实验三：Linux进程管理

1. 实验要求
2. 实现一个模拟的shell；
3. 实现一个管道通信程序；
4. 利用Linux的消息队列通信机制实现两个进程间的通信；
5. 利用Linux的共享内存通信机制实现两个进程间的通信。
6. 函数设计

本实验由四个任务组成。

2.1 实现一个模拟shell

本实验中首先编写了三个程序分别命名为cmd1.c~cmd3.c，并编译成可执行文件cmd1~cmd3。然后编写一个程序模拟shell的功能。此程序中主要用到两个函数vfork()与execlp()。利用vfork()函数创建子进程运行相应命令，利用execlp()函数执行命令对应的可执行文件。程序的流程图如下：



2.2 实现一个管道通信程序

实验要求利用管道实现父进程与三个子进程之间的通信，同时利用POSIX信号量机制实现进程对管道的互斥访问。

本程序中采用POSIX机制中的有名信号量实现进程对管道的互斥访问，以及父子进程之间的同步关系。父子进程之间通过由pipe()系统调用产生的无名管道进行通信。程序流程图如下：



2.3 利用Linux的消息队列通信机制实现两个进程间的通信

此实验要求程序创建3个线程：sender1、sender2、receive。sender1创建消息队列，并通过该队列将消息传递给receive线程；sender2共享sender1创建的消息队列，同样通过该队列将消息传递给receive线程。当sender1与sender2不再发送消息时，向receive发送“exit”，receive线程接收到后给予回应。消息接收完后删除消息队列，结束线程的运行。

除此之外，程序中使用POSIX信号量中的无名信号量‘mutex’，实现线程对消息队列的互斥使用。设置信号量‘r1、r2’，使线程sender1与sender2等待接收线程的应答消息。设置信号量‘f1、f2’，使得发送线程在两个接收线程接收到应答消息后再删除消息队列。

程序流程图如下：





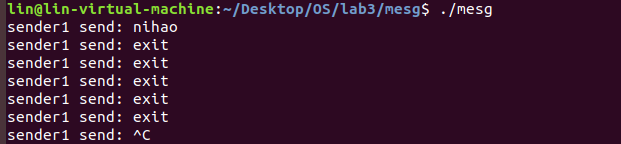
2.4 利用Linux的共享内存通信机制实现两个进程间的通信。

本实验中编写了两个程序，实现了两个进程通过共享内存机制进行通信。一个进程利用共享内存发送信息，另一个进程从共享内存中读出信息。发送进程可以多次发送信息，当用户要结束发送信息时，向终端输入“exit”，发送进程会向共享内存中写入“end”，当接收进程接收到“end”时，会向共享内存中写入“over”以作回应，写入完毕后便结束运行。发送进程从共享内存中读出此信息后，便将共享内存删除，之后便结束进程的运行。

由于两个进程之间存在同步关系，程序中使用POSIX信号机制中的有名信号量实现两个进程间的同步。当发送进程向共享内存中写入一串信息后，接收进程才能从共享内存中将该信息读出。



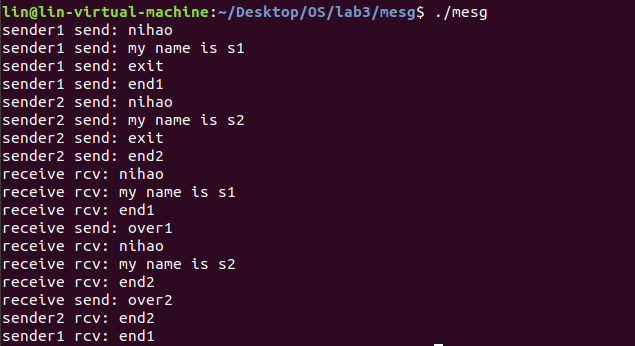
1. 在实验中遇到的问题
2. 实现消息队列通信时，输入“exit”无法标志进程sender1发送消息结束



该问题产生的原因是 程序中获取用户输入使用的语句是“fgets(msg.mtext,80,stdin);”，该语句会将用户输入的换行符‘\n’,也写入消息缓冲区，使得比较语句strcmp(msg.mtext,"exit")判断两个不相等，从而无法执行相应的语句。

解决方案：将读入用户输入的语句改为“gets(msg.mtext);”，gets()函数读入以换行符为结束的字符串，同时也可保证消息中可以正常使用空格。

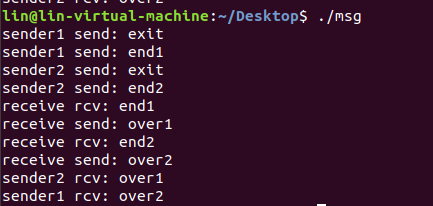
1. 实现消息队列通信时，最后两个发送进程无法接收到receiver进程发送的应答消息。



该问题产生的原因是，在程序中消息队列由receiver进程删除，在删除消息队列时，进程sender1与sender2还没有从消息队列中将receiver发送的应答消息读出。

解决方案：添加信号量f1与f2，使得当两个发送进程从消息队列中读出应答消息之后，接收进程才能将消息队列删除。

1. 实现消息队列通信时，发现发送进程接收到错误的响应消息，如下图所示：



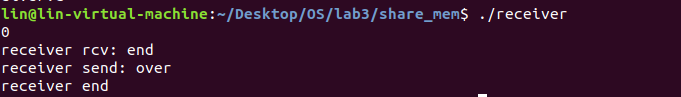
sender1进程错接收到sender2的应答消息“over2”；sender2进程错接收到

sender1的应答消息“over1”；

解决方案：对消息的类型加以区分，在消息缓冲区中增加消息类型mtype字

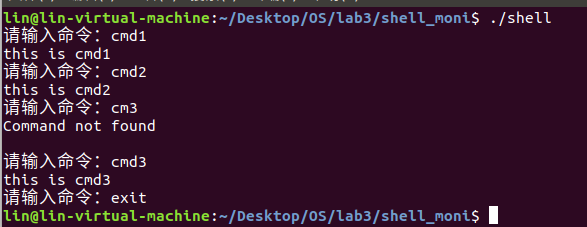
段：1代表普通消息、2代表receiver对sender1的应答消息；3代表receiver对sender2的应答消息。发送进程在接收应答消息的时候，根据消息类型有选择地从消息队列接收消息。

1. 实现共享内存通信时，receiver只能接收到sender发送的“end”，而无法接收到其他消息。



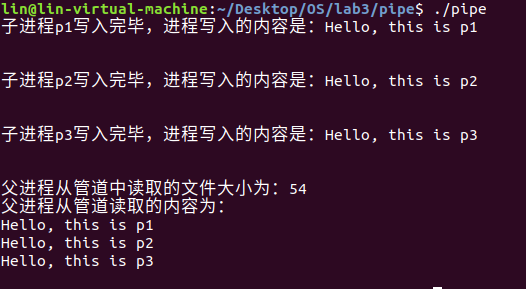
解决方法：一个发送进程，一个接收进程，两者之间仅存在进程同步，不需要互斥信号量，将互斥信号量删去。

1. 实验结果讨论
2. 模拟shell实验结果演示



程序运行后，若用户输入了正确的指令（cmd1、cmd2、cmd3），则会执行指令相对应的程序；若用户输入了错误指令，如上图中的cm3，则会提示“Command not found”；若用户输入的指令为“exit”，则模拟shell程序运行结束。

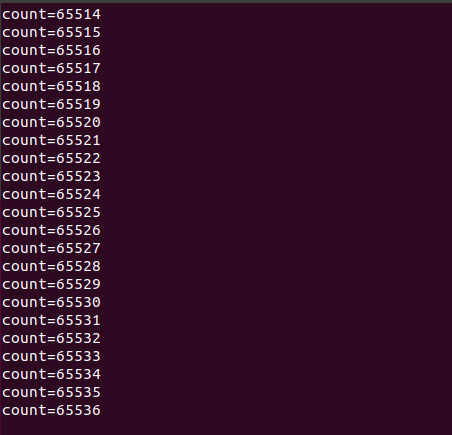
1. 管道通信实验结果演示



三个子进程写入完毕后，父进程从管道中读取子进程写入的数据，读取到的数据与子进程写入的数据一致。

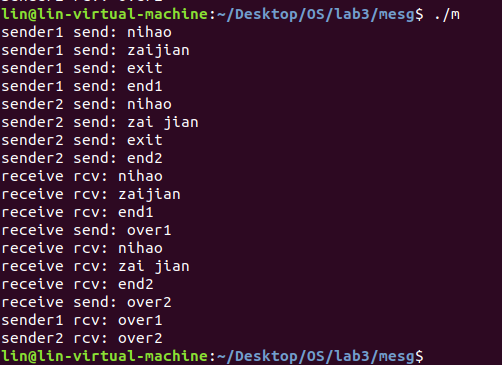
测试管道大小：

程序中采用循环的方式向管道写入数据，一次写入一个\*，同时对写入的\*数目进行计数。由于一个\*占一个字节的内存，因此可根据写入管道的\*的数目，测试出管道的大小，测试结果如下图所示。



可以看出，程序向管道中写入了65536个\*，即管道的容量为65536字节，即16KB。

1. 消息队列通信实验结果演示



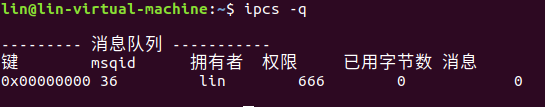
从上面的结果可看出，程序能完成预期目标。

程序所用消息队列建立情况：

1）运行程序前，系统中的消息队列情况如下图所示：



2）开始运行程序，再次查看系统中的消息队列情况，得到下图：



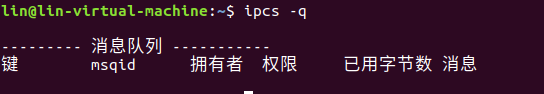
可以看到，程序运行时确实创建了一个消息队列。

3）在向消息队列中发送3条消息后，看到消息队列的信息如下：



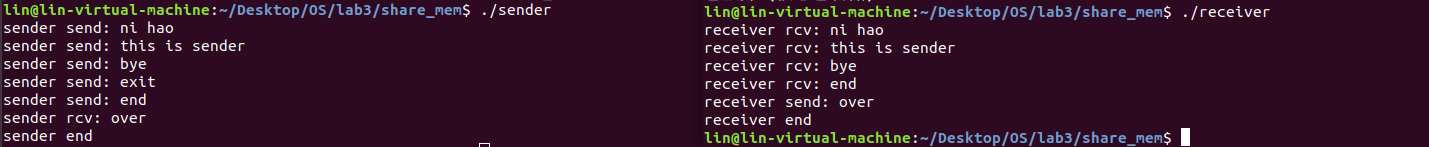
可以看到消息队列中确实已有3条信息，共占用3096个字节。

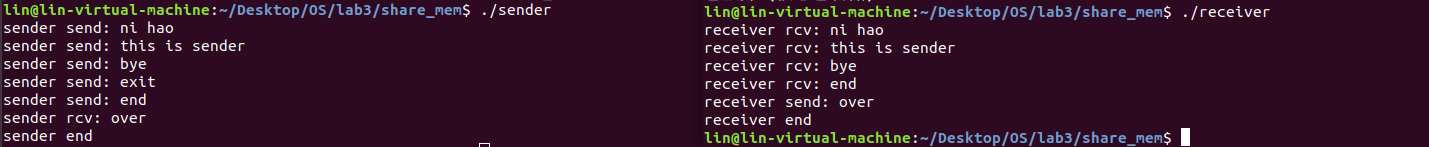
4）程序运行结束后，再次查看系统中消息队列的情况：



可以看到，程序创建的消息队列已经正常删除。

4. 共享内存通信实验结果演示：





对于sender进程写入到共享内存的数据，receiver进程可以正常接收并显示在终端屏幕上；sender进程要结束通信时，由用户输入“exit”，进程向共享内存中写入“end”，receiver接收到后，回复“over”，sender接收到后显示在屏幕上。实验结果符合预期。

程序中共享内存段创建情况：

1）程序运行之前，系统中的共享内存段的情况如下图所示：



2）运行程序后，系统中的共享内存段的情况如下图所示。

可以看到系统创建了一个共享内存段，shmid为53，大小为256字节，权

限为666，与程序中定义的一致，证明程序正常运行。

3）程序运行结束后，系统中的共享内存段情况如下图所示：



可以看出程序创建的共享内存段已经正常删除。