计算机操作系统原理

课程设计

题　　目 传统的进程间通信

学生姓名 _____

专业班级 计算机科学与技术　班

完成时间 _____

指导教师 _____

青岛大学　计算机学院

2019 年 7 月

设计目的

1、理解信号和管道的概念及用于实现进程通信的原理

2、掌握信号通信机制，实现进程之间通过信号进行通信

3、掌握匿名管道及有名管道通信机制，实现进程之间通过管道进行通信

进程通信--管道

一、无名管道

匿名管道的创建通过函数 pipe( )实现，其声明格式如下：

```
#include <unistd.h>
int pipe(int fd[2])
```

当调用成功时，函数 pipe 返回值为 0，否则返回值为–1。成功返回时，数组 fd 被填入两个有效的文件描述符。数组第 1 个元素中的文件描述符供应用程序读取，数组的第 2 个元素中的文件描述符可用来供应用程序写入。

1.只能用于具有亲缘关系的进程之间的通信
2.半双工通信模式
3.一种特殊的文件，是一种只存在于内核中的读写函数

管道基于文件描述符，管道建立时，有两个文件描述符：

匿名管道两端可分别用描述字 fd[0]及 fd[1]来描述，管道两端的任务是固定的。习惯上，描述符 fd[0]表示管道读端，描述符 fd[1]表示管道写端，按惯例编程，会使程序的可移植性更好。一般文件的 I/O 函数都可以用于管道，如 close( )、read( )和 write( )等。

a. fd[0]: 固定用于读管道
b. fd[1]: 固定用于写管道

一般步骤：
1. pipe()创建管道
2. fork()创建子进程
3. 子进程会继承父进程的管道

关闭管道：1. 逐个关闭文件描述符　　2. close()

父子进程间的管道通信：父子进程对管道分别有自己的读写通道，把无关的读端或写段关闭。
掌握函数 pipe()、read()、write()、close()。

【示例】

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <wait.h>
#define MAX_LINE 80

int main()
{
    int thePipe[2], ret;
```

```
    char buf[MAX_LINE+1];
    const char *testbuf="a test string.";
    if ( pipe( thePipe ) == 0 ) {
      printf("thePipe[0]=%d, thePipe[1]=%d \n", thePipe[0], thePipe[1]);
      if (fork() == 0) {
        ret = read( thePipe[0], buf, MAX_LINE );
        printf("read ret=%d \n", ret);
        buf[ret] = 0;
        printf( "Child read %s\n", buf );
      } else {
        ret = write( thePipe[1], testbuf, strlen(testbuf) );
        printf("write ret=%d \n", ret);
        ret = wait( NULL );
        printf("wait ret=%d \n", ret);
      }
    }
    close(thePipe[0] );
    close(thePipe[1] );
    return 0;
}
```

【示例 2】

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <unistd.h>

#define MAX_DATA_LEN 256
#define DELAY_TIME 1

int main() {
    pid_t pid;
    char buf[MAX_DATA_LEN];
    const char *data="Pipe Test program";
    int real_read,real_write;
    int pipe_fd[2];

    memset((void*)buf,0,sizeof(buf));

    if(pipe(pipe_fd)<0){
        perror("Pipe create error!\n");
        exit(1);
    }
```

```
    if ((pid=fork())<0) {
        perror("Fork error!\n");
        exit(1);
    } else if (pid==0) {
        close(pipe_fd[1]);
        sleep(DELAY_TIME*3);

        if ((real_read=read(pipe_fd[0],buf,MAX_DATA_LEN))>0) {
            printf("Child receive %d bytes from pipe: '%s'.\n",real_read,buf);
        }

        close(pipe_fd[0]);
        exit(0);
    } else {
        close(pipe_fd[0]);
        sleep(DELAY_TIME);

        if ((real_write=write(pipe_fd[1],data,strlen(data)))>0) {
            printf("Parent write %d bytes into pipe: '%s'.\n",real_write,data);
        }

        close(pipe_fd[1]);
        waitpid(pid,NULL,0);
        exit(0);
    }

}
```

二、有名管道 FIFO

1. 使不相关的两个进程彼此通信：
a. 通过路径名指出，在文件系统中可见
b. 管道建立后，两进程可按普通文件一样对其操作

2. FIFO 遵循先进先出规则：
a. 对管道读从开始处返回数据
b. 对管道写则把数据添加到末尾
c. 不支持如 lseek()等文件定位操作

创建有名管道：mkfifo()

命名管道的创建通过函数 mkfifo( )实现，该调用的函数声明格式如下：

```
#include <sys/types.h>
#include <sys/stat.h>
int mkfifo(const char * pathname, mode_t mode)
```

mkfifo( )的作用是在文件系统中创建一个文件，该文件用于提供命名管道功能。第 1 个参数(pathname)是将在文件系统中创建的一个专用文件，也就是创建后命名管道的名字。第 2 个参数(mode)用来规定命名管道的读写权限。mkfifo( )如果调用成功，返回值为 0；如果调用失败，返回值为–1。如果 mkfifo( )的第 1 个参数是一个已经存在的路径名，则返回 EEXIST 错误。因此，典型的调用代码首先会检查是否返回该错误。一般文件的 I/O 函数都可以用于命名管道，如 close( )、read( )和 write( )等。下例给出使用 mkfifo( )创建命名管道的代码框架：

创建管道成功后，可使用 open()、read()和 write()等函数。

为读而打开的管道可在 open()中设置 O_RDONLY

为写而打开的管道可在 open()中设置 O_WRONLY

与普通文件不同的是阻塞问题

- 普通文件的读写时不会出现阻塞问题
- 在管道的读写中却有阻塞的可能，
- 非阻塞标志：在 open()函数中设定为 O_NONBLOCK

阻塞打开与非阻塞打开:

对于读进程

- 若该管道是阻塞打开，且当前 FIFO 内没有数据，则对读进程而言将一直阻塞到有数据写入
- 若该管道是非阻塞打开，则不论 FIFO 内是否有数据，读进程都会立即执行读操作。即如果 FIFO 内没有数据，则读函数将立刻返回 0

对于写进程

- 若该管道是阻塞打开，则写操作将一直阻塞到数据可以被写入
- 若该管道是非阻塞打开而不能写入全部数据，则读操作进行部分写入或者调用失败

【实例 1】reader 进程

```
/*fifo_read.c  读管道程序*/
#include <unistd.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<errno.h>
#include<fcntl.h>
#include<string.h>
#include<stdlib.h>
#include<stdio.h>
#include<limits.h>
#define MYFIFO    "/tmp/myfifo"    /*有名管道文件名*/
/*在 limits.h 中有  #define PIPE_BUF    4096  即 4 个字节大小*/
#define MAX_BUFFER_SIZE             PIPE_BUF /*定义在 limits.h 中*/
```

```
int main()
{
    char buff[MAX_BUFFER_SIZE];
    int fd;
    int nread;

    /*判断有名管道是否已经存在，若尚未创建，则以相应的权限创建*/
    if(access(MYFIFO,F_OK)==-1)
    {
        if((mkfifo(MYFIFO,0666)<0)&&(errno!=EEXIST))
        {
            printf("Cannot create fifo file\n");
            exit(1);
        }
    }

    /*以只读阻塞方式打开有名管道*/
    fd=open(MYFIFO,O_RDONLY);
    if(fd==-1)
    {
        printf("Open fifo file error\n");
        exit(1);
    }
    while(1)
    {
        memset(buff,0,sizeof(buff));

        if((nread=read(fd,buff,MAX_BUFFER_SIZE))>0)
        {
            printf("Read '%s' from FIFO\n",buff);
        }
        //假设读取到 exit 的时候退出
                if(!strcmp(buff,"exit")) break;
    }
    close(fd);
    exit(0);

}/*end*/
```

【实例 2】writer 进程

```
/* fifo_write.c  写管道程序*/
#include <unistd.h>
#include<sys/types.h>
#include<sys/stat.h>
```

```
#include<errno.h>
#include<fcntl.h>
#include<stdlib.h>
#include<stdio.h>
#include<limits.h>
#define MYFIFO    "/tmp/myfifo"    /*有名管道文件名*/
/*在 limits.h 中有  #define PIPE_BUF    4096  即 4 个字节大小*/
#define MAX_BUFFER_SIZE             PIPE_BUF /*定义在 limits.h 中*/

int main(int argc,char *argv[]) /*参数为即将写入的字符串*/
{
    int fd;
    char buff[MAX_BUFFER_SIZE];
    int nwrite;

    if(argc<=1)
    {
        printf("Usage: ./fifo_write string\n");
        exit(1);
    }
    /*sscanf()表示从字符串中格式化输出，与 scanf 类似，都是用于输入的，只是
    scanf()以键盘为输入源，sscanf()是以固定字符串为输入源*/
    sscanf(argv[1],"%s",buff);/*将 argv[1]的内容以字符串(%s)的形式存入 buf 中*/

    /*以只写阻塞方式打开 FIFO 管道*/
    fd=open(MYFIFO,O_WRONLY);
    if(fd==-1)
    {
        printf("Open fifo file error\n");
        exit(1);
    }
    /*向管道中写入字符串*/
    if((nwrite=write(fd,buff,MAX_BUFFER_SIZE))>0)
    {
        printf("Write '%s' to FIFO\n",buff);
    }
    close(fd);
    exit(0);
} /*end*/
```

三、设计

1、设计说明

学会使用有名管道在多进程间建立通信

2、解决方案

为实现多进程间基于命名管道的通信，首先使用 mkfifo( )创建一个命名管道。随后可使用一般的文件 I/O 函数，如 open( )、close( )、read( )、write( )等，来对它进行操作，从而实现多进程间的通信。

3、程序框架

```
#include <sys/types.h>
#include <sys/stat.h>
#include <stdio.h>
#include <errno.h>
#include <fcntl.h>
#include <string.h>
#define FIFO_SERVER "/tmp/fifoserver"
#define BUFFERSIZE 80

void main( )
{

    if(创建命名管道失败) {
        /*打印“无法创建命名管道”错误提示信息*/
        /*退出*/
    }
    /*打印“成功创建命名管道”提示信息*/
    /*创建子进程*/
    if(子进程创建成功) {
        /*以写方式打开命名管道*/
        if(打开失败) {
            /*打印“无法打开命名管道”错误信息*/
            /*退出*/
        }
        /*向命名管道写入数据*/
        if(写入失败) {
            /*打印“写数据出错”提示信息*/
            /*退出*/
        }
        /*打印“成功写入数据”提示信息*/
        /*关闭命名管道*/
    }else
```

```
        if(父进程) {
            /*以只读方式打开命名管道*/
            /*输出读数据前缓冲区信息*/
            /*从命名管道读取数据到缓冲区*/
            /*输出读数据后缓冲区信息*/
            /*关闭命名管道*/
        }else {
            /*打印“创建进程出错”提示信息*/
            /*退出*/
        }
}
```