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: 201605130116

姓　　名: 杜洪超

同组成员: 王文嵩

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

一 实验内容与任务

1 实验内容

该实验以Linux 0.11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Linux 0.11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Linux 0.11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。

2 实验要求

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，可以根据自身的能力和兴趣在不同层次、规模、难度上完成本项实验。

- 可以探究系统某个模块的某个过程，如文件系统的读操作、键盘的输入、CPU的调度等。
- 可以选择组成大小不等的团队参与实验。
- 在可视化处理上，也可以做适当的简化。

二 实验流程与目的

1 实验安排

1. 实验前一学期

   在操作系统原理课程中，教师介绍Linux0.11源码结构及相关资料，并公布下一学期操作系统课程设计的任务,具备了自己分析源代码的基础。

2. 实验学期

   (1) 将Linux 0.11源代码分成基础模块和选读模块，必须分析基础模块，从选读模块中选择感兴趣的模块重点分析。（第1-4周，16课时）

   (2) 自由组合成团队，提出设计方案，每个团队说明感兴趣的系统运行过程。（第5-6周，8课时）

   (3) 讨论、评估设计方案（第7周，4课时）

   (4) 从感兴趣的系统运行过程中提取系统运行的状态数据，并生成系统运行日志。（第8-11周，16课时）

   (5) 根据日志实现运行过程的可视化。（第12-14周，12课时）

   (6) 演示运行结果（15-16周，8课时）

2 实验目的

1. 将操作系统原理与具体实现相结合，加深对理论知识的理解。

2. 掌握计算机系统的硬软件整体架构,培养全局观和系统能力。

3. 理解运行中的系统，锻炼解决实际问题的能力。

三 Linux 0.11简介

对于需要学习操作系统，或者希望对已经熟知的内核原理有更深刻的认识的人，阅读源代码是一种非常有效的方法。Linux作为一种被广泛使用的开源操作系统，就成了学习者的不二选择。而选择0.11版本的原因则更简单：麻雀虽小但五脏俱全，而且这种小带来了相当的友好性。Linux 0.11是第一个能稳定运行且实现了基本内核功能的版本，有着不到两万行代码，发布于1991年12月，发布时包括了以下内容：

```
bootimage.Z       - 具有美国键盘代码的压缩启动映像文件；
rootimage.Z       - 以 1200kB 压缩的根文件系统映像文件；
linux-0.11.tar.Z - 内核源代码文件。大小为 94KB，展开后也仅有 325KB；
as86.tar.Z        - Bruce Evans'二进制执行文件。是 16 位的汇编程序和装入程序；
INSTALL-0.11      - 更新过的安装信息文件。
```

我们主要分析的就是被打包成linux-0.11.tar.Z的源代码，就像Linux在一篇新闻组投稿中说的，要理解一个软件系统的真正运行机制，一定要“RTFSC - Reading The F**king Source Code!”。

四 基础模块分析与选读模块确定

本章节介绍本学期实验1-4周的工作，主要为阅读源代码，分析Linux0.11系统的运行过程以及确定选读模块。

1 Linux 0.11代码架构

解压linux-0.11.tar.Z,源代码目录结构如图4.1所示。一共14个子目录，总共包含102个代码文件。源代码经过编译链接后就生成了上一节中提到的bootimage.Z启动镜像文件，其中编译连接/组合结构如图4.2所示。

```
linux
├── boot            系统引导汇编程序
├── fs              文件系统
├── include         头文件(*.h)
│   ├── asm         与 CPU 体系结构相关的部分
│   ├── linux       Linux 内核专用部分
│   └── sys         系统数据结构部分
├── init            内核初始化程序
├── kernel          内核进程调度、信号处理、系统调用等程序
│   ├── blk_drv     块设备驱动程序
│   ├── chr_drv     字符设备驱动程序
│   └── math        数学协处理器仿真处理程序
├── lib             内核库函数
├── mm              内存管理程序
└── tools           生成内核 Image 文件的工具程序
```

图 4.1: 源代码目录结构

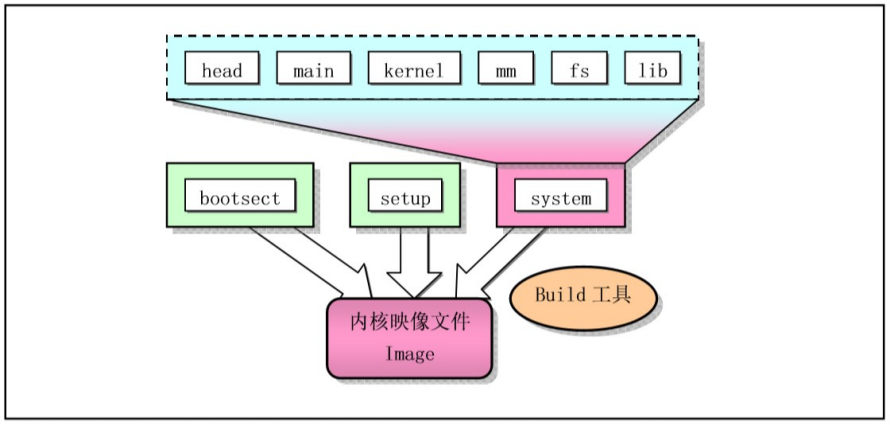


图 4.2: 编译连接/组合结构

2 基础模块分析

基础模块包括boot目录和init目录中的源代码，其中boot为启动引导目录，包含3个汇编文件：16位的bootsect.s和setup.s以及32位的head.s；主要功能是当计算机加电时引导内核启动，将内核代码加载到内存中，并做一些进入32位保护模式前的系统初始化工作。init目录只包含一个文件main.c，用于执行内核所有的初始化工作，然后以到用户模式创建新线程，并在控制台设备上运行shell程序。关于基础模块的分析详见https://github.com/MrDuGitHub/OS/blob/master/presentation/Linux%200.11.pdf

3 选读模块分析

经过对源代码进行简单分析后，确定了文件系统为选读模块，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4.3所示。因为对源代码的分析与设计方案有关，因此详细分析将放到下一章节讨论。

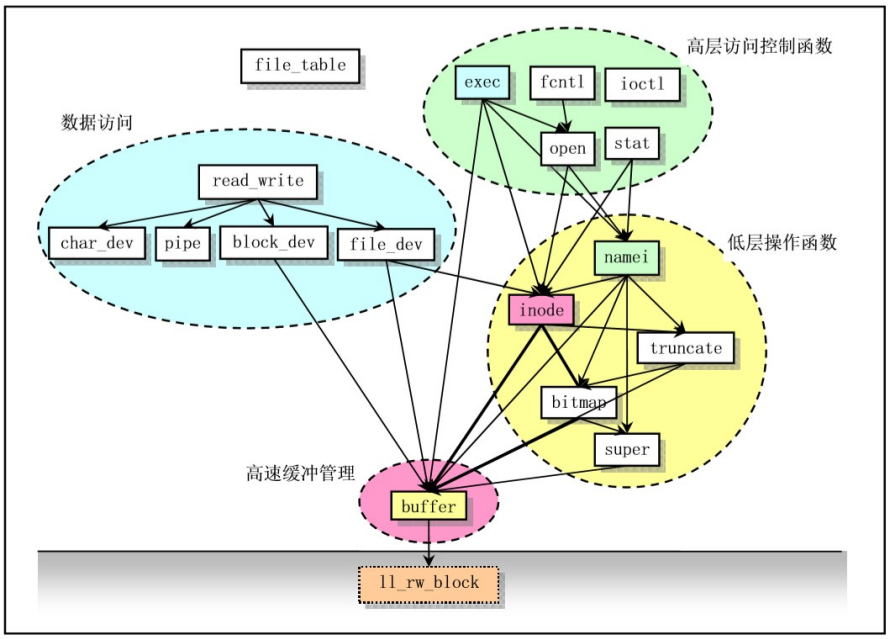


图 4.3: fs目录下源文件结构及引用关系

五 基础事件确定与图形化展示

本章节介绍本学期实验5-7周的工作，主要为设计展示方案，提取关键帧并以图形方式展示。从这一阶段开始小组分工合作完成，在本阶段我完成了实验平台的搭建与改进，数据提取的git框架，创建和打开文件的方案设计与图形化。

1 实验平台搭建

阅读完源代码之后，我们确定了小组并开始设计方案，为了验证方案的真实和可行性，以及考虑到后面的数据提取工作，我们搭建了Linux 0.11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5.1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。

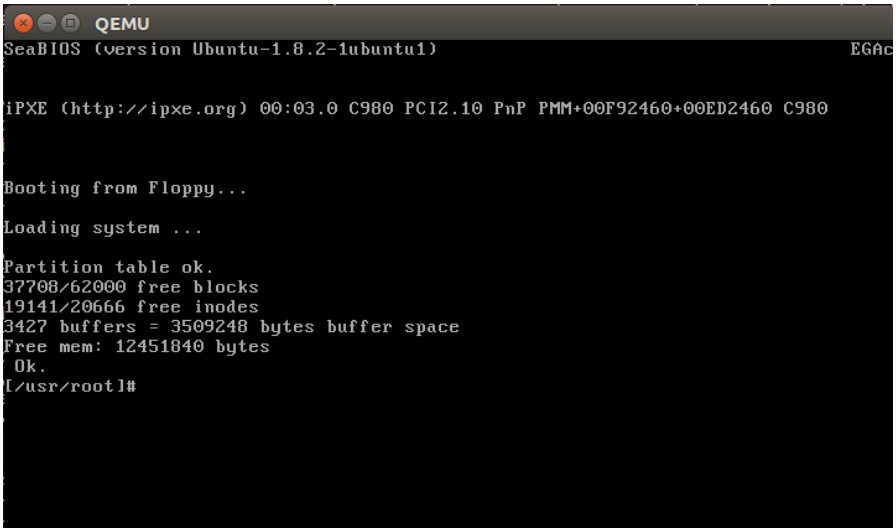


图 5.1: 实验平台

2 方案设计

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，打开文件等，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，我负责打开和创建文件的方案设计。静态分析只涉及数据结构，无需方案设计.

3 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5.2：

   (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

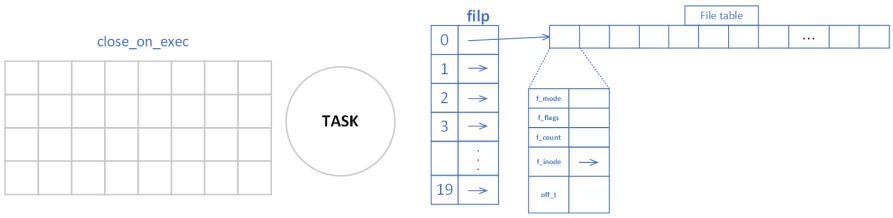


图 5.2: 创建文件的基础设置

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。

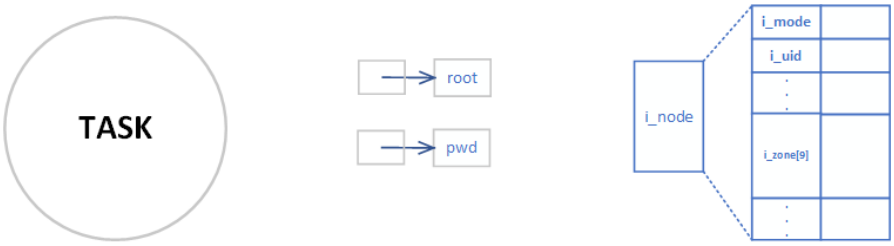


图 5.3: 当前进程的根inode节点和当前目录inode节点

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。

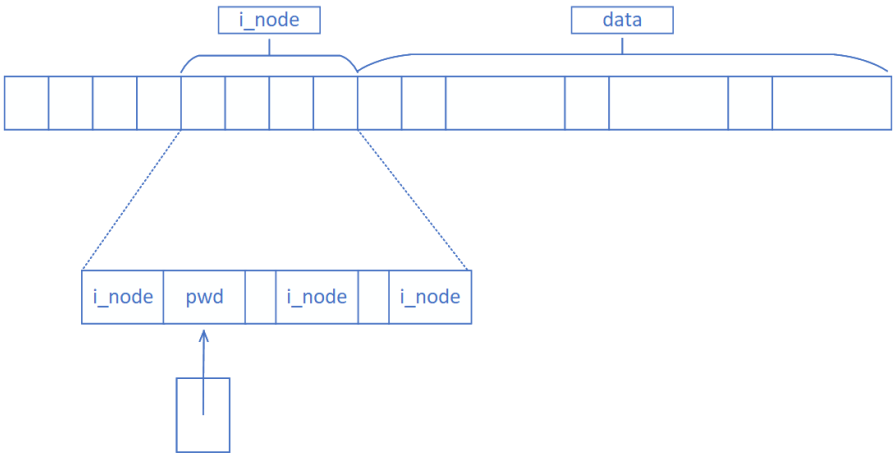


图 5.4: 找到pwd

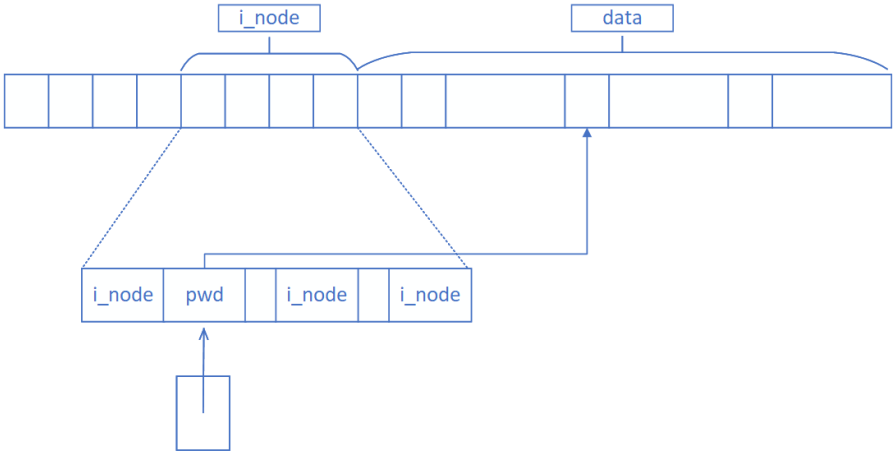


图 5.5: 找到pwd所指向的数据区

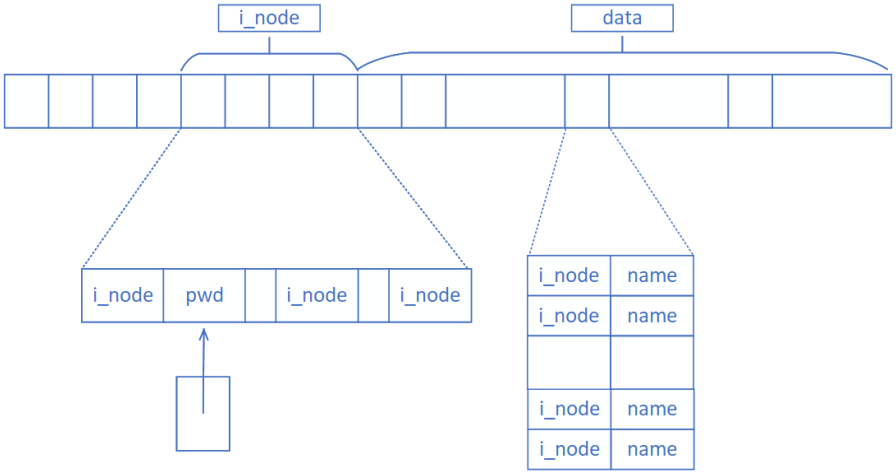


图 5.6: 从数据区中查找文件

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:

```
//open.c
int sys_creat(const char * pathname, int mode)
    sys_open(pathname, O_CREAT | O_TRUNC, mode);

/* Open a file */
//open.c
    int sys_open(const char * filename,int flag,int mode)
    // Find an empty file struction pointer for current progcess, return the index as handle
    // Set the close_on_exec
    // Find an empty file structioon in the file table
//namei.c
    open_namei(filename,flag,mode,&inode));
    // open file,set the inode,return wrong code if failed
        dir_namei(pathname,&namelen,&basename);
        // Find the inode of the file specified by filename
            get_dir(pathname);
            // Find the inode of the file specified by filename
                inode = current->pwd;
                return pwd;
        // Find the file in this dir
        find_entry(&dir,basename,namelen,&de);
        // find the target and return buffer
        // If this is a create operation, apply for a new inode and add it under the dir
            new_inode(dir->i_dev); // bitmap.c
//bitmap.c
            struct m_inode * new_inode(int dev)
                inode=get_empty_inode()
//inode.c
                struct m_inode * get_empty_inode(void)
//bitmap.c
                sb = get_super(dev)
//super.c
                struct super_block * get_super(int dev)
//namei.c
            add_entry(dir,basename,namelen,&de);
            return;
       // If not, the file already exists, return the point of inode
```

4 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。

六 数据提取与筛选

本章节介绍本学期实验8-11周的工作，主要为根据设计的方案进行相关数据的提取。在本阶段我完成了静态数据的提取，设计了动态数据格式,并完成了创建和打开文件的数据提取。

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6.1：

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 6.1: 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6.2：

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 6.2: 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

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6.3：

\	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 6.3: 动态数据JSON字段详解

图 6.1: JSON数据举例

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

4 统计数据

根据收集到的动态数据，统计信息如下：

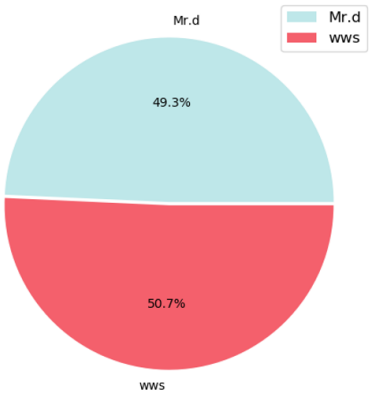


图 6.2: provider分布

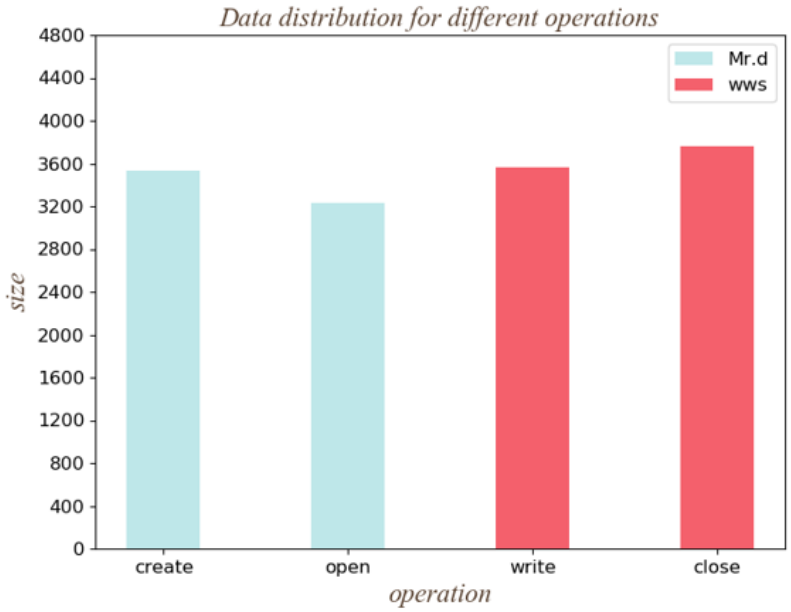


图 6.3: operation分布

七 可视化与展示

可视化是本学期课程设计的最后一部分，但最靠后的工作却并不最简单。可视化按计划是12到15周的工作，但实际上花费的时间不止于此。在这一部分我的工作主要包括:设计了基于Processing的可视化场景设计框架，实现了一系列的工具模块，并在此基础上进行了静态展示、动态展示中创建和打开文件的部分以及动画的开始和结束阶段的可视化实现。

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/

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。

3 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7.1和图7.2分别截取了二者的一个画面。

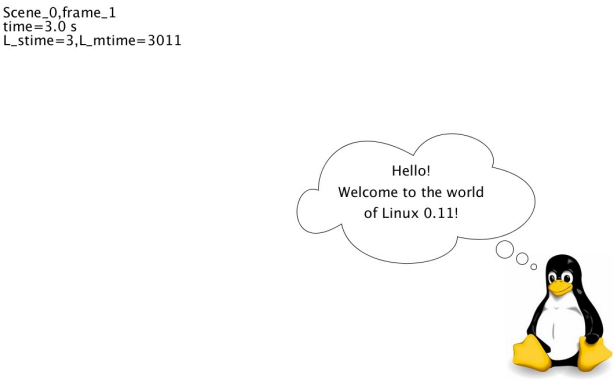


图 7.1: 欢迎语

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

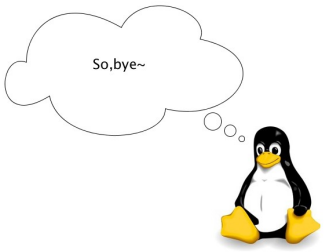
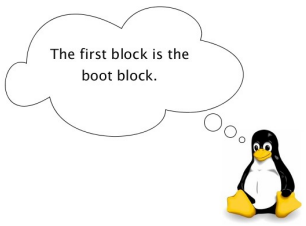


图 7.2: 结束语

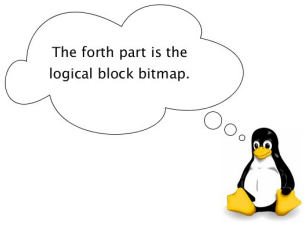
4 静态展示

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

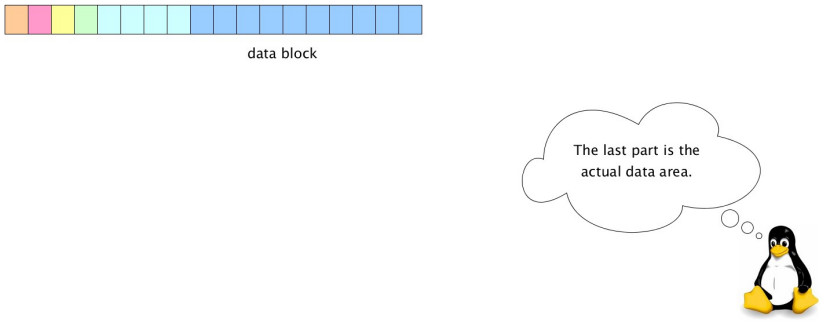
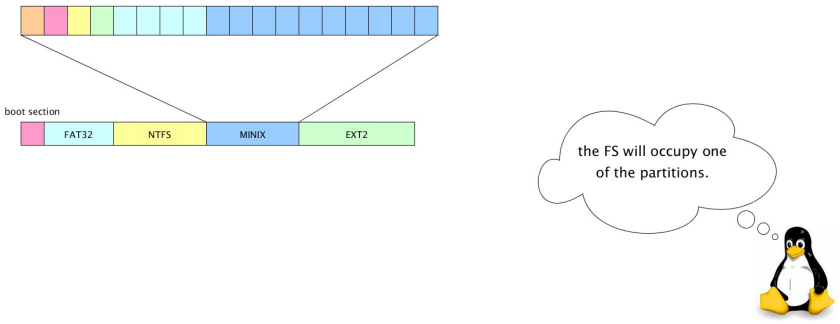
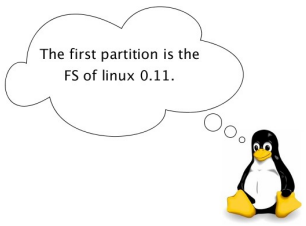


图 7.3: MINIX文件系统静态展示

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

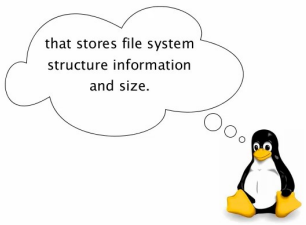


Scene_3,frame_0
time=7.9 s
L_stime=8,L_mtime=8000



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



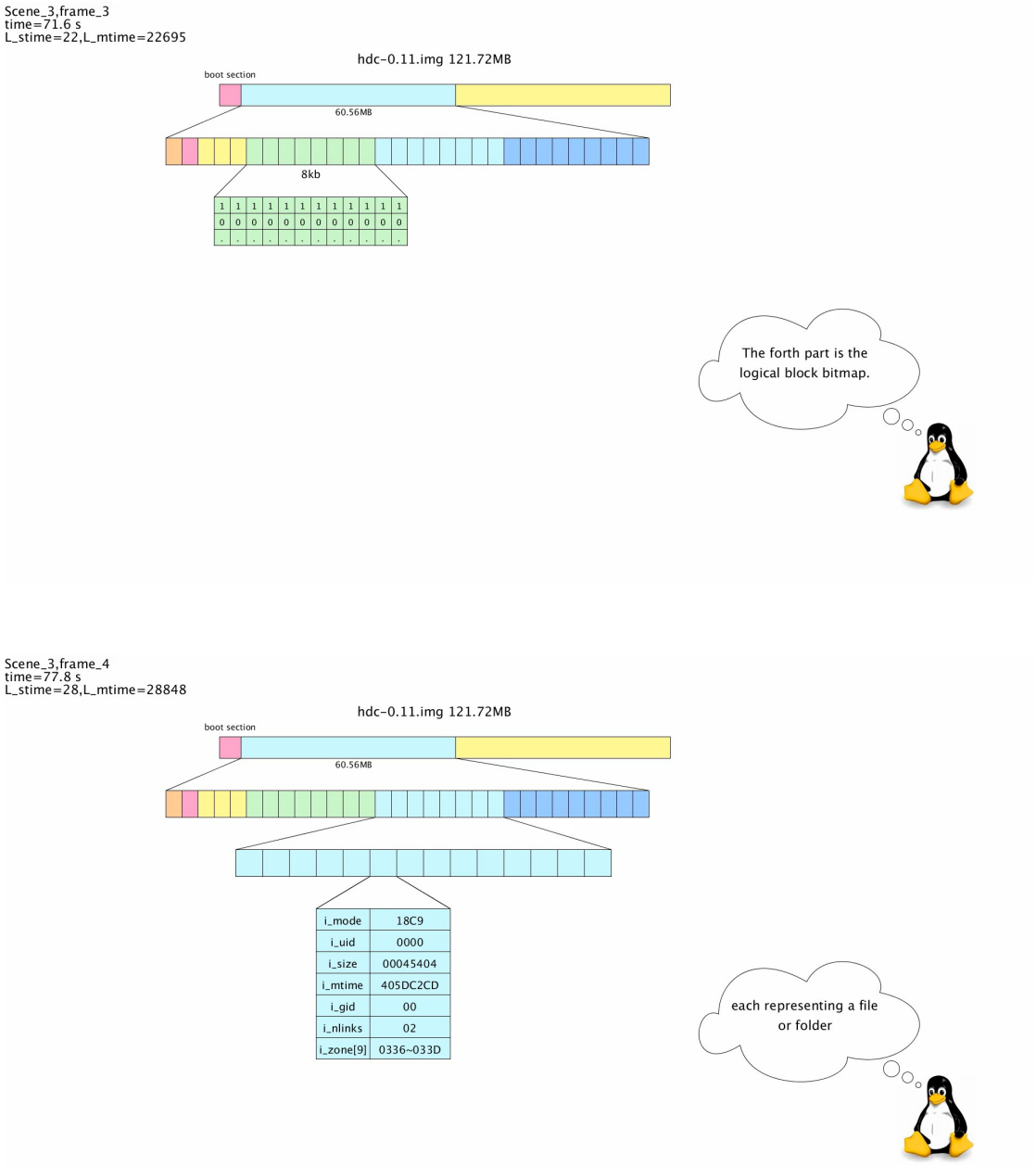


图 7.4: Linux 0.11文件系统静态展示

5 动态展示

静态展示共占用了3个场景，场景4到场景6，总时长78秒。

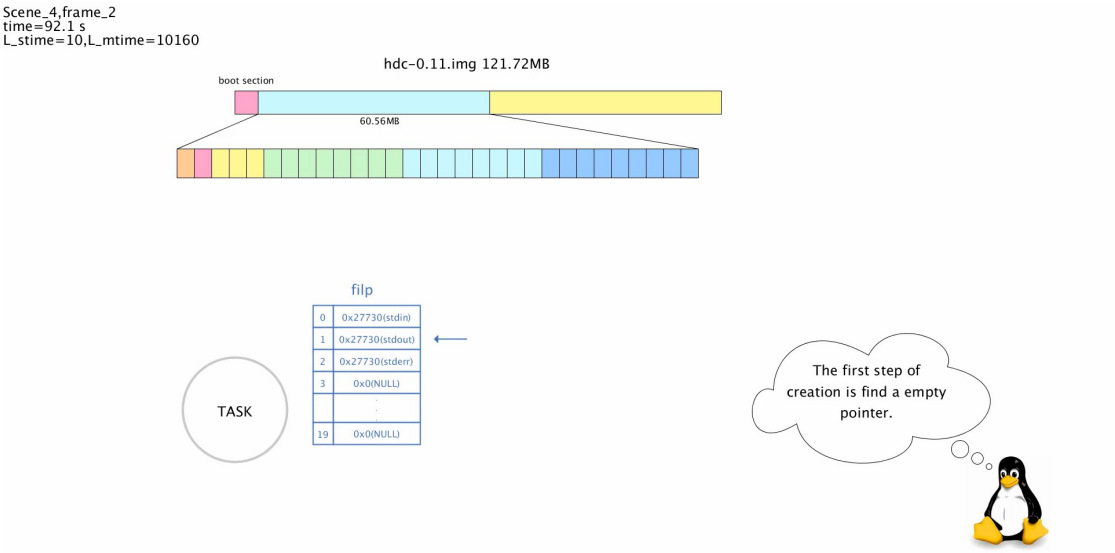


图 7.5: 查找文件指针数组

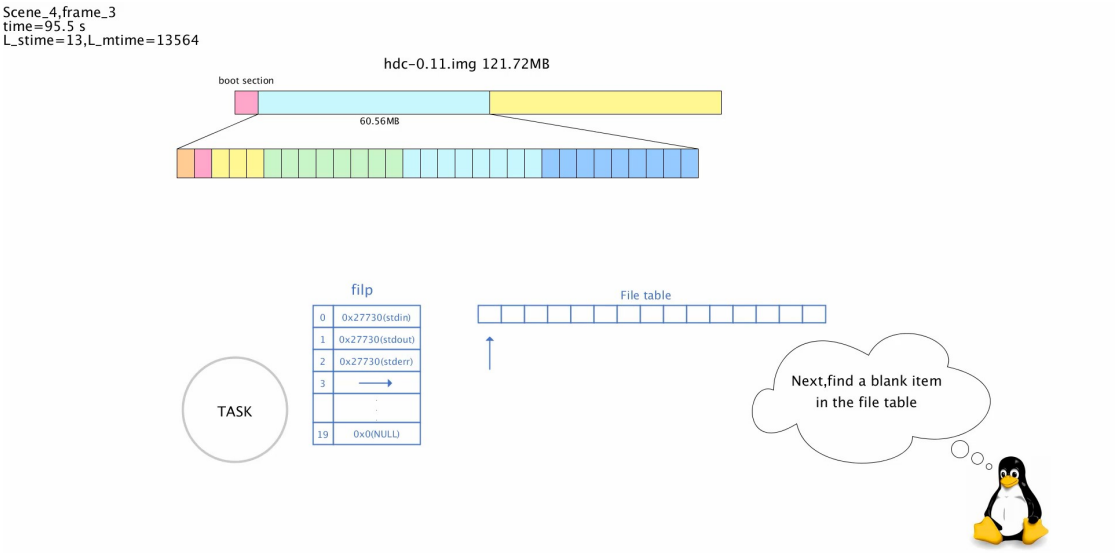


图 7.6: 查找文件表

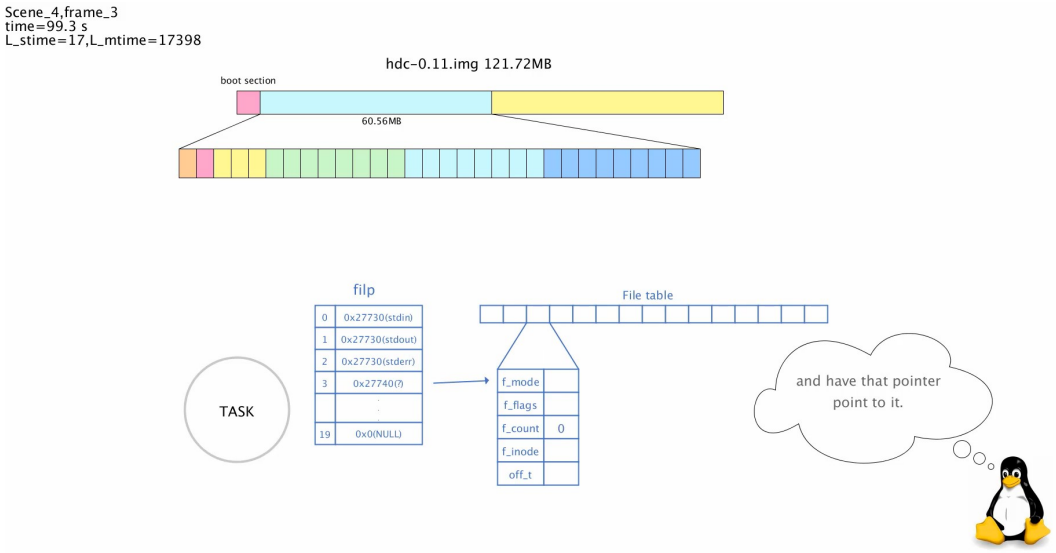


图 7.7: 设置文件指针数组

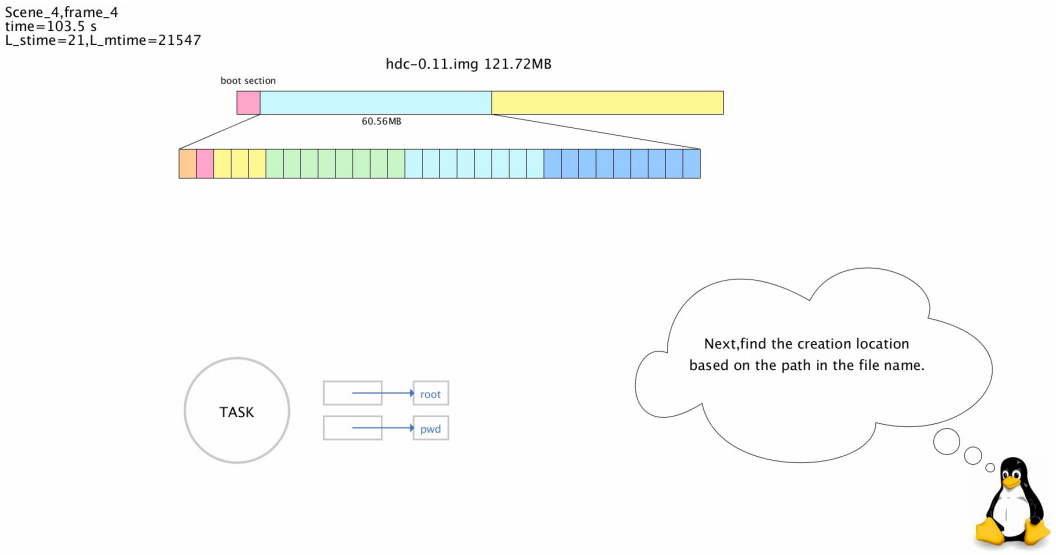


图 7.8: 查找文件的直接目录

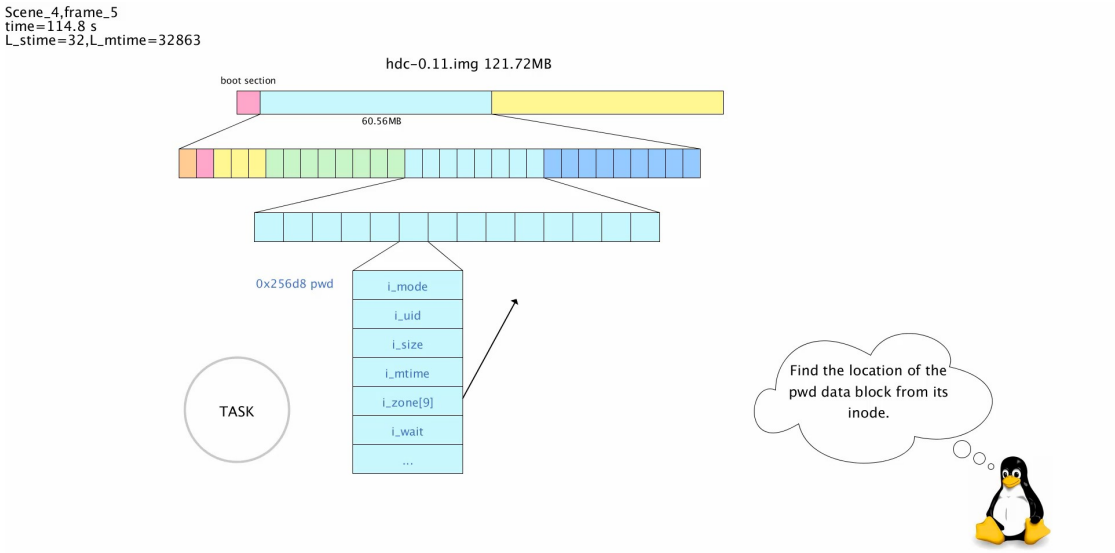


图 7.9: 寻找pwd的数据区

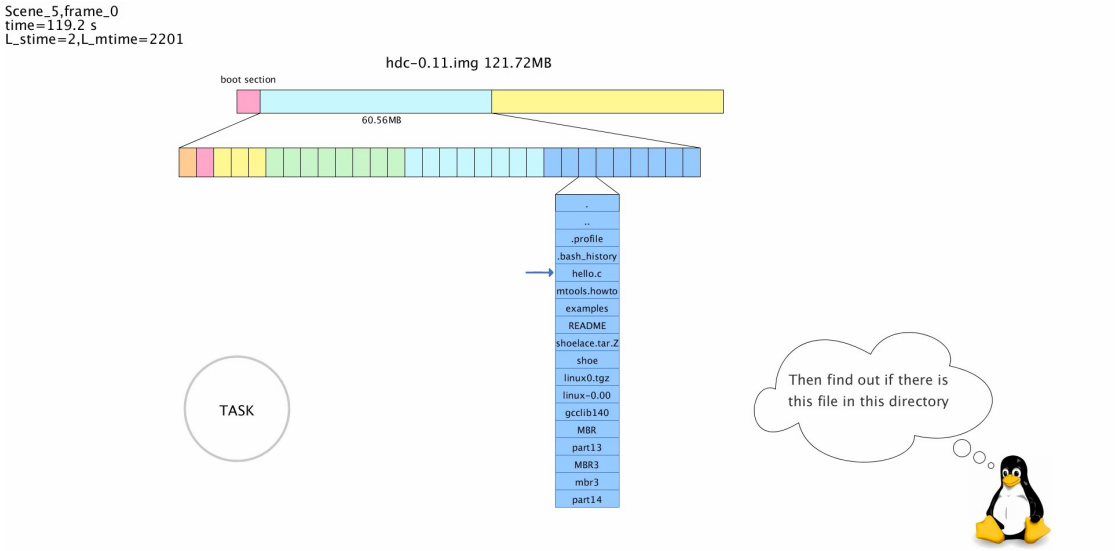


图 7.10: 在当前目录查找文件

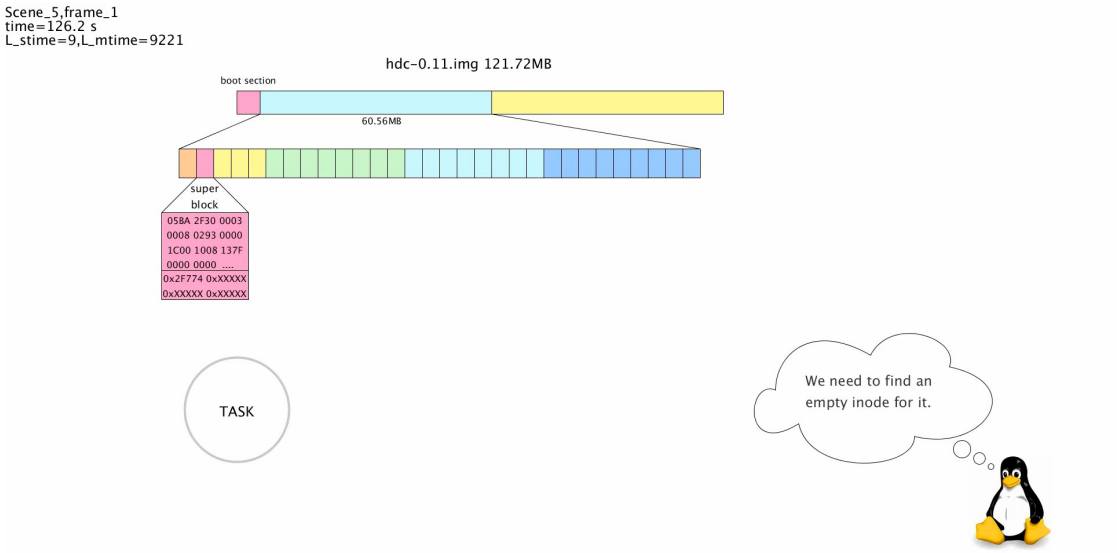


图 7.11: 查找超级块

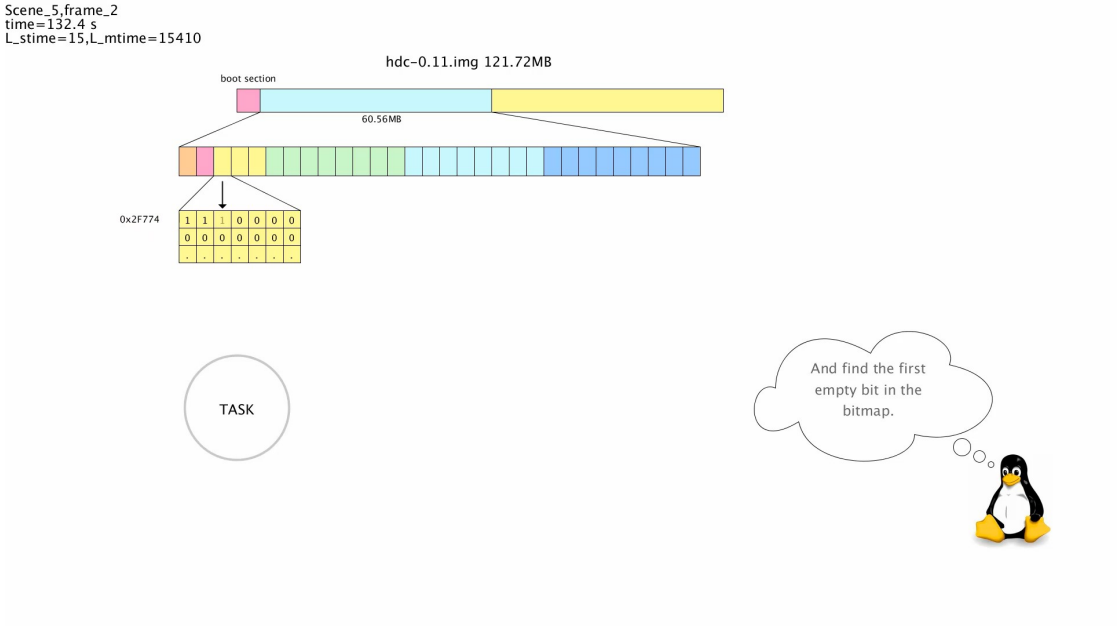


图 7.12: 查找i节点位图寻找空闲位

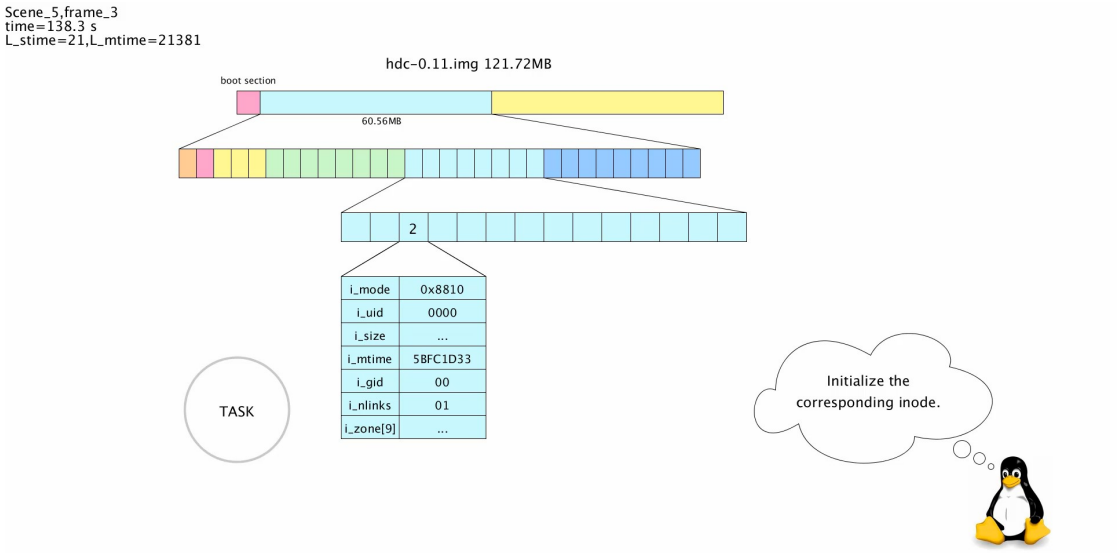


图 7.13: 初始化该inode节点

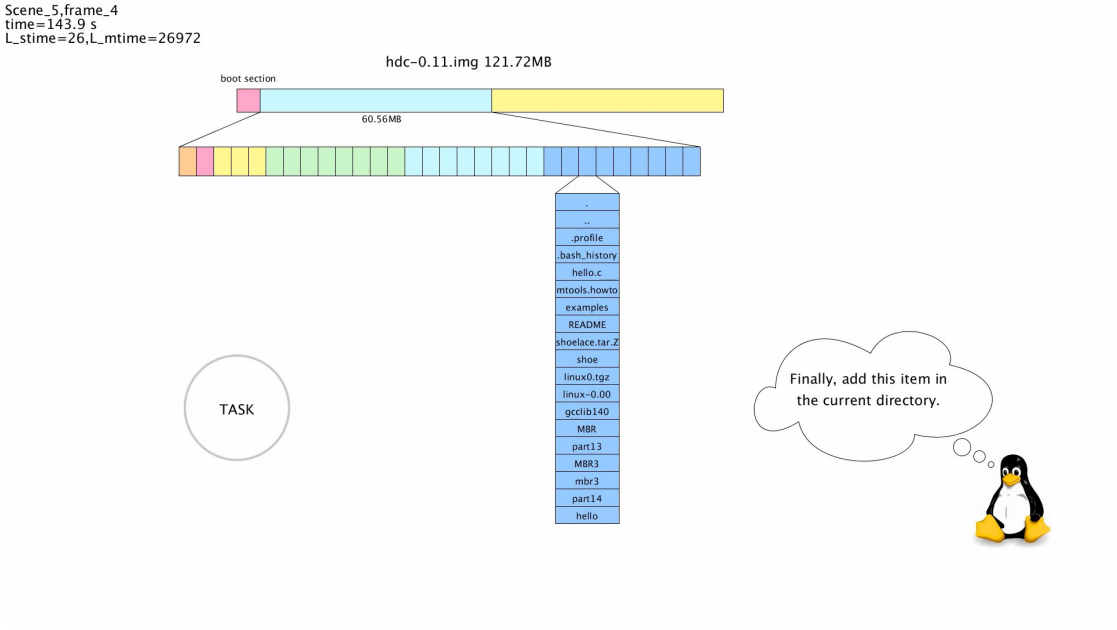


图 7.14: 在当前目录添加该文件

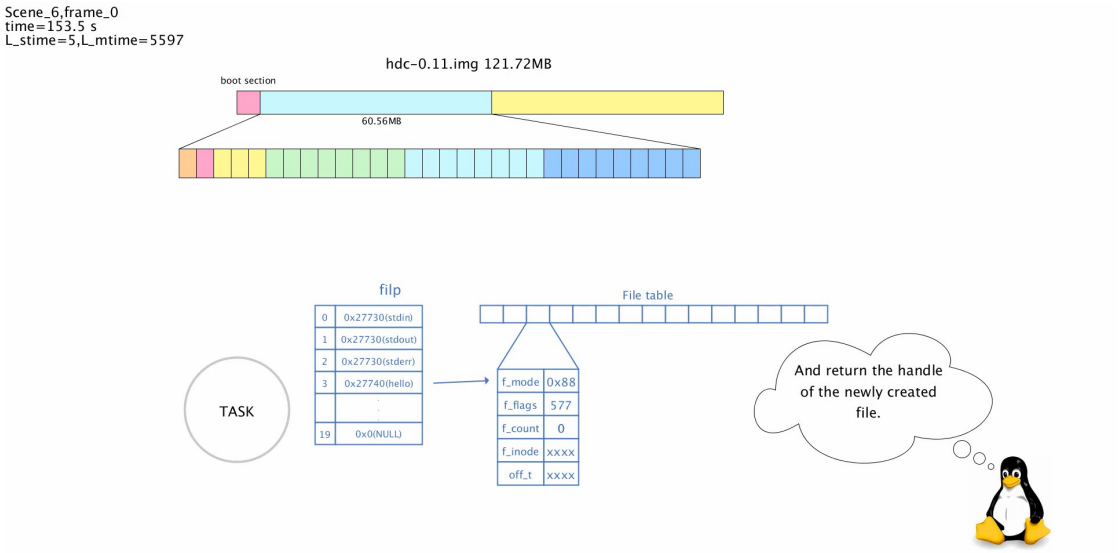


图 7.15: 设置当前进程的文件指针数组和内存文件表

八 实验总结

以上就是本学期操作系统课程设计的全部内容，期间还进行了四次课堂展示与报告。这次课程设计让我对操作系统的原理有了更深刻的认识，掌握了很多技能，提高了阅读代码以及解决问题的能力。

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/MrDuGitHub/sdu-os-fall-2018

本次课程设计的课堂展示见https://github.com/MrDuGitHub/OS/tree/master/presentation