M5: File Recovery (frecov)

截止日期

Soft Deadline: 6月2日23:59:59

收到的作业

引子

一个Makefile引发的血案：在某个月黑风高的夜晚，修改了代码，以为 OBJ_DIR 这个变量不再使用，遂删除。结果没有料到，Makefile里还留了这么一手：

```
clean:
    rm -rf ... $(OBJ_DIR)/* ...
```

在另一个月黑风高的夜晚， `make clean` 之后， `rm -rf /*` 开始了.....

背景

每个人(包括jyy和yzh)都有过在脑抽的时候误删过文件的经历。那可真是相当惨痛的经历——类似的还有把重要数据保存在优盘上，然后优盘损坏了.....

你误删过重要的文件吗?

自从用上了git、养成按时git push的习惯之后，重要数据的损失就降低到最小了——至少可以保证在任何时候，如果瞬间某台电脑彻底损坏，损失的工作量不会超过几个小时。以下是贴在实验室门上的忠告。每当git push成功以后，心里都会觉得很踏实。

另一方面，在《操作系统》课程中我们也学习到了文件系统在实现文件/目录的删除操作时，仅仅是从文件系统的数据结构中抹去文件相关的信息，并不会实际将每一个保存数据的块删除。

文件粉碎

文件系统中总有一些你不想让别人看到的文件——你不仅想在文件系统中删除它，还希望别人即便拿到你的磁盘做数据恢复，也不能恢复出文件的内容：明文存放了各大网站密码的文件、不能见人的照片……

因此，有些系统软件提供了“文件粉碎”的功能。文件粉碎的一个实现方法是用覆盖写的方式把文件的数据清除(即找到磁盘上所有存储了该文件数据的块，并且向块内写入随机数据)，然后再删除。

对于HDD上的文件系统，通常这样的“文件粉碎”就是相当安全的。但随着存储技术的发展，底层的文件系统(甚至是闪存硬件)可能采用copy-on-right方式保存数据，从而在硬件上保留文件的拷贝。虽然恢复它们需要很大的代价，但安全的“文件粉碎”变得更加困难。使用加密的文件系统是更好的办法。

为了验证上述猜想，我们试着在磁盘上写入一个很大的文件，然后把它删除：

```
$ dd if=/dev/zero of=a.img count=4194304 bs=4096
...
17179869184 bytes (17 GB, 16 GiB) copied, 14.2317 s, 1.2 GB/s
$ time bash -c 'rm a.img && sync'
...
0.694 total
```

写入文件花去了14s，而删除 + 全局文件系统同步(等待输入到达磁盘)仅用去不到1s。因此，我们知道文件系统中的文件即便被删除，文件的内容只要不被覆盖，则仍然可能从磁盘中读出。在这个实验中，我们模拟如下场景 (但做出相当的简化):

- 你无意中捡到了一个数码相机的SD卡，但插入计算机后发现其中没有任何数据(FAT32格式，已被格式化过)。
- 你忽然对其中的数据产生了很大的好奇，希望恢复其中的数据。于是你打算设计一个算法，能够尽可能地从一个磁盘镜像(数据区)中推导出可能的文件系统。

实验描述

获取实验代码与提交

本学期的所有代码(minilab, OSlab)都在同一个目录中完成。请参考代码获取与提交 (OS2019_Code)。

在你原先的os-workbench基础上，执行

```
git pull origin M5
```

将本实验的框架代码下载到本地。实验相关的文件/数据在 `frecov` 下。

实现 `frecov` 工具，给定一个FAT32文件系统镜像，请你尽可能地从文件系统中恢复出BMP格式的完整图片文件。命令行工具使用:

frecov FILE

输入一个FAT32文件系统镜像文件FILE，试图恢复尽可能多的图片文件。每恢复一张图片文件(完整的文件，包含BMP头和所有数据)，调用系统中的 `sha1sum` 命令获得它的校验和，在标准输出中逐行输出图片文件的校验和以及你恢复出的文件名。只有校验和与文件名**都恢复正确且一致**，才能获得这个图片的分数，例如:

```
206e31b7ad95b2b980d4d5afebc57c56f3663b6e  huangtu-1.bmp
620e1f842d7dbd1a60fd011fb30dac61d644e37a  huangtu-2.bmp
211beccd79721f296f93285a81bd5e9c21b1f8f5  fuli.bmp
...
```

我们如何测试你的代码?

我们会使用不超过64MB的镜像来测试你的文件，然后：

- 仅有你输出的前1,000行会被我们处理，因此如果你不能确信恢复的结果，你可能需要对恢复出的图片进行排序；
- 助教会在i7-6700配置的台式机(本地Linux)上运行你的程序，超过60s的程序将被提前终止(因此如果你的程序运行得很慢，你应该在每恢复出一个图片后flush你的stdout)，建议使用内存不要超过1GB。

你只需要恢复出一部分图片即可获得成绩——有相当多的图片是直接连续存储在磁盘上的，因此这些图片你没有道理恢复不正确。

一些约定

这个实验会不会太难了?

这个实验并不是让大家真正编写一个优秀的undelete工具。如果做到的话，你就可以开公司挣大钱了！easyrecovery就是一个相当大的赢家。实验的目的是为了让大家体验一下：

- 写代码分析FAT文件系统；
- 写代码分析BMP文件；
- 想一个(也许是非常简单)的算法恢复出文件系统中的BMP文件。也许能work的算法比你想象更简单。

我们对这个问题做了相当的简化，首先，我们保证我们用 `mkfs.fat` 的如下参数创建文件系统镜像：

```
$ cat /dev/zero | head -c $(( 1024 * 1024 * 64 )) > filesystem.img # 创
$ mkfs.fat -v -F 32 -S 512 filesystem.img # 在空文件上创建FAT32文件系统
mkfs.fat 4.1 (2017-01-24)
fs.img has 64 heads and 32 sectors per track,
hidden sectors 0×0000;
logical sector size is 512,
using 0×f8 media descriptor, with 131072 sectors;
drive number 0×80;
filesystem has 2 32-bit FATs and 1 sector per cluster.
FAT size is 1009 sectors, and provides 129022 clusters.
There are 32 reserved sectors.
Volume ID is 60fb68c4, no volume label.
$ mount filesystem.img /mnt/ # 挂载文件系统
$ mkdir /mnt/DCIM # 创建DCIM目录
```

上述是64MB镜像的例子，是一个“干净”的FAT文件系统镜像，包含一个根目录下的“DCIM”目录，然后我们会进行很多次如下的文件操作(图片文件分辨率、大小可能不同，但都**保证是真实世界中有意义的图片**)：

- 向DCIM中复制图片文件
- 删除DCIM中的图片文件
- ……
- (反复操作之后，文件系统中可能存在一些碎片化的情况)

最后，我们会重新执行

```
mkfs.fat -v -F 32 -S 512 filesystem.img
```

模拟一次格式化的过程，之后得到待恢复的文件系统镜像。测试程序中包含不同workload生成的不同大小的镜像——jyy才不会告诉你，有一个测试用例就是一个文件系统里有一个顺序存储的bmp文件，能恢复出来就给分。

关于BMP文件，我们是使用convert (ImageMagick)工具生成的24位图片，例如使用 `file` 查看 `fuli.bmp` 得到的结果：

```
fuli.bmp: PC bitmap, Windows 98/2000 and newer format, 403 x 350 x 24
```

参考镜像

为了方便上手，我们为大家提供了一个参考镜像，请点这里下载 (/static/wiki/os/2019/files/filesystem.zip)。镜像中包含很多黄色图片(真的是“黄色”的图片，纯粹的黄色、每一张图片的每一个像素颜色都相同)。这样，尽管参考镜像已经被格式化过(因此挂载看不到任何文件)，并且一个文件的数据可能散落在磁盘的多个地方，但你很容易把它们在磁盘中的位置标记出来：

- 扫描磁盘，找到并解析BMP的文件头；
- 试着读取BMP文件的前几个像素，它和BMP的文件头位于同一个数据块中；
- 在磁盘中寻找所有与这个像素相同的数据块，即恢复出一个BMP文件。

参考镜像DCIM目录中的文件大约是这样的：

```
DCIM/
  huangtu-1.bmp
  huangtu-23.bmp
  ...
  fuli.bmp
```

除了“黄色”图片外，还有一张福利图片(非黄色图片)。你如果文件恢复的成功，就能正确看到图片了。那么聪明的你，看到什么图片了呢？？？

sha1sum

在这个实验中，你需要调用外部工具sha1sum计算校验和。为此，你需要像sperf实验中类似的那样，和sha1sum建立管道连接(或者创建临时文件，但不推荐)，然后获取sha1sum的输出结果，和你恢复出的文件名一起打印出来：

```
$ echo "Hello, World" | sha1sum
4ab299c8ad6ed14f31923dd94f8b5f5cb89dfb54  -
```

sha1sum是一种计算文件“指纹”的算法。指纹(fingerprint)也称为校验和(checksum)，它是一个单向的hash function H，能把一个很大空间里的字符串映射到很小的空间中(计算$H(x)$)，并且目前而言，给定一个fingerprint后的字符串t，人类很难计算出一个字符串 x 满足 $H(x) = t$。指纹能在不传输整个文件的前提下，快速比较两个文件是否相等。除了校验文件的完整性之外，指纹还可以用来做数据的去重——例如大家在即时通信软件中传送文件(例如QQ)时，服务器会用校验和检查是否已经存在文件的副本，如果是就可以立即结束传输。

实验指南

BMP文件格式

为了完成实验，你首先需要了解BMP文件的格式。BMP文件的头部包含了文件的重要信息，但简单起见，你只需要考虑之前提到的24位存储的BMP文件。

为了锻炼大家独立思考、查找资料的能力，关于BMP文件的细节请STFW。

FAT32文件系统

你还需要了解FAT32的文件和目录存储。用C代码解析FAT32文件系统时，我们建议大家使用mmap系统调用，将镜像直接映射到进程的地址空间中，然后使用内存读取进行访问。

为了锻炼大家独立思考、查找资料的能力，关于FAT32文件系统的细节请STFW。

提示：文件名怎么来?

必须文件名和恢复的文件内容都正确且一致，才能获得一张图片的分数。

DCIM是一个**目录**。在课堂和教课书中都已经明确了“目录文件”的概念——用普通的数据文件存储目录的信息。FAT32文件系统目录文件中包含诸多目录项，其中包含了图片的文件名。

数据恢复

数据对每个人来说都是非常宝贵的。在数码相机等设备中，FAT32这样的文件系统依然广泛使用。因此，对数码相机相片的误删相对是很容易恢复的——如果你不小心删除了一整天拍摄的照片，当你恢复出它们的时候，一定感受到文件系统设计者的不杀之恩。

当然，能恢复数据也带来了滥用的可能。试想你的手机被盗或被作为二手处理掉之后，能够从手机的内部存储/存储卡中恢复出先前的文件——重要信息、聊天记录、照片……幸运的是，随着文件系统加密(在文件系统层写入数据前即执行数据的加密)的普及，大家越来越不用担心这些问题了。

加密的文件系统

如何实现文件系统的加密？原理上，我们需要一个秘钥key，然后利用一些算法(例如RSA, AES等)计算出加密后的密文存储在磁盘上，之后利用秘钥可以进行解密。

但是秘钥的安全性如何保证？如果直接保存在磁盘上，被人搜索出来不就暴露了么？有很多种办法：

- 在手机这样的移动设备中，通常会预留一部分存储在操作系统都无法访问的数据区(例如大家可能听说过ARM的TrustZone，类似于Ring -1的特权级)，只有厂商内置的代码可以访问，秘钥存储在此处，厂商提供API进行加密等操作；
- 把密码作为秘钥的一部分。由于验证登录并不需要保存明文密码，因此明文密码不会被保存在磁盘上，但在操作系统运行时可以作为内存的一部分，如此一来没有密码就无法实现数据恢复了。

为什么每次手机重启之后都需要输入密码?

关于文件系统加密，还有我们身边的一个案例：虽然我们总是用指纹/人脸解锁手机，但每当手机重启后，系统都会要求输入一次密码才能登录。这是因为加密存储会把密码作为秘钥的一部分，而且操作系统不会在文件系统中保存明文密码——明文密码仅保存在内存中，断电后就消失。因此系统的第一次登陆必须输入密码，是在保护你数据的安全性！