一、问题描述

用C语言编写一个程序，使其用管道实现父子进程间通信。在程序中建立一个管道，同时父进程生成一个子进程，子进程向管道中写入字符串“is sending a message to parent!”；父进程从管道中读出该字符串，并显示到屏幕上。通过一系列操作，实现子进程向父进程发送消息，父进程接受子进程发来的消息，然后终止的过程。

1. 工作原理
2. 相关概念

管道通信方式也称为共享文件(shared file)通信机制，是进程高级通信方式的一种。在该机制中，发送进程以字符流形式把大量数据送入管道（连接读写进程的一个特殊的共享文件），接收进程从管道中接收数据，实质上是利用外存进行数据通信。

2.Pipe的建立和使用

pipe文件在使用之前，必须先由使用者建立并打开，管道由程序调用pipe函数来创建，格式为：int pipe(int fd[2])，pipe函数会建立管道,并将文件描述词由fd[2]数组返回,参数数组fd[2]包含pipe使用的两个文件的描述符，fd[0]为读管道，fd[1]为写管道。若创建成功，返回0，否则返回-1。

建立pipe的主要工作是在系统打开文件表中建立该pipe的两个系统文件表目，一个表目用于控制该pipe的写操作（写入端），另一表目用于控制该pipe的读操作（读出端）：

(1)系统文件write(fd[1],buf,size)

功能：把buf中的长度为size字符的消息送入管道入口fd[1]

(2)系统文件read(fd[0],buf,size)

功能：从pipe出口fd[0]读出size字符的消息置入 buf中

3.具体实现

发送进程利用文件系统的系统调用write(fd[1],buf,size)，将buf中的长度为size个字符的消息送入管道入口fd[1]；接收进程则使用系统调用 read(fd[0],buf,size)从管道出口fd[0]读出size个字符的消息置入buf中。



1. 详细设计
2. 系统函数说明

注：以下函数按在代码中的出现顺序排序

1. pipe(fd)

调用pipe函数建立管道，使用fd数组接收文件描述词。若建立成功，则返回0，否则返回-1。

1. fork( )

使用fork系统调用，从已存在进程中创建一个新进程，新进程称为子进程，原进程称为父进程。

fork调用一次，返回两次，两次返回分别带回父子进程各自的返回值。若子进程创建成功，则父进程中的返回值是子进程的进程号（id号），子进程中的返回值为0；若子进程创建失败，则返回值为-1。因此，可以通过返回值来判定该进程是父进程还是子进程。

使用while循环判断(x=fork( ))==-1，即为判断子进程是否创建失败，若子进程创建失败，则一直循环该创建过程，保证子进程创建成功。

1. int sprintf(char \*buffer, const char \*format, [argument]...)

sprintf函数把格式化的数据写入某个字符，在子进程中执行sprintf(buf,"is sending a passage to parent!")，实现将字符串写入buf中。

1. write(fd[1],buf,size)

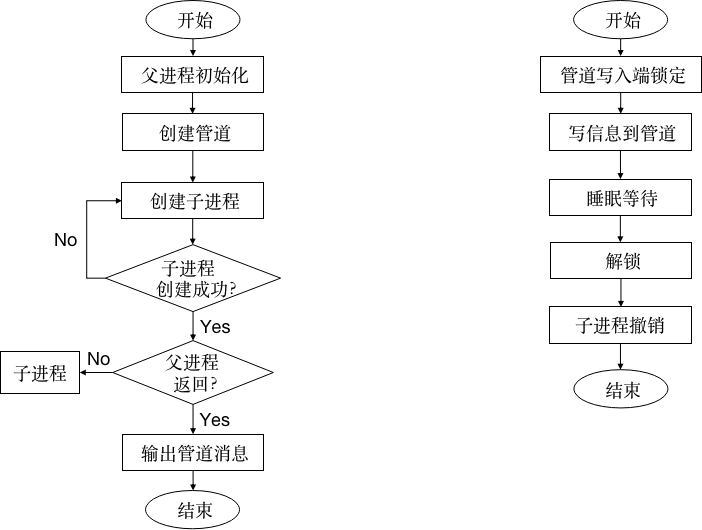
代码write(fd[1],buf,50)实现将buf中的长度为50的消息送入管道入口fd[1]中。

1. read(fd[0],buf,size)

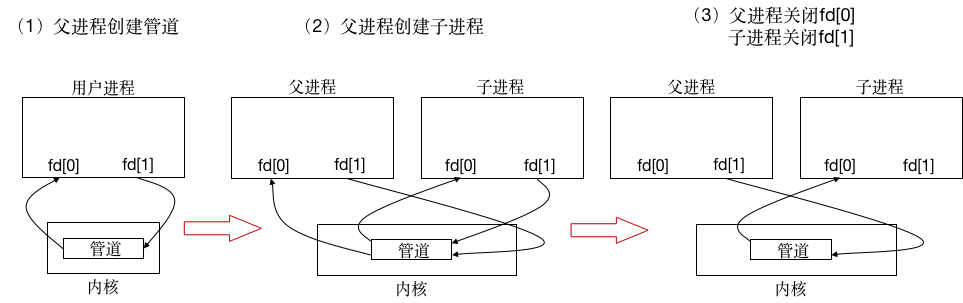
代码read(fd[0],s,50)实现从pipe出口fd[0]读出50字符的消息，并置入 buf中。

1. 程序流程图

父进程的程序流程图： 子进程的程序流程图：



程序运行图：



(1)父进程创建管道，得到两个⽂件描述符指向管道的两端

(2)父进程fork出子进程，⼦进程也有两个⽂件描述符指向同⼀管道。

(3)父进程关闭fd[0],子进程关闭fd[1]，即⽗进程关闭管道读端,⼦进程关闭管道写端（因为管道只支持单向通信）。⽗进程可以往管道⾥写,⼦进程可以从管道⾥读,管道是⽤环形队列实现的,数据从写端流⼊从读端流出,这样就实现了进程间通信。

1. 代码展示

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

main()

{

int x,fd[2];

char buf[50],s[50];

pipe(fd);

while((x=fork())==-1);

if(x==0)

{

printf("调用子进程，子进程ID:%d\n",getpid());

sprintf(buf,"is sending a passage to parent!");

write(fd[1],buf,50);

printf("子进程已将消息写入管道！\n");

exit(0);

}

else

{ printf("调用父进程，父进程ID：:%d\n",getpid());

wait(0);

printf("返回父进程\n");

read(fd[0],s,50);

printf("%s%s\n","父进程读取到的消息：",s);

}

}