电子科技大学

计算机科学与工程学院

标 准 实 验 报 告

（实验）课程名称 Unix 操作系统

电子科技大学教务处制表

电 子 科 技 大 学

实 验 报 告

学生姓名：刘芷溢　　学　号：2020080907009

指导教师：聂晓文　　实验地点：主楼 A2-412

实验时间：2022/5/28

一、实验室名称：计算机学院实验中心

二、实验项目名称：模拟管道操作及 makefile 管理工程

三、实验学时：4 学时

四、实验原理：C 语言模拟管道、makefile 基本原理

五、实验目的：

理解管道命令原理，利用 C 语言实现管道；采用 makefile 管理该工程；

六、实验内容：

1. 编程模拟管道
2. 利用 Makefile 管理工程

七、实验器材（设备、元器件）：

PC 微机一台；

八、实验步骤：

1.下载 VMware 虚拟机；
2.在虚拟机上安装 Ubuntu 服务器版本，进行环境配置；
3.执行实验要求的各项操作；

九、实验数据及结果分析：

1.模拟管道源程序，模拟 ls | wc -w 命令：

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <sys/time.h>
#include <fcntl.h>
#include <errno.h>
#include <string.h>
#include <dirent.h>
#include <signal.h>
#define ERR_EXIT(m)       \
        do                \
        {                 \
                perror(m); \
                exit(EXIT_FAILURE); \
        }while(0)\


int main(int argc, char* argv[]){
        int pipefd[2];          //创建数组作为pipe的参数
        //创建一个管道，错误时打印错误信息
        if (pipe(pipefd)==-1){
                ERR_EXIT("pipe error");
        }
        //创建子进程
        pid_t pid;
        pid=fork();
        //进入子进程
        if(pid==0){
                //将文件输出描述符重定向到写端，关闭读口和写口
                dup2(pipefd[1],STDOUT_FILENO);
                close(pipefd[1]);
                close(pipefd[0]);
                //执行ls命令
                execlp("ls","ls",NULL);
                fprintf(stderr,"error exec ls\n");
                exit(EXIT_FAILURE);
        }
        //重定向读端，关闭读口和写口
        dup2(pipefd[0],STDIN_FILENO);
        close(pipefd[0]);
        close(pipefd[1]);
        //执行wc -w命令
        execlp("wc","wc","-w",NULL);
        fprintf(stderr,"error execute wc \n");
        return 0;
}
```

Gdb 调试程序:

gcc -g test.c -o test

List：显示源码

```
unix@unix:~/exam4/exam5$ gdb test -q
Reading symbols from test...
(gdb) l
7       #include <sys/time.h>
8       #include <fcntl.h>
9       #include <errno.h>
10      #include <string.h>
11      #include <dirent.h>
12      #include <signal.h>
13      #define ERR_EXIT(m)       \
14              do                \
15              {                 \
16                      perror(m); \
(gdb)
17                      exit(EXIT_FAILURE); \
18              }while(0)\
19
20
21      int main(int argc, char* argv[]){
22              int pipefd[2];
23              if (pipe(pipefd)==-1){
24                      ERR_EXIT("pipe error");
25              }
26
(gdb)
27              pid_t pid;
28              pid=fork();
29              if(pid==0){
30                      dup2(pipefd[1],STDOUT_FILENO);
31                      close(pipefd[1]);
32                      close(pipefd[0]);
33                      execlp("ls","ls",NULL);
34                      fprintf(stderr,"error exec ls\n");
35                      exit(EXIT_FAILURE);
36              }
(gdb)
37              dup2(pipefd[0],STDIN_FILENO);
38              close(pipefd[0]);
39              close(pipefd[1]);
40              execlp("wc","wc","-w",NULL);
41              fprintf(stderr,"error execute wc \n");
42              return 0;
43      }
```

b 21：在 21 行的程序打断点

r：运行程序

n：下一行程序执行

```
Starting program: /home/unix/exam4/exam5/test

Breakpoint 1, main (argc=21845, argv=0x7ffff7fc02e8 <__exit_funcs_lock>) at test.c:21
21      int main(int argc, char* argv[]){
(gdb) n
23              if (pipe(pipefd)==-1){
(gdb) n
28              pid=fork();
(gdb) n
[Detaching after fork from child process 2942]
29              if(pid==0){
(gdb) n
37              dup2(pipefd[0],STDIN_FILENO);
(gdb) n
38              close(pipefd[0]);
(gdb) n
39              close(pipefd[1]);
(gdb) n
40              execlp("wc","wc","-w",NULL);
(gdb) n
process 2936 is executing new program: /usr/bin/wc
Error in re-setting breakpoint 1: No source file named /home/unix/exam4/exam5/test.c.
5
[Inferior 1 (process 2936) exited normally]
(gdb) n
The program is not being run.
```

gcc -E test.c -o test.i

gcc -S test.i -o test.s

gcc -c test.s -o test.o

gcc test.o -o test

产生的 test 为可执行文件;

在 test.c 的目录下创造 Makefile 文件，利用 makefile 管理工程;

Makefile：这是一个文件的依赖关系，也就是说，target 这一个或多个的目标文件依赖于 prerequisites 中的文件，其生成规则定义在 command 中。

```
test: test.o
	gcc -o test test.o
test.o: test.c
	gcc -c test.c

clean:
	rm *.o
	rm test
```

测试结果:

目录下文件

```
unix@unix:~/exam4/exam5$ ls
Makefile  test  test.c  test.i  test.o  test.s
unix@unix:~/exam4/exam5$
```

结果:

```
unix@unix:~/exam4/exam5$ ls | wc -w
6
unix@unix:~/exam4/exam5$ ./test
6
unix@unix:~/exam4/exam5$ make
gcc -c test.c
gcc -o test test.o
unix@unix:~/exam4/exam5$ make
make: 'test' is up to date.
```

十、实验结论：

通过对管道的模拟编程，我对管道的原理有了更深的认识；利用 makefile 管理工程，让我对 makefile 有了更深的理解。

十一、总结及心得体会：

通过这次实验我了解了 makefile 基于依赖对工程进行管理; 对管道的原理和命令有了进一步的认识。

十二、对本实验过程及方法、手段的改进建议：

希望可以实现 shell 的模拟和管理。

报告评分：

指导教师签字：