实验5 进程通信（1）无名管道和共享内存通信

实验目的

1.了解无名管道通信的基本原理

2.掌握父子进程使用管道通信的方法

3.掌握共享内存通信方式的原理

实验内容

1. 父子进程基于无名管道的简单通信。

/\*pipe1.c\*/

#include <sys/types.h>

#include <sys/wait.h>

#include <stdlib.h>

#include <unistd.h>

#include <signal.h>

#include <stdio.h>

#include <string.h>

main()

{

int fd[2],pid,n;

char outpipe[50],inpipe[50];

pipe(fd); /\*创建无名管道\*/

pid=fork();

if(pid==0)

{

sprintf(outpipe,"I am child process");

write(fd[1],outpipe,strlen(outpipe)); /\*向无名管道写入信息\*/

printf("child process writes%d bytes: %s\n",strlen(outpipe),outpipe);

}

else

{

wait(0);

n=read(fd[0],inpipe,25); /\*从无名管道读出信息\*/

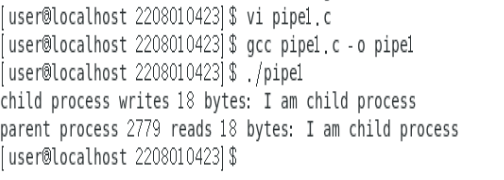
printf("parent process %d reads %d bytes:%s\n",getpid(),n,inpipe);

}

}

要求：写出程序的运行结果截图及分析

**运行结果**



**程序分析**

无名管道通信，首先创建无名管道，其中 fd[0] 用于读入，fd[1] 用于写入。

然后创建子进程，父进程会等待子进程先执行结束。子进程将I am child process

写入 outpipe, 之后父进程从管道中读取字符串I am child process 到 inpipe，然后输出结果。

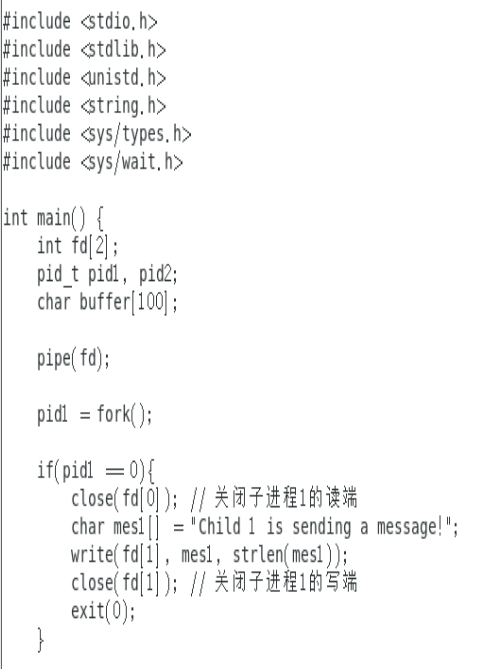
2. 编写程序实现多个进程基于无名管道进行通信。用系统调用pipe()建立一无名管道，两个子进程P1和P2分别向管道写入一句话：

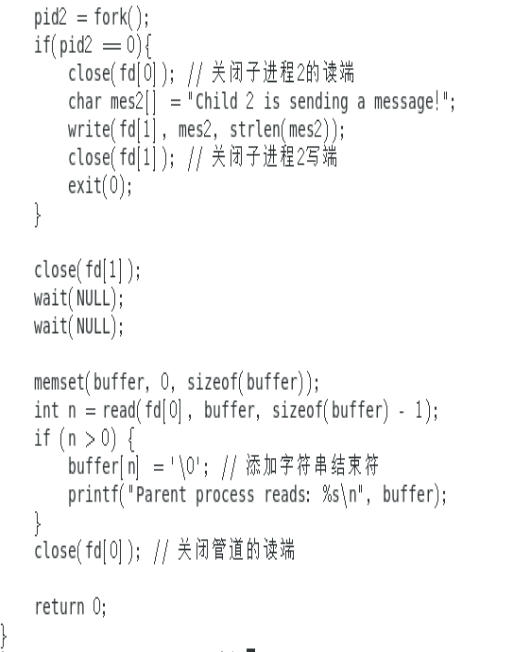
Child 1 is sending a message!

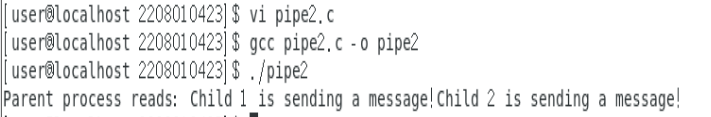
Child 2 is sending a message!

父进程从无名管道中读出两个来自子进程的信息并显示，子进程发送先后顺序没有要求。

要求：（1）写出主要程序代码及运行结果截图







1. 总结pipe管道通信的主要原理和特点

**主要原理**

管道是一种进程间通信（IPC）机制，允许数据在具有父子关系的进程之间进行传递。

使用 pipe() 系统调用创建一个无名管道，它提供了两个文件描述符：fd[0]（读端）和 fd[1]（写端）。

子进程可以通过 write() 将数据写入管道，父进程可以通过 read() 从管道中读取数据。

**特点**

1, 单向通信: 无名管道是单向的，数据只能从一端写入，从另一端读取。要实现双向通信，必须创建两个管道。

2, 半双工通信: 数据传输是单向的，读写操作不能同时进行。

3, 仅限亲缘进程: 无名管道只能在具有亲缘关系（如父子进程）的进程之间使用。

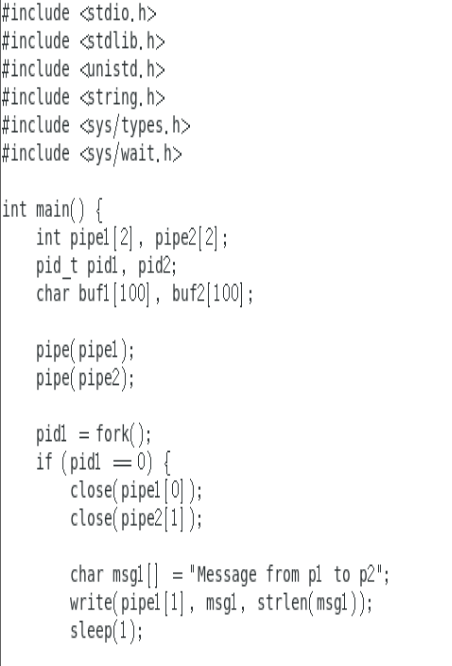
4, 内核缓冲区: 管道使用内核缓冲区来传输数据，如果缓冲区满，写操作将被阻塞；如果缓冲区为空，读操作将被阻塞。

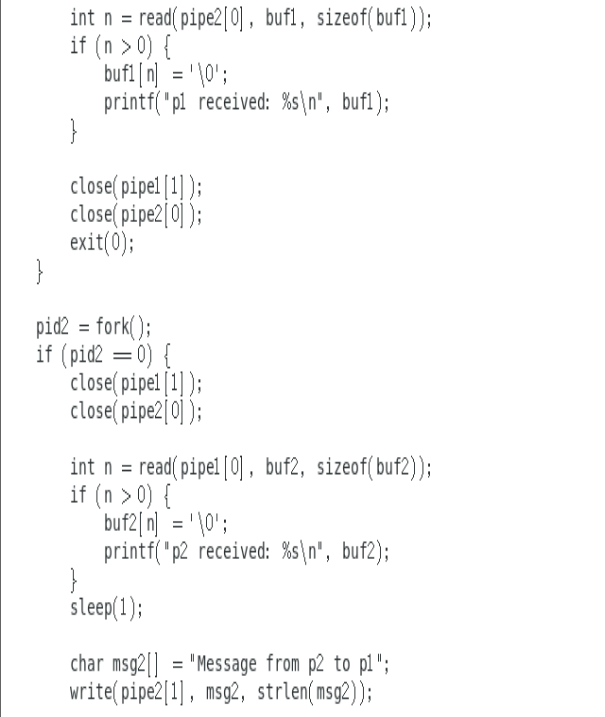
5, 自动同步: 管道通信提供自动同步机制，当一个进程写入数据时，另一个进程可以读取数据，而不需要使用显式的同步工具。

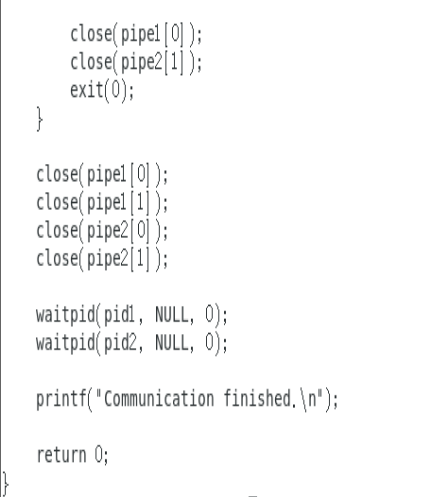
1. 如果从管道中读出乱码如何解决？（memset函数）

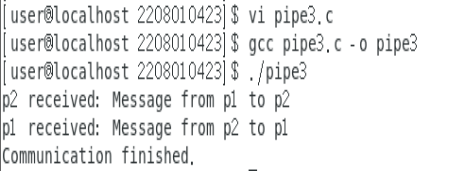
**当从管道中读取数据时，如果未正确初始化缓冲区或者未添加字符串结束符 \0，可能会导致输出乱码。可以使用 memset() 函数来清空缓冲区，确保读取的数据格式正确。**

1. （选做）编程实现父进程建立的两个子进程p1，p2利用pipe实现双向通信（p1发一条消息收一条消息，p2收一条发一条消息，或者相反；可用sleep控制同步），父进程等待子进程结束后输出通信结束信息终止。









**开通两个管道 p1, p2，进程 1 向 p1 写入，然后 sleep, 等待进程 2 读取，进程 2 读取之后，再向管道 2 写入，等待进程 1 读取。完成上述操作之后，父进程输出通信结束。**

4. 运行以下程序，分析程序的运行结果。

接收进程receive.c:

#include<stdio.h>

#include<stdlib.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

int main()

{

int shmid; char \*viraddr;

shmid=shmget(1234,BUFSIZ,0666|IPC\_CREAT); viraddr=shmat(shmid, 0,0);

printf("Your message is :\n%s",viraddr);   
 shmdt(viraddr);

shmctl(shmid,IPC\_RMID,0);

exit(0);

return 0;

}

发送进程send.c：

#include<string.h>

#include<stdio.h>

#include<stdlib.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

int main()

{ int shmid; /\*定义共享内存内部标识shmid \*/

char \*viraddr; /\*定义附接共享内存的虚拟地址\*/

char buffer[BUFSIZ]; shmid=shmget(1234,BUFSIZ,0666|IPC\_CREAT);

viraddr=(char\*)shmat(shmid, 0,0);

while(1)

{ puts("Enter some text:");

fgets(buffer,BUFSIZ,stdin);

strcat(viraddr,buffer);

if(strncmp(buffer,"end",3)==0) break; }

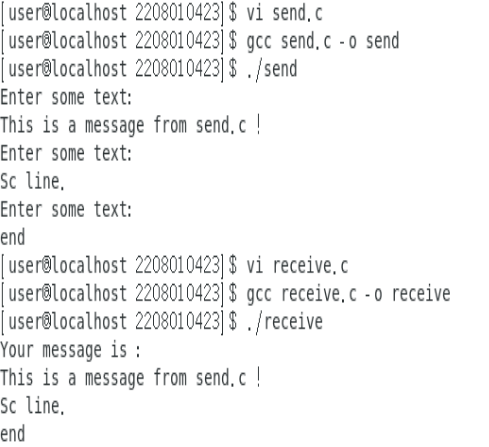
shmdt(viraddr); exit(0);

return 0;

}

要求：

1. 运行截图。



1. 根据程序分析共享内存的实现原理

对于 send.c, 通过 shmget 函数创建一个内存共享段，标识符为 1234.

之后通过 shamt 函数将共享内存段附加到当前进程的地址空间进进行操作。

对于 receive.c, 打开共享内存区，并映射到当前地址空间，之后就可以使用了。