操作系统 实验三 文件系统的用户界面和Unix实用程序

3.1 文件系统的用户界面

1. 本实验中直接使用系统调用 `read(fd, buf, nbytes)`，`write(fd, buf, nbytes)` 和文件的库函数 `fread(buf, size, nitems, fp)`，`fwrite(buf, size, nitems, fp)` 完成文件复制功能。
2. 分别使用 `fscanf` 和 `fprintf`，`fgetc` 和 `fputc`，`fgets` 和 `fputs`（仅限于行结构的文本文件），实现上述的文件复制程序。
3. 编写一个父子进程之间用无名管道进行数据传送的 *C* 程序。父进程逐一读出一个文件的内容，并通过管道发送给子进程。子进程从管道中读出信息，再将其写入一个新的文件。程序结束后，对原文件和新文件的内容进行比较。
4. 在两个用户的独立程序之间，使用有名管道，重新编写一个 *C* 程序，实现题三的功能。

算法思想

1 直接使用系统调用和文件的库函数

分别使用文件的系统调用 `read(fd, buf, nbytes)`，`write(fd, buf, nbytes)` 和文件的库函数 `fread(buf, size, nitems, fp)`，`fwrite(buf, size, nitems, fp)`，编写一个文件的复制程序。

当上述函数中 `nbytes`，`size` 和 `nitems` 都取值为 1 时（即一次读写一个字节），比较这两种程序的执行效率。当 `nbytes` 取 1024 字节，`size` 取 1024 字节，且 `nitems` 取 1 时（即一次读写 1024 字节），再次比较这两种程序的执行效率。

2 使用 `fscanf` 和 `fprintf`，`fgetc` 和 `fputc`，`fgets` 和 `fputs`

分别使用 `fscanf` 和 `fprintf`，`fgetc` 和 `fputc`，`fgets` 和 `fputs`（仅限于行结构的文本文件），实现上述的文件复制程序。

3 父子进程之间用无名管道进行数据传送

父子进程之间用无名管道进行数据传送。父进程逐一读出一个文件的内容，并通过管道发送给子进程。子进程从管道中读出信息，再将其写入一个新的文件。

4 两个用户的独立程序使用有名管道进行数据传送

两个用户的独立程序使用有名管道进行数据传送。进程一逐一读出一个文件的内容，并通过管道发送给进程二。进程二从管道中读出信息，再将其写入一个新的文件。

源程序

函数说明

```
ssize_t read(int fd, void *buf, size_t count);
ssize_t write(int fd, void *buf, size_t count);
```

`read`，`write` 等文件操作函数是直接读写的，即没有缓冲区的读写。

- `fd`：通过 `open` 函数打开文件返回的文件描述符。
- `buf`：指定存储器读出或写入数据的缓冲区，但是与基于缓冲区读写的概念不同。
- `count`：指定读出或写入的字节数。

```
size_t fread ( void * ptr, size_t size, size_t count, FILE * stream );
size_t fwrite ( const void * ptr, size_t size, size_t count, FILE * stream
);
```

- `ptr`：指向保存结果的指针；
- `size`：每个数据类型的大小；
- `count`：数据的个数；
- `stream`：文件指针
- 函数返回读取/写入数据的个数。

写操作 `fwrite()` 后必须关闭流 `fclose()`，不关闭流的情况下，每次读或写数据后，文件指针都会指向下一个待写或者读数据位置的指针。

```
int fscanf(FILE * stream, const char * format, [argument...]);
```

其功能为根据数据格式(`format`)，从输入流(`stream`)中读入数据，存储到`argument`中，遇到空格和换行时结束。成功返回读入的参数的个数，失败返回`EOF(-1)`。

```
char *fgets(char *str, int n, FILE *stream);
```

从指定的流`stream`读取一行，并把它存储在`str`所指向的字符串内。当读取`(n-1)`个字符时，或者读取到换行符时，或者到达文件末尾时，它会停止，具体视情况而定。

函数成功将返回`stream`，失败或读到文件结尾返回`NULL`。

主程序

1 直接使用系统调用和文件的库函数

```
// 使用文件的库函数`fread(buf, size, nitems, fp)`, `fwrite(buf, size, nitems, fp)`, size取1
#include <sys/stat.h>
#include <sys/fcntl.h>
#include <stdio.h>
#include <time.h>
#define NUM 1

main()
{
  clock_t start,end;
  start=clock();
  int n;
  char buf4[NUM];
  FILE *fp1, *fp2;
  fp1 = fopen("file1.txt", "r");
  fp2 = fopen("file2.txt", "w");
  while((n=fread(buf4,NUM,1,fp1)) > 0) fwrite(buf4, NUM, n, fp2);
  fclose(fp1);
  fclose(fp2);
  end=clock();
  printf("%d\n",end-start);
}
```

```
// 使用文件的库函数`fread(buf, size, nitems, fp)`, `fwrite(buf, size, nitems, fp)`, size取1024
#include <sys/stat.h>
#include <sys/fcntl.h>
#include <stdio.h>
#include <time.h>
#define NUM 1024

```

```c
main()
{
  clock_t start,end;
  start=clock();
  int n;
  char buf4[NUM];
  FILE *fp1, *fp2;
  fp1 = fopen("file1.txt", "r");
  fp2 = fopen("file2.txt", "w");
  while((n=fread(buf4,NUM,1,fp1)) > 0) fwrite(buf4, NUM, n, fp2);
  fclose(fp1);
  fclose(fp2);
  end=clock();
  printf("%d\n",end-start);
}
```

```c
// 使用系统调用`read(fd, buf, nbytes)`, `write(fd, buf, nbytes)`, nbytes取1
#include <sys/stat.h>
#include <sys/fcntl.h>
#include <stdio.h>
#include <time.h>
#define NUM 1

main()
{
  clock_t start,end;
  start=clock();
  int fd1, fd2;
  int n;
  char buf1[NUM];
  FILE *fp1, *fp2;
  fd1 = open("file1.txt", O_RDONLY);
  fd2 = open("file2.txt", O_WRONLY|O_CREAT, 0644);
  while((n=read(fd1,buf1,NUM)) > 0) write(fd2, buf1, n);
  close(fd1);
  close(fd2);
  end=clock();
  printf("%d\n",end-start);
}
```

```c
// 使用系统调用`read(fd, buf, nbytes)`, `write(fd, buf, nbytes)`, nbytes取1024
#include <sys/stat.h>
#include <sys/fcntl.h>
#include <stdio.h>
#include <time.h>
#define NUM 1024

```

```c
main()
{
  clock_t start,end;
  start=clock();
  int fd1, fd2;
  int n;
  char buf1[NUM];
  FILE *fp1, *fp2;
  fd1 = open("file1.txt", O_RDONLY);
  fd2 = open("file2.txt", O_WRONLY|O_CREAT, 0644);
  while((n=read(fd1,buf1,NUM)) > 0) write(fd2, buf1, n);
  close(fd1);
  close(fd2);
  end=clock();
  printf("%d\n",end-start);
}
```

2 使用 `fscanf` 和 `fprintf`，`fgetc` 和 `fputc`，`fgets` 和 `fputs`

```c
#include <sys/stat.h>
#include <sys/fcntl.h>
#include <stdio.h>

main()
{
  int n;
  char c;
  char buf1[256], buf2[256];
  FILE *fp1, *fp2, *fp3, *fp4;
  fp1 = fopen("file1.txt", "r");
  fp2 = fopen("file2.txt", "w");
  fp3 = fopen("file3.txt", "w");
  fp4 = fopen("file4.txt", "w");
  // `fscanf` 和 `fprintf`
  while(fscanf(fp1, "%s", buf1) > 0) fprintf(fp2, "%s", buf1);
  fclose(fp1);
  // `fgetc` 和 `fputc`
  fp1 = fopen("file1.txt", "r");
  while((c=fgetc(fp1)) > 0) fputc(c, fp3);
  fclose(fp1);
  // `fgets` 和 `fputs`
  fp1 = fopen("file1.txt", "r");
  while(fgets(buf2, 1024, fp1)) fputs(buf2, fp4);
  fclose(fp1);
}
```

3 父子进程之间用无名管道进行数据传送

```c
#include <sys/stat.h>
#include <stdio.h>
#include <fcntl.h>
#include <sys/types.h>
#include<string.h>

main()
{
  int fd1,fd2;
  char buf1[1];
  int chanel[2],n;
  char buf[100];
  fd1 = open("file1.txt", O_RDONLY);
  fd2 = open("file2.txt", O_WRONLY|O_CREAT, 0644);
  pipe(chanel);
  if(fork())
  {
    close(chanel[0]);
    while((n=read(fd1,buf1,1)) > 0) write(chanel[1], buf1, n);
    printf("Parent sends over.\n");
    close(chanel[1]);
  }
  else
  {
    close(chanel[1]);
    while((n=read(chanel[0],buf,sizeof(buf))) > 0) write(fd2, buf, n);
    printf("Child receives over.\n");
    close(chanel[0]);
  }
  close(fd1);
  close(fd2);
}
```

4 两个用户的独立程序使用有名管道进行数据传送

```c
// 进程一
#include <stdio.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <string.h>

main()
{
  int fd, fd1, n;
  char buf1[1];
  fd1 = open("file1.txt", O_RDONLY);
  mknod("fifo", S_IFIFO|0666,0);
  fd = open("fifo", O_WRONLY);
```

```
  while(n=read(fd1,buf1,1) > 0) write(fd, buf1, n);
  close(fd);
  close(fd1);
}
```

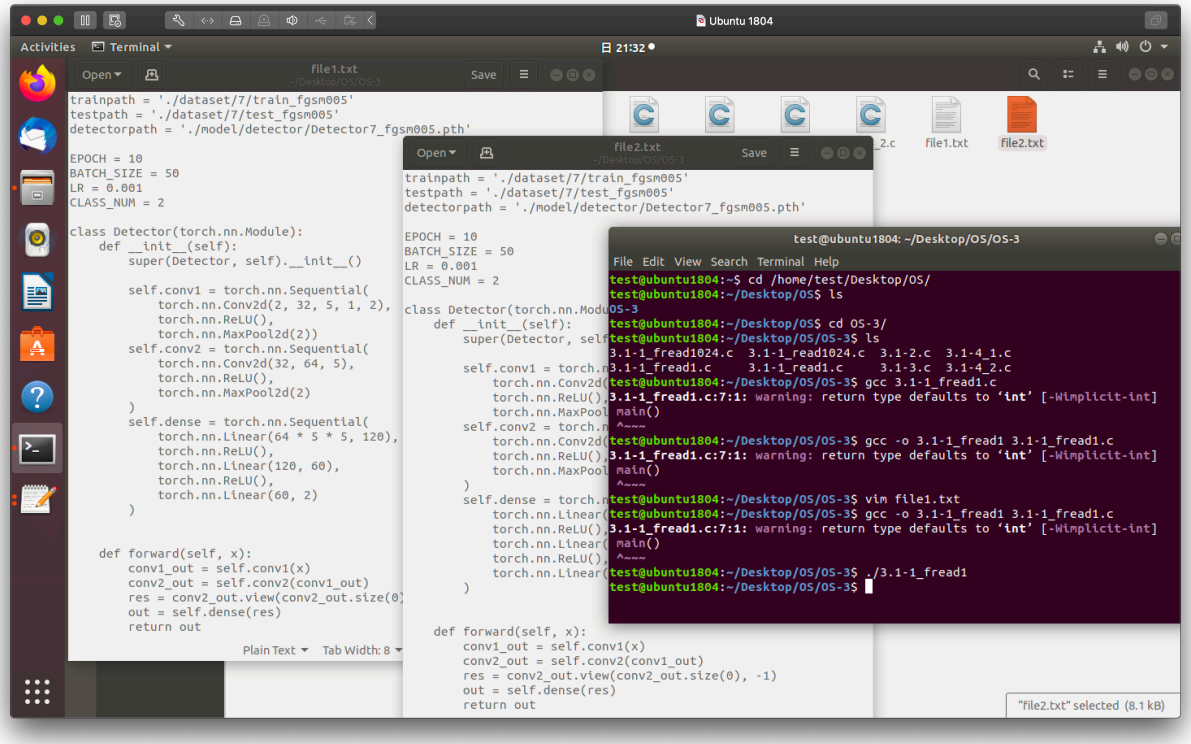
```
// 进程二
#include <fcntl.h>

main()
{
  int fd, fd2, n;
  char buf1[1];
  fd = open("fifo", O_RDONLY);
  fd2 = open("file2.txt", O_WRONLY|O_CREAT, 0644);
  while((n=read(fd,buf1,sizeof(buf1))) != -1) write(fd2, buf1, n);
  close(fd);
  close(fd2);
}
```

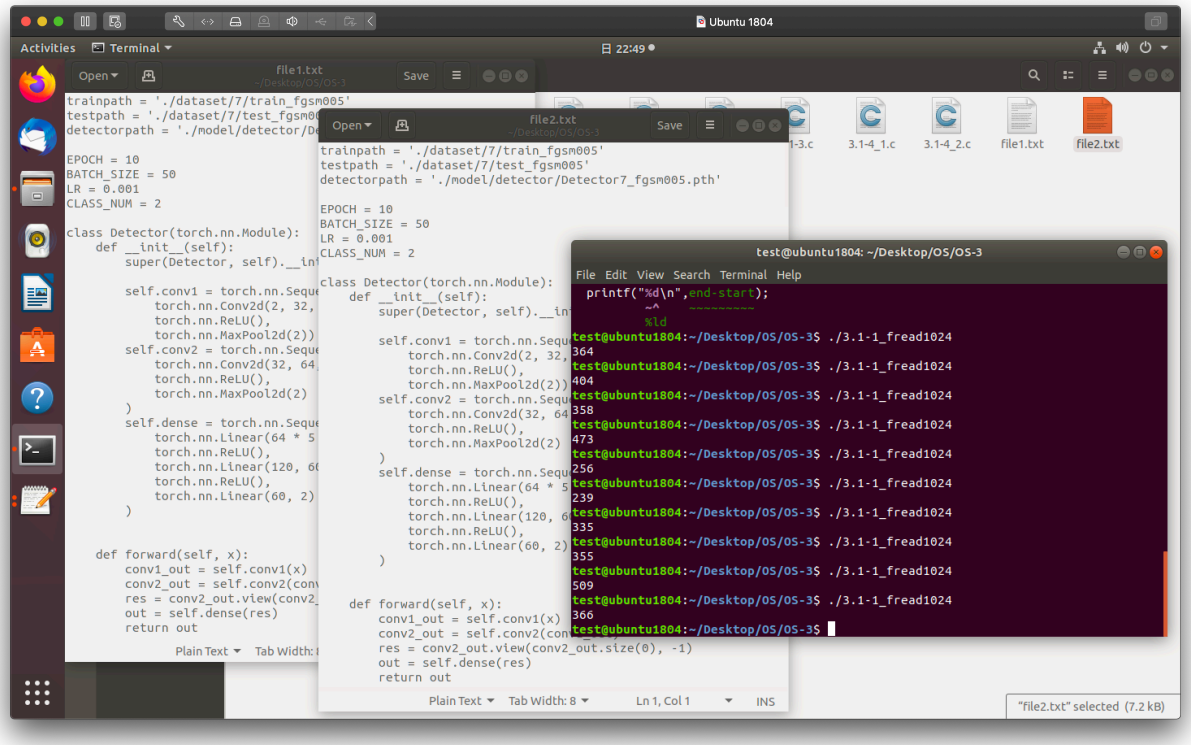
测试

1 直接使用系统调用和文件的库函数

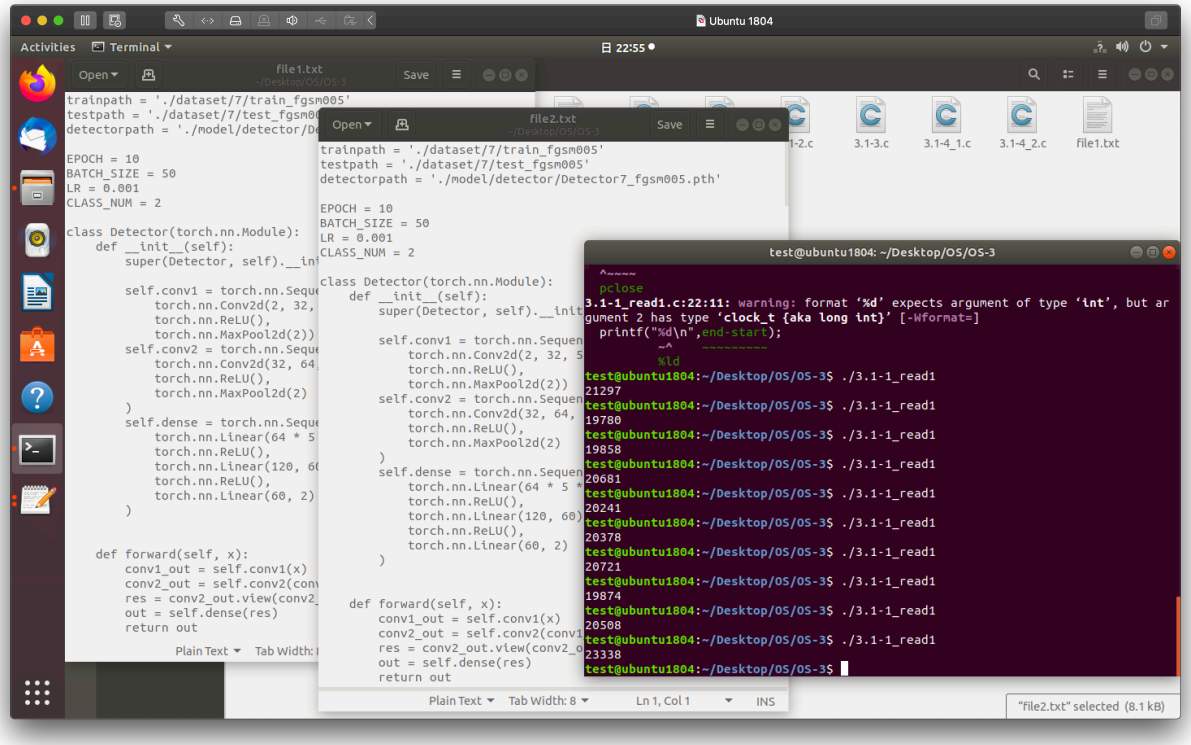
如图运行 `3.1-1_fread1`，即使用文件的库函数`fread(buf, size, nitems, fp)`，`fwrite(buf, size, nitems, fp)`，`size` 取 1。`file1.txt` 中是一些以前的实验数据，可见被成功复制到 `file2.txt`。观察到每次执行时间都不一样，这是进程运行的不确定性导致的，于是运行十次，取平均值为 $668.6ms$。



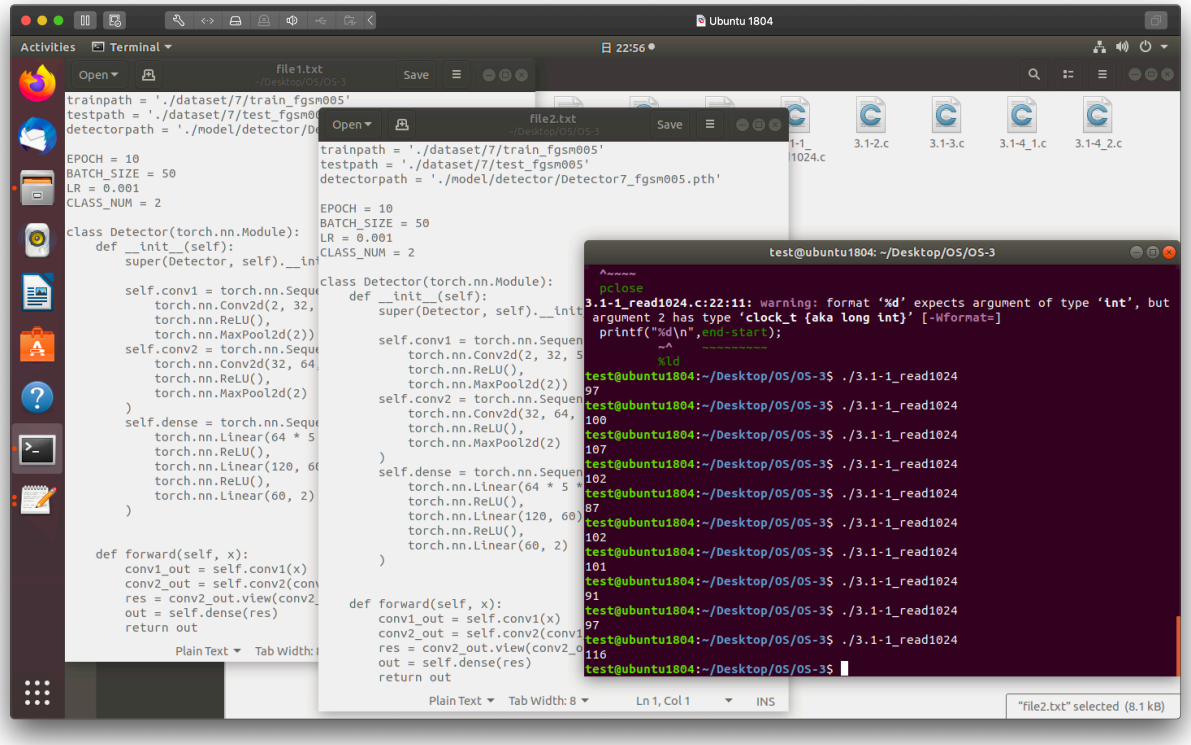
运行 `3.1-1_fread1024`，即使用文件的库函数`fread(buf, size, nitems, fp)`，`fwrite(buf, size, nitems, fp)`，`size` 取 1。平均运行时间为 $365.9ms$，明显小于`size` 取 1 的运行时间。



运行 `3.1-1_read1`，即使用系统调用`read(fd, buf, nbytes)`，`write(fd, buf, nbytes)`，`nbytes` 取 1。平均运行时间为 $20667.6ms$。

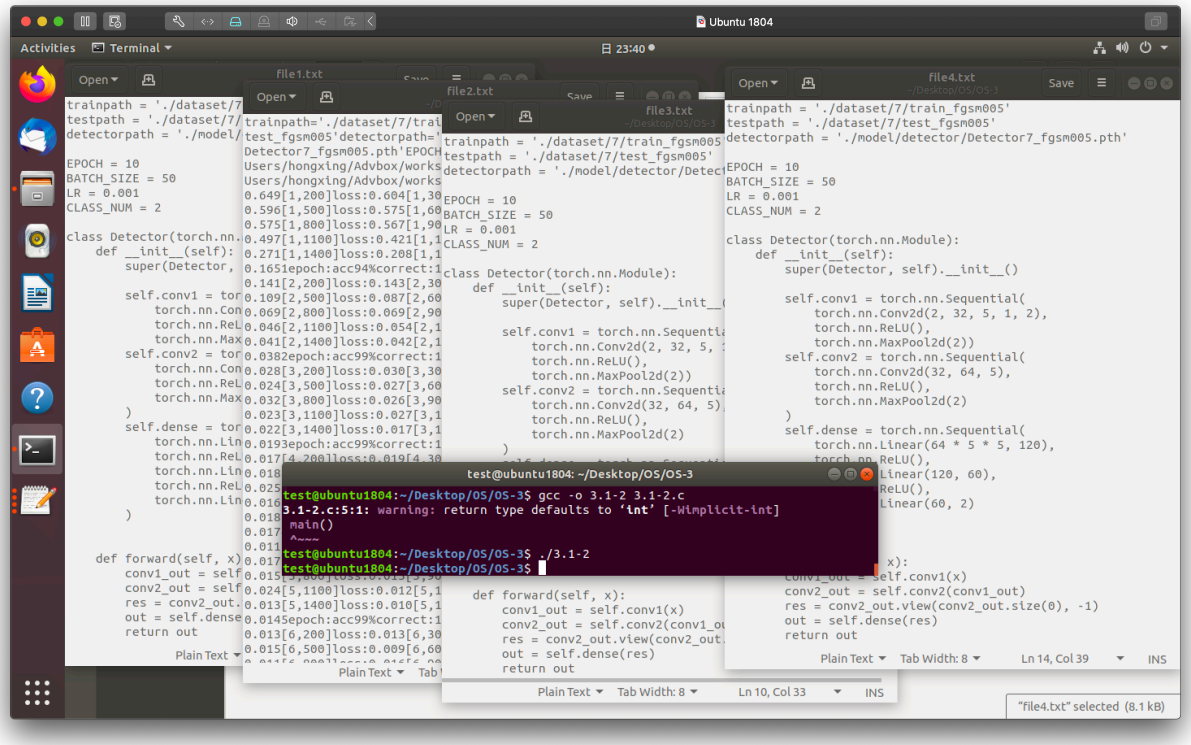


运行 `3.1-1_read1024`，即使用系统调用`read(fd, buf, nbytes)`，`write(fd, buf, nbytes)`，`nbytes` 取 1024。平均运行时间为 $100ms$，远小于 `nbytes` 取 1 所需时间。



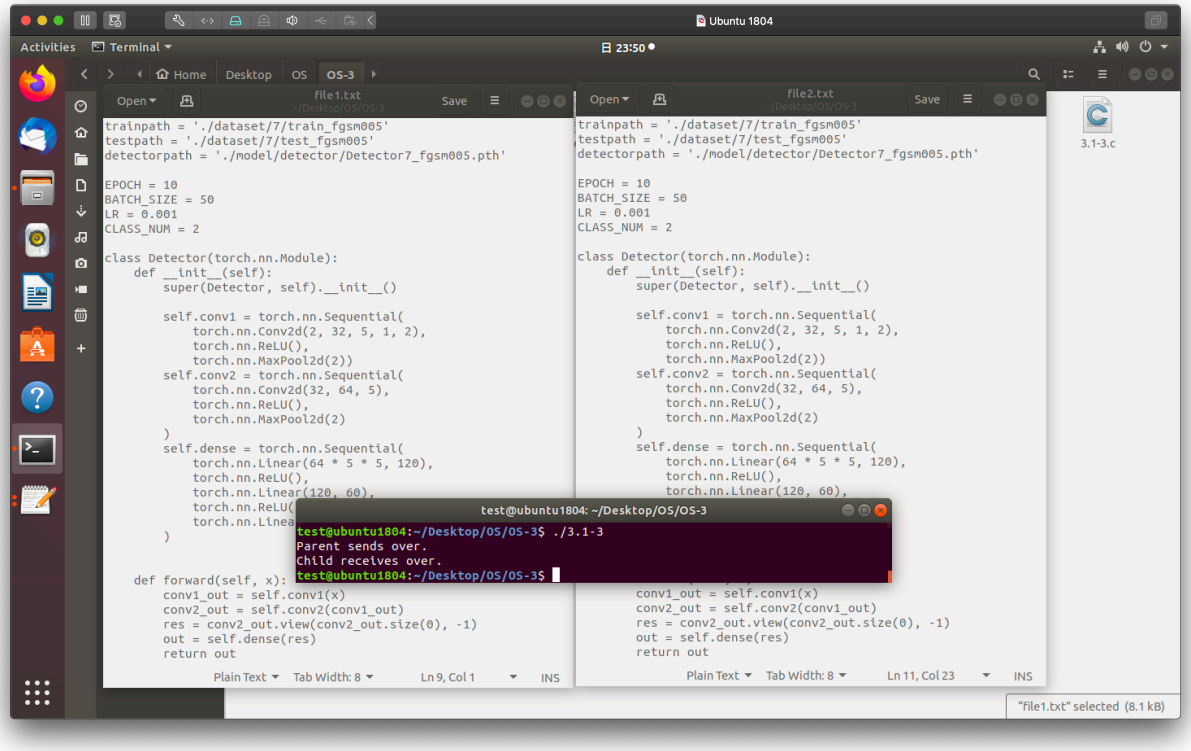
2 使用 `fscanf` 和 `fprintf`，`fgetc` 和 `fputc`，`fgets` 和 `fputs`

运行 `3.1-2`，分别复制 `file1.txt` 为三个文件。实验过程发现，`fscanf` 会跳过所有空格和换行。



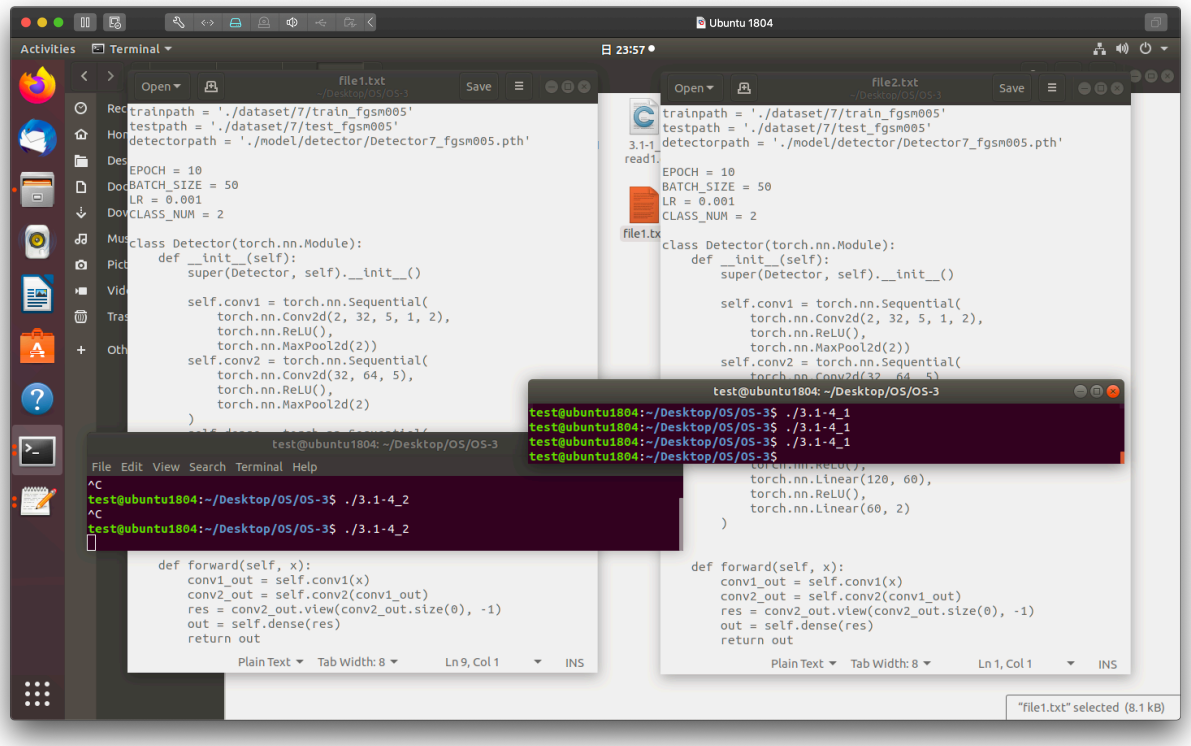
3 父子进程之间用无名管道进行数据传送

父子进程之间用无名管道进行数据传送。父进程逐一读出 `file1.txt` 的内容，并通过管道发送给子进程。子进程从管道中读出信息，再将其写入 `file2.txt`。



4 两个用户的独立程序使用有名管道进行数据传送

先运行进程一，再运行进程二，进程一读出 `file1.txt` 的内容，并通过管道发送给进程二。进程二从管道中读出信息，再将其写入 `file2.txt`。

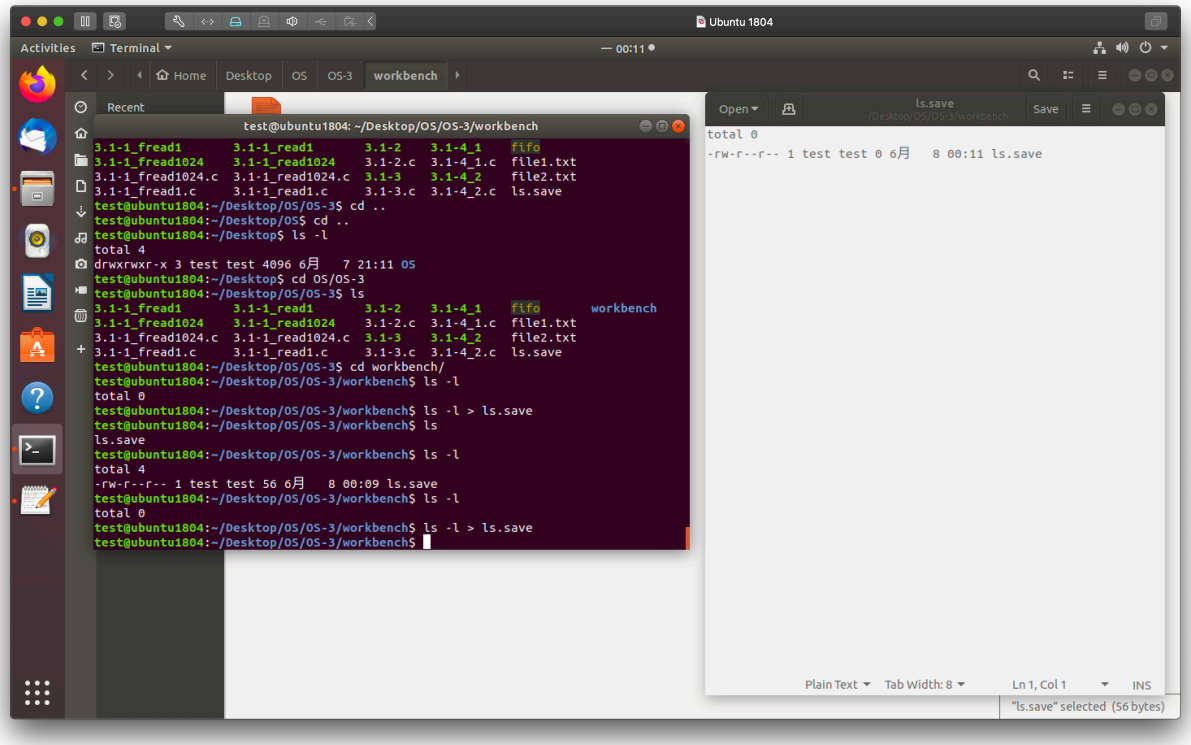


3.2 Unix实用程序

1. 上机练习 `ls, ln, ln –s, diff, file, chown, chmode, head, taill, pr, od` 等命令。

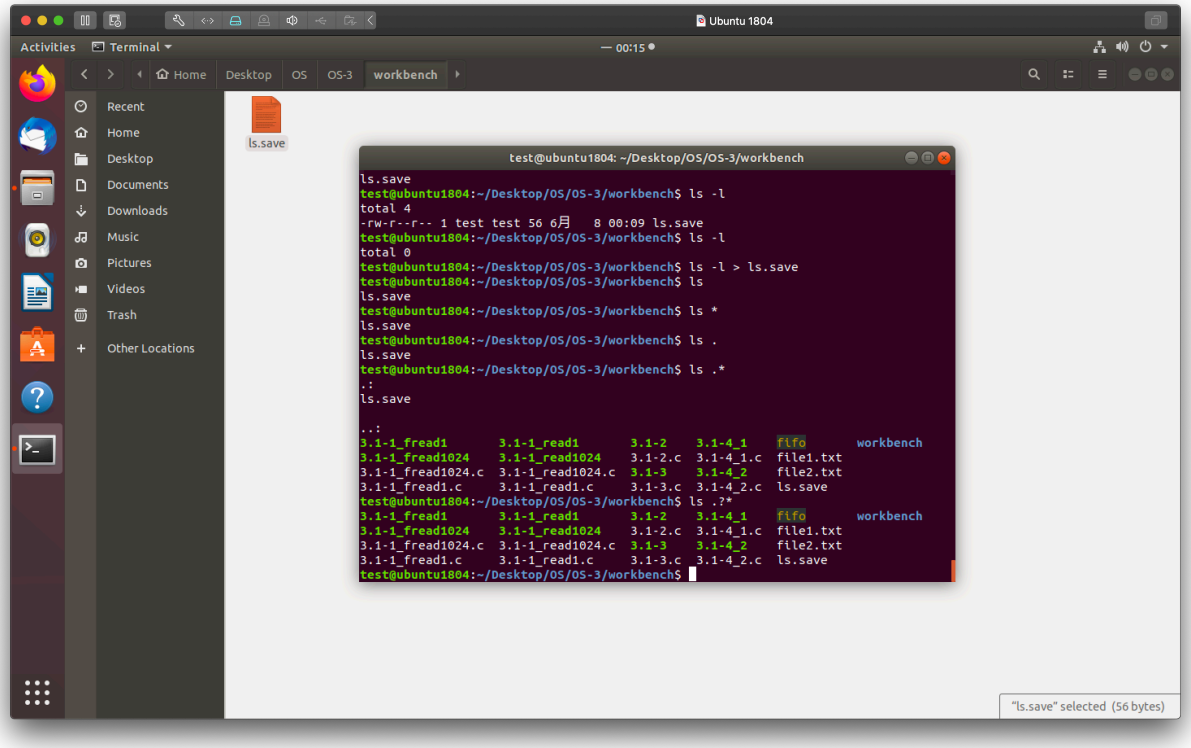
(1) 假定当前目录中没有 `ls.save` 文件，先后键入 `ls -l` 和 `ls -l > ls.save` 两个命令。请将第一个 `ls` 命令的输出和第二个命令的输出文件 `ls.save` 中的内容作仔细比较。两者有何不同？并加以解释。

会多出 `ls.save` 文件。说明输出 `ls.save` 文件时会把自己也计入。



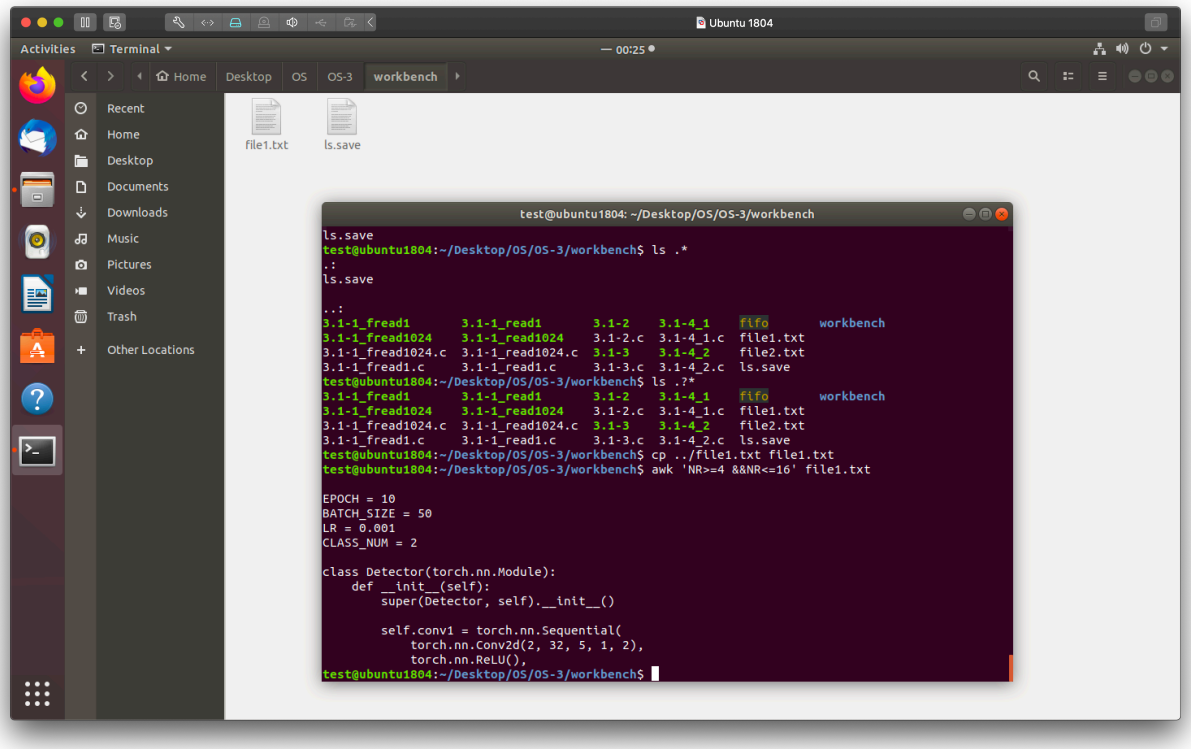
(2) 比较 `ls, ls *, ls ., ls .* , ls .?*` 这些命令的输出有何不同?

`ls, ls *, ls .` 即列出当前目录下所有文件与目录名；`ls .*` 即列出当前目录下所有以 `.` 开头的文件与目录名；`ls .?*` 即列出当前目录下所有以 `.` 开头，且 `.` 后还有字符的文件与目录名。



(3) 显示某一文件从 `n1` 行到 `n2` 行的内容。

```
awk 'NR>=4 &&NR<=16' file1.txt
```



(4) 以下两条命令的执行效果是否总是相同的？请仔细推敲并上机验证。

```
mv file1 file2
```

```
cp  file1 file2; rm file1
```

不总是一样。

`cp` 命令是用来拷贝数据，在 `cp` 命令执行过程中：分配一个空闲的 `inode` 号，在 `inode` 表中生成新条目，在目录中创建一个目录项，将名称与 `inode` 编号关联，拷贝数据生成新的文件。

`rm` 命令是删除数据的工具，在 `rm` 命令执行过程中；数据链接数递减，从而释放 `inode` 号,并且 `inode` 号可以被重用，把数据块放在空闲列表中，删除目录项，数据实际上不会马上被删除，但当另一个文件使用数据块时将被会被覆盖。

`cp` 命令和 `rm` 命令可以看出来在运行这两种工具时，底层数据都发生了改变。

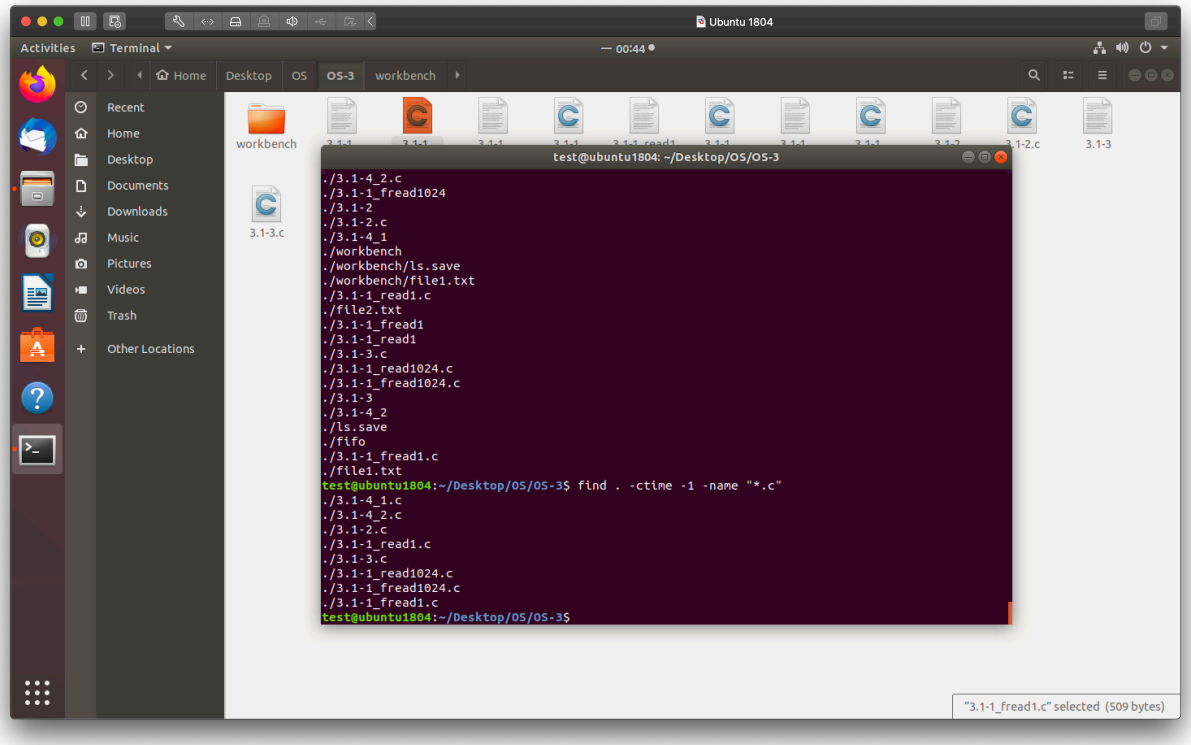
`mv` 命令是经常用来将数据从一个地方挪到另外一个地方的工具，而 `mv` 命令在挪动数据的时候底层工作分为两种情况：

1. 第一种是 `mv` 命令的目标和源在相同的文件系统，可以理解为同一分区，`mv` 命令在执行过程中用新的文件名创建对应新的目录项，删除旧目录条目对应的旧的文件名，并不影响 `inode` 表（除时间戳）或磁盘上的数据位置，也就是说没有数据被移动。
2. 第二种如果目标和源不在一个文件系统，也就是不在一个分区，那么 `mv` 就相当于 `cp` 和 `rm`。即改变了 `inode` 表的条目，也发生了数据移动。

2. 上机练习 `find` 命令。

（1）在当前的目录树中显示当天修改的 *c* 源程序名。

```
find . -ctime -1 -name "*.c"
```

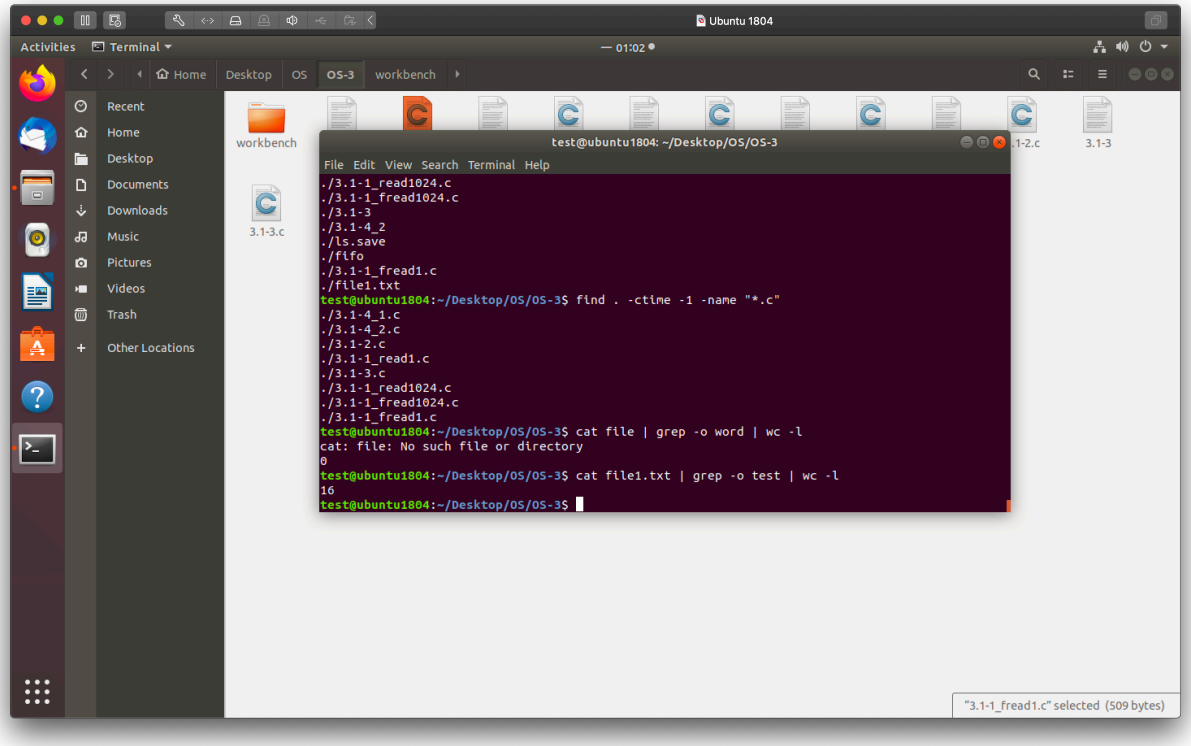


掌握正则表达式的匹配规则，上机练习 `grep，sed，awk` 等命令。

（1） 按相反的次序显示输入行及行号。

（2） 统计输入单词的频度。

```
cat file1.txt | grep -o test | wc -l
```



上机练习 `date, df, du, tar` 命令。

`date` 格式化输出：

```
date +"%Y-%m-%d"
```

`date` 输出昨天日期：

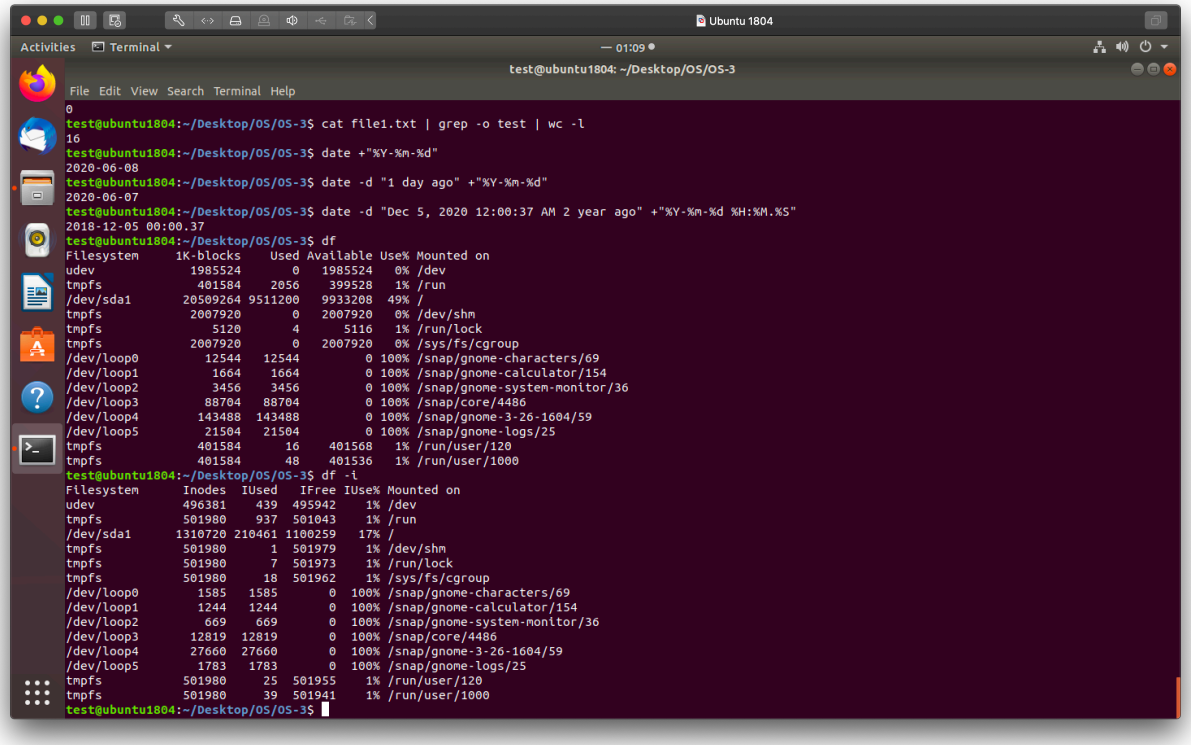
```
date -d "1 day ago" +"%Y-%m-%d"
```

`date` 格式转换：

```
date -d "Dec 5, 2020 12:00:37 AM 2 year ago" +"%Y-%m-%d %H:%M.%S"
```

`df`

```
# 显示文件系统的磁盘使用情况统计：
df
# 用一个-i选项的df命令的输出显示inode信息而非块使用量
df -i
```

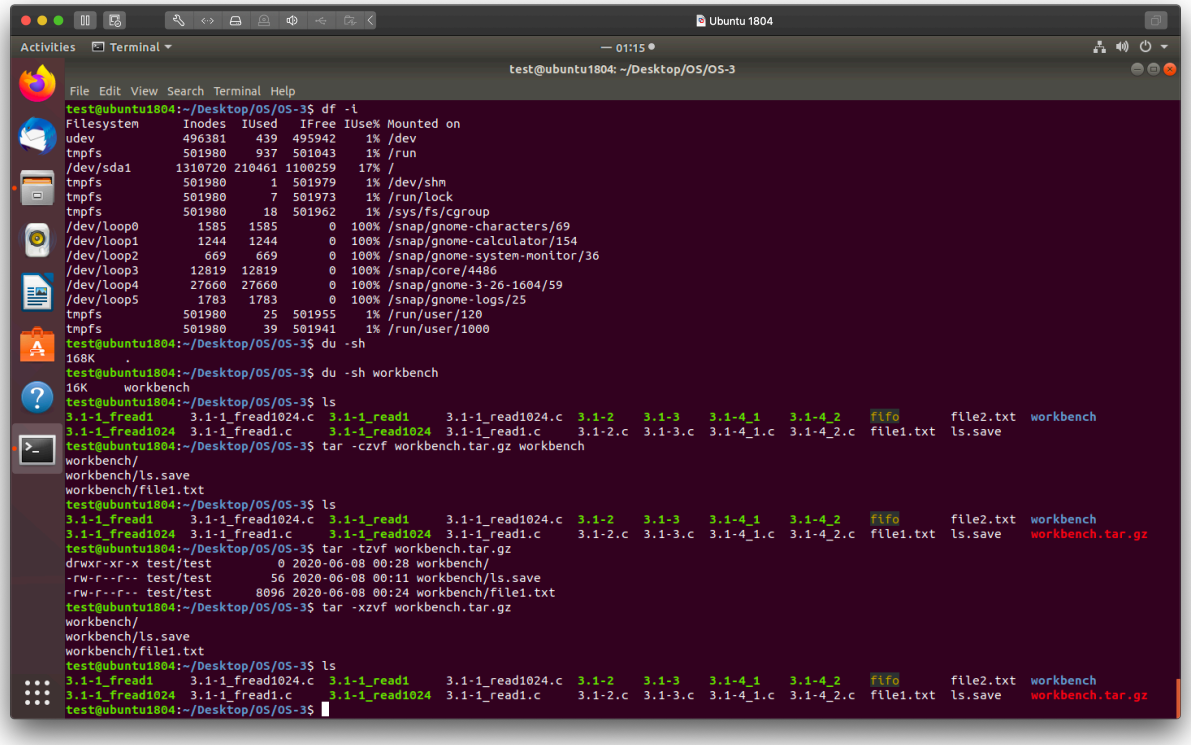


`du`

```
# 显示当前目录的大小
du -sh
# 显示某个目录或文件的大小
du -sh workbench
```

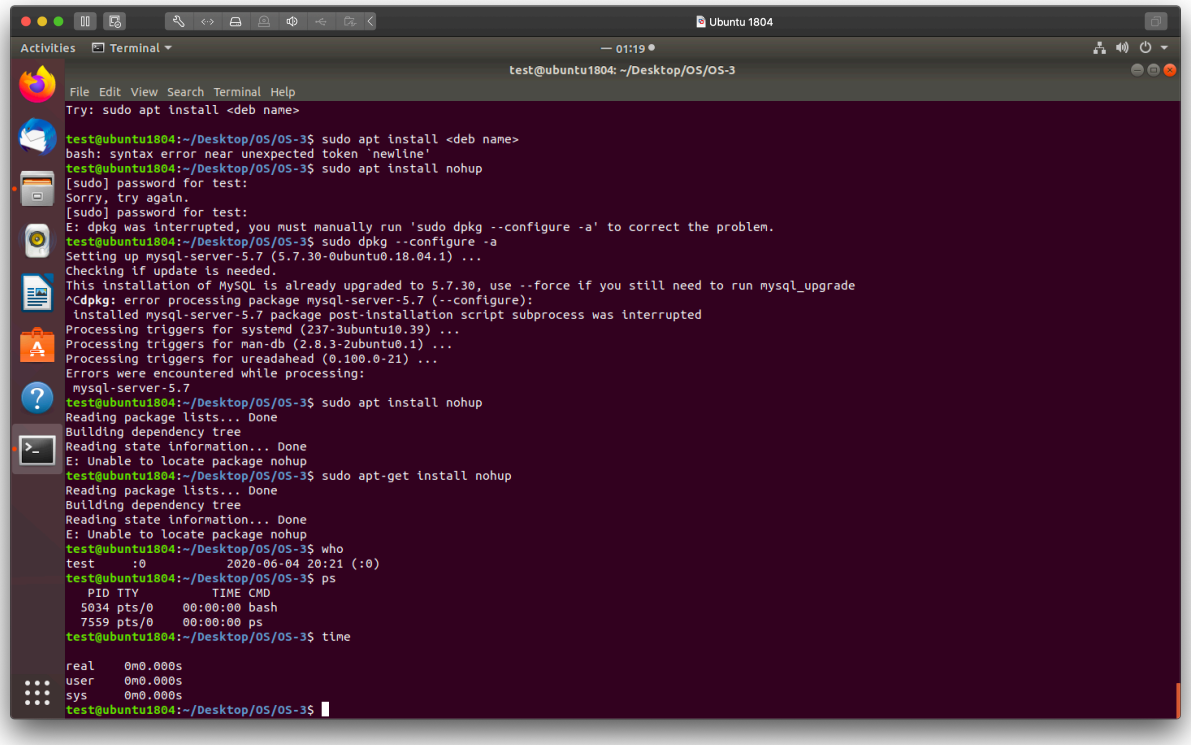
`tar`

```
# 压缩文件，非打包
tar -czvf workbench.tar.gz workbench
# 列出压缩文件内容
tar -tzvf workbench.tar.gz
# 解压文件
tar -xzvf workbench.tar.gz
```



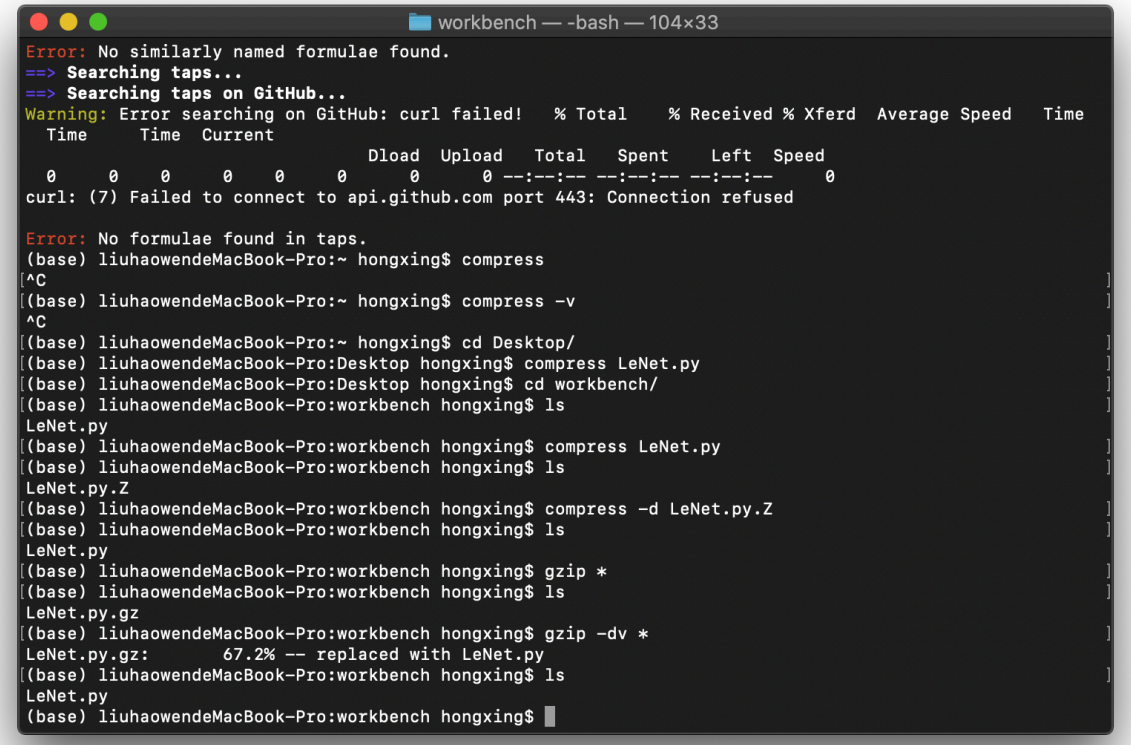
上机练习 `who, ps, time, nohub` 命令。

由于 **Linux** 虚拟机出现安装问题，`nohub` 命令无法运行。



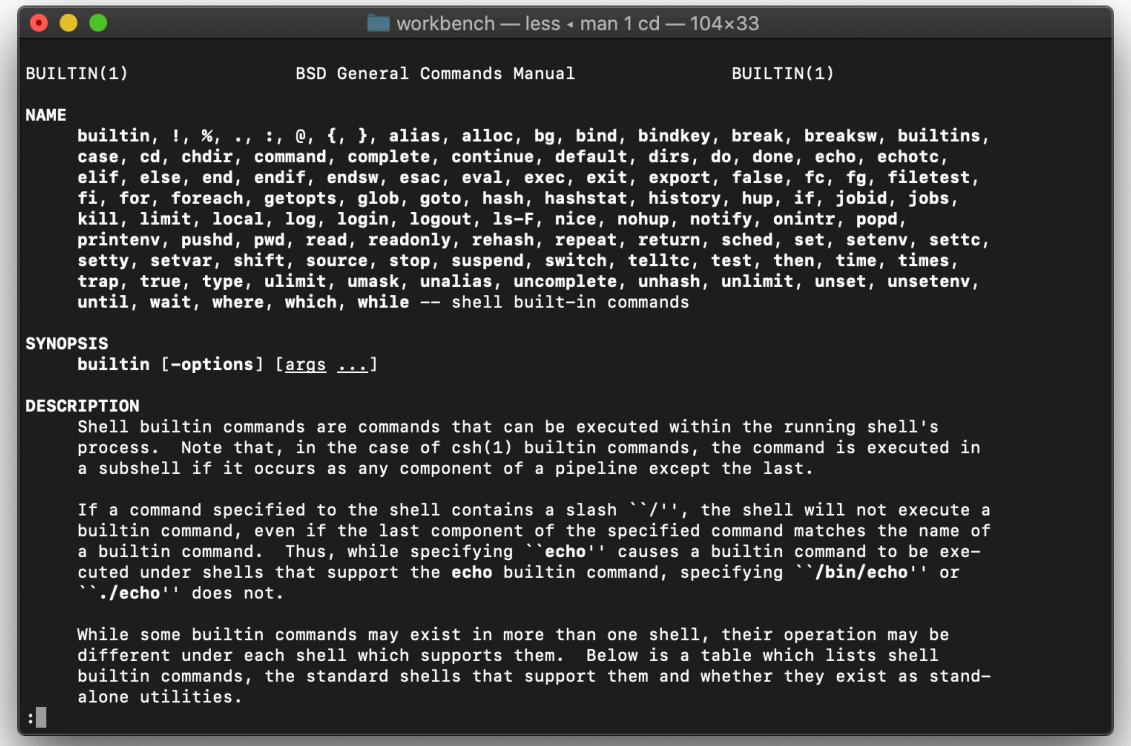
上机练习 `compress, gzip` 命令。

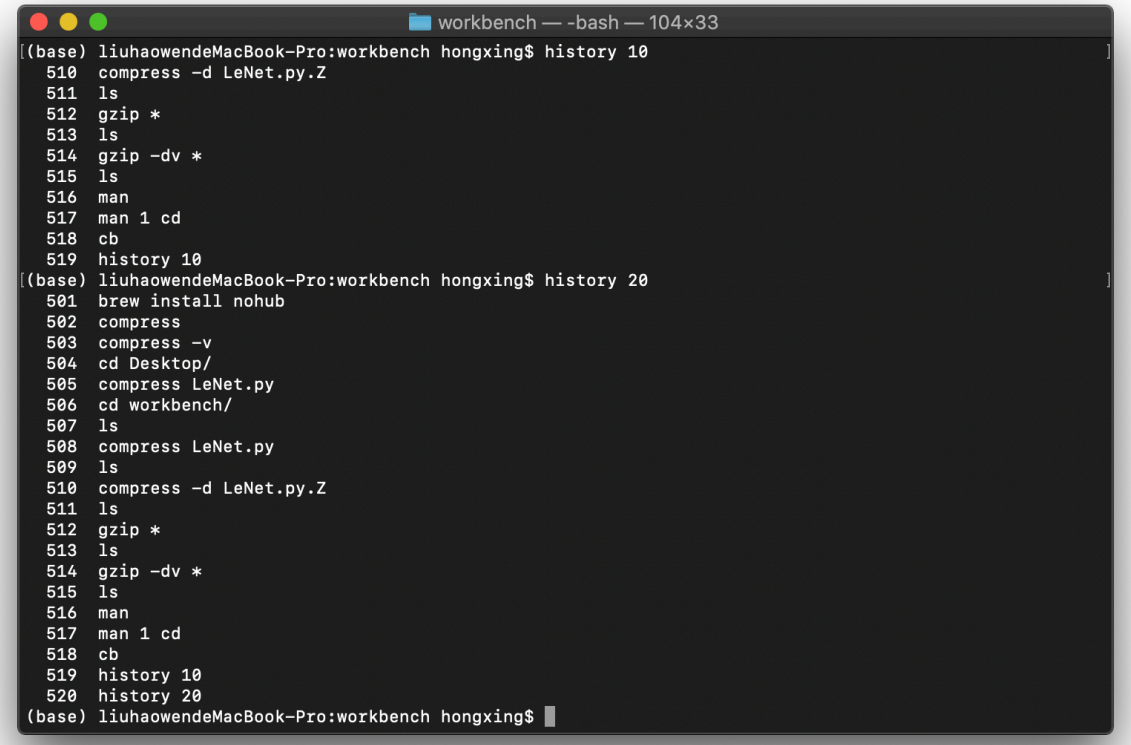
由于虚拟机出现问题，之后的命令都在 **macOS** 机器上练习。



上机练习 `man, cb, history` 命令。

```
man 1 cd
```





改进与体会

改进

一开始不太清楚`fgetc`和`fputc`，`fgets`和`fputs`等函数的用法，发现输出的文件只有一个单词或只有一行。查询资料，了解了这些函数的处理方法、用法和返回值后才能正确写出程序。

体会

这次实验让我非常有效地复习了操作系统的文件系统。同时也学习了 **Linux** 的不少命令。虽然以前经常使用 **Linux** 命令完成一些任务，但都是些常用命令，且对它们的真正用法也不太了解。这次实验不仅让我了解了很多陌生的命令，也对常用的命令有了更深的了解：原来它们的功能如此强大，至今我用到的只是它们的一些皮毛。

by 刘浩文 517021911065