

# 实验三

---

## 实验目的

- 通过对 Linux 进程控制的相关系统调用的编程应用，进一步加深对进程概念的理解，明确进程和程序的联系与区别，理解进程并发执行的具体含义。
- 通过对 Linux 管道通信机制、消息队列通信机制、共享内存通信机制的应用，加深对不同类型的进程通信方式的理解。
- 通过对 Linux 的 Posix 信号量及 IPC 信号量的应用，加深对信号量同步机制的理解。

## 实验内容

- 模拟shell：基本功能+find、grep命令，并给结果显示
- 管道通信：基本功能+有名管道通信(独立进程)
- 共享内存：基本功能+双向通信

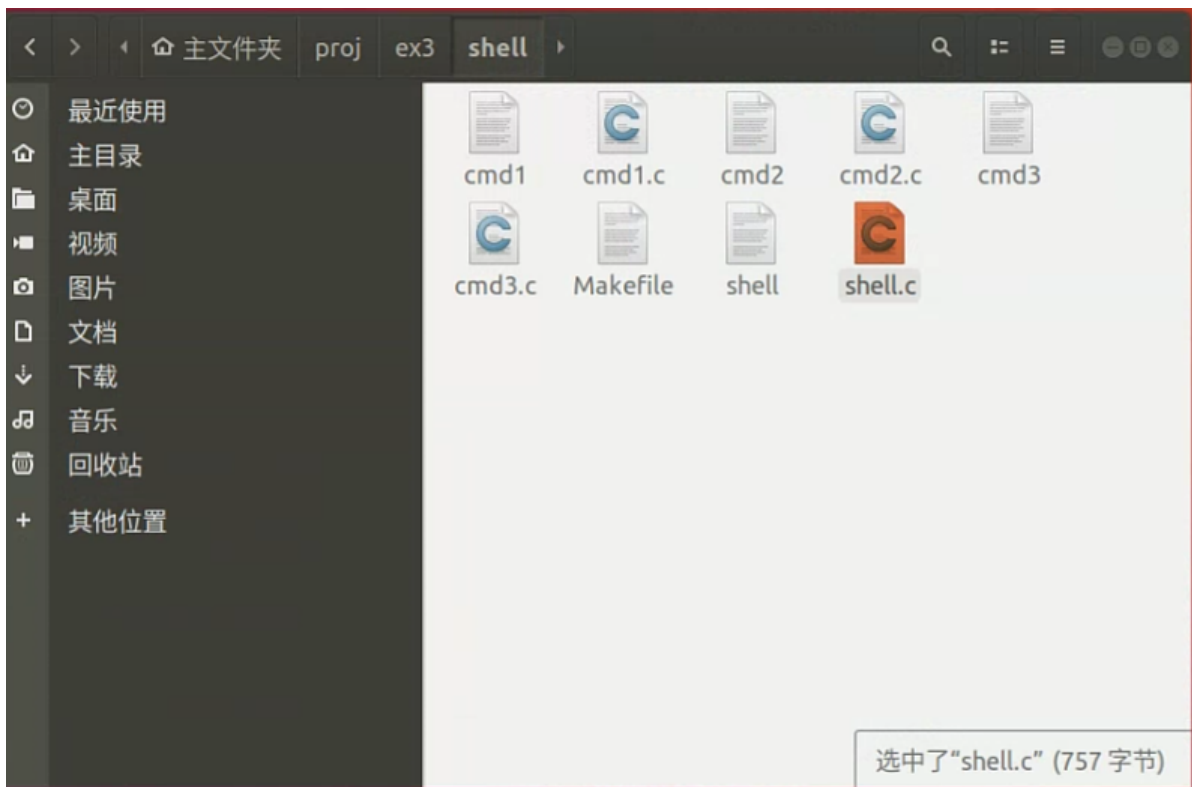
## 实验方法

在 linux 下使用C语言实现上述功能。

## 实验过程和结果

### 模拟shell

编写相应的cmd1.c, cmd2.c, cmd3.c, shell.c:



shell测试结果:

```
dobycho@S1-pro: ~/proj/ex3/shell
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
dobycho@S1-pro:~/proj/ex3/shell$ make
make: 对“all”无需做任何事。
dobycho@S1-pro:~/proj/ex3/shell$ ./shell
$ cmd1
cmd1
$ cmd2
cmd2
$ cmd3
cmd3
$ find "cmd"
find: 'cmd': 没有那个文件或目录
$ find "cmd1.c"
cmd1.c
$ ls | grep cmd
cmd1
cmd1.c
cmd2
cmd2.c
cmd3
cmd3.c
$ exit
shell exit
dobycho@S1-pro:~/proj/ex3/shell$
```

## 管道通信

### 无名管道通信

只需编写一个程序 pipe.c, 并 fork 出三个子进程, 注意编译时要加上 `-pthread` 选项。

```
dobychao@S1-pro: ~/proj/ex3/pipe/noname
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
dobychao@S1-pro:~/proj/ex3/pipe/noname$ ./pipe
pid:20790 进程1写入数据: 1
pid:20791 进程2写入数据: 2
pid:20792 进程3写入数据: 3
pid:20789 父进程接收数据:
1
2
3
dobychao@S1-pro:~/proj/ex3/pipe/noname$
```

## 有名管道通信

这个实验比较复杂，需要写父进程的程序以及三个子进程的程序，共享一个 share.h 头文件是一种比较好的实现方式。

```
dobychao@S1-pro: ~/proj/ex3/pipe/name
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
dobychao@S1-pro:~/proj/ex3/pipe/name$ ./father
pid:9627 父进程接收数据:
1
2
3
dobychao@S1-pro:~/proj/ex3/pipe/name$

dobychao@S1-pro: ~/proj/ex3/pipe/name
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
dobychao@S1-pro:~/proj/ex3/pipe/name$ ./son1
pid:9822 进程1写入数据: 1
dobychao@S1-pro:~/proj/ex3/pipe/name$

dobychao@S1-pro: ~/proj/ex3/pipe/name
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
dobychao@S1-pro:~/proj/ex3/pipe/name$ ./son2
pid:9937 进程2写入数据: 2
dobychao@S1-pro:~/proj/ex3/pipe/name$

dobychao@S1-pro: ~/proj/ex3/pipe/name
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
dobychao@S1-pro:~/proj/ex3/pipe/name$ ./son3
pid:10057 进程3写入数据: 3
dobychao@S1-pro:~/proj/ex3/pipe/name$
```

## 统计管道大小

用非阻塞方式给管道写数据，每次写1024字节，写到管道满为止。

```
dobychao@S1-pro: ~/proj/ex3/pipe/count
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
; dobychao@S1-pro:~/proj/ex3/pipe/count$ ./count
64KB
dobychao@S1-pro:~/proj/ex3/pipe/count$
```

## 进程通信（共享内存）

进程1往共享内存的前半写数据，进程2往共享内存的后半写数据。需要注意的是，进程的任意一方可以随时停止通信。

```
dobychao@S1-pro: ~/proj/ex3/share
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
dobychao@S1-pro:~/proj/ex3/share$ ./job1
收到消息：2
发送：1
发送成功！
发送：exit
dobychao@S1-pro:~/proj/ex3/share$
dobychao@S1-pro: ~/proj/ex3/share
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
dobychao@S1-pro:~/proj/ex3/share$ ./job2
发送：2
发送成功！
收到消息：1
对方终止会话
dobychao@S1-pro:~/proj/ex3/share$
```

## 实验体会

在实验中，我通过使用Linux管道通信和进程共享内存，发现了它们的优点和局限性。

使用Linux管道通信，我们可以轻松实现两个进程之间的交互，比如一个进程向另一个进程发送消息，并等待对方的回复。这种方式非常灵活，因为它可以被用于多种不同的场景，比如实现命令行工具之间的交互。

但是，Linux管道通信也有一些局限性。它只能用于进程之间的单向通信，无法实现双向通信。另外，管道的容量也有限，如果发送的数据量过大，会导致管道满，造成数据丢失。

使用进程共享内存，我们可以轻松实现多个进程之间的数据共享，比如一个进程向共享内存中写入数据，另一个进程从共享内存中读取数据。这种方式非常高效，因为它可以直接访问内存，避免了系统调用和网络传输的开销。

但是，进程共享内存也有一些局限性。它需要操作系统的支持，因此只能在Linux系统中使用。另外，共享内存的容量也有限制，如果某个进程写入了过多的数据，可能会导致内存空间耗尽，造成系统崩溃。此外，由于多个进程可以同时访问共享内存，因此需要设置好同步机制，避免出现数据竞争和冲突的问题。

总的来说，Linux管道通信和进程共享内存都是非常有用的进程间通信方式，它们各有优点和局限性。在实际的应用中，我们应该根据不同的需求，选择合适的通信方式，并进行适当的调优，以达到最优的性能。

## 参考文献

[https://blog.csdn.net/qq\\_34851605/article/details/113577095?spm=1001.2014.3001.5501](https://blog.csdn.net/qq_34851605/article/details/113577095?spm=1001.2014.3001.5501)