Homework1 - Shell

1. Executing simple commands

在实验要求中曾出现过一下片段

> This prints a prompt and waits for input. sh.c prints as prompt 6.828$ so that you don't get confused with your computer's shell. Now type to your shell:
>
> ```
> 6.828$ ls
> ```
>
> Your shell may print an error message (unless there is a program named ls in your working directory or you are using a version of exec that searches PATH). Now type to your shell:
>
> ```
> 6.828$ /bin/ls
> ```
>
> This should execute the program /bin/ls, which should print out the file names in your working directory. You can stop the 6.828 shell by typing ctrl-d, which should put you back in your computer's shell.

可以得知，如果shell的指令能去调用/bin目录下的指令就可以执行。

那么我们可以用一些接口函数去调用相应的指令，从而达到实现指令执行的目的。

```
case ' ':
        ecmd = (struct execcmd *)cmd;
        if (ecmd->argv[0] == 0)
            _exit(0);

        if (access(ecmd->argv[0], F_OK) == 0) //用access函数查看对应文件是否可以访问
        {
            char *file_path = ecmd->argv[0];
            execv(file_path, ecmd->argv); //执行对应文件的指令
        }
        else
        {
            char *file_path = (char *)malloc(sizeof(char) * (5 + strlen(ecmd-
>argv[0])));
            strcpy(file_path, "/bin/");
            file_path = strcat(file_path, ecmd->argv[0]); //使用bin目录下的指令
            if (access(file_path, F_OK) == 0)
            {
                execv(file_path, ecmd->argv);
            }
            else
            {
                fprintf(stderr, "%s: Command not found\n", ecmd->argv[0]); //调用指令失
败
            }
        }
        break;
```

2. I/O redirection

根据xv6 book出现的片段来说

```
char *argv[2];
argv[0] = "cat";
argv[1] = 0;
if(fork() == 0) {
  close(0);
  open("input.txt", O_RDONLY);
  exec("cat", argv);
}
```

如果想重定向，那么就要把对应的file descriptor关掉，并且赋予新的输入/输出源

因为每一次分配都是最小的未分配的file descriptor， 所以close之后再open就一定是刚才关掉的fd

```
case '>':
case '<':
        rcmd = (struct redircmd *)cmd;

        // Your code here ...
        close(rcmd->fd); // 关掉对应fd
        if (open(rcmd->file, rcmd->flags, 0777) < 0) //重新打开fd，  并且要赋予权限，不然
可能会失败
        {
            fprintf(stderr, "ERROR: %s Open failed!\n", rcmd->file);
            exit(0);
        }

        runcmd(rcmd->cmd);
        break;
```

3. Implement pipes

对于管道左边的部分，我们需要把左边的输出当做管道的输入，那么这时候我们就要把管道的写端口重定向到标准输出端口，同理，我们需要把管道的读端口重定向到标准输出端口，从而实现管道通信

```
case '|':
        pcmd = (struct pipecmd *)cmd;
        // fprintf(stderr, "pipe not implemented\n");
        //  Your code here ...

        if (pipe(p) < 0)
        {
            fprintf(stderr, "Create pipe failed\n");
            exit(0);
        }

        int t1 = fork();
        if (t1 == 0)
        {
            close(1);
            dup(p[1]); //关闭输出接口，并重定向到管道写端口
            close(p[0]);
            close(p[1]);
            runcmd(pcmd->left);
        }
        else
        {
            wait(&r);
            int t2 = fork1();
            if (t2 == 0)
            {
                close(0);
                dup(p[0]); //关闭输入接口，并重定向到管道读接口
                close(p[0]);
                close(p[1]);
                runcmd(pcmd->right);
            }
            else
            {
                close(p[0]);
                close(p[1]);
                wait(&r);
            }
        }

        break;
```

4. 感想

比较清晰的认识了执行命令，重定向和管道通信的指令原理。