

# Week06 笔记 day33~day38

## Ep01 管道

- 标准流管道 `popen`

- 函数原型

```
1 FILE *popen(const char*command, const char*openmode);
2 //启动一个新进程，并且新进程也原进程之间有一条管道进行通信
```

- `command` 字符串是要运行的程序名
- `open_mode`必须是 "w"/"r" 类型
- 函数返回一个FILE\*文件流指针，可以通过stdio函数，比如 `fread` 来读取文件输出。
- 如果为 "w" 调用程序就可以用 `fwrite` 向被调用程序发送数据，被调用程序可以在自己的标准输入上读取这些数据

- 代码实现

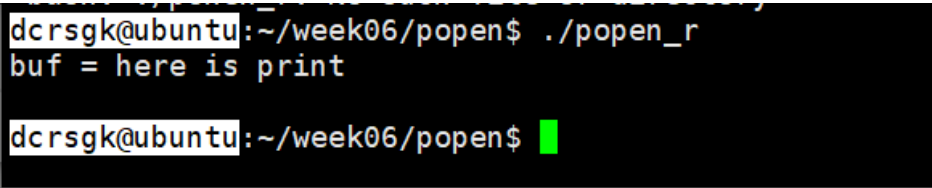
- `popen_r.c`

```
1 int main()
2 {
3     FILE *fp;
4     fp = popen("./peint","r");
5     //重定向print的输出，当作此文件的输入
6     char buf[64]={0};
7     fread(buf,sizeof(char),sizeof(buf));
8     printf("buf = %s\n",buf);
9     pclose(fp);
10    return 0;
11 }
```

- `print.c`

```
1 int main()
2 {
3     printf("i am print");
4     return 0;
5 }
```

- 运行结果

- 

- `popen_w.c`

```

1  int main()
2  {
3      FILE *fp;
4      //以"w"的方式启动新的进程
5      //fwrite写给文件流fp的数据会传递新进程的标准输入
6      fp = popen("./read", "w");
7      fwrite("hello read", sizeof(char), 10, fp);
8      pclose(fp);
9  }

```

#### ■ read.c

```

1  int main()
2  {
3      char buf[128]={0};
4      read(0, buf, sizeof(buf));
5      printf("here is read\n");
6      printf("%s", buf);
7  }

```

#### ■ 运行结果

```

dcrsgk@ubuntu:~/week06/popen$ ./popen_w
here is read
hello read,here is write
dcrsgk@ubuntu:~/week06/popen$

```

### • 无名管道

#### ○ 函数原型

```

1  #include<unistd.h>
2  int pipe(int fds[2]);
3  //此处2为数组大小

```

- 在程序中用一对文件描述符表示
- 其中一个文件描述符为读，一个为写，fds[0]是读，fds[1]是写。
- 创建成功则为0，否则返回-1

#### ○ 无名管道的特点

- 只能在亲缘关系进程间通信（父子/兄弟）
- 半双工（有固定的读写端）
- 是特殊的文件，可以用read/write等，
- 无名管道仅存在在内存中
- 进程终止后自动消失

#### ○ 代码实现

#### ■ pipe.c

```

1  int main()
2  {
3      int fds[2]={0};
4      int ret = pipe(fds);
5      ERROR_CHECK(ret,-1,"pipe Error,4");
6      printf("fds[0]=%d,fds[1]=%d\n",fds[0],fds[1]);
7      return 0;
8  }

```

```

dcrsgk@ubuntu:~/week06/pipe$ ./pipe
fds[0]=3,fds[1]=4
dcrsgk@ubuntu:~/week06/pipe$

```

- 因为进程一开始的时候就创建了 `STDIN`，`STDOUT`，`STDERR` 三个文件描述符，所以文件描述符从3开始

- `pipe_fork.c`：通过管道完成的进程间通信

```

1  int main()
2  {
3      int fds[2]={0};
4      int ret = pipe(fds);
5      ERROR_CHECK(ret,-1,"pipe Error,4");
6      char buf[64]={0};
7      if(fork())
8      {
9          printf("here is father process\n");
10         char message = {"hello child,here is father process"};
11         write(fds[1],message,strlen(message));
12     }
13     else
14     {
15         printf("here is child pricess\n");
16         read(fds[0],buf,sizeof(buf));
17         printf("the message form father:");
18         printf("%s\n",buf);
19     }
20     return 0;
21 }

```

- 运行结果

```

dcrsgk@ubuntu:~/week06/pipe$ ./pipe_fork
here is father pricess
dcrsgk@ubuntu:~/week06/pipe$ here is child pricess
here is the message form father:
hello child,here is father process

```

## Ep02: 基于system V的共享内存

- `ipcs`：查看消息队列/共享内存/信号集
- 函数原型

```

1  #include<sys/types.h>
2  #include<sys/ipc.h>

```

```

3 key_t ftok(const char*pathname, int proj_id);
4 //路径和大小
5 int shmget(key_t key,int size,int shmflg);
6 //创建/打开一段共享内存段，给内存段由函数的第一个参数位移创建
7 //key的值可以是
8 //shmflg可以是IPC_CREAT和IPC_EXCL
9 //返回值为shmid（共享内存id）失败则为-1;
10 void *shmat(int shmid,const void*shmaddr,int shmflg);
11 //映射共享内存
12 //shmid: 是shmget的返回值
13 int shmdt(const void*shmaddr);
14 //
15 int shmctl(int chmid,int cmd,struct shmid_ds *buf);
16 //删除共享内存
17 //cmd是命令语句：常用IPC_SHAT,IPC_SET,IPC_RMID
18 //IPC_RMID更为常用，为删除共享内存段
19 //buf是一个结构体指针，结构体内是共享内存的信息
20 //结算的时候，是以程序判定的
21 //仅有程序结束才完整删除
22 //共享内存的删除是标记删除

```

- `key_t ftok(const char*pathname, int proj_id);`

- `const char *pathname`: 路径
- `int proj_id`: 大小
- `fork.c`

```

1 int mian(int argc,char*argv[])
2 {
3     key_t key = ftok(argv[1],argv[2]);
4     //此处输入一个路径值，可以用
5     //(.)->当前目录
6     //(..) ->上一目录
7     //可以自定义大小
8     //成功则返回一个针对当前路径和大小的唯一的key
9     //失败则返回-1
10    printf("key=%d\n",key);
11 }

```

- 运行结果:

```

dcrsgk@ubuntu:~/week06/shared_memory$ ./ftok .
key = 16865102
dcrsgk@ubuntu:~/week06/shared_memory$ ./ftok ..
key = 16879260
dcrsgk@ubuntu:~/week06/shared_memory$ ./ftok .
key = 16865102
dcrsgk@ubuntu:~/week06/shared_memory$ ./ftok ..
key = 16879260
dcrsgk@ubuntu:~/week06/shared_memory$

```

- `int shmget(key_t key,int size,int shmflg);`

- 创建/打开一段共享内存段，给内存段由函数的第一个参数唯一创建
- `key_t key`
  - 是一个与共享内存段相关联关键字
  - 可以如果事先已经存在一个与指定关键字关联的共享内存段，则直接返回该内存段的标识，表示打开，如果不存在，则创建一个新的共享内存段

- key 的值既可以用ftok函数产生，也可以是IPC\_PRIVATE (用于创建一个只属于创建进程的共享内存，主要用于父子通信)。表示总是创建新的共享内存段
- int size: 表示共享内存的大小，注意此处以字节为单位
- int shmflg: 是掩码合成值，可以是IPC\_CREAT和IPC\_EXCL
  - IPC\_CREAT: 表示如果不存在该内存段，则创建它
  - IPC\_EXCL: 表示如果该内存段存在，则函数返回(-1)
- 返回值为shmid (共享内存id) 失败则为-1;
- 代码实现:
  - shmget.c

```

1  int main()
2  {
3      int shmid = shmget(1000,1<<20,IPC_CREAT);
4      //1000为一个关键字，可自定义
5      //1<<20为1mb大小
6      ERROR_CHECK(shmid,-1,"shmid Error");
7      printf("shmid = %d\n",shmid);
8      //shmget返回共享内存id
9      return 0;
10 }
```

- 运行结果

```
dcrrsgk@ubuntu:~/week06/shared_memory$ ./shmget
shmid = 62
```

- 

- void \*shmat(int shmid,const void\*shmaddr,int shmflg);

- 将共享内存段映射到进程空间的某一地址(使用共享内存的前提)
- int shmid: 为共享内存的段标识，一般为shmget的返回值
- const void\*shmaddr: 指定的是共享内存链接到当前进程中的地址位置
  - 通常为NULL，表示让系统决定共享内存的地址
- int shmflg: 为表示符，通常是0
- 如果调用成功则返回0，失败则返回-1
- 共享内存的分离并不是删除共享内存，只是让共享内存存在当前不可用
- 代码实现:

- shmget\_w.c

```

1  int main()
2  {
3      int shmid = shmget(1000,1<<20,IPC_CREAT);
4      ERROR_CHECK(shmid,-1,"shmid Error");
5      printf("shmid = %d\n",shmid);
6      printf("here is write\n");
7      //shmget返回共享内存id
8      int *p = (int*)shmat(shmid,NULL,0);
9      //映射文件
10     ERROR_CHECK(p,(int*)-1,"shmat Error~");
11     //需要强转int*类型
12     p[0]=4396;
```

```
13     return 0;
14 }
```

#### ■ shmget\_r.c

```
1  int main()
2  {
3      int shmid = shmget(1000,1<<20,IPC_CHEAT);
4      ERROR_CHECK(shmid,-1,"shmid Error");
5      printf("shmid = %d\n",shmid);
6      //shmget返回共享内存id
7      int *p = (int*)shmat(shmid,NULL,0);
8      //映射文件
9      ERROR_CHECK(p,(int*)-1,"shmat Error~");
10     printf("shm data = %d",p[0]);
11     return 0;
12 }
```

#### ■ 运行结果:

```
dcrsgk@ubuntu:~/week06/shared_memory$ ./shmat_w
shmid is 32768
here is write
dcrsgk@ubuntu:~/week06/shared_memory$ ./shmat_r
shmid = 32768
shm data = 4396
dcrsgk@ubuntu:~/week06/shared_memory$
```