

操作系统 实验三 文件系统的用户界面和Unix实用程序

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

算法思想

- 1 直接使用系统调用和文件的库函数
- 2 使用 `fscanf` 和 `fprintf`, `fgetc` 和 `fputc`, `fgets` 和 `fputs`
- 3 父子进程之间用无名管道进行数据传送
- 4 两个用户的独立程序使用有名管道进行数据传送

源程序

函数说明

主程序

- 1 直接使用系统调用和文件的库函数
- 2 使用 `fscanf` 和 `fprintf`, `fgetc` 和 `fputc`, `fgets` 和 `fputs`
- 3 父子进程之间用无名管道进行数据传送
- 4 两个用户的独立程序使用有名管道进行数据传送

测试

- 1 直接使用系统调用和文件的库函数
- 2 使用 `fscanf` 和 `fprintf`, `fgetc` 和 `fputc`, `fgets` 和 `fputs`
- 3 父子进程之间用无名管道进行数据传送
- 4 两个用户的独立程序使用有名管道进行数据传送

3.2 Unix实用程序

改进与体会

改进

体会

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

```
1 ssize_t read(int fd, void *buf, size_t count);
2 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。

```
1 size_t fread ( void * ptr, size_t size, size_t count, FILE * stream );
2 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。

```
1 | 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)`。

```
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

```
1 // 使用文件的库函数`fread(buf, size, nitems, fp)`, `fwrite(buf, size, nitems,
2   fp)`, size取1
3 #include <sys/stat.h>
4 #include <sys/fcntl.h>
5 #include <stdio.h>
6 #include <time.h>
7 #define NUM 1
8
9 main()
10 {
11     clock_t start,end;
12     start=clock();
13     int n;
14     char buf4[NUM];
15     FILE *fp1, *fp2;
16     fp1 = fopen("file1.txt", "r");
17     fp2 = fopen("file2.txt", "w");
18     while((n=fread(buf4,NUM,1,fp1)) > 0) fwrite(buf4, NUM, n, fp2);
19     fclose(fp1);
20     fclose(fp2);
21     end=clock();
22     printf("%d\n",end-start);
}
```

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

```
8 main()
9 {
10     clock_t start,end;
11     start=clock();
12     int n;
13     char buf4[NUM];
14     FILE *fp1, *fp2;
15     fp1 = fopen("file1.txt", "r");
16     fp2 = fopen("file2.txt", "w");
17     while((n=fread(buf4,NUM,1,fp1)) > 0) fwrite(buf4, NUM, n, fp2);
18     fclose(fp1);
19     fclose(fp2);
20     end=clock();
21     printf("%d\n",end-start);
22 }
```

```
1 // 使用系统调用`read(fd, buf, nbytes)`，`write(fd, buf, nbytes)`，nbytes取1
2 #include <sys/stat.h>
3 #include <sys/fcntl.h>
4 #include <stdio.h>
5 #include <time.h>
6 #define NUM 1
7
8 main()
9 {
10     clock_t start,end;
11     start=clock();
12     int fd1, fd2;
13     int n;
14     char buf1[NUM];
15     FILE *fp1, *fp2;
16     fd1 = open("file1.txt", O_RDONLY);
17     fd2 = open("file2.txt", O_WRONLY|O_CREAT, 0644);
18     while((n=read(fd1,buf1,NUM)) > 0) write(fd2, buf1, n);
19     close(fd1);
20     close(fd2);
21     end=clock();
22     printf("%d\n",end-start);
23 }
```

```
1 // 使用系统调用`read(fd, buf, nbytes)`，`write(fd, buf, nbytes)`，nbytes取
1024
2 #include <sys/stat.h>
3 #include <sys/fcntl.h>
4 #include <stdio.h>
5 #include <time.h>
6 #define NUM 1024
7
```

```

8 main()
9 {
10     clock_t start,end;
11     start=clock();
12     int fd1, fd2;
13     int n;
14     char buf1[NUM];
15     FILE *fp1, *fp2;
16     fd1 = open("file1.txt", O_RDONLY);
17     fd2 = open("file2.txt", O_WRONLY|O_CREAT, 0644);
18     while((n=read(fd1,buf1,NUM)) > 0) write(fd2, buf1, n);
19     close(fd1);
20     close(fd2);
21     end=clock();
22     printf("%d\n",end-start);
23 }
```

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

```

1 #include <sys/stat.h>
2 #include <sys/fcntl.h>
3 #include <stdio.h>
4
5 main()
6 {
7     int n;
8     char c;
9     char buf1[256], buf2[256];
10    FILE *fp1, *fp2, *fp3, *fp4;
11    fp1 = fopen("file1.txt", "r");
12    fp2 = fopen("file2.txt", "w");
13    fp3 = fopen("file3.txt", "w");
14    fp4 = fopen("file4.txt", "w");
15    // `fscanf` 和 `fprintf`
16    while(fscanf(fp1, "%s", buf1) > 0) fprintf(fp2, "%s", buf1);
17    fclose(fp1);
18    // `fgetc` 和 `fputc`
19    fp1 = fopen("file1.txt", "r");
20    while((c=fgetc(fp1)) > 0) fputc(c, fp3);
21    fclose(fp1);
22    // `fgets` 和 `fputs`
23    fp1 = fopen("file1.txt", "r");
24    while(fgets(buf2, 1024, fp1)) fputs(buf2, fp4);
25    fclose(fp1);
26 }
```

3 父子进程之间用无名管道进行数据传送

```

1 #include <sys/stat.h>
2 #include <stdio.h>
3 #include <fcntl.h>
4 #include <sys/types.h>
5 #include<string.h>
6
7 main()
{
8
9     int fd1,fd2;
10    char buf1[1];
11    int chanel[2],n;
12    char buf[100];
13    fd1 = open("file1.txt", O_RDONLY);
14    fd2 = open("file2.txt", O_WRONLY|O_CREAT, 0644);
15    pipe(chanel);
16    if(fork())
17    {
18        close(chanel[0]);
19        while((n=read(fd1,buf1,1)) > 0) write(chanel[1], buf1, n);
20        printf("Parent sends over.\n");
21        close(chanel[1]);
22    }
23    else
24    {
25        close(chanel[1]);
26        while((n=read(chanel[0],buf,sizeof(buf))) > 0) write(fd2, buf, n);
27        printf("Child receives over.\n");
28        close(chanel[0]);
29    }
30    close(fd1);
31    close(fd2);
32 }
```

4 两个用户的独立程序使用有名管道进行数据传送

```

1 // 进程一
2 #include <stdio.h>
3 #include <fcntl.h>
4 #include <sys/stat.h>
5 #include <string.h>
6
7 main()
{
8
9     int fd, fd1, n;
10    char buf1[1];
11    fd1 = open("file1.txt", O_RDONLY);
12    mknod("fifo", S_IFIFO|0666,0);
13    fd = open("fifo", O_WRONLY);
```

```

14     while(n=read(fd1,buf1,1) > 0) write(fd, buf1, n);
15     close(fd);
16     close(fd1);
17 }

```

```

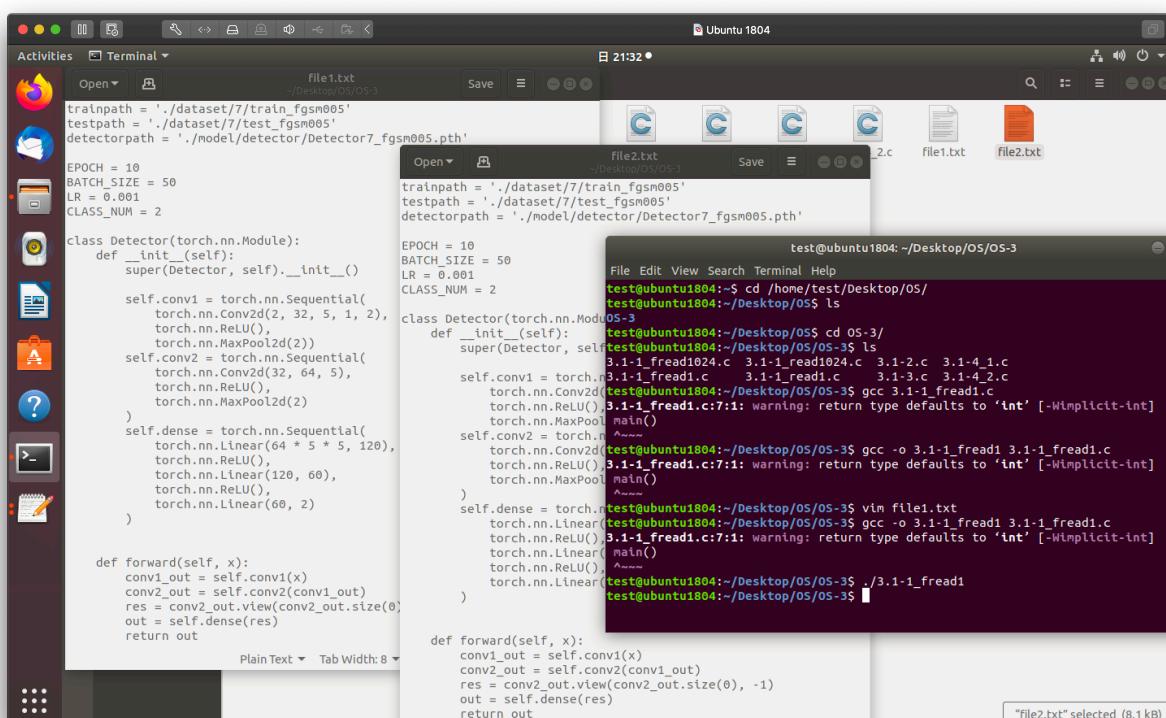
1 // 进程二
2 #include <fcntl.h>
3
4 main()
{
5     int fd, fd2, n;
6     char buf1[1];
7     fd = open("fifo", O_RDONLY);
8     fd2 = open("file2.txt", O_WRONLY|O_CREAT, 0644);
9     while((n=read(fd,buf1,sizeof(buf1))) != -1) write(fd2, buf1, n);
10    close(fd);
11    close(fd2);
12 }
13

```

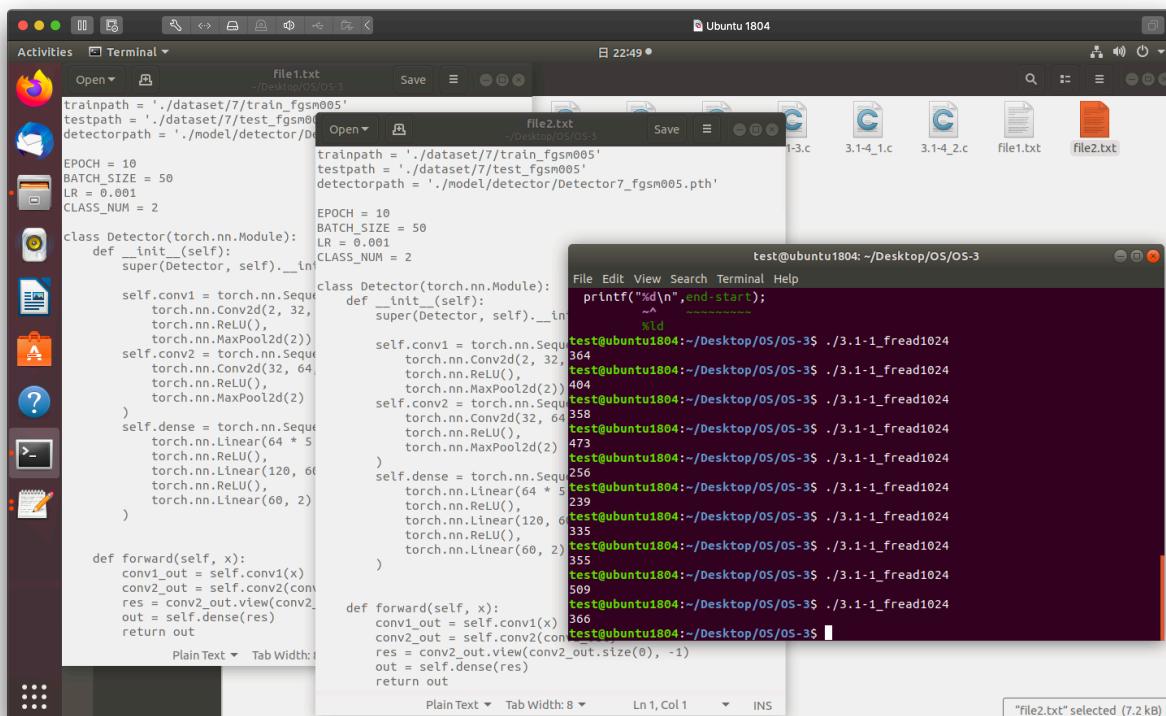
测试

1 直接使用系统调用和文件的库函数

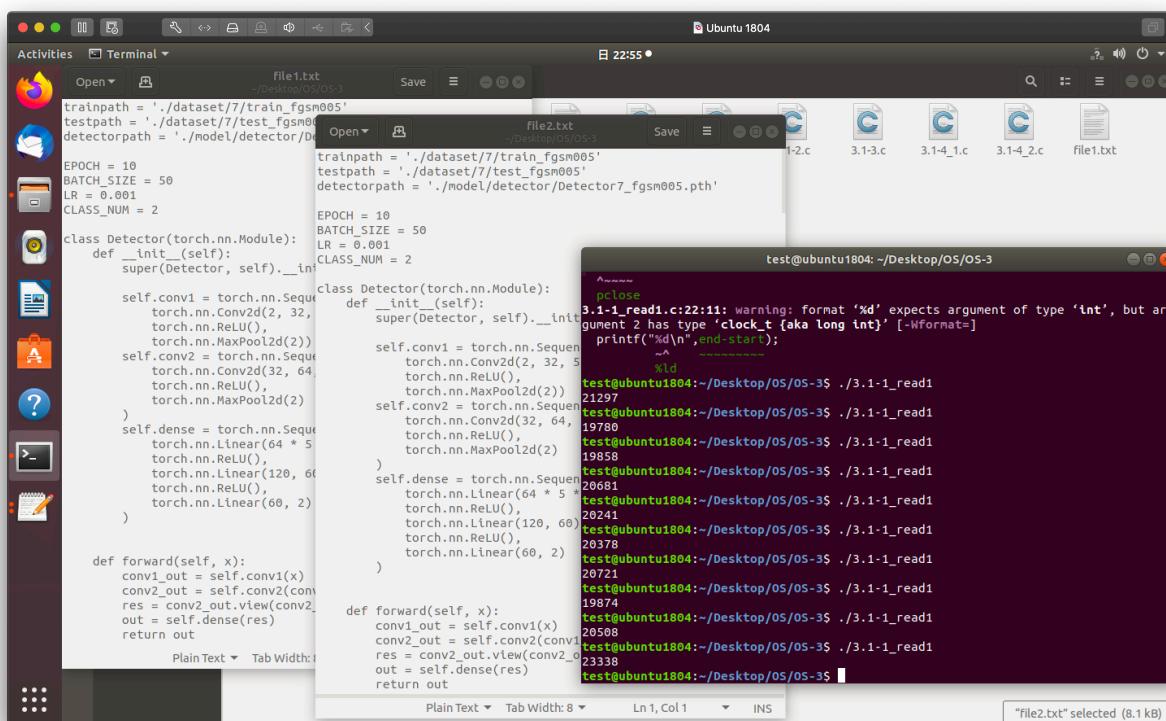
如图运行 `3.1-1_fread1`，即使用文件的库函数 `read(buf, size, nitems, fp)`, `write(buf, size, nitems, fp)`, `size` 取 1。`file1.txt` 中是一些以前的实验数据，可见被成功复制到 `file2.txt`。观察到每次执行时间都不一样，这是进程运行的不确定性导致的，于是运行十次，取平均值为 668.6ms。



运行 `3.1-1_fread1024`，即使用文件的库函数 `fclose(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。

```

trainpath = './dataset/7/train_fgsm005'
testpath = './dataset/7/test_fgsm005'
detectorpath = './model/detector/Detector7_fgsm005.pth'

EPOCH = 10
BATCH_SIZE = 50
LR = 0.001
CLASS_NUM = 2

class Detector(torch.nn.Module):
    def __init__(self):
        super(Detector, self).__init__()

        self.conv1 = torch.nn.Sequential(
            torch.nn.Conv2d(2, 32, 5, 1, 2),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(2))
        self.conv2 = torch.nn.Sequential(
            torch.nn.Conv2d(32, 64, 5, 1, 2),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(2))
        self.dense = torch.nn.Sequential(
            torch.nn.Linear(64 * 5 * 5, 120),
            torch.nn.ReLU(),
            torch.nn.Linear(120, 60),
            torch.nn.ReLU(),
            torch.nn.Linear(60, 2))

    def forward(self, x):
        conv1_out = self.conv1(x)
        conv2_out = self.conv2(conv1_out)
        res = conv2_out.view(conv2_out.size(0), -1)
        out = self.dense(res)
        return out

    PlainText ▾ Tab Width:8 ▾ Ln 1, Col 1 ▾ INS

```

```

trainpath = './dataset/7/train_fgsm005'
testpath = './dataset/7/test_fgsm005'
detectorpath = './model/detector/Detector7_fgsm005.pth'

EPOCH = 10
BATCH_SIZE = 50
LR = 0.001
CLASS_NUM = 2

class Detector(torch.nn.Module):
    def __init__(self):
        super(Detector, self).__init__()

        self.conv1 = torch.nn.Sequential(
            torch.nn.Conv2d(2, 32, 5, 1, 2),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(2))
        self.conv2 = torch.nn.Sequential(
            torch.nn.Conv2d(32, 64, 5, 1, 2),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(2))
        self.dense = torch.nn.Sequential(
            torch.nn.Linear(64 * 5 * 5, 120),
            torch.nn.ReLU(),
            torch.nn.Linear(120, 60),
            torch.nn.ReLU(),
            torch.nn.Linear(60, 2))

    def forward(self, x):
        conv1_out = self.conv1(x)
        conv2_out = self.conv2(conv1_out)
        res = conv2_out.view(conv2_out.size(0), -1)
        out = self.dense(res)
        return out

    PlainText ▾ Tab Width:8 ▾ Ln 1, Col 1 ▾ INS

```

```

3.1-1_read1024.c:22:11: warning: format ‘%d’ expects argument of type ‘int’, but argument 2 has type ‘clock_t [aka long int]’ [-Wformat=]
    printf("%d\n", end-star);
           ^
           %d
test@ubuntu1804:~/Desktop/OS/OS-3$ ./3.1-1_read1024
97
test@ubuntu1804:~/Desktop/OS/OS-3$ ./3.1-1_read1024
100
test@ubuntu1804:~/Desktop/OS/OS-3$ ./3.1-1_read1024
107
test@ubuntu1804:~/Desktop/OS/OS-3$ ./3.1-1_read1024
102
test@ubuntu1804:~/Desktop/OS/OS-3$ ./3.1-1_read1024
91
test@ubuntu1804:~/Desktop/OS/OS-3$ ./3.1-1_read1024
97
test@ubuntu1804:~/Desktop/OS/OS-3$ ./3.1-1_read1024
116
test@ubuntu1804:~/Desktop/OS/OS-3$ 
test@ubuntu1804:~/Desktop/OS/OS-3$ 

```

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。

```

trainpath = './dataset/7/train_fgsm005'
testpath = './dataset/7/test_fgsm005'
detectorpath = './model/detector/Detector7_fgsm005.pth'

EPOCH = 10
BATCH_SIZE = 50
LR = 0.001
CLASS_NUM = 2

class Detector(torch.nn.Module):
    def __init__(self):
        super(Detector, self).__init__()

        self.conv1 = torch.nn.Sequential(
            torch.nn.Conv2d(2, 32, 5, 1, 2),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(2))
        self.conv2 = torch.nn.Sequential(
            torch.nn.Conv2d(32, 64, 5, 1, 2),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(2))
        self.dense = torch.nn.Sequential(
            torch.nn.Linear(64 * 5 * 5, 120),
            torch.nn.ReLU(),
            torch.nn.Linear(120, 60),
            torch.nn.ReLU(),
            torch.nn.Linear(60, 2))

    def forward(self, x):
        conv1_out = self.conv1(x)
        conv2_out = self.conv2(conv1_out)
        res = conv2_out.view(conv2_out.size(0), -1)
        out = self.dense(res)
        return out

    PlainText ▾ Tab Width:8 ▾ Ln 1, Col 1 ▾ INS

```

```

trainpath = './dataset/7/train_fgsm005'
testpath = './dataset/7/test_fgsm005'
detectorpath = './model/detector/Detector7_fgsm005.pth'

EPOCH = 10
BATCH_SIZE = 50
LR = 0.001
CLASS_NUM = 2

class Detector(torch.nn.Module):
    def __init__(self):
        super(Detector, self).__init__()

        self.conv1 = torch.nn.Sequential(
            torch.nn.Conv2d(2, 32, 5, 1, 2),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(2))
        self.conv2 = torch.nn.Sequential(
            torch.nn.Conv2d(32, 64, 5, 1, 2),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(2))
        self.dense = torch.nn.Sequential(
            torch.nn.Linear(64 * 5 * 5, 120),
            torch.nn.ReLU(),
            torch.nn.Linear(120, 60),
            torch.nn.ReLU(),
            torch.nn.Linear(60, 2))

    def forward(self, x):
        conv1_out = self.conv1(x)
        conv2_out = self.conv2(conv1_out)
        res = conv2_out.view(conv2_out.size(0), -1)
        out = self.dense(res)
        return out

    PlainText ▾ Tab Width:8 ▾ Ln 1, Col 1 ▾ INS

```

```

3.1-2.c:5:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
    main()
    ^
    x:
    conv1_out = self.conv1(x)
    conv2_out = self.conv2(conv1_out)
    res = conv2_out.view(conv2_out.size(0), -1)
    out = self.dense(res)
    return out

```

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`。

```

File1.txt content:
trainpath = './dataset//train_fgsm005'
testpath = './dataset//test_fgsm005'
detectorpath = './model/detector/Detector7_fgsm005.pth'

EPOCH = 10
BATCH_SIZE = 50
LR = 0.001
CLASS_NUM = 2

class Detector(torch.nn.Module):
    def __init__(self):
        super(Detector, self).__init__()

        self.conv1 = torch.nn.Sequential(
            torch.nn.Conv2d(2, 32, 5, 1, 2),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(2))
        self.conv2 = torch.nn.Sequential(
            torch.nn.Conv2d(32, 64, 5),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(2))
    )
    self.dense = torch.nn.Sequential(
        torch.nn.Linear(64 * 5 * 5, 120),
        torch.nn.ReLU(),
        torch.nn.Linear(120, 60),
        torch.nn.ReLU(),
        torch.nn.Linear(60, 2)
    )

    def forward(self, x):
        conv1_out = self.conv1(x)
        conv2_out = self.conv2(conv1_out)
        res = conv2_out.view(conv2_out.size(0), -1)
        out = self.dense(res)
        return out

```

```

File2.txt content:
trainpath = './dataset//train_fgsm005'
testpath = './dataset//test_fgsm005'
detectorpath = './model/detector/Detector7_fgsm005.pth'

EPOCH = 10
BATCH_SIZE = 50
LR = 0.001
CLASS_NUM = 2

class Detector(torch.nn.Module):
    def __init__(self):
        super(Detector, self).__init__()

        self.conv1 = torch.nn.Sequential(
            torch.nn.Conv2d(2, 32, 5, 1, 2),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(2))
        self.conv2 = torch.nn.Sequential(
            torch.nn.Conv2d(32, 64, 5),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(2))
    )
    self.dense = torch.nn.Sequential(
        torch.nn.Linear(64 * 5 * 5, 120),
        torch.nn.ReLU(),
        torch.nn.Linear(120, 60),
        torch.nn.ReLU(),
        torch.nn.Linear(60, 2)
    )

    def forward(self, x):
        conv1_out = self.conv1(x)
        conv2_out = self.conv2(conv1_out)
        res = conv2_out.view(conv2_out.size(0), -1)
        out = self.dense(res)
        return out

```

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`。

```

File1.txt content:
trainpath = './dataset//train_fgsm005'
testpath = './dataset//test_fgsm005'
detectorpath = './model/detector/Detector7_fgsm005.pth'

EPOCH = 10
BATCH_SIZE = 50
LR = 0.001
CLASS_NUM = 2

class Detector(torch.nn.Module):
    def __init__(self):
        super(Detector, self).__init__()

        self.conv1 = torch.nn.Sequential(
            torch.nn.Conv2d(2, 32, 5, 1, 2),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(2))
        self.conv2 = torch.nn.Sequential(
            torch.nn.Conv2d(32, 64, 5),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(2))
    )
    self.dense = torch.nn.Sequential(
        torch.nn.Linear(64 * 5 * 5, 120),
        torch.nn.ReLU(),
        torch.nn.Linear(120, 60),
        torch.nn.ReLU(),
        torch.nn.Linear(60, 2)
    )

    def forward(self, x):
        conv1_out = self.conv1(x)
        conv2_out = self.conv2(conv1_out)
        res = conv2_out.view(conv2_out.size(0), -1)
        out = self.dense(res)
        return out

```

```

File2.txt content:
trainpath = './dataset//train_fgsm005'
testpath = './dataset//test_fgsm005'
detectorpath = './model/detector/Detector7_fgsm005.pth'

EPOCH = 10
BATCH_SIZE = 50
LR = 0.001
CLASS_NUM = 2

class Detector(torch.nn.Module):
    def __init__(self):
        super(Detector, self).__init__()

        self.conv1 = torch.nn.Sequential(
            torch.nn.Conv2d(2, 32, 5, 1, 2),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(2))
        self.conv2 = torch.nn.Sequential(
            torch.nn.Conv2d(32, 64, 5),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(2))
    )
    self.dense = torch.nn.Sequential(
        torch.nn.Linear(64 * 5 * 5, 120),
        torch.nn.ReLU(),
        torch.nn.Linear(120, 60),
        torch.nn.ReLU(),
        torch.nn.Linear(60, 2)
    )

    def forward(self, x):
        conv1_out = self.conv1(x)
        conv2_out = self.conv2(conv1_out)
        res = conv2_out.view(conv2_out.size(0), -1)
        out = self.dense(res)
        return out

```

3.2 Unix实用程序

- 上机练习 `ls`, `ln`, `ln -s`, `diff`, `file`, `chown`, `chmod`, `head`, `tail`, `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。

The screenshot shows a desktop environment with a terminal window and a file manager window. The terminal window displays the following command sequence:

```
test@ubuntu1804: ~/Desktop/OS/OS-3/workbench
total 0
-rw-r--r-- 1 test test 0 6月 8 00:11 ls.save

The file manager window shows a directory structure with several files and folders, including 3.1-1_fread1, 3.1-1_read1, 3.1-2, 3.1-4.1, file1.txt, 3.1-1_fread1024, 3.1-1_read1024, 3.1-2.c, 3.1-4.1.c, file1.txt, 3.1-1_fread1024.c, 3.1-1_read1024.c, 3.1-3, 3.1-4.2, file2.txt, 3.1-1_fread1.c, 3.1-1_read1.c, 3.1-3.c, 3.1-4.2.c, ls.save.
```

(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。

```
ls.save
test@ubuntu1804:~/Desktop/OS/OS-3/workbench$ ls -l
total 4
-rw-r--r-- 1 test test 56 6月 8 00:09 ls.save
test@ubuntu1804:~/Desktop/OS/OS-3/workbench$ ls -l > ls.save
total 0
test@ubuntu1804:~/Desktop/OS/OS-3/workbench$ ls
ls.save
test@ubuntu1804:~/Desktop/OS/OS-3/workbench$ ls *
ls.save
test@ubuntu1804:~/Desktop/OS/OS-3/workbench$ ls .
ls.save
test@ubuntu1804:~/Desktop/OS/OS-3/workbench$ ls .*
.:
ls.save
.:
3.1-1_fread1 3.1-1_read1 3.1-2 3.1-4_1 fifo workbench
3.1-1_fread1024 3.1-1_read1024 3.1-2.c 3.1-4_1.c file1.txt
3.1-1_fread1024.c 3.1-1_read1024.c 3.1-3 3.1-4_2 file2.txt
3.1-1_fread1.c 3.1-1_read1.c 3.1-3.c 3.1-4_2.c ls.save
test@ubuntu1804:~/Desktop/OS/OS-3/workbench$ ls .?*
3.1-1_fread1 3.1-1_read1 3.1-2 3.1-4_1 fifo workbench
3.1-1_fread1024 3.1-1_read1024 3.1-2.c 3.1-4_1.c file1.txt
3.1-1_fread1024.c 3.1-1_read1024.c 3.1-3 3.1-4_2 file2.txt
3.1-1_fread1.c 3.1-1_read1.c 3.1-3.c 3.1-4_2.c ls.save
test@ubuntu1804:~/Desktop/OS/OS-3/workbench$
```

"ls.save" selected (56 bytes)

(3) 显示某一文件从 n_1 行到 n_2 行的内容。

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

```
ls.save
test@ubuntu1804:~/Desktop/OS/OS-3/workbench$ ls .*
.:
ls.save

.:
3.1-1_fread1 3.1-1_read1 3.1-2 3.1-4_1 fifo workbench
3.1-1_fread1024 3.1-1_read1024 3.1-2.c 3.1-4_1.c file1.txt
3.1-1_fread1024.c 3.1-1_read1024.c 3.1-3 3.1-4_2 file2.txt
3.1-1_fread1.c 3.1-1_read1.c 3.1-3.c 3.1-4_2.c ls.save
test@ubuntu1804:~/Desktop/OS/OS-3/workbench$ ls .?*
3.1-1_fread1 3.1-1_read1 3.1-2 3.1-4_1 fifo workbench
3.1-1_fread1024 3.1-1_read1024 3.1-2.c 3.1-4_1.c file1.txt
3.1-1_fread1024.c 3.1-1_read1024.c 3.1-3 3.1-4_2 file2.txt
3.1-1_fread1.c 3.1-1_read1.c 3.1-3.c 3.1-4_2.c ls.save
test@ubuntu1804:~/Desktop/OS/OS-3/workbench$ cp ./file1.txt file1.txt
test@ubuntu1804:~/Desktop/OS/OS-3/workbench$ awk 'NR>=4 &&NR<=16' file1.txt
test@ubuntu1804:~/Desktop/OS/OS-3/workbench$
```

(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。

```
1 | mv file1 file2
```

```
1 | 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。

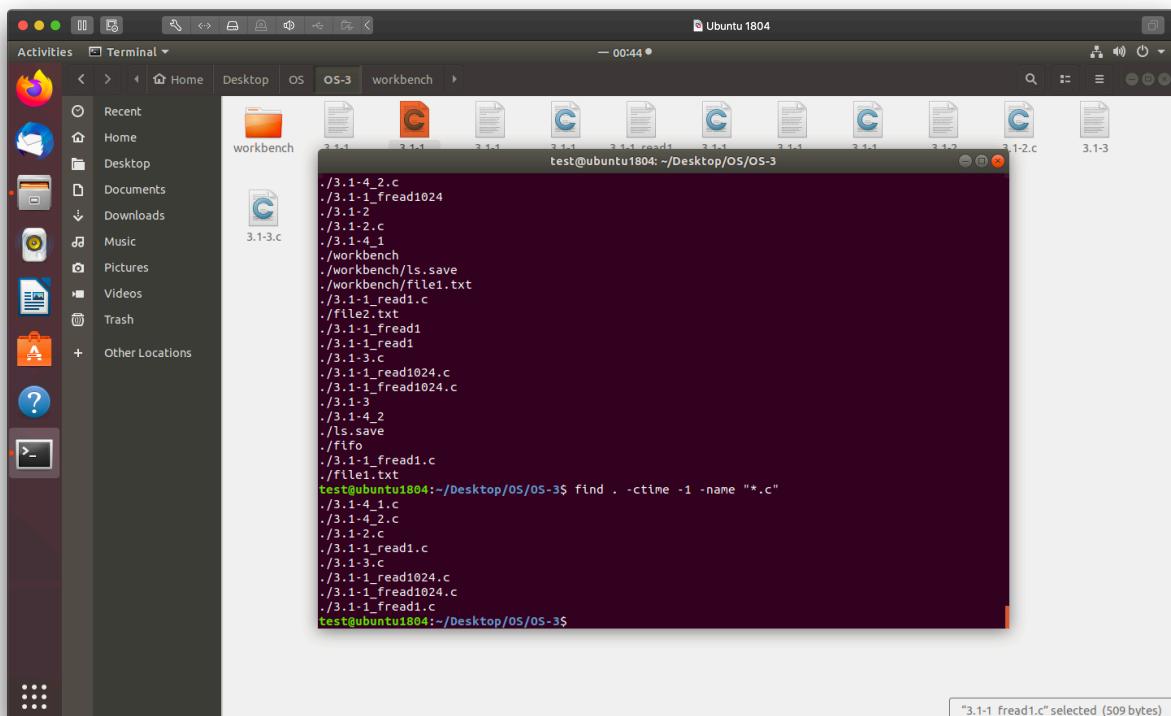
`mv` 命令是经常用来将数据从一个地方挪到另外一个地方的工具，而 `mv` 命令在挪动数据的时候底层工作分为两种情况：

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。

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

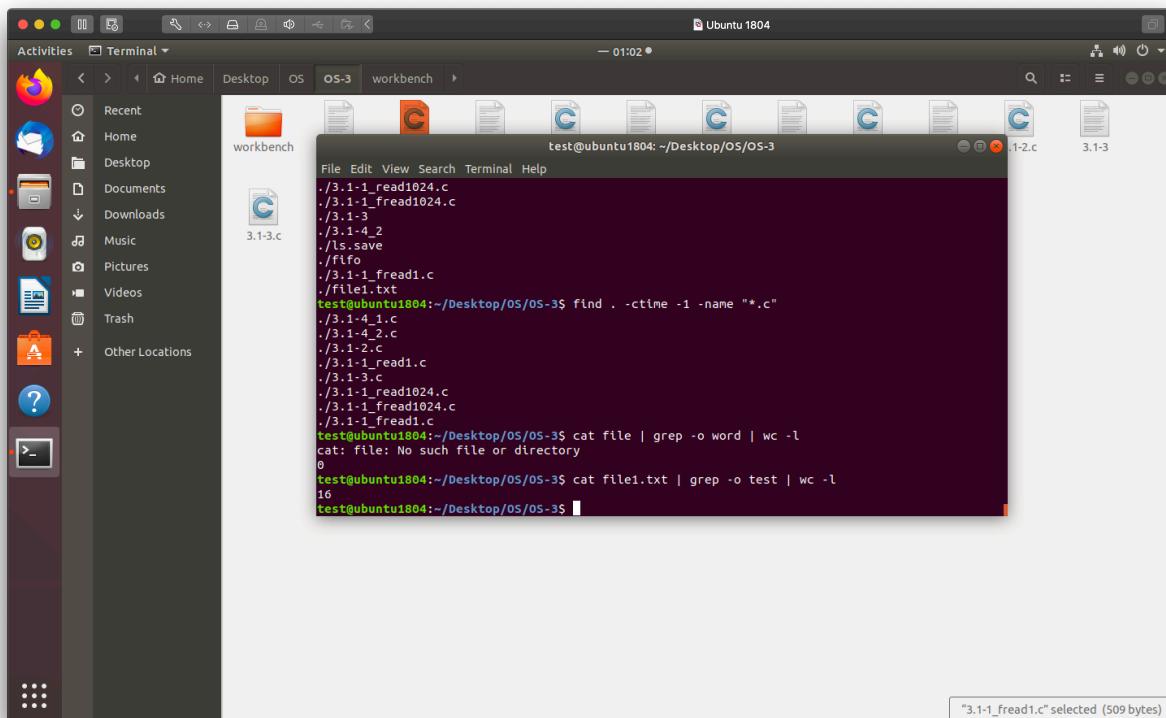


掌握正则表达式的匹配规则，上机练习 `grep`, `sed`, `awk` 等命令。

(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。

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



上机练习 `date`, `df`, `du`, `tar` 命令。

`date` 格式化输出:

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

`date` 输出昨天日期:

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

`date` 格式转换:

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

`df`

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

```
File Edit View Search Terminal Help
0
test@ubuntu1804:~/Desktop/OS/OS-3$ cat file1.txt | grep -o test | wc -l
16
test@ubuntu1804:~/Desktop/OS/OS-3$ date +"%Y-%m-%d"
2020-06-08
test@ubuntu1804:~/Desktop/OS/OS-3$ date -d "1 day ago" +"%Y-%m-%d"
2020-06-07
test@ubuntu1804:~/Desktop/OS/OS-3$ date -d "Dec 5, 2020 12:00:37 AM 2 year ago" +"%Y-%m-%d %H:%M.%S"
2018-12-05 00:00:37
test@ubuntu1804:~/Desktop/OS/OS-3$ df
Filesystem 1K-blocks Used Available Use% Mounted on
udev 1985524 0 1985524 0% /dev
tmpfs 401584 2056 395928 1% /run
/dev/sda1 20509264 9511200 9933208 49% /
tmpfs 2007920 0 2007920 0% /dev/shm
tmpfs 5120 4 5116 1% /run/lock
tmpfs 2007920 0 2007920 0% /sys/fs/cgroup
/dev/loop0 12544 12544 0 100% /snap/gnome-characters/69
/dev/loop1 1664 1664 0 100% /snap/gnome-calculator/154
/dev/loop2 3456 3456 0 100% /snap/gnome-system-monitor/36
/dev/loop3 88764 88764 0 100% /snap/core/4486
/dev/loop4 143488 143488 0 100% /snap/gnome-3-26-1604/59
/dev/loop5 21504 21504 0 100% /snap/gnome-logs/25
tmpfs 401584 16 401568 1% /run/user/120
tmpfs 401584 48 401536 1% /run/user/1000
test@ubuntu1804:~/Desktop/OS/OS-3$ df -i
Filesystem Inodes IUsed IFree IUse% Mounted on
udev 496381 439 495942 1% /dev
tmpfs 501980 937 501043 1% /run
/dev/sda1 1310720 210461 1100259 17% /
tmpfs 501980 1 501979 1% /dev/shm
tmpfs 501980 7 501973 1% /run/lock
tmpfs 501980 18 501962 1% /sys/fs/cgroup
/dev/loop0 1585 1585 0 100% /snap/gnome-characters/69
/dev/loop1 1244 1244 0 100% /snap/gnome-calculator/154
/dev/loop2 669 669 0 100% /snap/gnome-system-monitor/36
/dev/loop3 12819 12819 0 100% /snap/core/4486
/dev/loop4 27660 27660 0 100% /snap/gnome-3-26-1604/59
/dev/loop5 1783 1783 0 100% /snap/gnome-logs/25
tmpfs 501980 25 501955 1% /run/user/120
tmpfs 501980 39 501941 1% /run/user/1000
test@ubuntu1804:~/Desktop/OS/OS-3$
```

du

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

tar

```
1 # 压缩文件，非打包
2 tar -czvf workbench.tar.gz workbench
3 # 列出压缩文件内容
4 tar -tzvf workbench.tar.gz
5 # 解压文件
6 tar -xzvf workbench.tar.gz
```

```
test@ubuntu1804:~/Desktop/OS/OS-3$ df -i
Filesystem      Inodes IUsed  IFree IUse% Mounted on
udev            496381   439 495943    1% /dev
tmpfs           501980   937 501043    1% /run
/dev/sda1        1310720 210461 1100259   17% /
tmpfs           501980     1 501979    1% /dev/shm
tmpfs           501980     7 501973    1% /run/lock
tmpfs           501980    18 501962    1% /sys/fs/cgroup
/dev/loop0       1585   1585     0 100% /snap/gnome-characters/69
/dev/loop1       1244   1244     0 100% /snap/gnome-calculator/154
/dev/loop2       669    669     0 100% /snap/gnome-system-monitor/36
/dev/loop3       12819  12819     0 100% /snap/core/4486
/dev/loop4       27660  27660     0 100% /snap/gnome-3-26-1604/59
/dev/loop5       1783   1783     0 100% /snap/gnome-logs/25
tmpfs           501980    25 501955    1% /run/user/100
tmpfs           501980    39 501941    1% /run/user/1000
test@ubuntu1804:~/Desktop/OS/OS-3$ du -sh
168K .
test@ubuntu1804:~/Desktop/OS/OS-3$ du -sh workbench
16K workbench
test@ubuntu1804:~/Desktop/OS/OS-3$ ls
3.1-1_freadi  3.1-1_freadi024.c 3.1-1_read1  3.1-1_readi024.c 3.1-2 3.1-3 3.1-4_1 3.1-4_2  fifo  file2.txt workbench
3.1-1_freadi024 3.1-1_freadi.c 3.1-1_readi024 3.1-1_readi.c 3.1-2.c 3.1-3.c 3.1-4_1.c 3.1-4_2.c file1.txt ls.save
test@ubuntu1804:~/Desktop/OS/OS-3$ tar -czvf workbench.tar.gz workbench
workbench/
workbench/ls.save
workbench/file1.txt
test@ubuntu1804:~/Desktop/OS/OS-3$ ls
3.1-1_freadi  3.1-1_freadi024.c 3.1-1_read1  3.1-1_readi024.c 3.1-2 3.1-3 3.1-4_1 3.1-4_2  fifo  file2.txt workbench
3.1-1_freadi024 3.1-1_freadi.c 3.1-1_readi024 3.1-1_readi.c 3.1-2.c 3.1-3.c 3.1-4_1.c 3.1-4_2.c file1.txt ls.save workbench.tar.gz
test@ubuntu1804:~/Desktop/OS/OS-3$ tar -czvf workbench.tar.gz workbench/
workbench/
workbench/ls.save
workbench/file1.txt
test@ubuntu1804:~/Desktop/OS/OS-3$ ls
3.1-1_freadi  3.1-1_freadi024.c 3.1-1_read1  3.1-1_readi024.c 3.1-2 3.1-3 3.1-4_1 3.1-4_2  fifo  file2.txt workbench
3.1-1_freadi024 3.1-1_freadi.c 3.1-1_readi024 3.1-1_readi.c 3.1-2.c 3.1-3.c 3.1-4_1.c 3.1-4_2.c file1.txt ls.save workbench.tar.gz
test@ubuntu1804:~/Desktop/OS/OS-3$
```

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。

```
test@ubuntu1804:~/Desktop/OS/OS-3$ sudo apt install <deb name>
Try: sudo apt install <deb name>
test@ubuntu1804:~/Desktop/OS/OS-3$ sudo apt install nohub
[sudo] password for test:
Sorry, try again.
[sudo] password for test:
E: dpkg was interrupted, you must manually run 'sudo dpkg --configure -a' to correct the problem.
test@ubuntu1804:~/Desktop/OS/OS-3$ sudo dpkg --configure -a
Setting up mysql-server-5.7 (5.7.30-0ubuntu0.18.04.1) ...
Checking if update is needed.
This installation of MySQL is already upgraded to 5.7.30, use --force if you still need to run mysql_upgrade
*dpkg: error processing package mysql-server-5.7 (--configure):
 installed mysql-server-5.7 package post-installation script subprocess was interrupted
 Processing triggers for systemd (237-3ubuntu10.39) ...
 Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
 Processing triggers for ureadahead (0.100.0-21) ...
 Errors were encountered while processing:
 mysql-server-5.7
test@ubuntu1804:~/Desktop/OS/OS-3$ sudo apt install nohub
Reading package lists... Done
Building dependency tree
Reading state information... Done
E: Unable to locate package nohub
test@ubuntu1804:~/Desktop/OS/OS-3$ who
test  :0        2020-06-04 20:21 (:0)
test@ubuntu1804:~/Desktop/OS/OS-3$ ps
 PID TTY      TIME CMD
 5034 pts/0    00:00:00 bash
 7559 pts/0    00:00:00 ps
test@ubuntu1804:~/Desktop/OS/OS-3$ time
real 0m0.000s
user 0m0.000s
sys 0m0.000s
test@ubuntu1804:~/Desktop/OS/OS-3$
```

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。

```
workbench — bash — 104x33
Error: No similarly named formulae found.
==> Searching taps...
==> Searching taps on GitHub...
Warning: Error searching on GitHub: curl failed! % Total % Received % Xferd Average Speed Time
Time Time Current Dload Upload Total Spent Left Speed
0 0 0 0 0 0 0 --:--:-- --:--:-- --:--:-- 0
curl: (7) Failed to connect to api.github.com port 443: Connection refused

Error: No formulae found in taps.
(base) liuhawendeMacBook-Pro:~ hongxing$ compress
^C
((base) liuhawendeMacBook-Pro:~ hongxing$ compress -v
^C
((base) liuhawendeMacBook-Pro:~ hongxing$ cd Desktop/
((base) liuhawendeMacBook-Pro:Desktop hongxing$ compress LeNet.py
((base) liuhawendeMacBook-Pro:Desktop hongxing$ cd workbench/
((base) liuhawendeMacBook-Pro:workbench hongxing$ ls
LeNet.py
((base) liuhawendeMacBook-Pro:workbench hongxing$ compress LeNet.py
((base) liuhawendeMacBook-Pro:workbench hongxing$ ls
LeNet.py.Z
((base) liuhawendeMacBook-Pro:workbench hongxing$ compress -d LeNet.py.Z
((base) liuhawendeMacBook-Pro:workbench hongxing$ ls
LeNet.py
((base) liuhawendeMacBook-Pro:workbench hongxing$ gzip *
((base) liuhawendeMacBook-Pro:workbench hongxing$ ls
LeNet.py.gz
((base) liuhawendeMacBook-Pro:workbench hongxing$ gzip -dv *
LeNet.py.gz: 67.2% -- replaced with LeNet.py
((base) liuhawendeMacBook-Pro:workbench hongxing$ ls
LeNet.py
(base) liuhawendeMacBook-Pro:workbench hongxing$
```

上机练习 `man`, `cd`, `history` 命令。

1 | `man 1 cd`

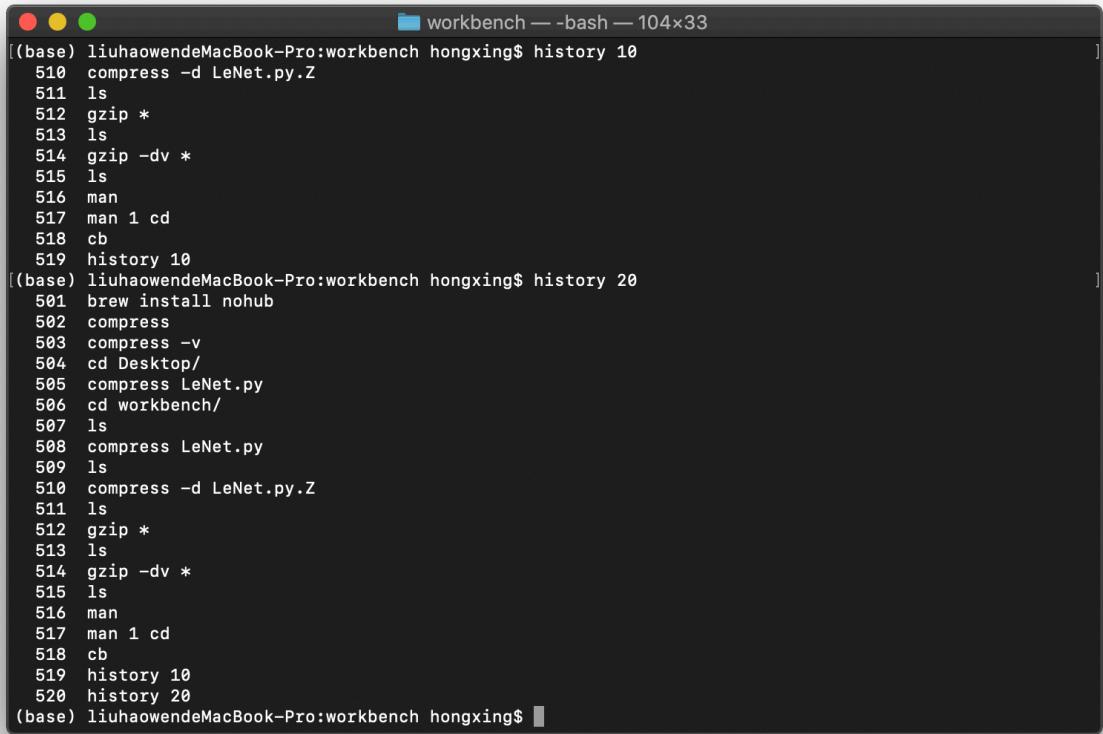
```
workbench — less - man 1 cd — 104x33
BUILTIN(1) BSD General Commands Manual BUILTIN(1)
NAME
builtin, !, %, ., :, @, {, }, alias, alloc, bg, bind, bindkey, break, breaksw, builtins,
case, cd, chdir, command, complete, continue, default, dirs, do, done, echo, echotc,
elif, else, end, endif, endsw, esac, eval, exec, exit, export, false, fc, fg, filetest,
fi, for, foreach, getopts, glob, goto, hash, hashstat, history, hup, if, jobid, jobs,
kill, limit, local, log, login, logout, ls-F, nice, nohup, notify, onintr, popd,
printenv, pushd, pwd, read, readonly, rehash, repeat, return, sched, set, setenv, settc,
setty, setvar, shift, source, stop, suspend, switch, telltc, test, then, time, times,
trap, true, type, ulimit, umask, unalias, uncomplete, unhash, unlimit, unset, unsetenv,
until, wait, where, which, while -- shell built-in commands

SYNOPSIS
builtin [-options] [args ...]

DESCRIPTION
Shell builtin commands are commands that can be executed within the running shell's
process. Note that, in the case of csh(1) builtin commands, the command is executed in
a subshell if it occurs as any component of a pipeline except the last.

If a command specified to the shell contains a slash `'', the shell will not execute a
builtin command, even if the last component of the specified command matches the name of
a builtin command. Thus, while specifying ``echo'' causes a builtin command to be exe-
cuted under shells that support the echo builtin command, specifying ``/bin/echo'' or
``./echo'' does not.

While some builtin commands may exist in more than one shell, their operation may be
different under each shell which supports them. Below is a table which lists shell
builtin commands, the standard shells that support them and whether they exist as stand-
alone utilities.
```



```
workbench — bash — 104x33
(base) liuhaowendeMacBook-Pro:workbench hongxing$ history 10
510 compress -d LeNet.py.Z
511 ls
512 gzip *
513 ls
514 gzip -dv *
515 ls
516 man
517 man 1 cd
518 cb
519 history 10
(base) liuhaowendeMacBook-Pro:workbench hongxing$ history 20
501 brew install nohub
502 compress
503 compress -v
504 cd Desktop/
505 compress LeNet.py
506 cd workbench/
507 ls
508 compress LeNet.py
509 ls
510 compress -d LeNet.py.Z
511 ls
512 gzip *
513 ls
514 gzip -dv *
515 ls
516 man
517 man 1 cd
518 cb
519 history 10
520 history 20
(base) liuhaowendeMacBook-Pro:workbench hongxing$
```

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