华北电力大学

课程设计报告

(2016 -- 2017 年度第 1 学期)

名　　称：Unix/Linux 编程课程设计

题　　目：LINUX 系统下多进程的创建与通信

院　　系：控制与计算机工程学院

班　　级：计算 1302 班

学　　号：1131220221

学生姓名：余涛

指导教师：贾静平

设计周数：1

成　　绩：

日期：2017 年 1 月　日

一、课程设计的目的与要求

1. 学习 UNIX/LINUX 系统下的多进程创建、控制和通信。

1．1 Linux 上的 bash 和 Windows 中的命令行有很大的不同。但是两者都有完成相似任务的命令，比如 Linux 上 bash 的 ls 命令的功能，类似于 Windows 命令行中的 dir 命令的功能。用 C 语言写一个简单的 Linux 终端软件，接收用户发出的类似于 Windows 命令行中的命令，转换成对应的 Linux 命令加以执行，并将执行的结果回显给用户。比如，用户输入“dir”，进程实际返回“ls”的内容。

1．2 软件包含前、后台两个进程，用户启动前台进程时，前台进程自行启动后台进程。前台进程提供界面，负责接收用户输入，对输入进行转换，并向后台进程发出实际要执行的指令，后台负责执行实际的指令，并将指令执行的结果返回给前台进程，由前台进程在终端显示。

二、设计正文

1. 任务分析

1．1 前台进程应实现的功能如下

创建消息队列

创建命名管道

接受用户输入

对输入进行装换

向后台发送实际要执行的指令

等待后台返回结果

输出指令执行的的结果

若输入的指令是 exit 则向后台发送 exit 命令

删除消息队列

删除命名管道

等待后台进程退出

因此，大致的进程执行流程如图 1 所示：

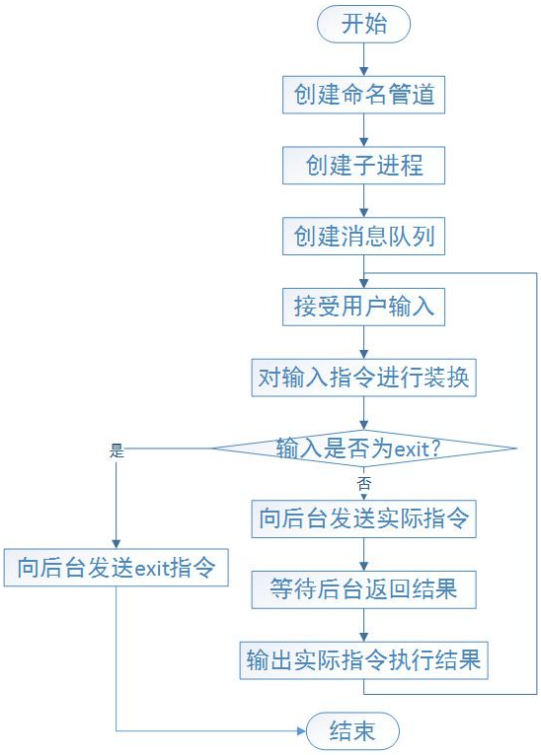

图 1：前台进程流程图

1．2　后台进程应实现的功能如下：

阻塞等待前台进程发出的实际执行的指令

执行收到的指令，并读取结果

将执行结果发送给前台进程

因此后台进程的执行流程如图 2 所示：

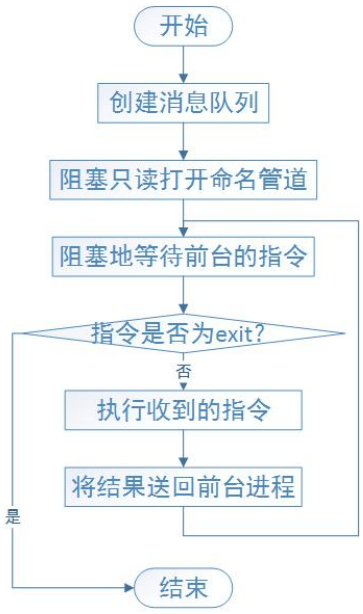

图 2：后台进程流程图

1．3 进程间的协助如图 3 所示：

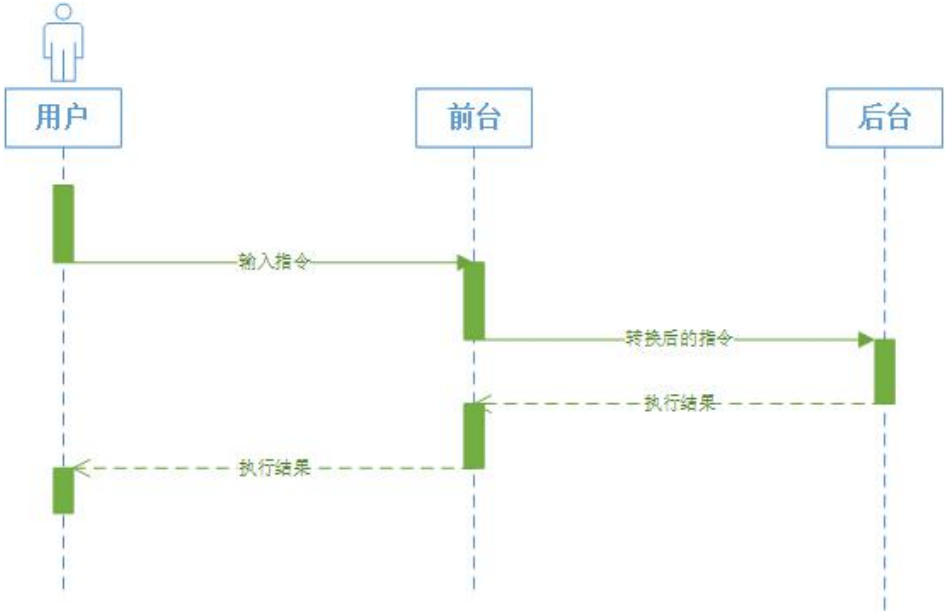

图 3：进程间的协助图

2． 功能实现分析

2．1 子进程创建

根据题目的要求，后台程序由前台程序创建，具体的是通过 fork 函数与 execl 函数组合使来实现。基本的程序结构如下所示：

```
int fpid = fork();
if (0==fpid) {//返回0，说明是子进程
      execl(const char *path, const char *arg, ...);
} else if(fpid > 0 ){ //返回大于0，说明是父进程
     ......//父进程代码
 } else if (fpid < 0) {
      ......//fork函数出错的情况
}
```

基本的思想在于使用 fork 函数创建一个子进程，通过 fork 函数的返回值可以将父子进程区分开来，在子进程中调用 execl 函数执行后台进程的可执行文件，可将当前进程映像替换为后台进程的映像，完成了后台进程的创建。

2．2 使用消息队列进行进程间通信

发送消息的数据结构

消息队列是进程间相互发送数据块的通信方式，发送消息使用的数据结构是一个满足图 4 所示要求的结构体，如下

```
struct msg_wrapper{
     long int msg_type;   //第一个成员变量必须是长整型，用于表示消息的类型
     ..... //具体的数据
};
```

消息队列的创建，使用 msgget 函数，如下所示：

```
#define MESSAGE_ID 973837
int msgid = msgget((key_t) MESSAGE_ID, 0666 | IPC_CREAT);
if (-1 == msgid) {//如果返回值等于-1，表示出错
      perror("create message queue");
      exit(errno);
}
```

消息队列的控制，使用 msgctl 函数，本次设计仅使用了这个函数来进行消息队列的删除，如下所示：

```
if (msgctl(msgid, IPC_RMID, 0) == -1) {
      perror("delete msg");
      exit(errno);
 }
```

前台进程发送消息，使用 msgsnd 函数，如下所示：

```
if (-1 == msgsnd(msgid, (void *) &data, BUFFER_SIZE, 0)) {
      perror("send message");
      exit(errno);
}
```

后台进程接收消息，使用 msgrcv 函数，如下所示：

```
msgrcv(msgid, (void *) &data, BUFFER_SIZE, msg_type, 1);
```

2．3 使用命名管道进行进程间通信

命名管道是通过 FIFO 文件来进行进程间通讯的方式，在文件系统中以文件名的形式存在。通过对 FIFO 文件的四种不同的打开方式（阻塞只读，非阻塞只读，阻塞只写，非阻塞只写），可以实现生产者-消费者的进程间同步方式。 在本次设计中，后台进程通过只写方式打开命名管道，作为生产者写入数据；前台进程通过只读方事打开命名管道，作为消费者读取数据，实现了进程间的通信与同步。

FIFO 文件的创建使用 mkfifo 函数，如下所示：

```
#define FIFO_NAME "/tmp/mine_fifo"
if (0 != mkfifo(FIFO_NAME, 0777)) {
      perror("mkfifo");
      exit(errno);
}
```

前台进程使用只读方式代，如下所示：

```
filedsc = open(FIFO_NAME, O_RDONLY);//返回了文件描述符
if(-1==filedsc){//若描述符为-1说明打开出错a
      perror("front open FIFO");
      exit(errno);
}
```

后台进程使用只写方式代，如下所示：

```
filedsc = open(FIFO_NAME, O_WRONLY);//返回了文件描述符
```

```
if(-1==filedsc){//若描述符为-1说明打开出错a
        perror("front open FIFO");
        exit(errno);
}
```

前台进程读取数据，使用 read 函数，如下

```
read(filedsc, buffer, BUFFER_SIZE);
```

后台进程写入数据，使用 write 函数，如下：

```
write(filedsc, buffer, BUFFER_SIZE);
```

关闭 FIFO 文件，使用 close 函数，如下：

```
if (-1 == close(filedsc)) {
        perror("background fifo close");
        exit(errno);
}
```

运行完成后要删除 FIFO 文件，如下：

```
unlink(FIFO_NAME);
```

2．4 对输入消息进行处理和转化

用户在终端的标准输入下输入命令时，缓冲方式为行缓冲，使用 fgets 函数从标识输入下获取用户所输入的指令。获取的字符串存在字符数值 buffer 中。

```
fgets(buffer, BUFFER_SIZE, stdin);
```

由于指令和各个参数之间通过空格隔开，通过 string.h 下的 strtok 函数可以按顺序取出指令和各个参数。本次设计使用如下的循环获取指令与参数，结果放在 stack 这个字符指针数组中。

```
1  char *stack[100];
2  int top = -1;
3  command = strtok(buffer, " \n");
4  while (command != NULL) {
5          stack[++top] = command;
6          command = strtok(NULL, " \n");
7  }
```

命令位于数值第一个元素的位置，通过逐个的比较来识别 windows 命令，并将相应的 linux 指令加入消息块中，如下所示。其中有 dir 命令对应着 linux 的 pwd 和 cd 命令，可以通过当前参数的个数来具体确定。

```
struct msg_wrapper data;
command = stack[0];
if (0 == strcmp(command, "dir")) {
      msg_cmd(&data, "ls");
} else if (0 == strcmp(command, "rename")) {
      msg_cmd(&data, "mv");
} else if (0 == strcmp(command, "move")) {
      msg_cmd(&data, "mv");
} else if (0 == strcmp(command, "del")) {
```

```
            msg_cmd(&data, "rm");
        } else if (0 == strcmp(command, "copy")) {
            msg_cmd(&data, "cp");
        } else if (0 == strcmp(command, "md")) {
            msg_cmd(&data, "mkdir");
        }else if (0 == strcmp(command, "cd")) {
            if (top == 0) {
                msg_cmd(&data, "pwd");
            } else {
                msg_cmd(&data, "cd");
            }
        } else if (0 == strcmp(command, "exit")) {
            flag = 0;
            msg_cmd(&data, "exit");
        } else {
            continue;
        }
```

本次设计将原有的参数原封不动地加回了消息的指令中，如下：

```
for (i = 1; i <= top; i++) {
            msg_add(&data, stack[i]);
        }
```

2．5 执行转化后指令

使用 popen 即可可以执行一条 shell 命令，如下：

```
File* pipe = popen(data.text, "r");
```

popen 执行结束后，可以通过 fgets 逐行读取执行的结果：

```
while (fgets(temp, BUFFER_SIZE, pipe) != NULL) {
        strcat(buffer, temp);
}
```

出了 exit 命令以外，cd 命令也不可以使用 popen 执行，因为 popen 相当于创建了一个子进程。而在子进程中执行 cd 命令，对父进程的环境变量没有影响。因此应当检查是否为 cd 指令，并且使用 chdir 函数来改变当前目录。

```
 if (0 == strcmp(data.text, "exit")) {
        running = 0;
        continue;
 } else if (0 == strncmp(data.text, "cd", 2)) {
        strtok(data.text, " ");
        str = strtok(NULL, "\n ");
        if (chdir(str) == -1) {
            perror("change directory");
        }
 }
```

3. 细化的流程图如图 5 所示：

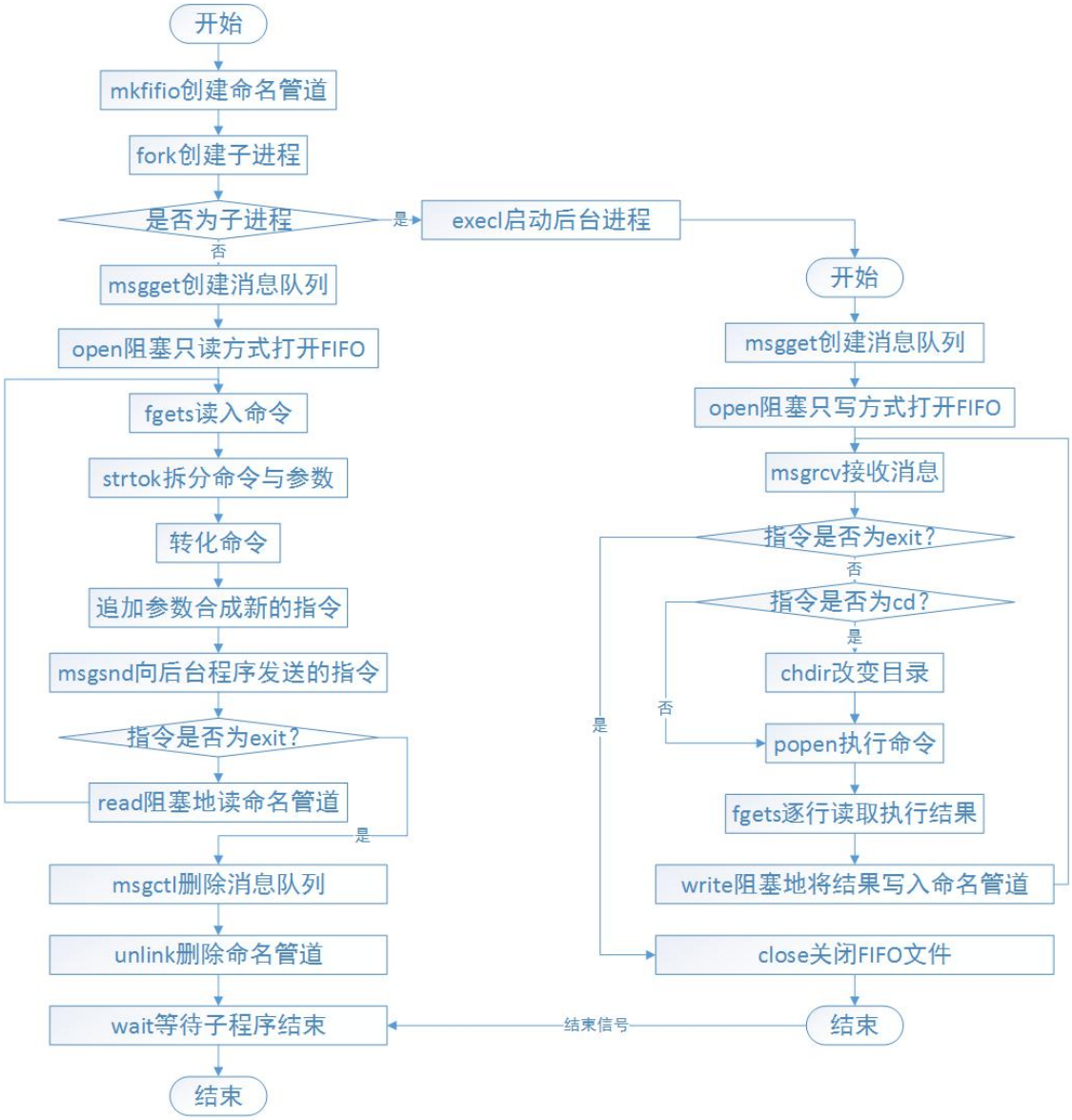

图 5：细化的前后台程序流程图

三、课程设计总结或结论

1. 本次课程设计编写了前台和后台两个进程，并使用了命名管道和消息队列两种进程间通信方式，实现了 linux 系统上多进程的创建与通信。

四、参考文献

[1] 宫虎波, Linux 编程从入门到精通. 化学工业出版社，第一版，2009 年 8 月出版

附录

1. msg_operation.h 头文件，定义了两个进程所公用的消息队列编号、命名管道名字、发送消息

的结构体，以及对这个结构体操作的几个函数。

```
//
// Created by yutao on 16-10-31.
//

#ifndef CMD_MESSAGE_H
#define CMD_MESSAGE_H
#endif //CMD_MESSAGE_H



#define BUFFER_SIZE 4096   //缓冲区的大小
#define MESSAGE_ID 973837 //消息队列的编号
#define FIFO_NAME "/tmp/mine10_fifo" //命名管道的名字

struct msg_wrapper{   //用于发送消息的数据结构
    long int type;
    char text[BUFFER_SIZE];
};
//以下为对消息内容进行操作的几个函数
int msg_del(struct msg_wrapper *msg);
int msg_add(struct msg_wrapper *msg, char *string);
int msg_cmd(struct msg_wrapper *msg, char *string);
```

2. msg_operation.c 源文件，实现了头文件中定义的几个函数。

```
//
// Created by yutao on 16-10-31.
//

#include "msg_operation.h"
#include "string.h"
//清空消息的内容
int msg_del(struct msg_wrapper *msg){
    memset(msg->text,0,BUFFER_SIZE);
    msg->type=1;
    return 1;
}
//在消息尾部追加内容，用空格隔开
int msg_add(struct msg_wrapper *msg, char *string) {
    strcat(msg->text," ");
    strcat(msg->text, string);
    return 1;
}
//在消息头部加入命令
int msg_cmd(struct msg_wrapper *msg, char *string){
    strcat(msg->text,string);
    return 1;
```

```
}
```

3. front.c，其中包含了前台进程的代码

```
//
// Created by yutao on 16-10-31.
//
#include <unistd.h>
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>
#include <sys/msg.h>
#include <string.h>
#include <sys/stat.h>
#include <fcntl.h>
#include "msg_operation.h"
#include <wait.h>

void main() {
    int flag = 1;
    int status = -1;
    int msgid, filedsc;
    char buffer[BUFFER_SIZE];
    char *command = NULL;
    struct msg_wrapper data;
    char *stack[100];
    int top, i;
    pid_t fpid;
    //使用fork创建
    fpid = fork();
    if (fpid == 0) {
        execl("./back", "back", "", NULL);
        perror("background");
        exit(errno);
    } else if (fpid < 0) {
        perror("fork");
        exit(errno);
    }
    //创建消息队列
    msgid = msgget((key_t) MESSAGE_ID, 0666 | IPC_CREAT);
    if (-1 == msgid) {
        perror("create message queue");
        exit(errno);
    }
    //创建FIFO
    if (0 != mkfifo(FIFO_NAME, 0777)) {
```

```
        perror("fifo");
        exit(errno);
    }
    //打开FIFO
    filedsc = open(FIFO_NAME, O_RDONLY);
    if(-1==filedsc){
        perror("front open FIFO");
        exit(errno);
    }
    while (flag) {
        printf(">>");
        msg_del(&data);
        //读入标准输入
        fgets(buffer, BUFFER_SIZE, stdin);
        //将输入拆分成各个字符
        top = -1;
        command = strtok(buffer, " \n");
        while (command != NULL) {
            stack[++top] = command;
            command = strtok(NULL, " \n");
        }
        if (top < 0) {
            continue;
        }
        //指令是第一个字符串
        command = stack[0];
        //指令是数
        if (0 == strcmp(command, "dir")) {
            msg_cmd(&data, "ls");
        } else if (0 == strcmp(command, "rename")) {
            msg_cmd(&data, "mv");
        } else if (0 == strcmp(command, "move")) {
            msg_cmd(&data, "mv");
        } else if (0 == strcmp(command, "del")) {
            msg_cmd(&data, "rm");
        } else if (0 == strcmp(command, "touch")) {
            msg_cmd(&data, "touch");
        } else if (0 == strcmp(command, "copy")) {
            msg_cmd(&data, "cp");
        } else if (0 == strcmp(command, "md")) {
            msg_cmd(&data, "mkdir");
        }else if (0 == strcmp(command, "cd")) {
            if (top == 0) {
                msg_cmd(&data, "pwd");
            } else {
                msg_cmd(&data, "cd");
```

```
            }
        } else if (0 == strcmp(command, "exit")) {
            flag = 0;
            msg_cmd(&data, "exit");
        } else {
            continue;
        }
        if (flag) {
            //将参数追加到消息的尾部
            for (i = 1; i <= top; i++) {
                msg_add(&data, stack[i]);
            }
        }
        //向子进程发送消息
        if (-1 == msgsnd(msgid, (void *) &data, BUFFER_SIZE, 0)) {
            perror("send message");
            exit(errno);
        }
        if (flag) {
        //从FIFO文件中阻塞地读取数据
            read(filedsc, buffer, BUFFER_SIZE);
            printf("%s", buffer);
        }
    }
    //循环结束，删除消息队列
    if (msgctl(msgid, IPC_RMID, 0) == -1) {
        perror("delete msg");
        exit(errno);
    }
    //关闭FIFO文件
    if(-1==close(filedsc)){
        perror("front close FIFO");
        exit(errno);
    }
    //删除FIFO文件
    unlink(FIFO_NAME);
    //等待后台进程退出
    wait(&status);
    //用于显示前后台进程的退出顺序°
    printf("foreground exited\n");
}
```

4. back.c，其中包含了后台进程的代码

```
//
// Created by yutao on 16-10-31.
//
#include <stdio.h>
```

```
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <stdlib.h>
#include <fcntl.h>
#include "msg_operation.h"
#include <errno.h>
#include <sys/msg.h>


void main() {
    int running = 1;
    int msg_type = 0;
    FILE *pipe;
    char buffer[BUFFER_SIZE];
    char temp[BUFFER_SIZE], *str;
    struct msg_wrapper data;
    int filedsc, msgid;

    //创建消息队列
    msgid = msgget((key_t) MESSAGE_ID, 0666 | IPC_CREAT);
    if (-1 == msgid) {
        perror("create message queue");
        exit(errno);
    }
    //阻塞写方式打开FIFO文件
    filedsc = open(FIFO_NAME, O_WRONLY);
    if(-1==filedsc){
        perror("back open FIFO");
        exit(errno);
    }
    while (running) {
        //接收消息
        msg_del(&data);
        msgrcv(msgid, (void *) &data, BUFFER_SIZE, msg_type, 1);
        if (0 == strcmp(data.text, "exit")) {
            running = 0;
            continue;
        } else if (0 == strncmp(data.text, "cd", 2)) {
            strtok(data.text, " ");
            str = strtok(NULL, "\n ");
            printf("%s\n", str);
            if (chdir(str) == -1) {
                perror("change directory");
            }
```

```
            strcpy(data.text, "pwd");
        }
        //执行命令，读取结果并写入FIFO
        pipe = popen(data.text, "r");
        memset(buffer, 0, BUFFER_SIZE);
        while (fgets(temp, BUFFER_SIZE, pipe) != NULL) {
            strcat(buffer, temp);
        }
        pclose(pipe);
        write(filedsc, buffer, BUFFER_SIZE);

    }
    //关闭FIFO文件
    if (-1 == close(filedsc)) {
        perror("background fifo close");
        exit(errno);
    }
    //用于显示前后台退出的顺序
    printf("background exited\n");
}
```