

实验五：管道通信

1. 阅读以下程序：

```
#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

#include "apue.h"

main()
{
    int filedes[2];

    char buffer[80];

    if (pipe(filedes) < 0)
        err_quit("pipe error");

    if (fork() > 0)
    {
        char s[] = "hello!\n";

        close(filedes[0]);

        write(filedes[1], s, sizeof(s));

        close(filedes[1]);
    }
    else
    {
        close(filedes[1]);

        read(filedes[0], buffer, 80);

        printf("%s", buffer);

        close(filedes[0]);
    }
}
```

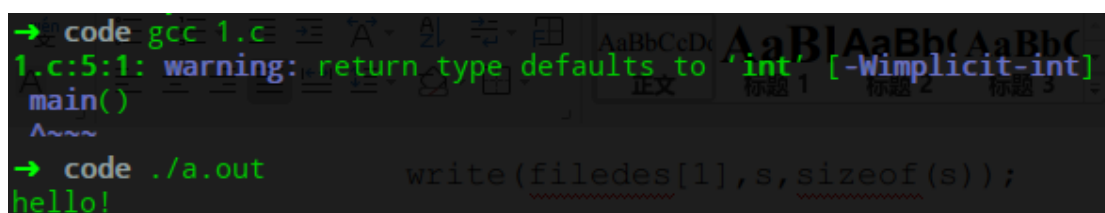
编译并运行程序，分析程序执行过程和结果，注释程序主要语句。

答:

程序代码注释如下:

```
1  #include <unistd.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include "apue.h"
5  main()
6  {
7      int filedес[2]; //filedes[0]为管道里的读取端,
                        filedес[1]为管道的写入端
8      char buffer[80];
9      if (pipe(filedes) < 0) //成功返回 0, 失败返回-1, 错误
                            原因存于 errno 中
10         err_quit("pipe error");
11     if (fork() > 0) //父进程
12     {
13         char s[] = "hello!\n";
14         close(filedes[0]); //关闭父进程读取端
15         write(filedes[1], s, sizeof(s)); //写入数据
16         close(filedes[1]); //关闭父进程写入端
17     }
18     else //子进程
19     {
20         close(filedes[1]); //关闭子进程写入端
21         read(filedes[0], buffer, 80); //读取数据
22         printf("%s", buffer); //输出读取的内容
23         close(filedes[0]); //关闭子进程读取端
24     }
25 }
```

这段代码使用无名管道, 通过从父进程中写入字符串数据"hello!\n", 并在子进程中
在屏幕上打印接收到的数据, 用于测试管道通信, 其执行效果如下:



```
→ code gcc 1.c
1.c:5:1: warning: return type defaults to 'int' [-Wimplicit-int]
main()
~~~~
→ code ./a.out
hello!
```

2. 阅读以下程序:

```

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>

main()
{
    char buffer[80];

    int fd;

    unlink("FIFO");

    mkfifo("FIFO", 0666); //FIFO 是管道名, 0666 是权限

    if (fork() > 0)
    {
        char s[] = "hello !\n";

        fd = open("FIFO", O_WRONLY);

        write(fd, s, sizeof(s));

        close(fd);
    }
    else
    {
        fd = open("FIFO", O_RDONLY);

        read(fd, buffer, 80);

        printf("%s", buffer);

        close(fd);
    }
}

```

编译并运行程序，分析程序执行过程和结果，注释程序主要语句。

答：

程序代码注释如下：

<pre> 1 #include <sys/types.h> 2 #include <sys/stat.h> </pre>

```

3  #include <fcntl.h>
4  #include <unistd.h>
5  main()
6  {
7      char buffer[80];
8      int fd;
9      unlink("FIFO"); //如果已经存在 FIFO 文件则先删除
10     mkfifo("FIFO", 0666); //创建权限为 0666 的名称为 FIFO
    的命名管道
11     if (fork() > 0) //父进程
12     {
13         char s[] = "hello !\n";
14         fd = open("FIFO", O_WRONLY); //以只写方式打开
        FIFO 命名管道
15         write(fd, s, sizeof(s)); //写入数据 s
16         close(fd); //关闭命名管道
17     }
18     else //子进程
19     {
20         fd = open("FIFO", O_RDONLY); //以只读方式打开
        FIFO 命名管道
21         read(fd, buffer, 80); //读出数据存入 buffer
22         printf("%s", buffer); //打印 buffer
23         close(fd); //关闭命名管道
24     }
25 }

```

这段代码使用有名管道，创建了一个名为 FIFO 的命名管道，通过从父进程中写入字符串数据“hello !\n”，并在子进程中在屏幕上打印接收到的数据，用于测试管道通信，其执行效果如下：

```

→ code gcc 2.c
2.c:5:1: warning: return type defaults to 'int' [-Wimplicit-int]
main()
~~~~~
2.c: In function 'main':
2.c:22:9: warning: implicit declaration of function 'printf' [-Wimplicit-function-declaration]
printf("%s", buffer); //打印buffer
~~~~~
2.c:22:9: warning: incompatible implicit declaration of built-in function 'printf'
2.c:22:9: note: include '<stdio.h>' or provide a declaration of 'printf'
→ code ./a.out
hello !

```

3. 阅读以下程序：

```
#include<stdio.h>
```

```
main()
{
    FILE * fp;

    char buffer[80];

    fp=popen("cat /etc/passwd","r");

    fgets(buffer,sizeof(buffer),fp);

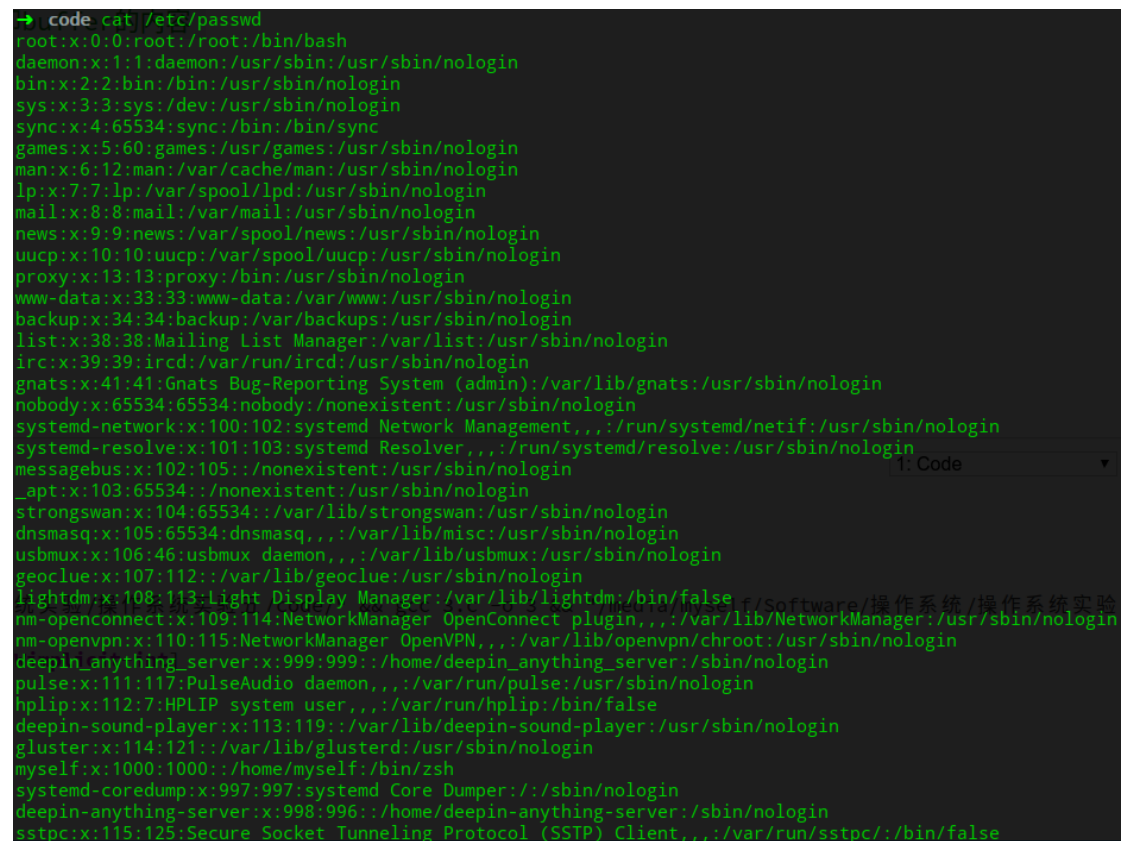
    printf("%s",buffer);

    pclose(fp);
}
```

编译并运行程序，分析程序执行过程和结果，注释程序主要语句。

答：

Shell 命令 `cat /etc/passwd` 的输出结果如下：



```
➔ code cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mail List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin)/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-network:x:100:102:systemd Network Management,,,:/run/systemd/netif:/usr/sbin/nologin
systemd-resolve:x:101:103:systemd Resolver,,,:/run/systemd/resolve:/usr/sbin/nologin
messagebus:x:102:105:,:/nonexistent:/usr/sbin/nologin
_apt:x:103:65534:,:/nonexistent:/usr/sbin/nologin
strongswan:x:104:65534:./var/lib/strongswan:/usr/sbin/nologin
dnsmasq:x:105:65534:dnsmasq,,,:/var/lib/misc:/usr/sbin/nologin
usbmux:x:106:46:usbmux daemon,,,:/var/lib/usbmux:/usr/sbin/nologin
geoclue:x:107:112:./var/lib/geoclue:/usr/sbin/nologin
lightdm:x:108:113:Light Display Manager:/var/lib/lightdm:/bin/false
nm-openconnect:x:109:114:NetworkManager OpenConnect plugin,,,:/var/lib/NetworkManager:/usr/sbin/nologin
nm-openvpn:x:110:115:NetworkManager OpenVPN,,,:/var/lib/openvpn/chroot:/usr/sbin/nologin
deepin-anything-server:x:999:999:./home/deepin-anything-server:/sbin/nologin
pulse:x:111:117:PulseAudio daemon,,,:/var/run/pulse:/usr/sbin/nologin
hplip:x:112:7:HPLIP system user,,,:/var/run/hplip:/bin/false
deepin-sound-player:x:113:119:./var/lib/deepin-sound-player:/usr/sbin/nologin
gluster:x:114:121:./var/lib/glusterd:/usr/sbin/nologin
myself:x:1000:1000:./home/myself:/bin/zsh
systemd-coredump:x:997:997:systemd Core Dumper:./sbin/nologin
deepin-anything-server:x:998:996:./home/deepin-anything-server:/sbin/nologin
sstpc:x:115:125:Secure Socket Tunneling Protocol (SSTP) Client,,,:/var/run/sstpc:/bin/false
```

程序代码注释如下：

```
1 #include<stdio.h>
2 main()
3 {
```

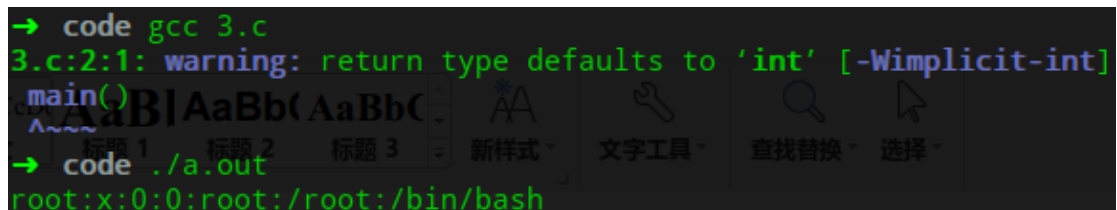
```

4     FILE * fp;
5     char buffer[80];
6     fp=popen("cat /etc/passwd","r");    //子进程执行 cat
    /etc/passwd, 建立管道 I/O 连接到子进程标准输出设备
7     fgets(buffer,sizeof(buffer),fp);    //读取一行内容
8     printf("%s",buffer);    //打印 buffer 的内容
9     pclose(fp); //关闭管道 I/O
10 }

```

这段程序使用 popen 建立管道 I/O 连接到执行 cat /etc/passwd 命令的子进程的标准输出设备，并使用 fgets 函数从中读取一行内容打印在屏幕上。

程序的输出结果如下：



```

→ code gcc 3.c
3.c:2:1: warning: return type defaults to 'int' [-Wimplicit-int]
main()
AaBb(AaBbC
→ code ./a.out
root:x:0:0:root:/root:/bin/bash

```

4. 编写一个程序，读取一个数据文件，对每一个数据进行某种运算，再在屏幕输出计算结果。要求以上工作用两个进程实现，父进程负责读文件和显示，子进程进行计算，进程间通信使用无名管道。（使用系统调用）

答：

程序代码如下：

```

1  #include <stdio.h>
2  #include <unistd.h>
3  #include <math.h>
4  #include <sys/wait.h>
5  #include "apue.h"
6  int main(void)
7  {
8      int fda[2], fdb[2];
9      int pid, i;
10     if(pipe(fda) < 0 || pipe(fdb) < 0)    //建立无名管道
11         err_quit("pipe error");
12     while((pid = fork()) == -1) ;    //创建子进程
13
14     if(pid > 0) //父进程
15     {

```

```

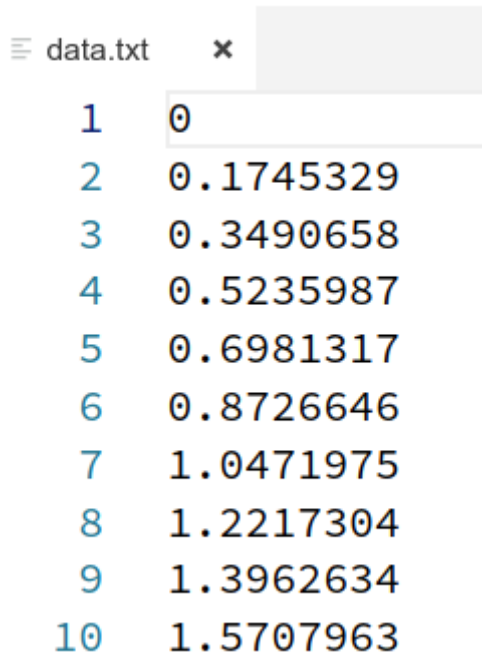
16     FILE *fpread = fopen("data.txt", "r"); //打开
    文件 data.txt
17     float data, result;
18     if(fpread == NULL) //打开文件失败
19     {
20         perror("open file error!\n");
21         exit(1);
22     }
23     close(fda[0]); //关闭父进程管道 a 读取端
24     close(fdb[1]); //关闭父进程管道 b 写入端
25     for(i = 0; i < 10; i++)
26     {
27         fscanf(fpread, "%f", &data); //读入一个数
    数据存入 data
28         write(fda[1], &data, sizeof(data)); //data
    通过管道 a 发送给子进程
29         read(fdb[0], &result, sizeof(result)); //
    通过管道 b 读计算结果到 result
30         printf("sin(%f)=%f\n", data, result);
31     }
32     close(fdb[0]); //关闭父进程管道 b 读取端
33     close(fda[1]); //关闭父进程管道 a 写入端
34     fclose(fpread); //关闭文件 data.txt
35 }
36 else //子进程
37 {
38     float data, result;
39     close(fda[1]); //关闭子进程管道 a 写入端
40     close(fdb[0]); //关闭子进程管道 b 读取端
41     for(i = 0; i < 10; i++)
42     {
43         read(fda[0], &data, sizeof(data));
44         result = sinf(data);
45         write(fdb[1], &result, sizeof(result));
46     }
47     close(fdb[1]); //关闭子进程管道 b 写入端
48     close(fda[0]); //关闭子进程管道 a 读取端
49 }

```

```
50     return 0;
51 }
```

这段代码建立了 2 个无名管道，由于无名管道是半双工的，数据只能在一个方向传送，因此 2 个无名管道分别对应父进程写数据子进程读数据（`fda`）和子进程写数据父进程读数据（`fdb`）。

`data.txt` 文件中存储了 10 条数据，如下图所示：

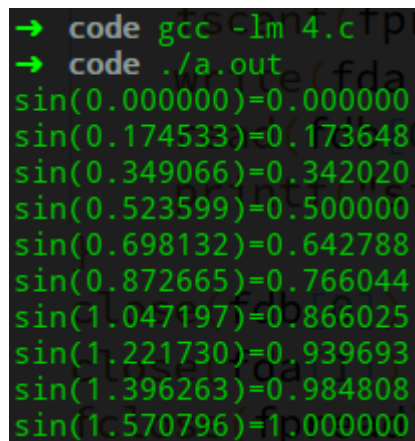


A screenshot of a text editor window titled 'data.txt'. The window contains 10 lines of text, each consisting of a line number followed by a space and a numerical value. The line numbers are in blue, and the values are in black. The values are: 0, 0.1745329, 0.3490658, 0.5235987, 0.6981317, 0.8726646, 1.0471975, 1.2217304, 1.3962634, and 1.5707963.

Line	Value
1	0
2	0.1745329
3	0.3490658
4	0.5235987
5	0.6981317
6	0.8726646
7	1.0471975
8	1.2217304
9	1.3962634
10	1.5707963

父进程和子进程中均循环了 10 次，每次循环父进程从文件中读取一行浮点型数据，使用 `fda` 发送给子进程，子进程接收到这个浮点数对其求正弦值并将计算结果发回给父进程并显示在屏幕上。

程序的输出结果如下：



A screenshot of a terminal window showing the execution of a program. The first two lines are green, indicating they are prompts: `→ code gcc -lm 4.c` and `→ code ./a.out`. The subsequent lines show the output of the program, which consists of 10 lines of text, each showing a sine value calculation: `sin(0.000000)=0.000000`, `sin(0.174533)=0.173648`, `sin(0.349066)=0.342020`, `sin(0.523599)=0.500000`, `sin(0.698132)=0.642788`, `sin(0.872665)=0.766044`, `sin(1.047197)=0.866025`, `sin(1.221730)=0.939693`, `sin(1.396263)=0.984808`, and `sin(1.570796)=1.000000`.

5. 编写两个程序，一个创建一个 FIFO，并读管道，并显示在屏幕上，另一个每过一段时间

向该管道写数据（进程 PID）。运行多个写程序和一个读程序，观察运行结果。（使用系统调用）

答：

读程序的代码如下：

```
1  #include <stdio.h>
2  #include <unistd.h>
3  #include <sys/stat.h>
4  #include <fcntl.h>
5
6  int main(void)
7  {
8      int fd, pid;
9      unlink("FIFO"); //如果已经存在 FIFO 文件则先删除
10     mkfifo("FIFO", 0666); //创建权限为 0666 的名称为 FIFO
    的命名管道
11     fd = open("FIFO", O_RDONLY); //以只读方式打开 FIFO
    命名管道
12     while(read(fd, &pid, sizeof(pid)))
13     {
14         printf("pid: %d\n", pid);
15     }
16     close(fd);
17     return 0;
18 }
```

写程序的代码如下：

```
1  #include <stdio.h>
2  #include <unistd.h>
3  #include <fcntl.h>
4
5  int main(void)
6  {
7      int pid, fd;
8      pid = getpid(); //获得当前进程的 pid
9      fd = open("FIFO", O_WRONLY); //以只写方式打开 FIFO
    命名管道
10     while(write(fd, &pid, sizeof(pid))) //持续写数据
11     {
```

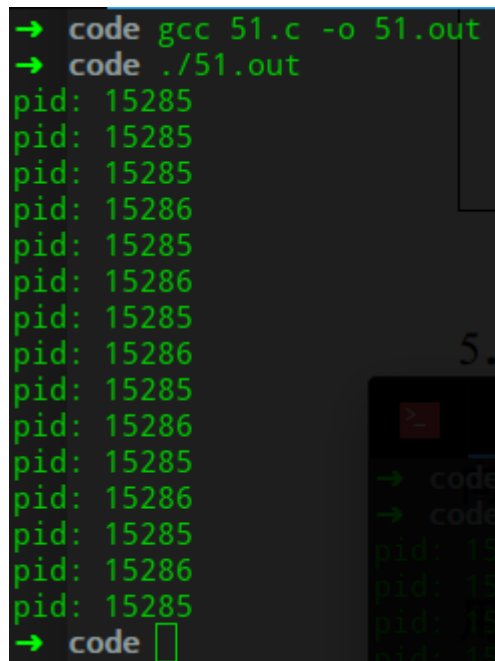
```

12         printf("pid: %d\n", pid);
13         sleep(3);    //等待 3 秒
14     }
15     close(fd);
16     return 0;
17 }

```

读程序和写程序中都创建了一个名为 FIFO 的命名管道，读程序连续不断的从 FIFO 中读取数据并打印，而写程序则写入自己的 pid，最终执行写程序一段时间后通过 Ctrl+C 停止两个写程序的执行，读程序也随之停止执行。读程序输出了两个写程序的 PID，输出的数量与两个写程序写入的次数之和一致。

读程序的执行效果如下：

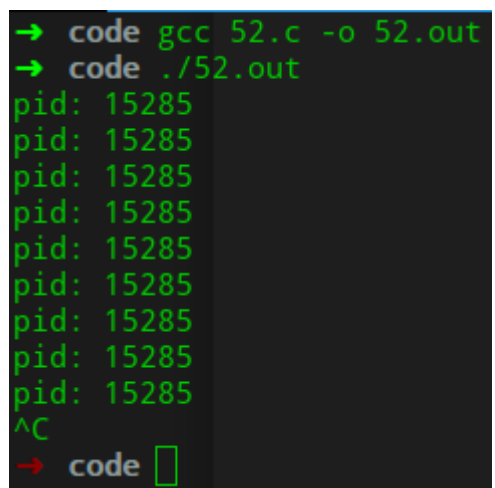


```

→ code gcc 51.c -o 51.out
→ code ./51.out
pid: 15285
pid: 15285
pid: 15285
pid: 15286
pid: 15285
pid: 15286
pid: 15285
pid: 15286
pid: 15285
pid: 15286
pid: 15285
pid: 15286
pid: 15285
pid: 15286
pid: 15285
→ code

```

第一个写程序的执行效果如下：



```

→ code gcc 52.c -o 52.out
→ code ./52.out
pid: 15285
pid: 15285
pid: 15285
pid: 15285
pid: 15285
pid: 15285
pid: 15285
pid: 15285
^C
→ code

```

第二个写程序的执行效果如下：

```
→ code ./52.out  
pid: 15286  
pid: 15286  
pid: 15286  
pid: 15286  
pid: 15286  
pid: 15286  
^C  
→ code 
```