**这个任务将使您更加熟悉Unix系统调用接口和shell，在一个小型shell中实现了几个功能。您可以在支持Unix API的任何操作系统上完成此任务。**

**下载6.828 shell，并查看其中内容。6.828 shell包含两个主要部分：解析shell命令和实现它们。解析器仅识别如下所示的简单shell命令：**

**ls > y**

**cat < y | sort | uniq | wc > y1**

**cat y1**

**rm y1**

**ls | sort | uniq | wc**

**rm y**

**将这些命令复制粘贴到一个名为t.sh的文件中。**

**如果您正在使用自己的计算机，可能需要安装gcc。**

**一旦安装了gcc，可以按以下方式编译骨架shell：**

**$ gcc sh.c**

**这会产生一个名为a.out的文件，可以运行它：**

**$ ./a.out < t.sh**

**这个执行过程会打印错误消息，因为您还没有实现几个功能。**

**在后续的任务中，您将实现这些功能。**

**执行简单命令**

**实现简单命令，例如：**

**$ ls**

**解析器已经为您构建了一个execcmd，所以您只需在runcmd中为' '情况编写代码。您可能会发现查看exec的手册页面很有用；键入"man 3 exec"，并阅读有关execv的信息。当exec失败时，打印错误消息。**

**要测试程序，请编译并运行生成的a.out：**

**6.828$./a.out**

**这将打印一个提示并等待输入。sh.c以6.828$为提示打印，这样您就不会与计算机的shell混淆。现在在您的shell中键入：**

**6.828$ ls**

**您的shell可能会打印一个错误消息（除非您的工作目录中有一个名为ls的程序，或者您正在使用一个在PATH中搜索的exec版本）。现在在您的shell中输入：**

**6.828$ /bin/ls**

**这将执行程序/bin/ls，它会打印出您工作目录中的文件名。您可以通过键入ctrl-d来停止6.828 shell，这会将您带回计算机的shell中。**

**您可能希望更改6.828 shell，以便始终尝试使用/bin路径，如果该程序在当前工作目录中不存在，以便在下面的步骤中不必为每个程序输入"/bin"。如果您有雄心壮志，可以实现对PATH变量的支持。**

**I/O重定向**

**实现I/O重定向命令，以便您可以运行：**

**echo "6.828 is cool" > x.txt**

**cat < x.txt**

**解析器 解析器已经识别">"和"<"，并为您构建了并为您构建了redircmd，因此您只需在runcmd中填写与这些符号相对应的缺失代码。您可能会发现open和close的man页很有用。**

**请注意，redircmd中的mode字段包含访问模式（例如O\_RDONLY），您应该将其作为flags参数传递给open函数；参考parseredirs中shell使用的模式值以及open的手册页面中有关flags参数的说明。**

**确保在使用的系统调用失败时打印错误消息。**

**确保您的实现能够正确运行上述测试输入。常见错误是忘记指定创建文件时的权限（即open的第三个参数）。**

**实现管道功能**

**实现管道功能，以便您可以运行如下的命令流水线：**

**$ ls | sort | uniq | wc**

**解析器已经识别了"|"符号，并为您构建了pipecmd，所以您只需在runcmd中为"|"情况编写代码。您可能会发现pipe、fork、close和dup的man页很有用。**

**测试您能否运行上述命令流水线。sort程序可能在目录/usr/bin/中，如果是这样的话，您可以输入绝对路径名/usr/bin/sort来运行sort。（在您计算机的shell中，您可以输入which sort来找出shell搜索路径中具有名为"sort"的可执行文件的目录。）**

**现在，您应该能够正确运行以下命令：**

**6.828$ a.out < t.sh**

**请确保您使用的是正确的程序绝对路径名。**

#include<stdio.h>

#include<unistd.h>

#include<stdlib.h>

#include<fcntl.h>

int main() {

int fileDescriptor;

int savedStdout;

// 打开要写入的文件

fileDescriptor = open("output.txt", O\_WRONLY | O\_CREAT | O\_TRUNC, S\_IRUSR | S\_IWUSR);

if (fileDescriptor == -1) {

perror("open");

exit(1);

}

// 保存原始的stdout文件描述符

savedStdout = dup(STDOUT\_FILENO);

if (savedStdout == -1) {

perror("dup");

exit(1);

}

// 用文件描述符替换stdout

if (dup2(fileDescriptor, STDOUT\_FILENO) == -1) {

perror("dup2");

exit(1);

}

// 在stdout中写入内容

printf("This will be written to the file.\n");

// 恢复原始的stdout文件描述符

if (dup2(savedStdout, STDOUT\_FILENO) == -1) {

perror("dup2");

exit(1);

}

// 关闭文件描述符

close(fileDescriptor);

return 0;

    if(pipe(p) == -1){

      perror("pipe");

      exit(1);

    }

    //p[0]为管道的读取端，p[1]为管道的写入端

    pid\_t pid = fork();

    int savein = STDIN\_FILENO;

    int saveout = STDOUT\_FILENO;

    if(pid == -1){

      perror("fork");

      exit(1);

    }

    if(pid == 0){

      //子进程

      close(p[0]);//关闭读取端

      if(dup2(p[1], STDOUT\_FILENO) == -1){

        perror("dup2");

      }

      runcmd(pcmd -> left);

      close(p[1]);//关闭写入端

    }

    else if(pid == 1){

      //父进程

      wait(NULL);//等待子进程结束

      close(p[1]);//关闭写入端

      if(dup2(p[0], STDIN\_FILENO) == -1){

        perror("dup2");

        exit(1);

      }

      runcmd(pcmd -> right);

      close(p[0]);//关闭读取端

    }

    if(dup2(STDOUT\_FILENO, saveout) == -1){

      perror("dup2");

      exit(1);

    }

    if(dup2(STDIN\_FILENO, savein) == -1){

      perror("dup2");

      exit(1);

    }

}