操作系统实验报告：实验一

姓名 周华平　　学号 13061108

1 需求说明

1.1 基本要求和提高要求

基本要求：

1. 实现以下内部命令：exit, jobs, history, fg, bg
2. 通过组合键实现前后台切换
3. 提供对 IO 重定向的支持
4. 使用 YACC 进行语法分析

提高要求：

1. 将 flex 和 bison 结合起来使用进行语法和词法分析
2. 通配符的支持与实现
3. 实现对管道的支持
4. 实现 ctrl+c

1.2 完成情况

本次实验完成了以上所有要求

2 设计说明

2.1 基本要求实现说明

1. fg 指令的完善

在 fg_exec 函数中 waitpid(fgPid, NULL, 0);前增加一句 sleep(1);，目的是使父进程等待子进程，避免提前结束。

2. ctrl+z 的完善

需要在 rmJob 函数 free(now);语句前增加 wait(NULL);语句，目的是捕获到后台进程的结束信号，避免僵尸进程。

3. 屏蔽子进程中的 SIGINT 以及 SIGTSTP 信号

目的是使子进程的挂起以及结束只能依靠父进程发出的信号，避免产生僵尸进程。

信号屏蔽用以下的代码实现：

```
sigemptyset(&sint);
sigaddset(&sint, SIGINT);
sigprocmask(SIG_BLOCK, &sint, NULL);
sigemptyset(&sstp);
sigaddset(&sstp, SIGTSTP);
```

4. 使用 GDB 工具发现并解决了源代码中若干内存泄露以及重复 free 的问题。

2.2 提高要求实现说明

1. 将 flex 和 bison 结合起来使用进行语法和词法分析

编写 lex_analysis.l 文件，来替代 bison.y 中 yylex()函数的功能。并使用 yy_scan_string 函数将输入重定向到 inputBuff 字符串中。

2. 通配符的支持与实现

在执行指令之前，对 CMD 中的 args 进行处理，对其中可以扩展的参数进行扩展，然后再执行指令。由于使用了系统调用 glob()函数来实现通配符,所以除了可以支持*, ?通配符号之外，还可以支持系统自带 Shell 所支持的所有通配符。

3. 实现对管道的支持

在实现管道的时候我采用了逐段处理的原则，通过改变各段指令的读写指针将管道连接起来。主要代码如下：

```
void execute_cmplx(){
    int i=0,j=0;
    int flag=0;//flag 用来判断是否关闭上一个管道的读指针
    int pipe_fd[2]={0,1};
    //初始化管道指针初值
    inPipe=0;
    outPipe=1;
    for(j=0;inputBuff[j]!='\0';++j){
        if(inputBuff[j]=='|'){//找到管道标志
            if(pipe(pipe_fd)<0){//创建管道
                perror("pipe failed");
```

```
            exit(errno);
        }
        outPipe=pipe_fd[1];//将这条指令的写指针改为 pipe 的写指针
        execute(i,j);//执行这条命令
        if(flag==1){//该指令前存在管道指令
            close(inPipe);//关闭上一个管道的读指针?关于顺序有疑问
        }
        else{
            flag=1;//否则置位
        }
        close(outPipe);//关闭父进程的写指针
        inPipe=pipe_fd[0];//将管道读指针改成下一条指令的读指针
        i=j+1;//移动字符串头指针
    }
}
outPipe=1;//重置写指针
execute(i,strlen(inputBuff));
if(flag==1)
    close(inPipe);
}
```

4. 实现 ctrl+c

ctrl+c 与 ctrl+z 的实现方法差别不大，最大的区别就是发送给子进程的信号不同，所以能够产生不同的行为。主要代码如下：

```
void ctrl_C(){
    Job *now = NULL;

    if(fgPid == 0){ //前台没有作业则直接返回
        return;
    }
    now = head;
```

```
    while(now != NULL && now->pid != fgPid)
        now = now->next;

    if(now == NULL){ //未找到前台作业，则根据 fgPid 添加前台作业
        now = addJob(fgPid);
    }

    //修改前台作业的状态及相应的命令格式，并打印提示信息
    strcpy(now->state, KILLED);
    printf("[%d]\t%s\t\t%s\n", now->pid, now->state, now->cmd);

    //发送 SIGKILL 信号给正在前台运作的工作，将其杀死
    kill(fgPid, SIGKILL);
    fgPid = 0;
}
```

3 收获与感想

3.1 给予你帮助的人

在做实验的过程中遇到过不少困难，比如说一开始我对于信号通信的概念不太了解，GDB 工具也不太会使。在这期间周围寝室的同学们给我提供了很大的帮助，使得我能够跟上实验的进度。当然，度娘和谷歌娘的协助也是必不可少的。

3.2 从实验中学到的东西

在这次试验中，我学会了 GitHub 的使用方法，了解了 Linux 中的一些系统调用。在真正写了一个 shell 程序之后，我才开始对于 shell 的实现机制有了初步的了解。虽然写出来的程序依然很简单，其中可能依旧存在隐藏的 BUG，但是我认为作为 OS 的第一个实验，完成度还算不错，也算是一个好的开始吧。希望之后的 OS 实验能够做得更好。