《操作系统原理》实验报告

------实验一 shell 模拟

年级：2022 级　班级：智算大类十二班 学号：3523403837 姓名：邢俊杰

一、实验题目及要求

(1)题目：编写一个 Linux 下的简单 shell

(2)要求：实现运行原生的 Linux 系统，在代码 sh.c 的基础上完善代码，实现输入输出重定向>、<，管道|等命令要求的简单 shell

二、设计说明

(1)功能：用于实现输出重定向、输入重定向和管道功能。

(2)结构：代码分为三个结构：在输出重定向中，我们打开一个文件，然后调整进程的 stdio 文件描述符以便该进程从或向这个文件读取或写入数据；在管道中，我们创建一个管道，然后调整进程的 stdio 文件描述符以便左边的命令从管道的读取端读取数据，右边的命令向管道的写入端写入数据。

(3)原理：核心原理在于使用系统调用和文件描述符重定向来实现输出、输入的重定向，以及通过管道实现进程间通信。这样，就可以模拟一些基本的 shell 功能，例如将一个命令的输出传递给另一个命令，或者将输出写入到文件中。

三、编译、运行、测试说明

1.准备源代码：将您的代码保存在一个以 .c 为扩展名的源代码文件中，比如 sh.c。

2.打开终端：在 Linux 系统上打开终端。

3.使用编译器编译代码：使用 gcc 来编译代码。使用./a.out 来验证代码的正确性。

4.运行可执行文件：一旦编译成功，可以运行 shell，这将启动简单 shell，并可以开始输入命令和测试功能。

四、实验过程

（1）执行简单命令

```
//普通的执行命令，它会打印 "exec not implemented" 消息
case ' ':
    ecmd = (struct execcmd*)cmd;
    if(ecmd->argv[0] == 0)
      exit(0);
    fprintf(stderr, "exec not implemented\n");
    // Your code here ...
    if(execvp(ecmd->argv[0],ecmd->argv) < 0) {
      fprintf(stderr,"Command not found: %s\n",ecmd->argv[0]);
    }
    break;
```

通过修改 runcmd 中 case ‘’ 中的代码使函数可以执性用户输入的命令，并且在执性失败时会提示打印信息。

```
2440078$ ls
exec not implemented
a.out  sh.c  t.sh  x.txt  操作系统原理作业.pdf
2440078$ /bin/ls
exec not implemented
a.out  sh.c  t.sh  x.txt  操作系统原理作业.pdf
2440078$ " "
exec not implemented
Command not found: "
2440078$ hhh
exec not implemented
Command not found: hhh
2440078$
```

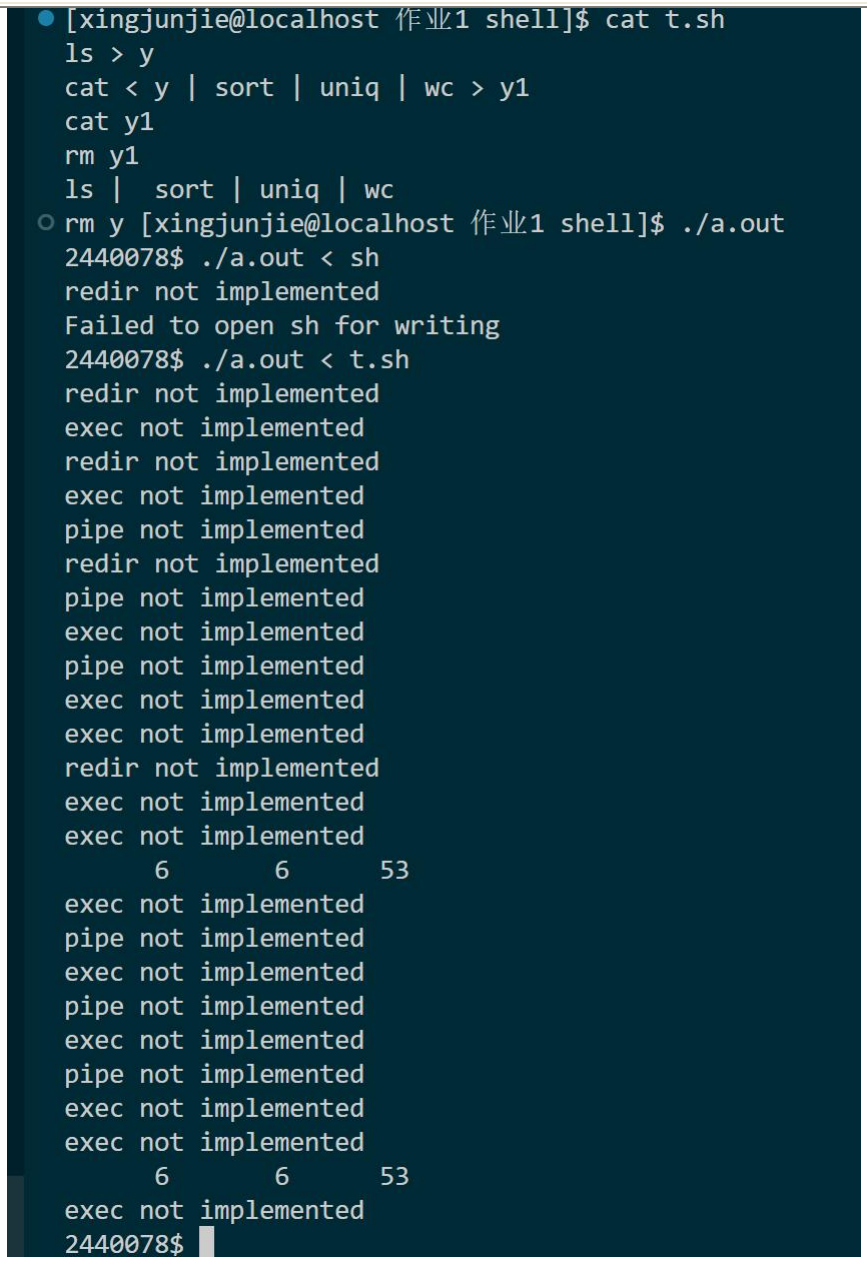
测试成功，在找不到命令时会提示

（2）输入输出重定向

重定向命令，它会递归调用 runcmd 来处理子命令，在原有的 runcmd 代码中解析器已经可以识别<和>符号，实现其功能，我们需要打开一个文件，然后调整进程的 stdio 文件描述符以便该进程从或向这个文件读取或写入数据，在这个过程中，我们使用 open 跟 close 来打开关闭文件以完成输入输出流

```
75      // Your code here ...
76      //输出
77      int fd_out = open(rcmd->file, rcmd->mode, 0666);
78      if (fd_out < 0) {
79        fprintf(stderr, "Failed to open %s for writing\n", rcmd->file);
80        exit(-1);
81      }
82      if (dup2(fd_out, rcmd->fd) < 0) {
83        fprintf(stderr, "Failed to redirect output\n");
84        exit(-1);
85    }
86      close(fd_out);
87    //输入
88      int fd_in = open(rcmd->file, rcmd->mode);
89      if (fd_in < 0) {
90        fprintf(stderr, "Failed to open %s for reading\n", rcmd->file);
91        exit(-1);
92      }
93      if (dup2(fd_in, rcmd->fd) < 0) {
94        fprintf(stderr, "Failed to redirect input\n");
95        exit(-1);
96      }
97      close(fd_in);
98      runcmd(rcmd->cmd);
99      break;
```

经过测试



信息打印成功

（3）管道操作

对于管道，我们首先需要创建一个管道，然后调整进程的 stdio 文件描述符以便左边的命令从管道的读取端读取数据，右边的命令向管道的写入端写入数据。在其中我们用到了 pipe、fork、close、dup 等系统的调用，其中我们使用使用 pipe 系统调用创建管道，fork 创建子进程，关闭不需要的文件描述符，最后使用 dup()系统调用将文件描述符复制到新的文件描述符，这样就可以打开新的文件或者关闭原有的文件描述符

```
// Your code here ...
int pipefd[2];
if (pipe(pipefd) < 0) {
  fprintf(stderr, "Failed to create pipe\n");
  exit(-1);
}
int pid1 = fork();
if (pid1 == 0) {
  close(pipefd[0]); // Close the read end
  dup2(pipefd[1], 1); // Redirect stdout to the pipe
  close(pipefd[1]);
  runcmd(pcmd->left);
  exit(0);
} else if (pid1 < 0) {
  fprintf(stderr, "Failed to fork\n");
  exit(-1);
}
int pid2 = fork();
if (pid2 == 0) {
  close(pipefd[1]); // Close the write end
  dup2(pipefd[0], 0); // Redirect stdin to the pipe
  close(pipefd[0]);
  runcmd(pcmd->right);
  exit(0);
} else if (pid2 < 0) {
  fprintf(stderr, "Failed to fork\n");
  exit(-1);
}
close(pipefd[0]);
close(pipefd[1]);
waitpid(pid1, NULL, 0);
waitpid(pid2, NULL, 0);
```

根据上次运行结果，该运行可以成功！

五、实验总结

通过本次实验，我们成功实现了一个基本的 Unix Shell，具备了运行一个原生 Linux 程序、输入输出重定向和管道命令的功能。其中在使用输出或输入重定向时，要确保指定的文件路径是正确的。而相对路径会相对于当前工作目录，绝对路径则是从根目录开始的完整路径，并且运行代码时可能会遇到各种运行时错误，如命令无法执行、文件无法打开等。我们需要添加错误处理机制或者输出明显提示信息来处理这些情况，例如使用 perror 来打印错误消息。本次实验使我们更紧密深入的了解到了 Shell 命令的实现，在接下来的学习中，将继续学习和拓展 Shell 命令的功能，以便更好地理解操作系统是怎么实现的。

完成日期：2023 年 10 月 15 日