它巧妙的好处，也是设计的初衷之一，那就是用这样的场景与帧的概念去对时间进行合适的划分,时间划分在动画设计中是十分关键的元素。一个场景实际上就是一个函数，每个场景都有一个基本的函数框架，其中实现了场景的时间显示，在每个场景左上角都有动画的总时间，当前处于那个场景的哪个帧，以及该场景已经出现了多长时间。经过合理的设计后，我们在主函数中使用基本的switch语句，只需在场景函数中加入该场景的帧的内容，就可以将一个场景加入到动画中来，在保证了简单易用的基础上提供了相当的灵活性。

因为我们设计方案中有静态展示和动态展示两个部分，再加上分析的是Linux系统，我们在动画中引入了Linux的小企鹅作为串起所有场景的精灵。通过给小企鹅添加对话框就可以在动画的合适位置与时间添加引导和解释语，使动画变得生动易懂。

除了场景与帧的框架设计，为可视化还需要的一些其它功能实现了一系列的功能函数，以供场景-帧的结构调用。

- 时间模块，实现时间控制与显示；
- 动画模块，包括图片文字和图形的的淡入淡出，以及画线效果；
- 云朵模块，为小企鹅添加云朵对话框
- 箭头模块，实现箭头

- 文本模块，实现在各种情景下显示文本

最终的动画一共设计了13个场景，从场景0到场景12，总时长264秒，详细代码见https://github.com/MrDuGitHub/OS/tree/master/visualization/sketch。

动画的开始与结束

在动画的开始和结束，设计了一些欢迎语和结束语。分别是场景0和场景1，总时长19秒，以及最后的场景12，时长11秒。图13和图14分别截取了二者的一个画面。

Scene_0,frame_1
time=3.0 s
L_stime=3,L_mtime=3011

图 13: 欢迎语

Scene_12,frame_3
time=8.0 s
L_stime=8,L_mtime=8003

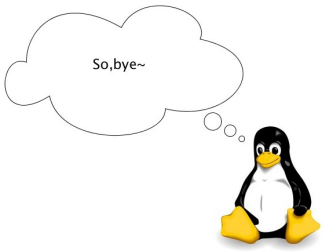
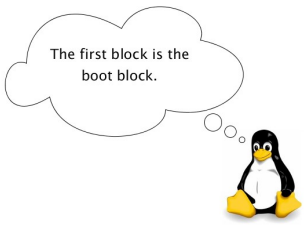


图 14: 结束语

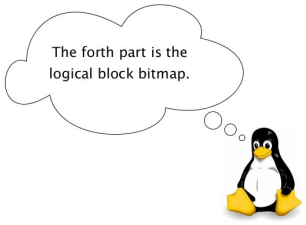
静态展示

静态展示共占用了2个场景，场景2和场景3，总时长63秒。

Scene_2,frame_2
time=8.0 s
L_stime=8,L_mtime=8001



Scene_2,frame_5
time=16.0 s
L_stime=16,L_mtime=16000



Scene_2,frame_7
time=23.0 s
L_stime=23,L_mtime=23001

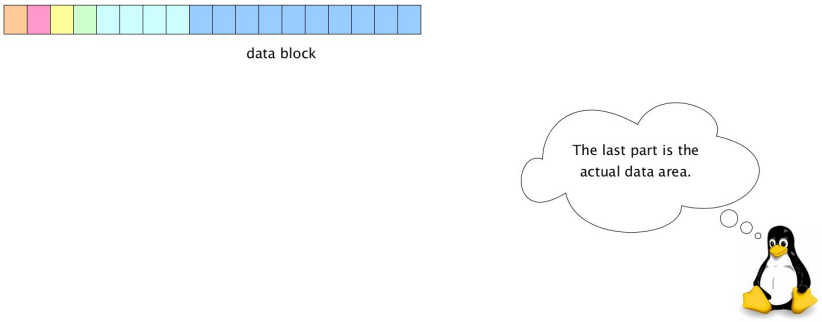
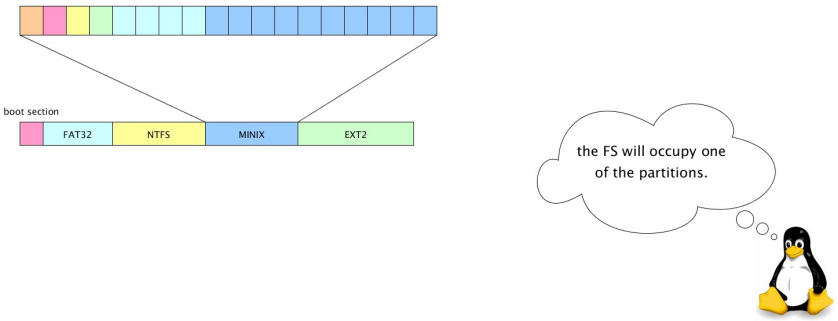
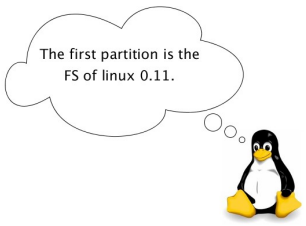


图 15: MINIX文件系统静态展示

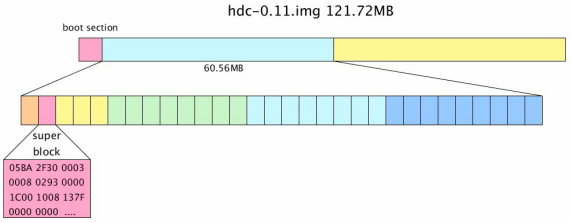
Scene_2,frame_9
time=29.0 s
L_stime=29,L_mtime=29002



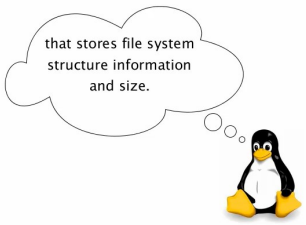
Scene_3,frame_0
time=7.9 s
L_stime=8,L_mtime=8000



Scene_3,frame_1
time=65.6 s
L_stime=16,L_mtime=16680



iNode number	1466	05BA
logical blocks	12080	2F30
inode bitmap blocks	3	0003
logical bitmap blocks	8	0008
first logical block	659	0293
log	0	0000
max_length	4697661 52	1C00 1008
magic number	4991	137F



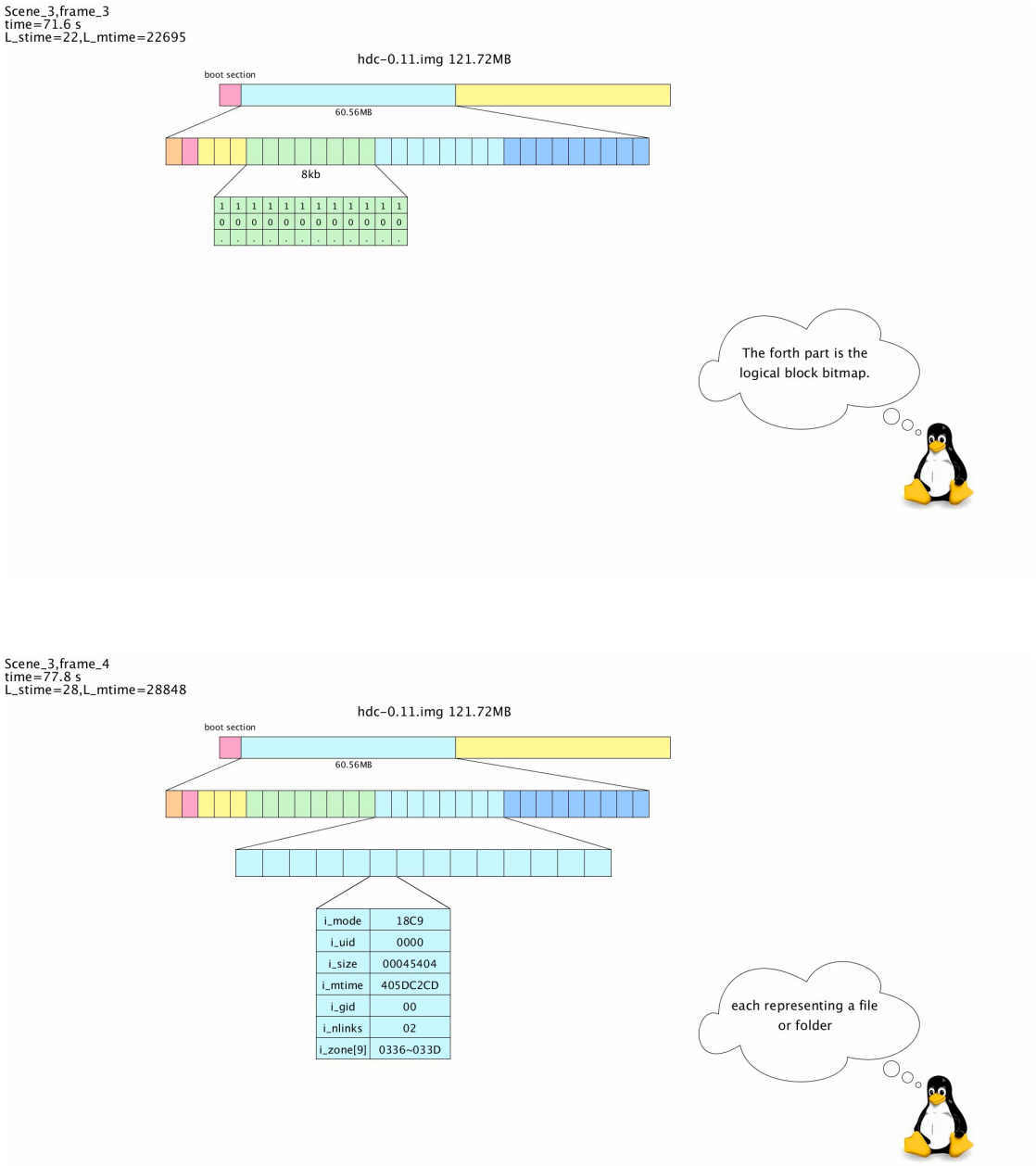


图 16: Linux 0.11文件系统静态展示

动态展示

动态态展示共占用了9个场景，场景4到场景6，总时长78秒由mr.d负责编写，针对打开

文件，创建文件，场景7-12由我负责编写，针对写入文件、关闭文件。

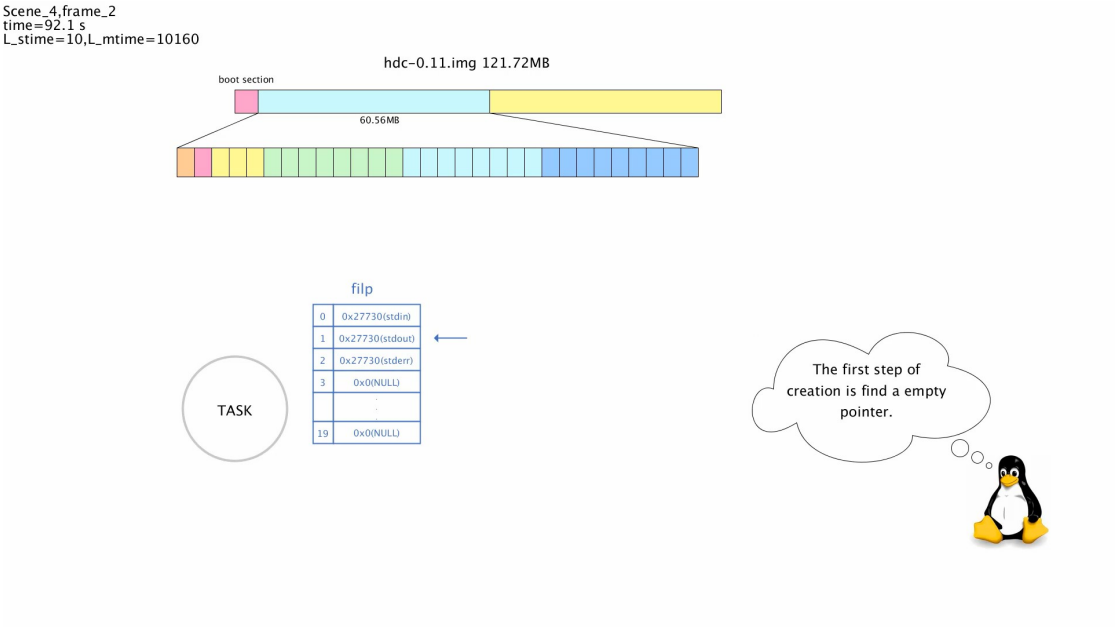


图 17: 查找文件指针数组

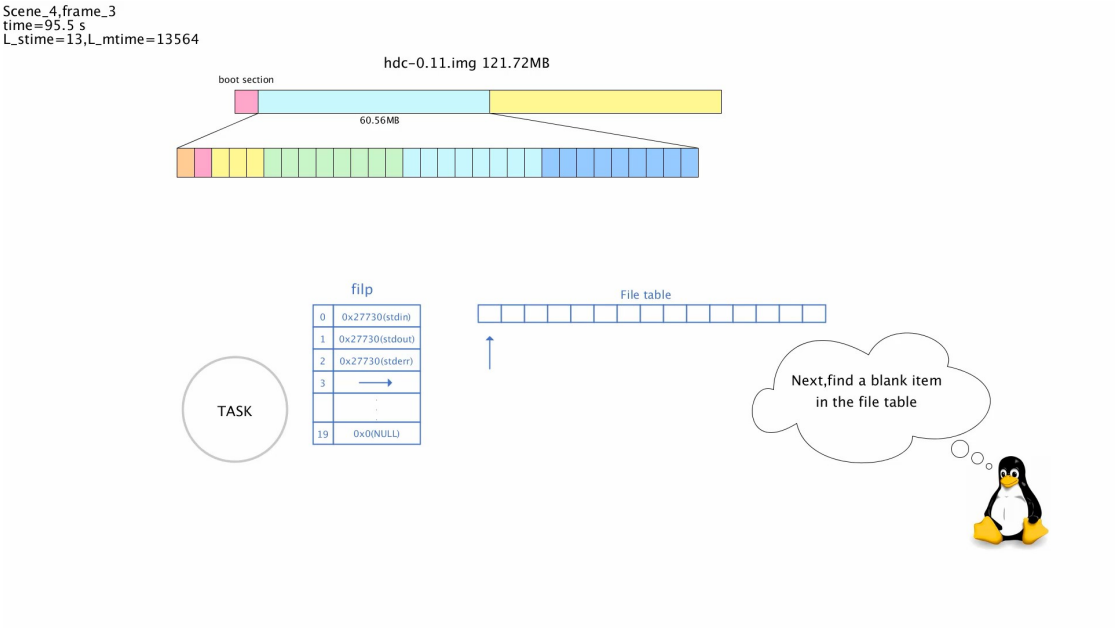


图 18: 查找文件表

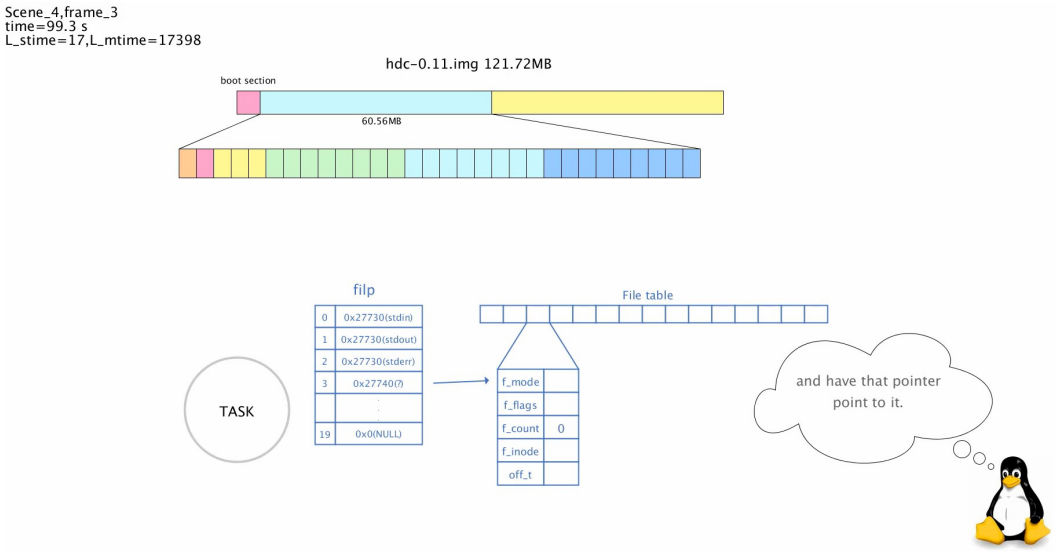


图 19: 设置文件指针数组

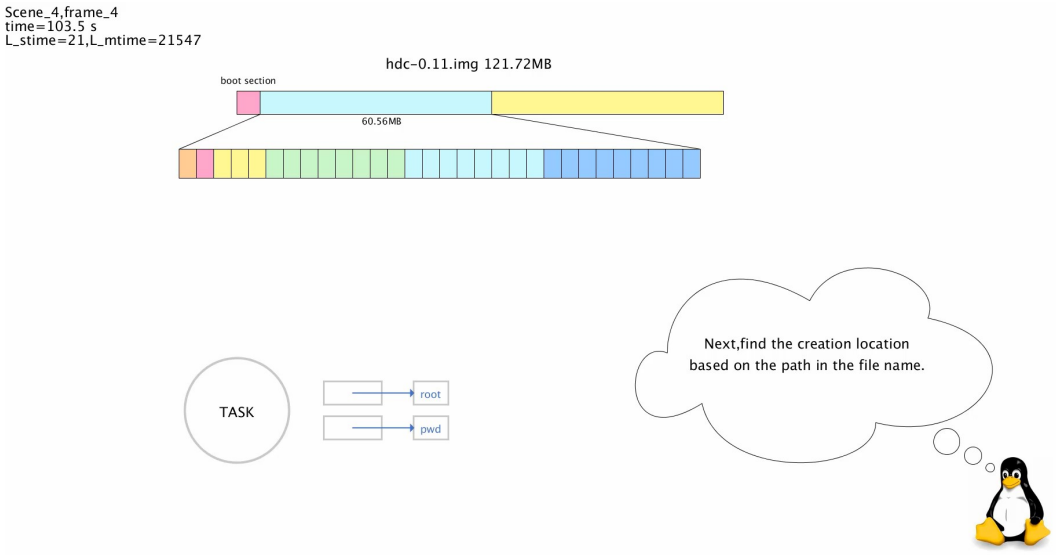


图 20: 查找文件的直接目录

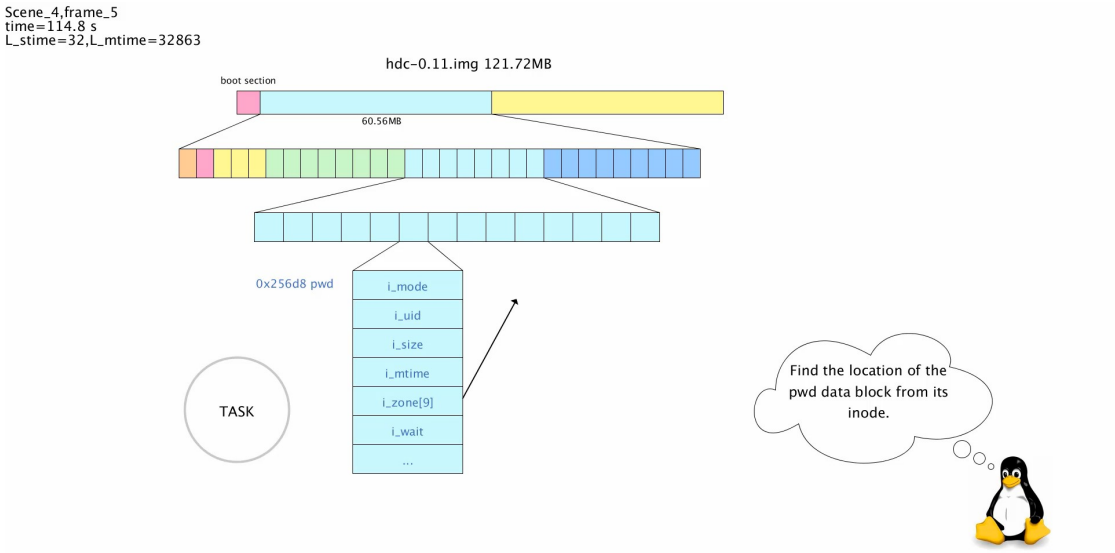


图 21: 寻找pwd的数据区

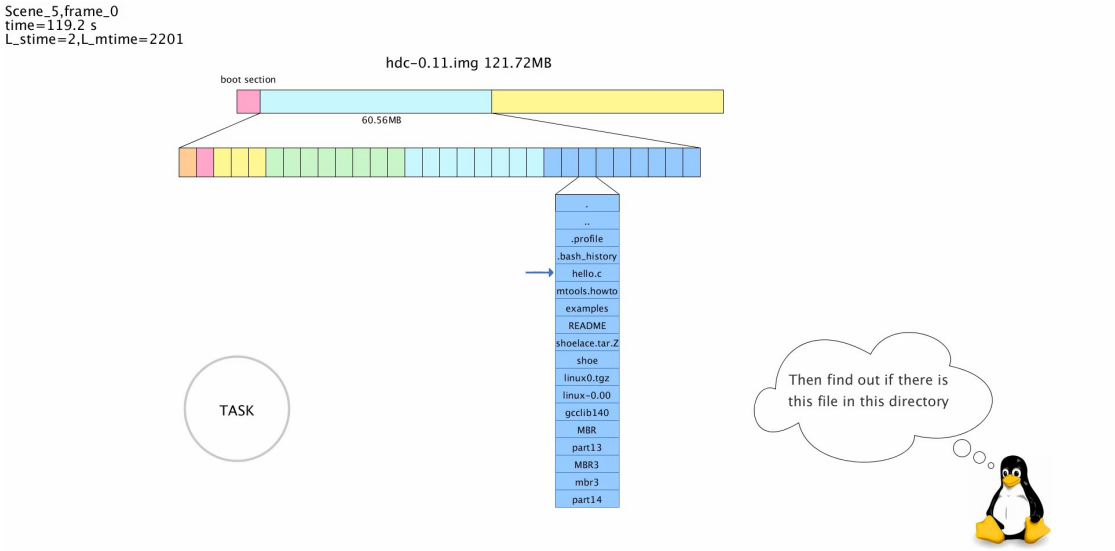


图 22: 在当前目录查找文件

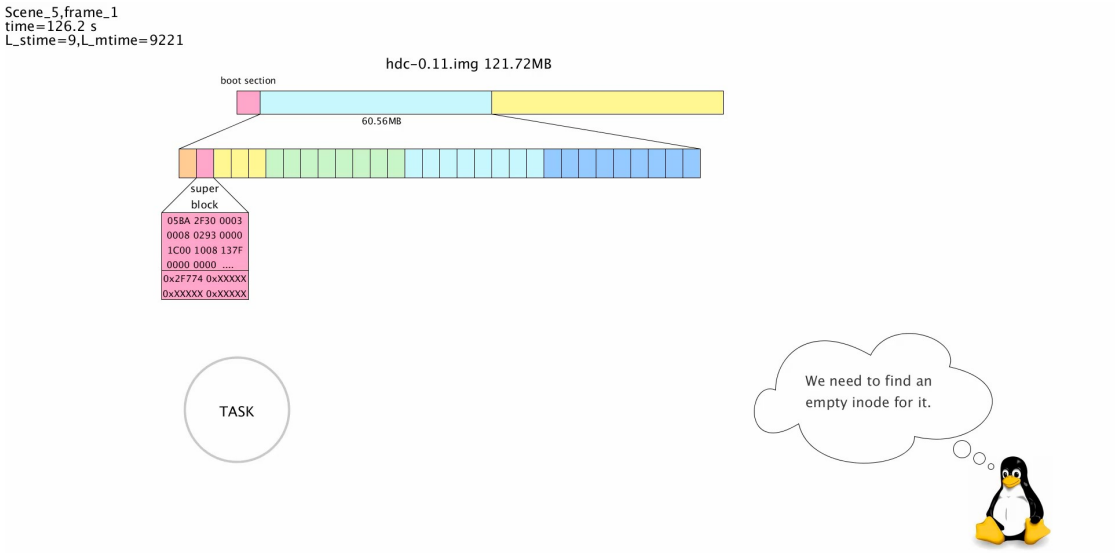


图 23: 查找超级块

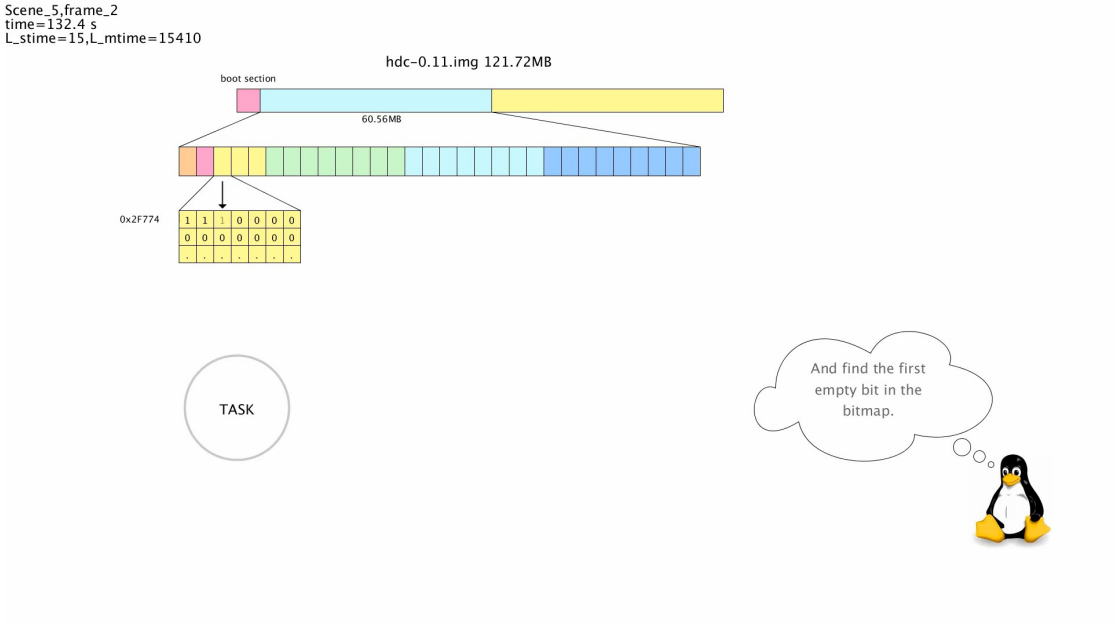


图 24: 查找i节点位图寻找空闲位

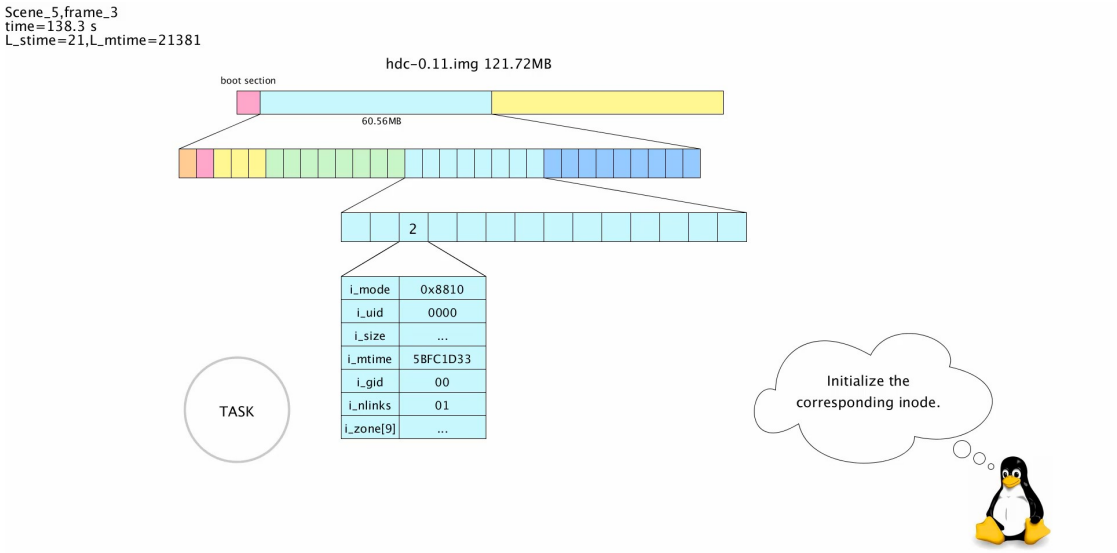


图 25: 初始化该inode节点

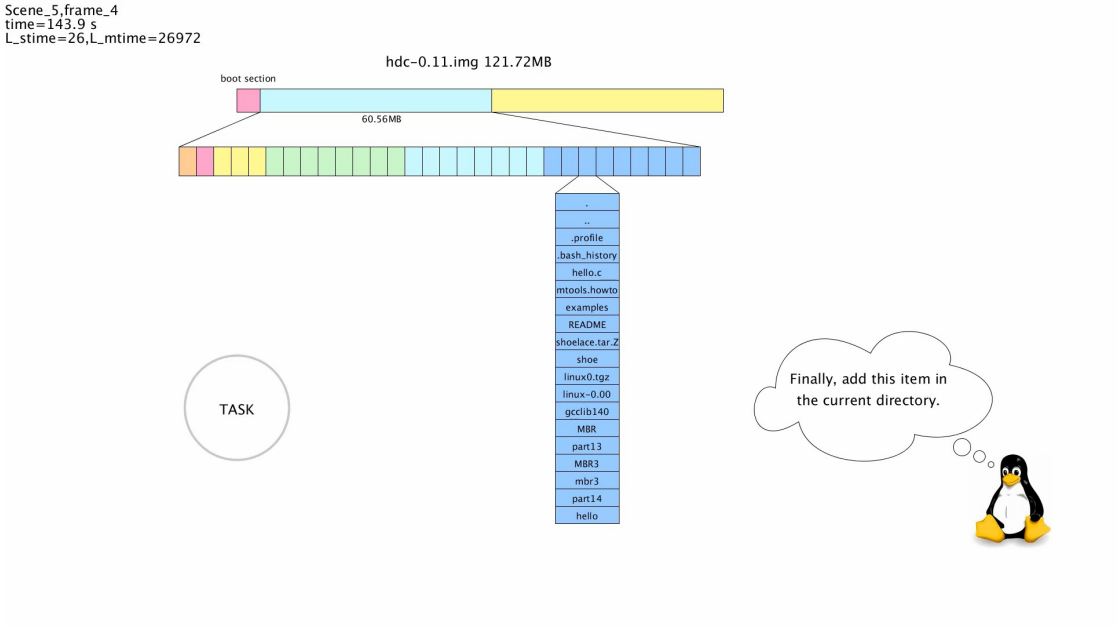


图 26: 在当前目录添加该文件

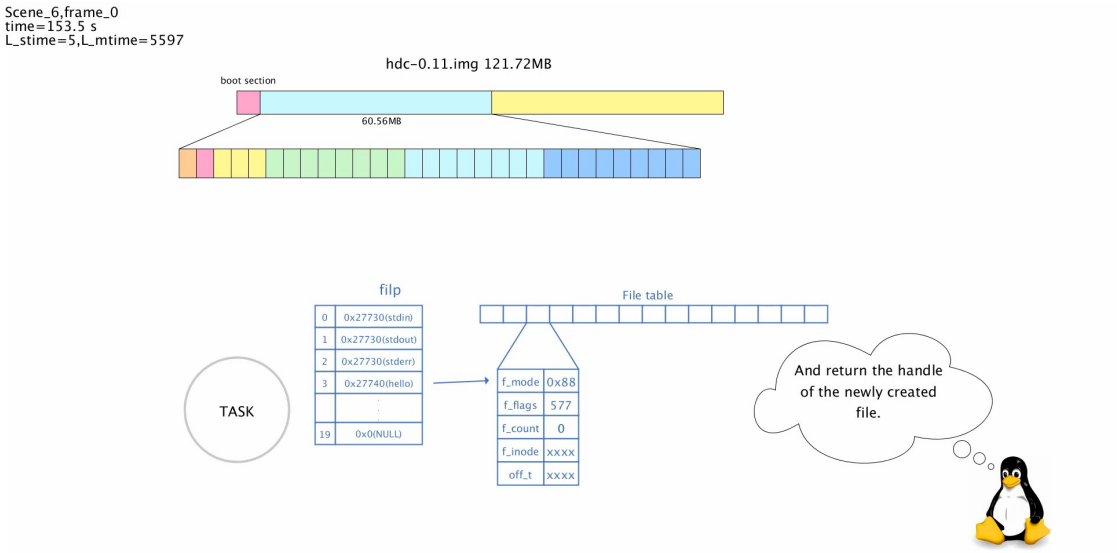


图 27: 设置当前进程的文件指针数组和内存文件表

Scene_7,frame_3
time=12.4 s
L_stime=12,L_mtime=12487

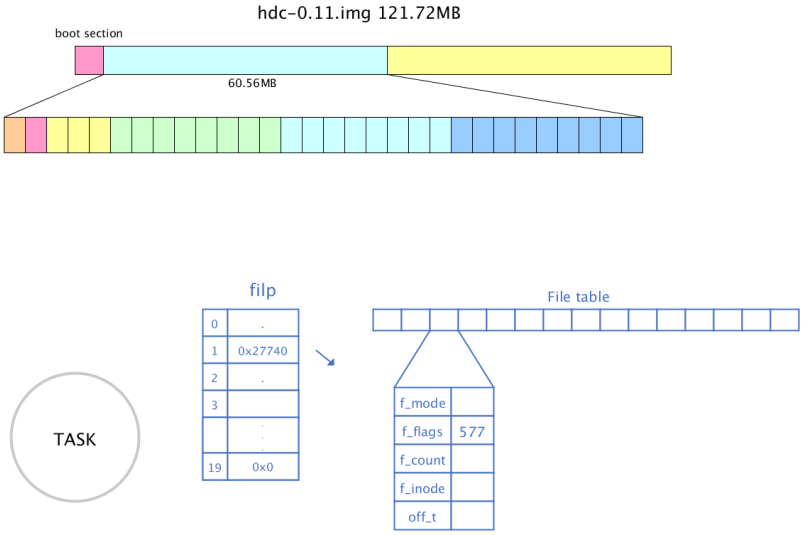


图 28: 查找文件结构表

Scene_7,frame_4
time=20.7 s
L_stime=20,L_mtime=20760

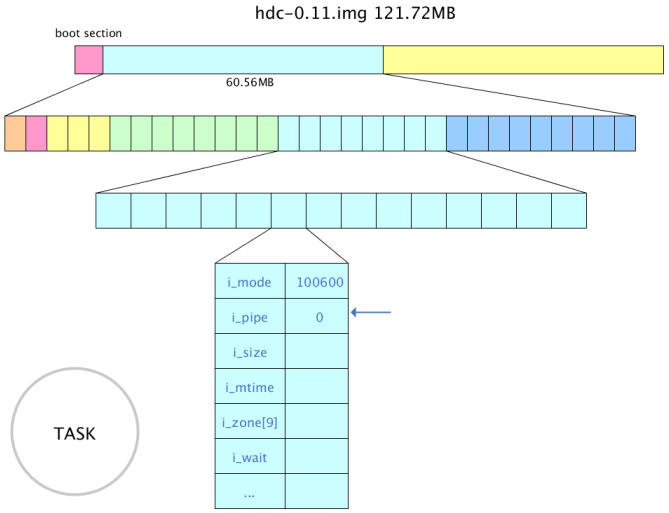


图 29: 查找文件类型

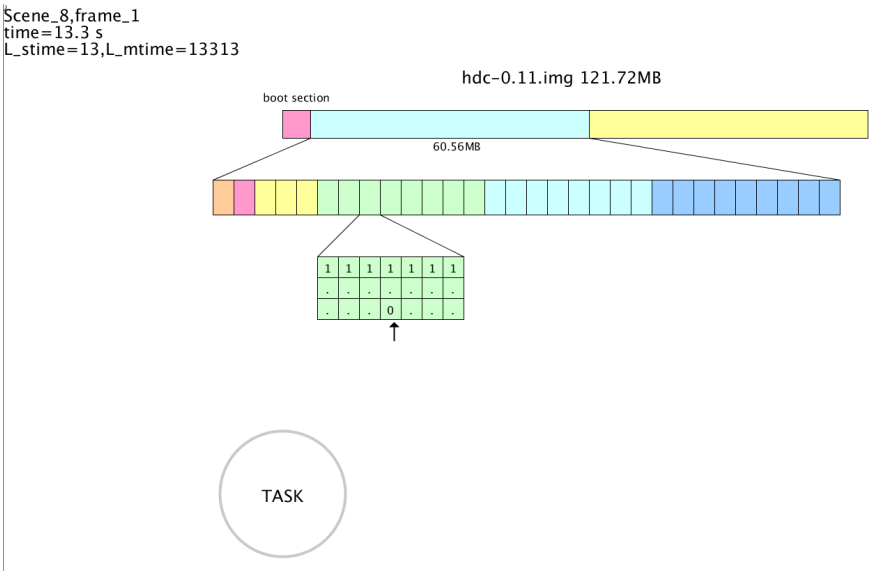


图 30: 找到超级块的逻辑位图中寻找第一个0值比特位

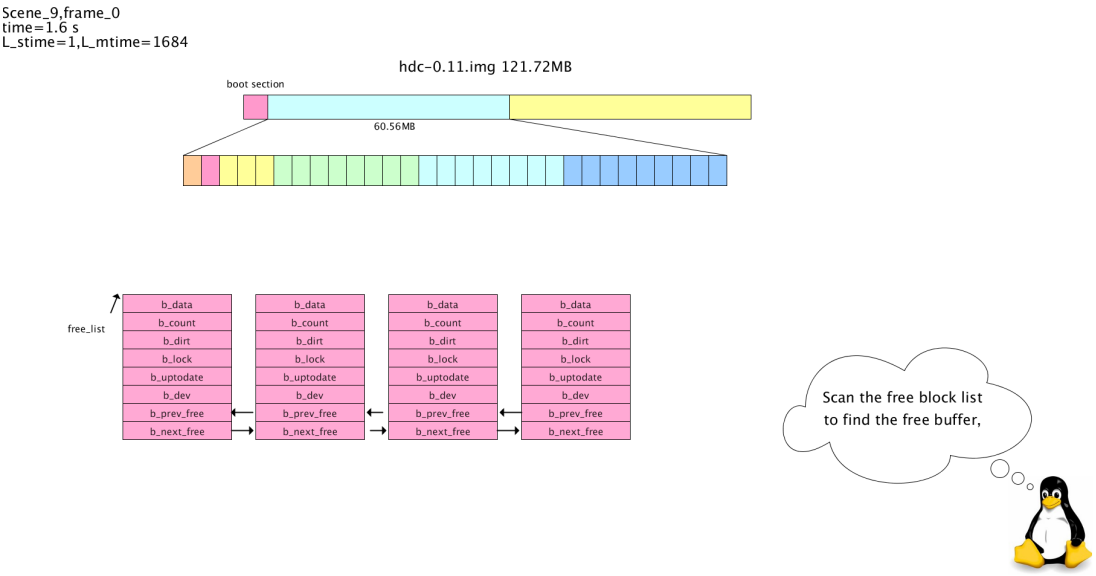


图 31: 获得一块空闲缓冲块

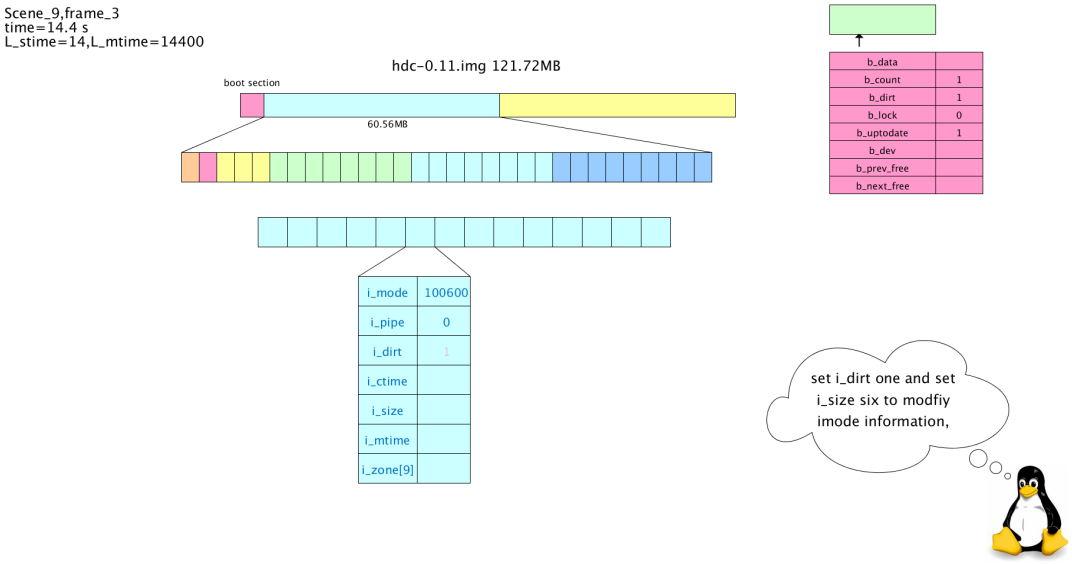


图 32: 设置inode中地标志位

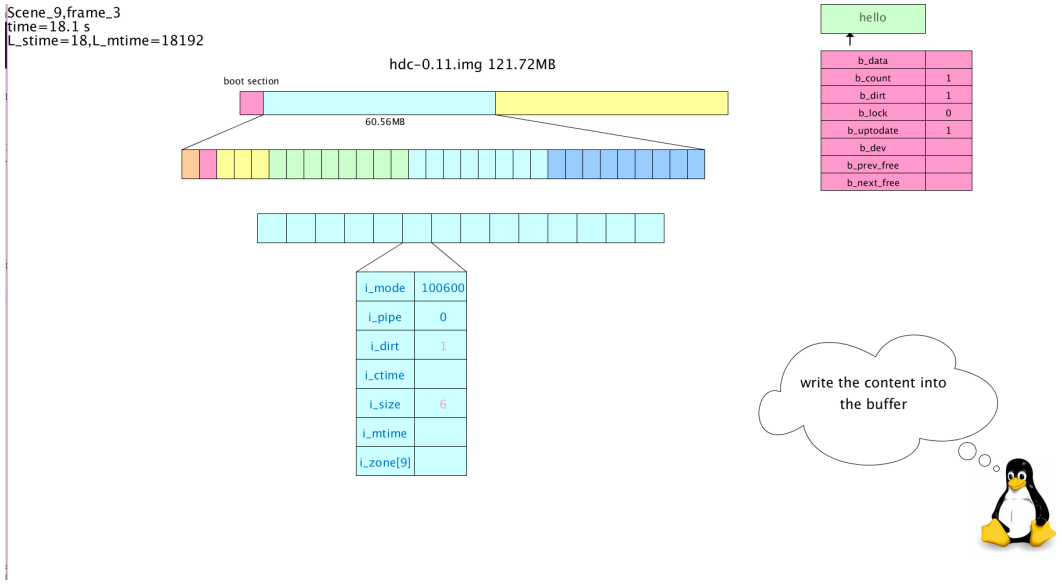


图 33: 将指定字符串写入缓冲区

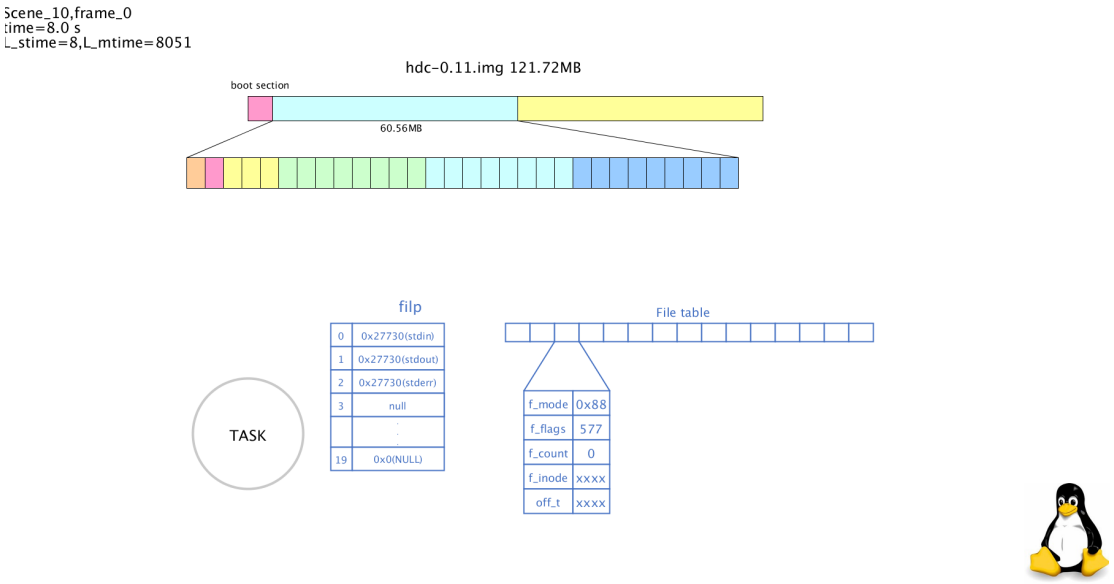


图 34: 文件句柄置为null

5 总结与收获

通过本次课程设计，有很多收获

1. 对Linux 0.11有了更深刻的理解, 加深了对操作系统课程内容知识的掌握
2. 理解linux操作系统的基本工作原理，深入理解了linu系统中对于文件地操作
3. 能够熟练地运用github，方便以后地多人合作
4. 在队友地帮助下，学习到了很多东西，尤其是在基于他给搭建好的平台下，减少了很多无用功
5. 学会使用processing
6. 学会使用visio
7. 学会录制视频等一系列技能吧

本次课程设计的全部实现见https://github.com/MrDuGitHub/OS
本次课程设计的进度情况见https://github.com/chengluyu/sdu-os-fall-2018
本次课程设计的课堂展示见https://github.com/MrDuGitHub/OS/tree/master/presentation