北京科技大学实验报告

学院： 计算机与通信工程学院　　专业： 计算机科学与技术　　班级： 计 184

姓名： 王丹琳　　学号： 41824179　　实验日期： 2021 年 4 月 1 日

实验名称：

实验二 用 LINUX 命令行操作处理日常业务

实验目的：

结合课程讲授第 2-5 讲的复习和理解，熟悉 LINUX 文件系统 ext2、ext3、ext4 的存储结构和优缺点，掌握有关 LINUX 命令的使用,实现用户/组管理；设计完成定位文件与目录；浏览文件与目录；搜索文件内容；操作文件和目录等操作处理任务，熟练使用 vim 等文本编辑工具，为日常的文件操作维护打下扎实的基础。

实验仪器：

PC 机一台：ThinkPad T480

实验环境：

VMware 虚拟机

Linux Ubuntu 64 位

实验原理：

Linux，全称 GNU/Linux，是一种免费使用和自由传播的类 UNIX 操作系统，其内核由林纳斯·本纳第克特·托瓦兹于 1991 年 10 月 5 日首次发布，它主要受到 Minix 和 Unix 思想的启发，是一个基于 POSIX 的多用户、多任务、支持多线程和多 CPU 的操作系统。它能运行主要的 Unix 工具软件、应用程序和网络协议。它支持 32 位和 64 位硬件。Linux 继承了 Unix 以网络为核心的设计思想，是一个性能稳定的多用户网络操作系统。

实验内容与步骤：

1）定位文件与目录

2）浏览文件与目录

3）精准搜索、编辑文件内容

4）熟练操作文件和目录的权限设置

5）使用命令进行文件压缩和解压缩

6）学习理解 vim 等文本编辑器基本功能，能熟练操作 Linux 中的文本文件。

7）收集整理 LLNUX 文件系统 ext2、ext3、ext4 的存储结构和优缺点，试着写出其对应的 C 语言描述结构体程序代码段。

实验数据及处理：

1）定位文件与目录

Linux find 命令用来在指定目录下查找文件

```
loongson@loongson-VirtualBox:~$ locate --help
Usage: locate [OPTION]... [PATTERN]...
Search for entries in a mlocate database.

  -A, --all              only print entries that match all patterns
  -b, --basename         match only the base name of path names
  -c, --count            only print number of found entries
  -d, --database DBPATH  use DBPATH instead of default database (which is
                         /var/lib/mlocate/mlocate.db)
  -e, --existing         only print entries for currently existing files
  -L, --follow           follow trailing symbolic links when checking file
                         existence (default)
  -h, --help             print this help
  -i, --ignore-case      ignore case distinctions when matching patterns
  -l, --limit, -n LIMIT  limit output (or counting) to LIMIT entries
  -m, --mmap             ignored, for backward compatibility
  -P, --nofollow, -H     don't follow trailing symbolic links when checking
 file
                         existence
  -0, --null             separate entries with NUL on output
  -S, --statistics       don't search for entries, print statistics about e
ach
                         used database
  -q, --quiet            report no error messages about reading databases
  -r, --regexp REGEXP    search for basic regexp REGEXP instead of patterns
      --regex            patterns are extended regexps
  -s, --stdio            ignored, for backward compatibility
  -V, --version          print version information
  -w, --wholename        match whole path name (default)
```

查找 d.txt 文件

```
loongson@loongson-VirtualBox:~/桌面$ find d.txt
d.txt
```

2）浏览文件与目录

ls 列出目录内容

```
loongson@loongson-VirtualBox:~/桌面$ ls --help
用法：ls [选项]... [文件]...
List information about the FILEs (the current directory by default).
Sort entries alphabetically if none of -cftuvSUX nor --sort is specified.

必选参数对长短选项同时适用。
  -a, --all                  不隐藏任何以. 开始的项目
  -A, --almost-all           列出除. 及.. 以外的任何项目
      --author               与-l 同时使用时列出每个文件的作者
  -b, --escape               以八进制溢出序列表示不可打印的字符
      --block-size=SIZE      scale sizes by SIZE before printing them; e.g.
,
                               '--block-size=M' prints sizes in units of
                             1,048,576 bytes; see SIZE format below
  -B, --ignore-backups       do not list implied entries ending with ~
  -c                         with -lt: sort by, and show, ctime (time of la
st
                               modification of file status information);
                             with -l: show ctime and sort by name;
                             otherwise: sort by ctime, newest first
  -C                         list entries by columns
      --color[=WHEN]         colorize the output; WHEN can be 'always' (def
ault
                               if omitted), 'auto', or 'never'; more info b
elow
  -d, --directory            list directories themselves, not their content
s
  -D, --dired                generate output designed for Emacs' dired mode
  -f                         do not sort, enable -aU, disable -ls --color
```

```
loongson@loongson-VirtualBox:~/桌面$ ls -all
总用量 28
drwxr-xr-x  4 loongson loongson 4096 3月  23 19:57 .
drwxr-xr-x 24 loongson loongson 4096 3月  31 18:12 ..
-rwxr-xr-x  1 loongson loongson    4 3月  15 20:09 D1.txt
-rw-rw-rw-  1 loongson loongson    4 3月  15 20:11 d.txt
drwx------  4 loongson loongson 4096 2月  28 19:15 func
drwx------  5 loongson loongson 4096 3月   4 14:11 func2
-rw-rw-r--  1 loongson loongson   69 3月  15 20:01 test.s
loongson@loongson-VirtualBox:~/桌面$
```

3）精准搜索、编辑文件内容

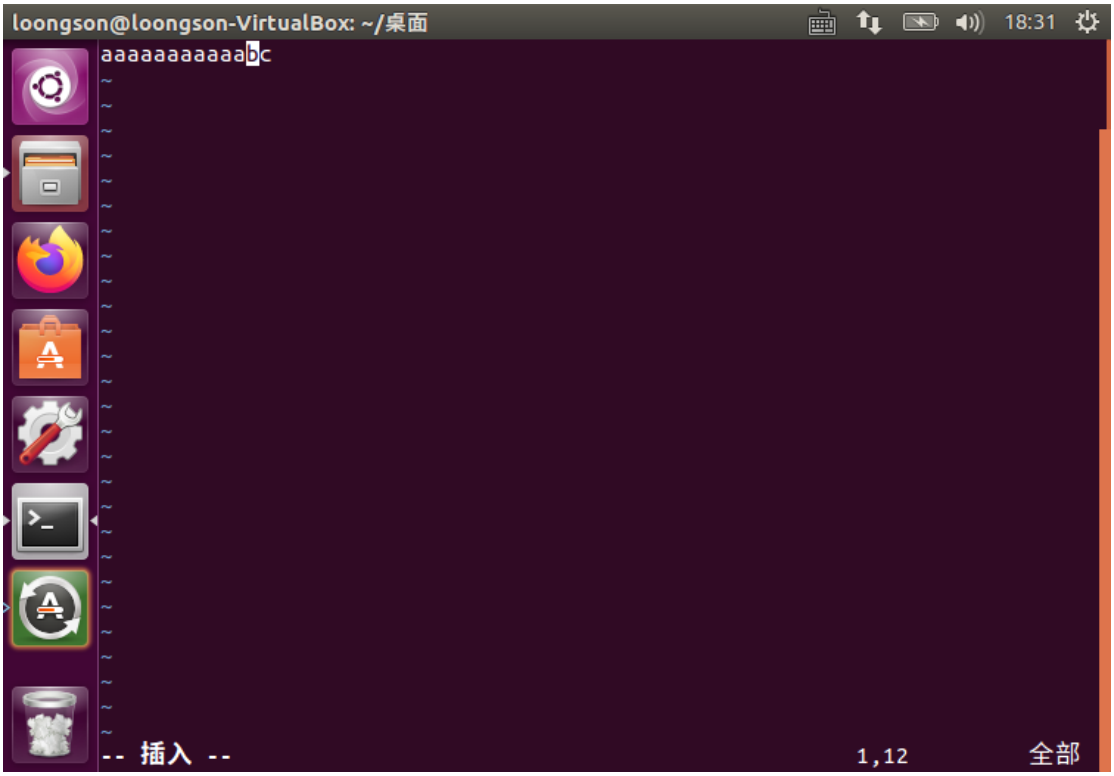
```
loongson@loongson-VirtualBox:~/桌面$ grep --help
用法: grep [选项]... PATTERN [FILE]...
在每个 FILE 或是标准输入中查找 PATTERN。
默认的 PATTERN 是一个基本正则表达式(缩写为 BRE)。
例如: grep -i 'hello world' menu.h main.c

正则表达式选择与解释:
  -E, --extended-regexp     PATTERN 是一个可扩展的正则表达式(缩写为 ERE)
  -F, --fixed-strings       PATTERN 是一组由断行符分隔的字符串。
  -G, --basic-regexp        PATTERN 是一个基本正则表达式(缩写为 BRE)
  -P, --perl-regexp         PATTERN 是一个 Perl 正则表达式
  -e, --regexp=PATTERN      用 PATTERN 来进行匹配操作
  -f, --file=FILE           从 FILE 中取得 PATTERN
  -i, --ignore-case         忽略大小写
  -w, --word-regexp         强制 PATTERN 仅完全匹配字词
  -x, --line-regexp         强制 PATTERN 仅完全匹配一行
  -z, --null-data           一个 0 字节的数据行，但不是空行

杂项:
  -s, --no-messages         不显示错误信息
  -v, --invert-match        选中不匹配的行
  -V, --version             显示版本信息并退出
      --help                显示此帮助并退出

输出控制:
  -m, --max-count=NUM       NUM 次匹配后停止
  -b, --byte-offset         输出的同时打印字节偏移
  -n, --line-number         输出的同时打印行号
      --line-buffered       每行输出清空
```

查找文件内容 a

```
loongson@loongson-VirtualBox:~/桌面$ grep A d.txt
loongson@loongson-VirtualBox:~/桌面$ grep a d.txt
abc
loongson@loongson-VirtualBox:~/桌面$
```

编辑文件内容

```
loongson@loongson-VirtualBox:~/桌面$ vim d.txt
```



4）熟练操作文件和目录的权限设置

使用字符形式修改文件权限

```
loongson@loongson-VirtualBox:~/桌面$ chmod a+x D1.txt
loongson@loongson-VirtualBox:~/桌面$ ls-l
ls-l: 未找到命令
loongson@loongson-VirtualBox:~/桌面$ ls -l
总用量 20
-rwxrwxr-x 1 loongson loongson    4 3月  15 20:09 D1.txt
-rw-rw-rw- 1 loongson loongson    4 3月  15 20:11 d.txt
drwx------ 4 loongson loongson 4096 2月  28 19:15 func
drwx------ 5 loongson loongson 4096 3月   4 14:11 func2
-rw-rw-r-- 1 loongson loongson   69 3月  15 20:01 test.s
loongson@loongson-VirtualBox:~/桌面$
```

使用数字形式修改文件权限

```
loongson@loongson-VirtualBox:~/桌面$ chmod 755 D1.txt
loongson@loongson-VirtualBox:~/桌面$ ls -l
总用量 20
-rwxr-xr-x 1 loongson loongson    4 3月  15 20:09 D1.txt
-rw-rw-rw- 1 loongson loongson    4 3月  15 20:11 d.txt
drwx------ 4 loongson loongson 4096 2月  28 19:15 func
drwx------ 5 loongson loongson 4096 3月   4 14:11 func2
-rw-rw-r-- 1 loongson loongson   69 3月  15 20:01 test.s
loongson@loongson-VirtualBox:~/桌面$
```

5）使用命令进行文件压缩和解压缩

tar 命令

```
loongson@loongson-VirtualBox:~/桌面$ tar --help
用法: tar [选项...] [FILE]...
GNU 'tar' saves many files together into a single tape or disk archive, and
 can
restore individual files from the archive.

Examples:
  tar -cf archive.tar foo bar  # Create archive.tar from files foo and bar.
  tar -tvf archive.tar         # List all files in archive.tar verbosely.
  tar -xf archive.tar          # Extract all files from archive.tar.

 主操作模式:

  -A, --catenate, --concatenate   追加 tar 文件至归档
  -c, --create                创建一个新归档
  -d, --diff, --compare       找出归档和文件系统的差异
      --delete                从归档(非磁带！)中删除
  -r, --append                追加文件至归档结尾
  -t, --list                  列出归档内容
      --test-label            测试归档卷标并退出
  -u, --update                仅追加比归档中副本更新的文件
  -x, --extract, --get        从归档中解出文件

 操作修饰符:

      --check-device          当创建增量归档时检查设备号(默认)
  -g, --listed-incremental=FILE   处理新式的 GNU 格式的增量备份
  -G, --incremental           处理老式的 GNU 格式的增量备份
      --ignore-failed-read
```

将 d.txt 进行文件压缩

```
loongson@loongson-VirtualBox:~/桌面$ tar czvf d_yasuo.tar d.txt
d.txt
loongson@loongson-VirtualBox:~/桌面$ ls
D1.txt  d.txt  d_yasuo.tar  func  func2  test.s
```

将 d_yasuo.tar 进行解压

```
loongson@loongson-VirtualBox:~/桌面$ tar zxvf d_yasuo.tar
d.txt
```

6）学习理解 vim 等文本编辑器基本功能，能熟练操作 Linux 中的文本文件。

基本上 vi/vim 共分为三种模式，分别是命令模式（Command mode），输入模式（Insert mode）和底线命令模式（Last line mode）。

进入 vim 编辑器：vim filename 打开/新建一个文件

命令模式 (命令模式)

- 保存退出

ZZ	保存并退出

- 光标定位

vim filename +n	打开文件，并将光标定位到第 n 行
vim filename +	打开文件，并将光标定位到最后一行

gg	定位到第一行
ngg	定位到第 n 行
G	定位到最后一行

- 内容处理

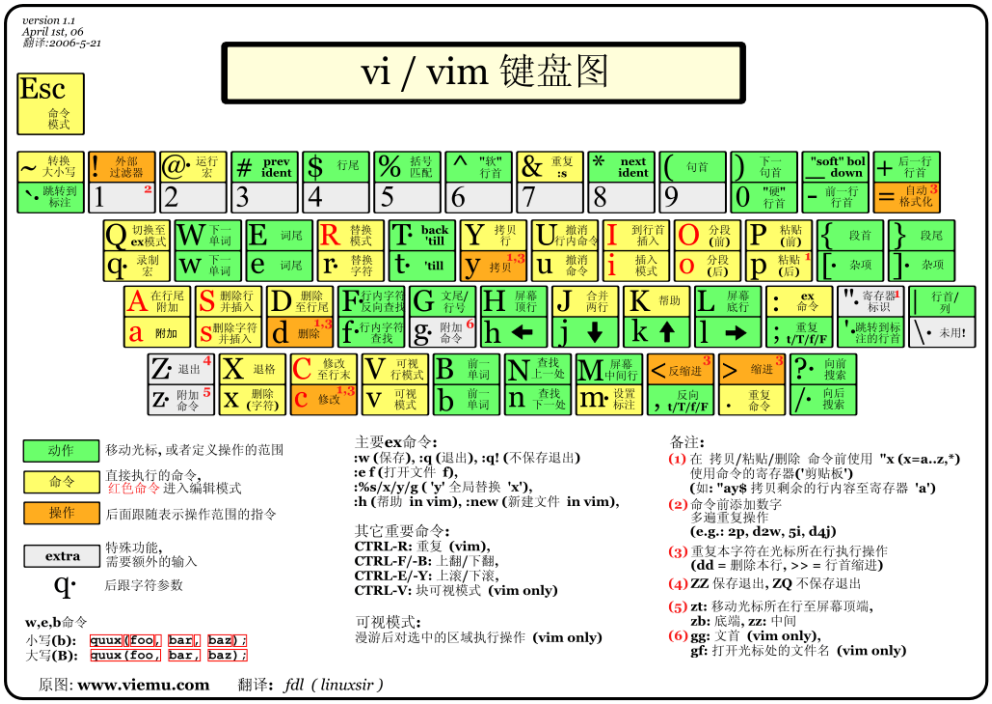
x	向右删除一个字符
nx	向右删除 n 个字符
X	向左删除一个字符
nX	向左删除 n 个字符

进入输入模式 (编辑模式)

i	在光标位置插入
I（大写的 i）	在当前行的第一个非空字符前插入

编辑模式

:n	将光标定位到第 n 行
:w	保存



用 vim 打开桌面上名为“d.txtl”的文件。

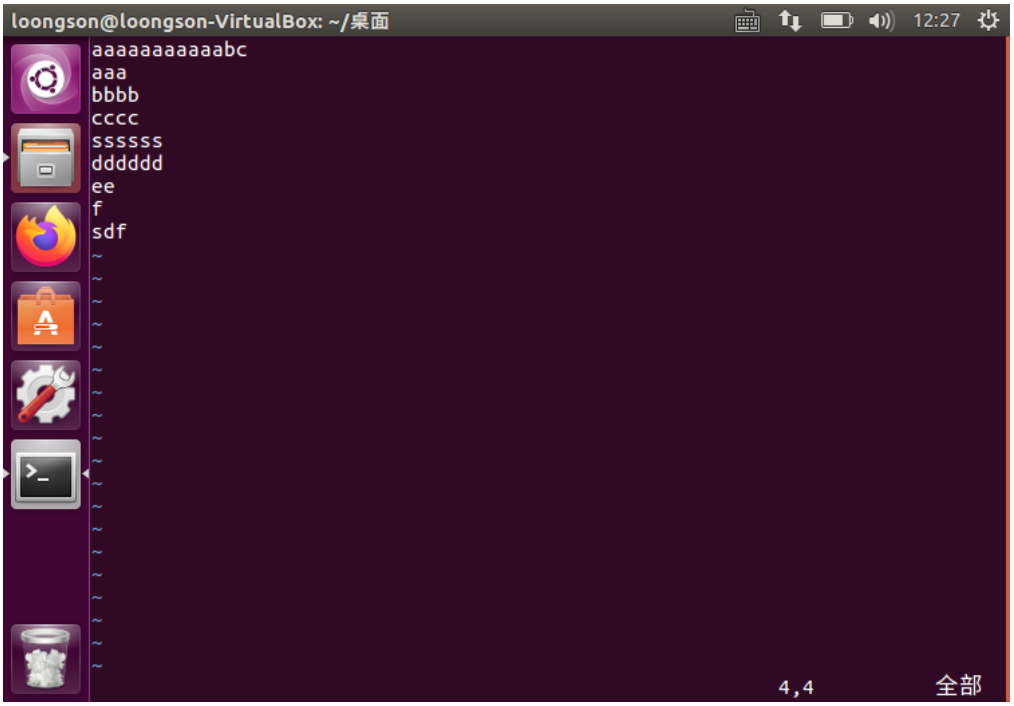
输入命令“vim d.txt' ”并执行。

```
loongson@loongson-VirtualBox:~/桌面$ vim d.txt
```

```
aaaaaaaaaabc
aaa
bbbb
cccc
ssssss
dddddd
ee
f
sdf
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"d.txt" 9L, 51C                                1,1           全部
```

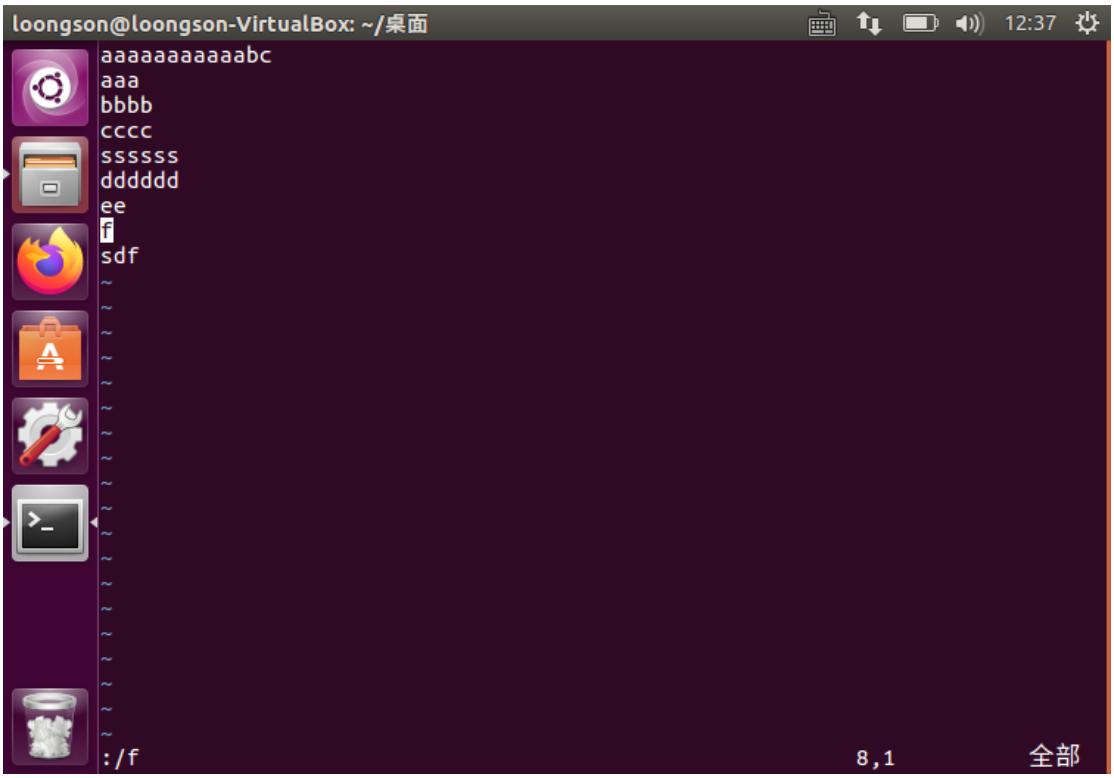
移动到第 3 行，向右移动 4 个字符

在命令模式下输入 3G 4l ，执行如下：



光标在 cccc 处搜索 f

/f 向光标之下寻找一个名称为 f 的字符串。

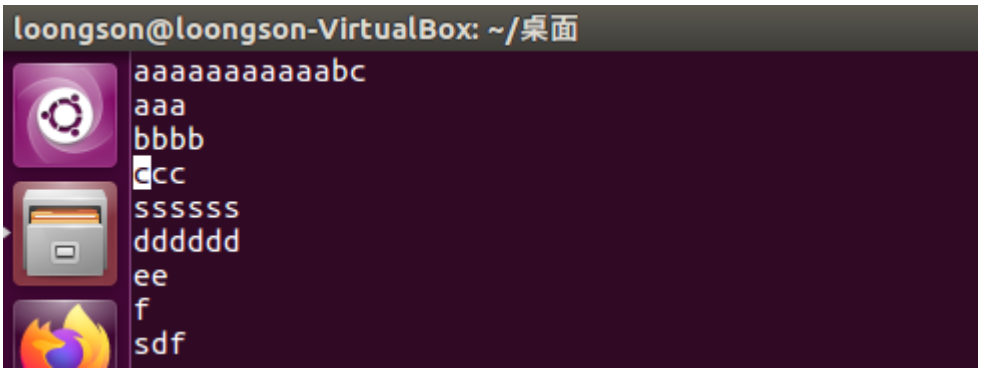


n （英文按键 n）代表重复前一个搜寻的动作。

N （英文按键 N）与 n 相反，为『反向』进行前一个搜寻动作。

x, X 相当于[del]， X 相当于 [backspace]

x 在命令模式下删除光标所处位置的字符：



保存后离开

在底线命令模式下输入“wq”，退出编辑，关闭文件。执行如下：

7）收集整理 LLNUX 文件系统 ext2、ext3、ext4 的存储结构和优缺点，试着写出其对应的 C 语言描述结构体程序代码段。

Linux kernel 自 2.6.28 开始正式支持新的文件系统 Ext4。 Ext4 是 Ext3 的改进版，修改了 Ext3 中部分重要的数据结构，而不仅仅像 Ext3 对 Ext2 那样，只是增加了一个日志功能而已。Ext4 可以提供更佳的性能和可靠性，还有更为丰富的功能：

1. 与 Ext3 兼容。执行若干条命令，就能从 Ext3 在线迁移到 Ext4，而无须重新格式化磁盘或重新安装系统。原有 Ext3 数据结构照样保留，Ext4 作用于新数据，当然，整个文件系统因此也就获得了 Ext4 所支持的更大容量。

2. 更大的文件系统和更大的文件。较之 Ext3 目前所支持的最大 16TB 文件系统和最大 2TB 文件，Ext4 分别支持 1EB（1,048,576TB，1EB=1024PB， 1PB=1024TB）的文件系统，以及 16TB 的文件。

3. 无限数量的子目录。Ext3 目前只支持 32,000 个子目录，而 Ext4 支持无限数量的子目录。

4. Extents。Ext3 采用间接块映射，当操作大文件时，效率极其低下。比如一个 100MB 大小的文件，在 Ext3 中要建立 25,600 个数据块（每

个数据块大小为 4KB）的映射表。而 Ext4 引入了现代文件系统中流行的 extents 概念，每个 extent 为一组连续的数据块，上述文件则表示为“该文件数据保存在接下来的 25,600 个数据块中”，提高了不少效率。

5. 多块分配。当写入数据到 Ext3 文件系统中时，Ext3 的数据块分配器每次只能分配一个 4KB 的块，写一个 100MB 文件就要调用 25,600 次数据块分配器，而 Ext4 的多块分配器“multiblock allocator”（mballoc） 支持一次调用分配多个数据块。

6. 延迟分配。Ext3 的数据块分配策略是尽快分配，而 Ext4 和其它现代文件操作系统的策略是尽可能地延迟分配，直到文件在 cache 中写完才开始分配数据块并写入磁盘，这样就能优化整个文件的数据块分配，与前两种特性搭配起来可以显著提升性能。

7. 快速 fsck。以前执行 fsck 第一步就会很慢，因为它要检查所有的 inode，现在 Ext4 给每个组的 inode 表中都添加了一份未使用 inode 的列表，今后 fsck Ext4 文件系统就可以跳过它们而只去检查那些在用的 inode 了。

8. 日志校验。日志是最常用的部分，也极易导致磁盘硬件故障，而从损坏的日志中恢复数据会导致更多的数据损坏。Ext4 的日志校验功能可以很方便地判断日志数据是否损坏，而且它将 Ext3 的两阶段日志机制合并成一个阶段，在增加安全性的同时提高了性能。

9. “无日志”（No Journaling）模式。日志总归有一些开销，Ext4 允许关闭日志，以便某些有特殊需求的用户可以借此提升性能。

10. 在线碎片整理。尽管延迟分配、多块分配和 extents 能有效减少文件系统碎片，但碎片还是不可避免会产生。Ext4 支持在线碎片整理，并将提供 e4defrag 工具进行个别文件或整个文件系统的碎片整理。

11. inode 相关特性。Ext4 支持更大的 inode，较之 Ext3 默认的 inode 大小 128 字节，Ext4 为了在 inode 中容纳更多的扩展属性（如纳秒时间戳或 inode 版本），默认 inode 大小为 256 字节。Ext4 还支持快速扩展属性（fast extended attributes）和 inode 保留（inodes reservation）。

12. 持久预分配（Persistent preallocation）。P2P 软件为了保证下载文

件有足够的空间存放，常常会预先创建一个与所下载文件大小相同的空文件，以免未来的数小时或数天之内磁盘空间不足导致下载失败。 Ext4 在文件系统层面实现了持久预分配并提供相应的 API（libc 中的 posix_fallocate()），比应用软件自己实现更有效率。

13. 默认启用 barrier。磁盘上配有内部缓存，以便重新调整批量数据的写操作顺序，优化写入性能，因此文件系统必须在日志数据写入磁盘之后才能写 commit 记录，若 commit 记录写入在先，而日志有可能损坏，那么就会影响数据完整性。Ext4 默认启用 barrier，只有当 barrier 之前的数据全部写入磁盘，才能写 barrier 之后的数据。（可通过 "mount -o barrier=0" 命令禁用该特性。）

ext2、ext3、ext4 对应的 C 语言描述结构体程序代码段

ext4：

```
struct ext4_super_block {
/*00*/ __le32   s_inodes_count;              /* Inodes count 文件系统中 inode 的总数 */
        __le32   s_blocks_count_lo;         /* Blocks count 文件系统中块的总数*/
        __le32   s_r_blocks_count_lo;       /* Reserved blocks count 保留块的总数*/
        __le32   s_free_blocks_count_lo; /*Free blocks count 未使用的块的总数（包括保留块）*/
/*10*/ __le32   s_free_inodes_count;      /* Free inodes count 未使用的 inode 的总数*/
        __le32   s_first_data_block;        /* First Data Block 第一块块 ID，在小于 1KB 的文件系统中为 0，大于 1KB 的文件系统中为 1*/
        __le32   s_log_block_size;          /* Block size 用以计算块的大小（1024 算术左移该值即为块大小）(0=1K, 1=2K, 2=4K) */
        __le32   s_obso_log_frag_size;    /* Obsoleted fragment size 用以计算段大小（为正则 1024 算术左移该值，否则右移）*/
/*20*/ __le32   s_blocks_per_group;       /* # Blocks per group 每个块组中块的总数 */
        __le32   s_obso_frags_per_group; /*Obsoleted fragments per group 每个块组
```

中段的总数*/

__le32 s_inodes_per_group; /* # Inodes per group 每个块组中 inode 的总数*/

__le32 s_mtime; /* Mount time POSIX 中定义的文件系统装载时间*/

/*30*/ __le32 s_wtime; /* Write time POSIX 中定义的文件系统最近被写入的时间*/

__le16 s_mnt_count; /* Mount count 最近一次完整校验后被装载的次数*/

__le16 s_max_mnt_count; /* Maximal mount count 在进行完整校验前还能被装载的次数*/

__le16 s_magic; /* Magic signature 文件系统标志*/

__le16 s_state; /* File system state 文件系统的状态*/

__le16 s_errors; /* Behaviour when detectingerrors 文件系统发生错误时驱动程序应该执行的操作*/

__le16 s_minor_rev_level; /* minor revision level 局部修订级别*/

/*40*/ __le32 s_lastcheck; /* time of last check POSIX 中定义的文件系统最近一次检查的时间*/

__le32 s_checkinterval; /* max. time between checks POSIX 中定义的文件系统最近检查的最大时间间隔*/

__le32 s_creator_os; /* OS 生成该文件系统的操作系统*/

__le32 s_rev_level; /* Revision level 修订级别*/

/*50*/ __le16 s_def_resuid; /* Default uid for reserved blocks 报留块的默认用户 ID */

__le16 s_def_resgid; /* Default gid for reserved blocks 保留块的默认组 ID */

__le32 s_first_ino; /* First non-reserved inode 标准文件的第一个可用 inode 的索引（非动态为 11）*/

__le16 s_inode_size; /* size of inode structure inode 结构的大

小（非动态为 128）*/

__le16 s_block_group_nr; /* block group # of this superblock 保存此超级块的块组号*/

__le32 s_feature_compat; /* compatible feature set 兼容特性掩码*/

/*60*/ __le32 s_feature_incompat; /* incompatible feature set 不兼容特性掩码*/

__le32 s_feature_ro_compat; /* readonly-compatible feature set 只读特性掩码*/

/*68*/ __u8 s_uuid[16]; /* 128-bit uuid for volume 卷 ID，应尽可能使每个文件系统的格式唯一*/

/*78*/ char s_volume_name[16]; /* volume name 卷名（只能为 ISO-Latin-1 字符集，以'\0'结束）*/

/*88*/ char s_last_mounted[64]; /* directory where last mounted 最近被安装的目录*/

/*C8*/ __le32 s_algorithm_usage_bitmap;/* For compression 文件系统采用的压缩算法*/

/*

* Performance hints. Directorypreallocation should only

* happen if the EXT4_FEATURE_COMPAT_DIR_PREALLOC flag is on.

*/

__u8 s_prealloc_blocks; /* Nr of blocks to try to preallocate 预分配的块数*/

__u8 s_prealloc_dir_blocks; /* Nr topreallocate for dirs 给目录预分配的块数*/

__le16 s_reserved_gdt_blocks; /* Pergroup desc for online growth */

/*

* Journaling support valid if EXT4_FEATURE_COMPAT_HAS_JOURNAL set.

*/

/*D0*/ __u8 s_journal_uuid[16]; /* uuid of journal superblock 日志超级块的

卷 ID */

/*E0*/ __le32 s_journal_inum; /* inode number of journal file 日志文件的 inode 数目*/

__le32 s_journal_dev; /* device number of journal file 日志文件的设备数*/

__le32 s_last_orphan; /* start of list of inodes to delete 要删除的 inode 列表的起始位置*/

__le32 s_hash_seed[4]; /* HTREE hash seed HTREE 散列种子*/

__u8 s_def_hash_version; /* Default hash version to use 默认使用的散列函数*/

__u8 s_jnl_backup_type;

__le16 s_desc_size; /* size of group descriptor */

/*100*/ __le32 s_default_mount_opts;

__le32 s_first_meta_bg; /* First metablock block group 块组的第一个元块*/

__le32 s_mkfs_time; /* Whenthe filesystem was created */

__le32 s_jnl_blocks[17]; /* Backup of the journal inode */

/* 64bit support valid if EXT4_FEATURE_COMPAT_64BIT */

/*150*/ __le32 s_blocks_count_hi; /* Blocks count */

__le32 s_r_blocks_count_hi; /* Reserved blocks count */

__le32 s_free_blocks_count_hi; /*Free blocks count */

__le16 s_min_extra_isize; /* All inodes have at least # bytes */

__le16 s_want_extra_isize; /* New inodes should reserve # bytes */

__le32 s_flags; /* Miscellaneous flags */

__le16 s_raid_stride; /* RAID stride */

__le16 s_mmp_update_interval; /* #seconds to wait in MMP checking */

__le64 s_mmp_block; /* Blockfor multi-mount protection */

__le32 s_raid_stripe_width; /* blocks on all data disks (N*stride)*/

__u8 s_log_groups_per_flex; /* FLEX_BGgroup size */

__u8 s_reserved_char_pad;

```
        __le16    s_reserved_pad;
        __le64    s_kbytes_written;          /* nr of lifetime kilobytes written */
        __le32    s_snapshot_inum;           /* Inode number of active snapshot */
        __le32    s_snapshot_id;             /* sequential ID of active snapshot*/
        __le64    s_snapshot_r_blocks_count;/* reserved blocks for active
                                                snapshot's future use */
        __le32    s_snapshot_list;           /* inode number of the head of the
                                                on-disk snapshot list */
#define EXT4_S_ERR_START offsetof(structext4_super_block, s_error_count)
        __le32    s_error_count;             /* number of fs errors */
        __le32    s_first_error_time;        /* first time an error happened */
        __le32    s_first_error_ino;         /* inode involved in first error */
        __le64    s_first_error_block;       /* block involved of first error */
        __u8      s_first_error_func[32]; /*function where the error happened */
        __le32    s_first_error_line;        /* line number where error happened */
        __le32    s_last_error_time;         /* most recent time of an error */
        __le32    s_last_error_ino;          /* inode involved in last error */
        __le32    s_last_error_line;         /* line number where error happened */
        __le64    s_last_error_block;        /* block involved of last error */
        __u8    s_last_error_func[32];    /*function where the error happened */
#define EXT4_S_ERR_END offsetof(structext4_super_block, s_mount_opts)
        __u8      s_mount_opts[64];
        __le32    s_reserved[112];           /* Padding to the end of the block */
};

struct ext4_group_desc
{
        __le32    bg_block_bitmap_lo;        /* Blocks bitmap block 块位图所在的第
一个块的块 ID */
        __le32    bg_inode_bitmap_lo;        /* Inodes bitmap block inode 位图所在的
```

```
第一个块的块 ID */
        __le32  bg_inode_table_lo;      /* Inodes table block inode 表所在的第一
个块的块 ID */
        __le16  bg_free_blocks_count_lo;/*Free blocks count 块组中未使用的块数
*/
        __le16  bg_free_inodes_count_lo;/*Free inodes count 块组中未使用的
inode 数*/
        __le16 bg_used_dirs_count_lo;  /*Directories count 块组分配的目录的
inode 数*/
        __le16  bg_flags;               /* EXT4_BG_flags (INODE_UNINIT,etc) */
        __u32   bg_reserved[2];         /* Likely block/inode bitmap checksum*/
        __le16  bg_itable_unused_lo;    /* Unused inodes count */
        __le16  bg_checksum;            /* crc16(sb_uuid+group+desc) */
        __le32  bg_block_bitmap_hi;     /* Blocks bitmap block MSB */
        __le32  bg_inode_bitmap_hi;     /* Inodes bitmap block MSB */
        __le32  bg_inode_table_hi;      /* Inodes table block MSB */
        __le16  bg_free_blocks_count_hi;/*Free blocks count MSB */
        __le16  bg_free_inodes_count_hi;/*Free inodes count MSB */
        __le16 bg_used_dirs_count_hi;  /*Directories count MSB */
        __le16  bg_itable_unused_hi;    /* Unused inodes count MSB */
        __u32   bg_reserved2[3];
};
struct ext4_inode {
        __le16  i_mode;         /* File mode 文件格式和访问权限*/
        __le16  i_uid;          /* Low 16 bits of Owner Uid 文件所有者 ID 的低 16
位*/
        __le32  i_size_lo;      /* Size in bytes 文件字节数*/
        __le32  i_atime;        /* Access time 文件上次被访问的时间*/
        __le32  i_ctime;        /* Inode Change time 文件创建时间*/
        __le32  i_mtime;        /* Modification time 文件被修改的时间*/
```

```
        __le32  i_dtime;        /* Deletion Time 文件被删除的时间（如果存在则为 0）*/
        __le16  i_gid;          /* Low 16 bits of Group Id 文件所有组 ID 的低 16 位*/
        __le16  i_links_count;  /* Links count 此 inode 被连接的次数*/
        __le32  i_blocks_lo;    /* Blocks count 文件已使用和保留的总块数（以 512B 为单位）*/
        __le32  i_flags;        /* File flags */
        union {
                struct {
                        __le32  l_i_version;
                } linux1;
                struct {
                        __u32   h_i_translator;
                } hurd1;
                struct {
                        __u32   m_i_reserved1;
                } masix1;
        } osd1;                         /*OS dependent 1 */
        __le32  i_block[EXT4_N_BLOCKS];/*Pointers to blocks 定位存储文件的块的数组*/
        __le32  i_generation;   /* File version (for NFS) 用于 NFS 的文件版本*/
        __le32  i_file_acl_lo;  /* File ACL 包含扩展属性的块号，老版本中为 0*/
        __le32  i_size_high;
        __le32  i_obso_faddr;   /* Obsoleted fragment address */
        union {
                struct {
                        __le16  l_i_blocks_high; /* were l_i_reserved1 */
                        __le16  l_i_file_acl_high;
                        __le16  l_i_uid_high;   /* these 2 fields */
```

```
                __le16  l_i_gid_high;    /* were reserved2[0] */
                __u32   l_i_reserved2;
        } linux2;
        struct {
                __le16  h_i_reserved1;  /* Obsoleted fragment
number/size which areremoved in ext4 */
                __u16   h_i_mode_high;
                __u16   h_i_uid_high;
                __u16   h_i_gid_high;
                __u32   h_i_author;
        } hurd2;
        struct {
                __le16  h_i_reserved1;  /* Obsoleted fragment
number/size which areremoved in ext4 */
                __le16  m_i_file_acl_high;
                __u32   m_i_reserved2[2];
        } masix2;
    } osd2;                             /*OS dependent 2 */
    __le16  i_extra_isize;
    __le16  i_pad1;
    __le32  i_ctime_extra;  /* extra Change time      (nsec << 2 | epoch) */
    __le32  i_mtime_extra;  /* extra Modification time(nsec << 2 |epoch) */
    __le32  i_atime_extra;  /* extra Access time      (nsec << 2 | epoch) */
    __le32  i_crtime;       /* File Creation time */
    __le32  i_crtime_extra; /* extraFileCreationtime (nsec << 2 | epoch) */
    __le32  i_version_hi;   /* high 32 bits for 64-bit version */
};
struct ext4_dir_entry {
    __le32  inode;                      /* Inode number 文件入口的 inode 号，
0 表示该项未使用*/
```

```
        __le16    rec_len;                    /* Directory entry length 目录项长度*/
        __le16    name_len;                   /* Name length 文件名包含的字符数*/
        char      name[EXT4_NAME_LEN];        /* File name 文件名*/
};
    struct ext2_acl_header      /* Header of Access Control Lists */
     {
            __u32       aclh_size;
            __u32       aclh_file_count;
            __u32       aclh_acle_count;
            __u32       aclh_first_acle;
      };
     struct ext2_acl_entry      /* Access Control List Entry */
     {
         __u32       acle_size;
            __u16       acle_perms;      /* Access permissions */
            __u16       acle_type;       /* Type of entry */
            __u16       acle_tag;       /* User or group identity */
            __u16       acle_pad1;
            __u32       acle_next;       /* Pointer on next entry for the */
     };

    struct ext2_group_desc
    {
            __u32       bg_block_bitmap;            /* Blocks bitmap block */
            __u32       bg_inode_bitmap;            /* Inodes bitmap block */
            __u32       bg_inode_table;             /* Inodes table block */
            __u16       bg_free_blocks_count;       /* Free blocks count */
            __u16       bg_free_inodes_count;       /* Free inodes count */
            __u16       bg_used_dirs_count;       /* Directories count */
            __u16       bg_pad;
```

```
        __u32       bg_reserved[3];
};
```

Inode：

```
struct ext2_inode {
        __u16       i_mode;             /* File mode */
        __u16       i_uid;              /* Low 16 bits of Owner Uid */
        __u32       i_size;             /* Size in bytes */
        __u32       i_atime;       /* Access time */
        __u32       i_ctime;       /* Creation time */
        __u32       i_mtime;       /* Modification time */
        __u32       i_dtime;       /* Deletion Time */
        __u16       i_gid;              /* Low 16 bits of Group Id */
        __u16       i_links_count;       /* Links count */
        __u32       i_blocks;       /* Blocks count */
        __u32       i_flags;       /* File flags */
        union {
        struct {
                __u32    l_i_reserved1;
            } linux1;
        struct {
                __u32    h_i_translator;
            } hurd1;
          struct {
                __u32    m_i_reserved1;
            } masix1;
          } osd1;                         /* OS dependent 1 */
        __u32       i_block[EXT2_N_BLOCKS];/* Pointers to blocks */
        __u32       i_generation;       /* File version (for NFS) */
        __u32       i_file_acl;       /* File ACL */
        __u32       i_dir_acl;       /* Directory ACL */
```

```
        __u32       i_faddr;       /* Fragment address */
        union {
                struct {
                    __u8       l_i_frag;       /* Fragment number */
                    __u8       l_i_fsize;       /* Fragment size */
                    __u16       i_pad1;
                    __u16       l_i_uid_high;       /* these 2 fields       */
                    __u16       l_i_gid_high;       /* were reserved2[0] */
                    __u32       l_i_reserved2;
                } linux2;
                struct {
                     __u8       h_i_frag;       /* Fragment number */
                     __u8       h_i_fsize;       /* Fragment size */
                     __u16       h_i_mode_high;
                     __u16       h_i_uid_high;
                     __u16       h_i_gid_high;
                     __u32       h_i_author;
                } hurd2;
               struct {
                     __u8       m_i_frag;       /* Fragment number */
                     __u8       m_i_fsize;       /* Fragment size */
                     __u16       m_pad1;
                     __u32       m_i_reserved2[2];
               } masix2;
            } osd2;                         /* OS dependent 2 */
};
```

实验结果与分析：

通过本次实验，我对第 2-5 讲的内容进行了复习和理解。通过练习，掌握了有关 LINUX 命令的使用,如用户/组管理；文件与目录的定位；文件与目录的浏览；

文件内容的搜索；文件和目录等操作处理方式。 同时，通过查询资料，对 LINUX 文件系统 ext2、ext3、ext4 的存储结构和优缺点有了浅显的认识。

Vim 是从 vi 发展出来的一个文本编辑器。其代码补完、编译及错误跳转等方便编程的功能特别丰富，被人们广泛使用。Linux 提供了丰富的系统指令，用户可以在终端下使用这些指令完成对计算机的操作。在实验中，我熟悉了 vim 文本编辑工具的使用，为日常的文件操作维护打下扎实的基础。但要真正熟练使用 Vim 编辑器，还要在实际操作中多去使用。