# 操作系统原理实验报告

## 实验一 小型 shell 模拟

3022206045-陆子毅

**实验内容：**

实现运行原生的 linux 程序，实现输入输出重定向，管道符

**实验过程：**

**一、执行简单命令**

根据手册指引，找到了 runcmd 所在位置，并用 man 3 exec 查看了 exec 函数原型。在 runcmd 函数中，编写了代码来处理简单命令。使用 execv 函数来执行用户输入的命令，并在执行失败时打印错误消息。

```
SYNOPSIS
    #include <unistd.h>

    extern char **environ;

    int execl(const char *pathname, const char *arg, ...
              /* (char  *) NULL */);
    int execlp(const char *file, const char *arg, ...
              /* (char  *) NULL */);
    int execle(const char *pathname, const char *arg, ...
              /*, (char *) NULL, char *const envp[] */);
    int execv(const char *pathname, char *const argv[]);
    int execvp(const char *file, char *const argv[]);
    int execvpe(const char *file, char *const argv[],
              char *const envp[]);

  Feature Test Macro Requirements for glibc (see feature_test_macros(7)):


  v - execv(), execvp(), execvpe()
      The char *const argv[] argument is an array of pointers to null-terminated strings
that
      represent  the argument list available to the new program.  The first argument, by
con-
      vention, should point to the filename associated with the file being executed.  The
ar-
      ray of pointers must be terminated by a null pointer.
```

```
case ' ':
  ecmd = (struct execcmd*)cmd;
  if(ecmd->argv[0] == 0)
    exit(0);
  execvp(ecmd->argv[0], ecmd->argv);
  perror("execvp");
  //fprintf(stderr, "exec not implemented\n");
  // Your code here ...
  break;
```

测试成功，并且在运行命令时不用写/bin

**二、输入输出重定向**

解析器已经能够识别>和<符号，并构建了 redircmd。只需要填写 runcmd 函数中>和<的代码部分，根据实验提示，应该使用 open 和 close 等系统调用。我们确保在系统调用失败时打印错误消息。

使用命令查看 open，close 的函数原型使用说明。

```
SYNOPSIS
    #include <sys/types.h>
    #include <sys/stat.h>
    #include <fcntl.h>

    int open(const char *pathname, int flags);
    int open(const char *pathname, int flags, mode_t mode);

    int creat(const char *pathname, mode_t mode);

    int openat(int dirfd, const char *pathname, int flags);
    int openat(int dirfd, const char *pathname, int flags, mode_t mode);

    /* Documented separately, in openat2(2): */
    int openat2(int dirfd, const char *pathname,
            const struct open_how *how, size_t size);


    SYNOPSIS
        #include <unistd.h>

        int close(int fd);
```

仔细阅读下面的说明后，填写代码

```
case '>':
case '<':
  rcmd = (struct redircmd*)cmd;
  int file_fd = open(rcmd->file, rcmd->mode, 0666);
  if (file_fd < 0) {
      perror("open");
      exit(-1);
  }
  if (dup2(file_fd, rcmd->fd) < 0) {
      perror("dup2");
      exit(-1);
  }
  close(file_fd);
  // Your code here ...
  runcmd(rcmd->cmd);
  //fprintf(stderr, "redir not implemented\n");
  break;
```

测试运行，成功

```
2440078$ ./a.out < sh
open: No such file or directory
2440078$ ./a.out < t.sh
    6    10    58
    6    10    58
rm: cannot remove 'y1': No such file or directory
cat: y1: No such file or directory
rm: cannot remove 'y1': No such file or directory
    5     9    56
rm: cannot remove 'yrm': No such file or directory
rm: cannot remove 'y': No such file or directory
2440078$
```

**三、管道操作**

管道操作需要用到使用了 pipe、fork、close 和 dup 等系统调用。使用 man 命令查询具体函数。

使用 pipe 系统调用创建管道，读取数据的文件描述符在子进程中使用，而写入数据的文件描述符在父进程中使用。需要创建两个子进程，一个用于执行前一个命令，另一个用于执行后一个命令。父进程则用于等待这两个子进程的完成。在子进程中，需要将标准输出或标准输入重定向到管道的读或写文件描述符。这可以使用 dup2 系统调用完成。例如，将标准输出重定向到管道的写入端。

如果有多个管道命令，可以使用循环创建多个管道和子进程来处理它们。

```c
case '|':
  pcmd = (struct pipecmd*)cmd;
  int pipe_fd[2];


  if (pipe(pipe_fd) < 0) {
      perror("pipe");
      exit(-1);
  }


  int left_pid = fork1();
  if (left_pid == 0) {
      // 子进程：左侧命令
      close(pipe_fd[0]); // 关闭管道的读端
      dup2(pipe_fd[1], 1); // 将标准输出重定向到管道写端
      close(pipe_fd[1]); // 关闭管道的写端
      runcmd(pcmd->left);
  }


  int right_pid = fork1();
  if (right_pid == 0) {
      // 子进程：右侧命令
      close(pipe_fd[1]); // 关闭管道的写端
      dup2(pipe_fd[0], 0); // 将标准输入重定向到管道读端
      close(pipe_fd[0]); // 关闭管道的读端
      runcmd(pcmd->right);
  }


  // 关闭父进程中的管道文件描述符
  close(pipe_fd[0]);
  close(pipe_fd[1]);


  // 等待左右两个子进程结束
  int status;
  waitpid(left_pid, &status, 0);
  waitpid(right_pid, &status, 0);
  // Your code here ...
  break;
  //fprintf(stderr, "pipe not implemented\n");
}
```

## 实验总结：

实验不是很难，但是需要我们去认证阅读 linux 编程手册，在这期间，准确快速地阅读英文手册是一个难点，希望能够通过不断的锻炼渐渐摆脱翻译软件。

通过本次实验，我们成功实现了一个基本的 Unix Shell，具备了运行原生 Linux 程序、输入输出重定向和管道操作的功能。我们熟悉了系统调用接口和 Shell 的工作原理，提高了操作系统原理的实际编程能力。本次实验使我们更深入地理解了操作系统原理，特别是 Shell 的实现。我们成功地实现了所需的功能，同时也学到了如何查看系统调用文档和调试 Shell 程序。在接下来的实验中，我们将继续学习和扩展 Shell 的功能，以更好地理解操作系统的工作原理。