

**操作系统课程设计报告**

**设计题目**  操作系统接口：

兼容Unix/Linux 命令接口2

**学生姓名**  陈嘉乐

**学 号**  2021218152

**专业班级**  计科21-3班

**指导教师**  田卫东

**完成日期**  2023/12/30

**合肥工业大学 计算机与信息学院**

**1. 课程设计任务、要求、目的**

**1.1课程设计任务**

1. 为 Windows 操作系统建立一个兼容 Unix/Linux 命令的命令接口；
2. 实现杂项命令包括 login,password,logout,sort,more,print,>,>>,<,<<,|等，命令的内容与详细格式请查阅 Unix/Linux 命令手册；
3. 可以字符形式接收命令，执行命令，然后显示命令执行结果；

**1.2课程设计目的和要求**

**1.2.1 设计目的**

（1）理解并掌握如何在Windows操作系统上模拟Unix/Linux命令接口。

（2）掌握基本的命令行命令和操作，如login、password、logout、sort、more、print等。

（3）理解并实现管道（pipe）和重定向（redirection）操作。

**1.2.2 设计要求**

1. 实现Linux/Unix的命令窗口，设置当前所在的磁盘目录和命令窗口字符显示的格式。
2. 实现登录命令login，登录的用户名和密码文件存储在本地磁盘上，在命令窗口接收命令，读取文件完成用户的登录。实现密码修改命令password，通过修改磁盘上的用户文件中的密码选项完成对用户密码的修改。实现登出命令logout，在用户登录后，调用登出命令在原来登录的基础上进行登出。
3. 实现查看文件内容命令more，在用户命令窗口接收more命令和它的文件名称参数，把文件里的内容按照文件内容的格式在命令行窗口中输出。
4. 实现排序命令sort，在用户命令窗口接收sort命令和它的参数，它的参数为当前所在目录的文件名称，使用相关的算法对文件内容按照一定的规则进行排序。
5. 实现字符串格式化输出命令print，按照字符串输出格式对print后面的字符串进行输出到命令行窗口中。
6. 实现输出重定向命令>、>>，结合前面所述的命令，把重定向符号左侧的输出命令的输出内容重新定向到重定向命令右侧的文件里面，其中>>命令是把输出内容添加到文件内容的末尾，>命令是把输出内容添加到文件里面并覆盖掉原来的文件内容。
7. 实现输入重定向命令<、<<，结合前面所述的命令，把重定向符合右侧的内容作为输入参数重新定向到左侧的输入命令里，其中<命令是把文件内容作为输入重定向到左侧命令中，<<命令是把键盘输入内容作为输入重定向到左侧命令中。
8. 实现管道命令|，结合前述所述从命令，把管道命令左侧命令的输出内容作为管道命令右侧命令的输入，完成相应指令的功能。

**2. 开发环境**

编程语言：C++语言

编译环境：使用Code::Blocks集成开发环境

计算机配置：Windows10操作系统，Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz 2.59 GHz，16.0GRAM。

**3. 相关原理及算法**

**3.1操作系统命令行窗口设计**

当打开命令行时，首要任务是设置当前目录路径。我将初始路径设为D盘的根目录，通过调用操作系统API SetCurrentDirectory()进行设置。同时，我将当前目录存储在path数组中，以便后续命令调用。

由于我们是在命令行窗口中输入命令，而输入的命令本质上是一个字符串。这个字符串中包含空格来分割命令和参数。在解析命令时，我们实际上是对字符串进行处理。这里，我使用gets()函数来接收输入的字符串，并使用string中的stok()函数进行字符串分割。

**3.2 Linux/Unix杂项命令设计**

**（1）login，password，logout命令设计**

这三个命令是对用户和密码进行操作，而我们所了解的是在操作系统中，账户的用户名和密码是存储在本地磁盘目录下的某个文件中。这三个命令涉及到用户和密码的操作，我们知道在操作系统中，用户名和密码通常存储在本地磁盘目录下的某个文件中。

login命令用于用户登录。系统首先提示用户输入用户名。在收到用户名后，系统会将其与存储在文件中的用户名进行比对。如果用户名匹配，系统会提示输入密码，并再次进行比对。如果用户名和密码都正确，则登录成功。此时，命令行窗口的提示符会显示用户名和主机名称，并且设置一个标志is\_login=True表示已登录。

password命令用于修改密码。首先，系统会检查用户是否已登录。如果已登录，系统会提示输入当前密码。输入后，系统会将其与存储在文件中的密码进行比对。如果密码一致，系统会提示输入新密码，并将新密码覆盖到用户文件中原有的密码。

logout命令表示用户登出。此时，我们只需将is\_login标志设置为FALSE，并在命令行窗口中去掉显示的用户名即可。

1. **sort命令设计**

sort命令为对文件内容的每一行按照asci码的顺序进行排序，当我们通过命令行窗口得到文件名时，跟当前路径进行拼接得到文件的绝对路径，然后通过文件流获取文件中每一行的内容，然后对这些内容进行排序，这里我使用冒泡排序算法，按照asci码不断比对相邻两行文件的内容，然后进行交换，数轮迭代后文件每行内容按照顺序排列。

1. **more命令设计**

more命令为查看当前目录中某个文件的内容，并把文件的内容按照在文件中的顺序进行输出到命令行窗口中，这里我使用文件流的方式依次读取文件中每一行文件内容并依次输出。

1. **print命令设计**

print命令为格式化输出字符串，print命令有两种格式，一种时直接把参数作为字符串进行输出，一种是把第二个参数按照第一个参数的格式进行输出，在这里我设计了三种格式，分别为%d，%f，%s，分别表示整型、浮点型、字符串型，把字符串按照这三种命令的格式进行输出。

1. **输出重定向命令**

输出重定向命令内容为把左侧输出命令的输出内容重定向到右侧文件中，而输出重定向命令有两种，分别为>>、>，他们分别表示把输出内容是增添到文件的末尾还是覆盖掉原来的文件内容，我设计这条命令时首先调用左侧输出命令，并设置一个标志flag=1，当flag=1时，把左侧输出命令的输出内容保存到一个数组里，而不是直接输出到终端里，然后通过文件流的方式把该数组里的内容保存到输出重定向命令右侧文件中。

1. **输入重定向命令**

输入重定向命令内容为把输入重定向命令右侧的输入设备作为左侧输入命令的输入，而输入重定向命令分为两种，分别为<、<<，其中<输入设备为文件，表示把文件内容作为输入命令的输入，而<<输入设备为键盘，表示把键盘输入内容作为输入命令的输入，这里我首先重输入设备（文件、键盘）获取输入的内容，把他保存到一个输入数组中，然后调用左侧的输入命令，把输入数组的内容作为参数。

1. **管道命令**

管道命令是一种特殊的命令，它的作用是将一个命令的输出作为另一个命令的输入。在执行管道命令时，相当于同时执行了左侧的输出命令和右侧的输入命令。首先，我执行左侧的输出命令，并将标志flag设置为1。然后，将输出命令的输出内容保存到一个数组中。接下来，我将该数组作为参数传递给右侧的输入命令，并执行该命令。

**4. 系统结构和主要的算法设计思路**

**4.1系统结构**

命令终端输入

执行命令

解析输入字符

串

login/logout

passwd

sort

more

print

<、<<、>、>>

管道命令 |

图1. 系统结构和功能

如图1所示，按照题目要求完成了课设要求实现的每个功能模块。

在从命令行终端输入字符串时，对输入的字符串进行解析，解析后成为各个命令，并按照各个命令的功能进行执行，它们分别为：login、logout、password、sort、more、print、>>、>、<<、<；

**4.2 设计思路及操作系统命令执行流程图**

对于用户输入的命令字符串，先对它进行解析，先判断其是否为重定向命令或是管道命令，若都不包含，则判断为一般命令，根据命令调用相关函数即可实现。若为重定向命令，则判断其是输入重定向还是输出重定向命令，若为输入重定向，则获取输入设备的具体内容，再调用相关命令执行得到结果；若为输出重定向，则先执行输出命令，再将结果保存到目标文件中去。若为管道命令，则将前面命令的输出作为后面命令的输入。

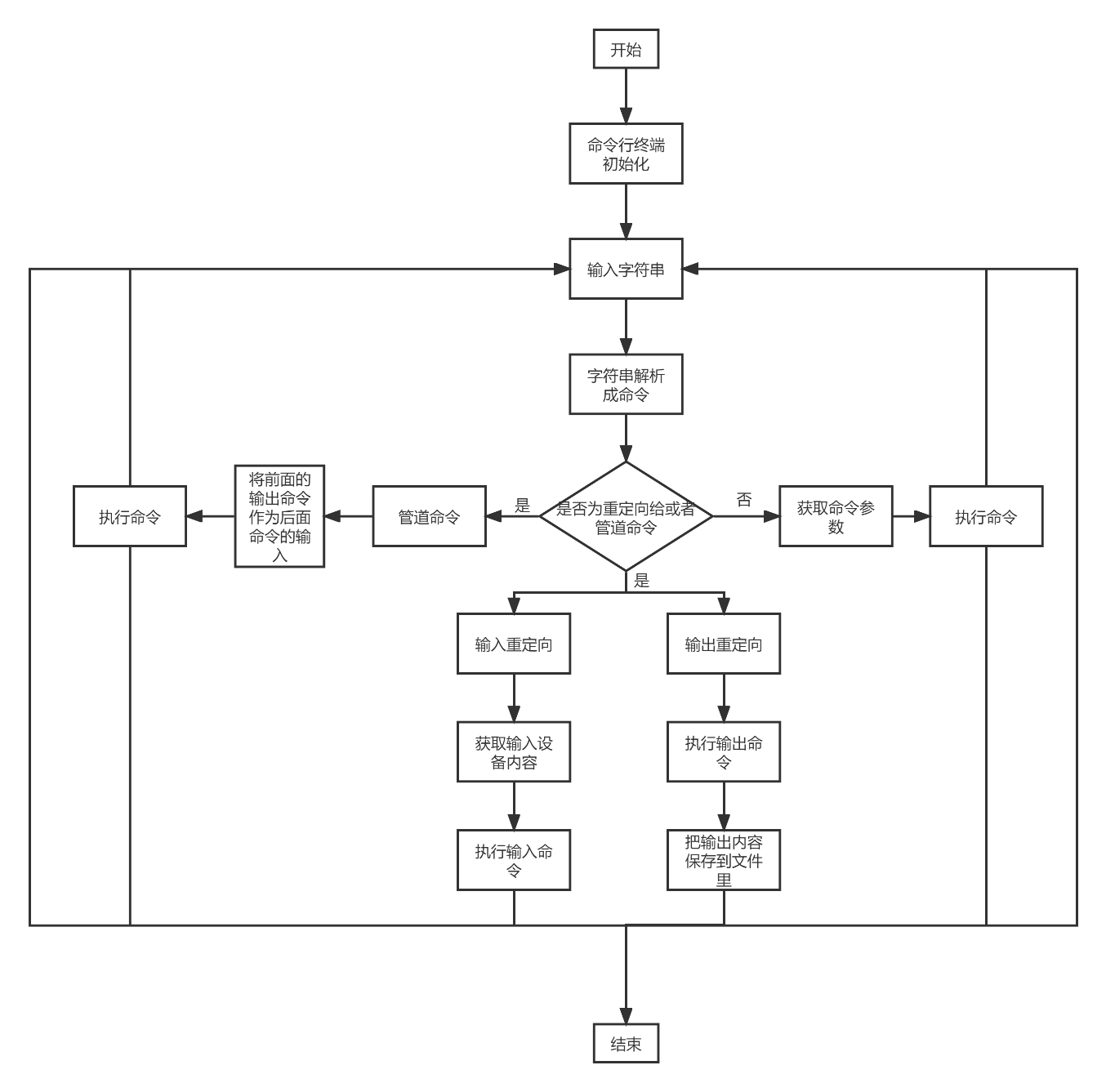


图2 操作系统命令执行流程图

**4.3主要算法思想**

**1.命令解析：**有限自动机用于识别命令的开始和结束，以及其中的特殊字符（如管道符|、重定向符<、>等）。有限自动机是一种可以识别正文的模型，它由一组状态组成，每个状态都有一个输入符号和转换函数，根据当前状态和输入符号，转换函数决定下一个状态。通过这种方式，有限自动机可以识别输入字符串中的命令和参数。

**2.词法分析：**将输入的字符串分解成一个个的词法单元或标记，例如识别出命令名、参数等。词法分析也称为词法扫描，是编译过程的第一步，将源代码分解成一系列的词素，每个词素都有特定的含义和语法规则。词法分析是编译过程中的一个重要阶段，其主要任务是将输入的字符串分解成一个个的词素或标记。这些词素具有一定的语法和语义含义，是构成程序的基本元素。在进行词法分析时，可以采用有限自动机或正则表达式等算法工具。有限自动机是一个可以识别正文的模型，它由一组状态组成，每个状态都有一个输入符号和转换函数。当有限自动机读取到输入字符串的某个字符时，会根据当前状态和该字符执行相应的转换，直到达到终止状态。正则表达式则是一种用于描述词法规则的元语言，它可以匹配字符串中的特定模式。在词法分析过程中，通常会将输入字符串按照空格、制表符等分隔符进行分割，得到一个个的单词或标记。这些单词或标记可以是关键字、标识符、常量、运算符等，它们构成了程序的基本语法单位。

**3.冒泡排序：**冒泡排序是一种简单的排序算法，它重复地走访过要排序的元素列，依次比较相邻的两个元素，如果顺序（如从大到小、首字母从Z到A）错误就把他们交换过来。

冒泡排序的基本思想是通过重复地走访过要排序的数列，比较每对相邻元素并交换位置（如果需要），直到没有再需要交换的元素为止。这个过程可以分解为以下几个步骤：

比较相邻元素：从第一个元素开始，比较相邻的两个元素。如果前一个元素大于后一个元素，则交换他们的位置。

重复走访：重复步骤1，直到没有再需要交换的元素为止。也就是说，该数列已经排序完成。

排序完成：这个算法会走访每一对相邻元素，直到没有需要交换的元素，说明该数列已经排序完成。

冒泡排序的性能分析表明，它的时间复杂度为O(n^2)，其中n为要排序元素的数量。这是因为冒泡排序需要通过多次走访和比较来排序元素，特别是当数据集较大时，效率较低。然而，冒泡排序是一种稳定的排序算法，即相等元素的相对位置在排序前后不会改变。

在实际应用中，由于冒泡排序的时间复杂度较高，对于大规模数据集的排序通常使用更高效的算法，如快速排序、归并排序等。但是，冒泡排序在某些情况下仍然是一种有用的算法，例如在处理小规模数据集或用于教学目的时。

**4.重定向：**

文件操作：使用C++的文件流来处理输入输出重定向。对于输入重定向，可以打开指定的文件并将其关联到一个输入流对象；对于输出重定向，可以打开一个文件并将其关联到一个输出流对象。

I/O重定向：通过系统调用或库函数来重定向标准输入、标准输出和标准错误输出。例如，在C语言中，可以使用 freopen() 函数来重定向文件流。

临时文件：对于更复杂的重定向需求，可能需要使用临时文件来存储中间结果。例如，可以将命令的输出写入临时文件，然后再将该文件用作另一个命令的输入。

**5. 程序实现——主要数据结构**

|  |  |  |
| --- | --- | --- |
| 变量名 | 类型 | 作用 |
| path | 数组array | 存储当前路径，以便进行切换目录 |
| username | 数组array | 用来存储当前的用户名 |
| is\_login | bool型变量 | 判断是否登录 |
| flag | int型变量 | 用来判断输出命令是否要在命令行窗口输出，在输出重定向时不输出，而是把输出内容保存到文件，其他情况则输出到命令行窗口中。 |
| passwd | 数组array | 用来存储当前的用户密码 |
| output | 数组array | 用来存储输出命令的输出内容以便进行输出重定向 |
| str | string型 | 用来存储当前输入到终端的字符串，以便进行解析成命令 |

**6. 程序实现——程序实现细节描述**

**1. 输出重定向函数**

输出重定向函数int cmd\_right(char \*data,char \*filename,int sign,char \*path)

char \*data：表示要写入文件的数据。

char \*filename：表示要写入的文件名。

int sign：表示重定向操作符，如果为0，表示>，表示覆盖写入；如果为其他值，表示>>，表示追加写入。

char \*path：表示文件的路径。

获取当前目录：GetCurrentDirectory(100,path);使用GetCurrentDirectory函数获取当前目录的路径，并将其存储在path变量中。

拼接文件路径：char\* path1=path; strcat(path1,"\\"); strcat(path1,filename);将当前目录的路径和文件名拼接起来，形成完整的文件路径。这里使用了字符串连接函数strcat。

文件写入操作：根据传入的sign参数判断是覆盖写入还是追加写入：

覆盖写入 (sign == 0)：使用ios::out模式打开文件，如果文件打开失败，输出错误信息并返回-1。然后使用<<操作符将数据写入文件，最后关闭文件流。

追加写入 (sign != 0)：使用ios::app模式打开文件，如果文件打开失败，输出错误信息并返回-1。然后使用<<操作符将数据写入文件，最后关闭文件流。

错误处理：在文件打开失败时，输出错误信息并返回-1。这可以帮助调试并指出问题所在。

**2.排序函数**

排序函数int cmd\_sort(char \*array,char output[],char \*path,int flag)

char \*array: 指向一个字符串的指针，表示要读取的文件名。

char output[]: 一个字符数组，用于存储排序后的结果。

char \*path: 指向一个字符串的指针，表示文件的路径。

int flag: 一个标志位，用于决定是否将排序结果输出到控制台。

获取当前目录：使用GetCurrentDirectory函数获取当前目录的路径，并将其存储在path变量中。

打开文件：拼接文件路径，并尝试打开文件。如果文件不存在或无法打开，函数返回-1并结束。

读取文件内容：使用ifstream对象读取文件内容，将每一行存储在line数组中。同时，记录行数lnum。

排序：使用冒泡排序算法对文件内容进行排序。排序是基于每行内容的ASCII码值进行的。

输出到文件：使用ofstream对象将排序后的结果写入到文件中。如果文件无法打开，函数返回-1并结束。

输出到控制台：如果flag为0，将排序后的结果输出到控制台。

结果存储：将排序后的结果存储在output数组中，每一行的内容后面跟着一个换行符。

关闭文件和返回：关闭文件流，并返回0表示成功。

**3.格式化输出函数**

格式化输出函数int cmd\_print(char \*p2,char \*p3,char output[],int flag)

p2: 格式化字符串，包含格式化指令，如 %d、%f 或 %s。

p3: 格式化参数，用于填充p2中的格式化指令。

output: 用于存储输出结果的字符数组。

flag: 一个标志位，用于决定是否将结果输出到控制台。

函数流程

判断条件：首先判断p3是否为空。

如果p3为空，那么：

如果flag为0，直接将p2输出到控制台。

清空output数组。

将p2复制到output数组中。

如果p3不为空，进入下一步。

格式化字符串处理：

将p2转换为字符串，并去掉其中的双引号字符。

根据字符串中的格式化指令（如 %d, %f, %s）进行相应的处理。

格式化输出：

如果字符串中包含整型格式化指令（%d），则进行整型格式化输出。如果flag为0，将结果输出到控制台；否则，将结果存储在output数组中。

如果字符串中包含浮点型格式化指令（%f），则进行浮点型格式化输出。同样，如果flag为0，将结果输出到控制台；否则，将结果存储在output数组中。

如果字符串中包含字符串型格式化指令（%s），则进行字符串型格式化输出。同样，如果flag为0，将结果输出到控制台；否则，将结果存储在output数组中。

错误处理：如果字符串中没有有效的格式化指令，函数返回-1并结束。

返回值：如果一切正常，函数返回1。

**4.列出目录中的文件和子目录**

int ls\_cmd(char \*array,char output[],int flag)

函数ls\_cmd接收三个参数：array，output和flag。

array是一个字符指针，表示命令的名称。如果这个参数是"ls"，则函数将列出当前目录下的所有文件和文件夹。

output是一个字符数组，用于存储列出的文件和文件夹的名称。

flag是一个整数，用于控制输出格式。

函数的执行过程如下：

初始化一些变量，包括文件句柄handle、文件编号filenum、文件时间lpCreationTime、lpLastAccessTime、lpLastWriteTime、系统时间stime等。

获取当前目录路径，并将其存储在path1中。如果获取失败，则返回-1。

在当前目录路径后添加通配符"."，以便列出所有文件和文件夹。

调用Windows API函数FindFirstFile来查找第一个匹配的文件或文件夹，并将结果存储在path2中。如果查找失败，则返回-1。

进入一个循环，遍历所有匹配的文件和文件夹。对于每个文件或文件夹，执行以下操作：

a. 调用Windows API函数CreateFile打开文件或文件夹，并获取其句柄。

b. 调用Windows API函数GetFileTime获取文件的最后访问时间、最后修改时间和创建时间，并将其转换为本地时间和系统时间。

c. 如果输入参数array是"ls"，则将文件或文件夹的名称添加到输出数组output中，并根据文件数量格式化输出。如果每三个文件输出一次换行符，则添加制表符。

d. 关闭文件句柄。

调用Windows API函数FindNextFile查找下一个匹配的文件或文件夹。如果查找失败，则退出循环。

调用Windows API函数FindClose关闭文件查找句柄。

打印一个换行符以结束输出。

调用Windows API函数GetCurrentDirectory获取当前目录路径，并将其存储在path1中。

返回0表示成功执行。

**7.程序运行的主要界面和实验结果截图**

**7.1 主要界面 （主目录在D盘）**

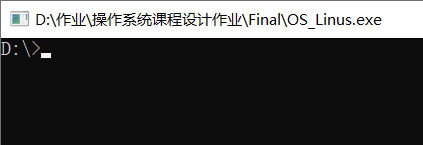


图3 主要界面

**7.2 ls命令查看当目录下所有的文件**

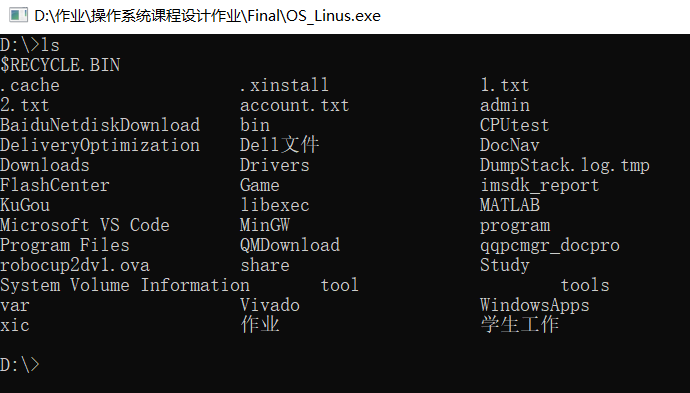


图4 ls命令

**7.3 用户登录**

用户文件是存储在D:/account.txt文件里

在这里account文件内容的第一行是用户名，第二行是密码；

表示用户名：Faye，密码：123456

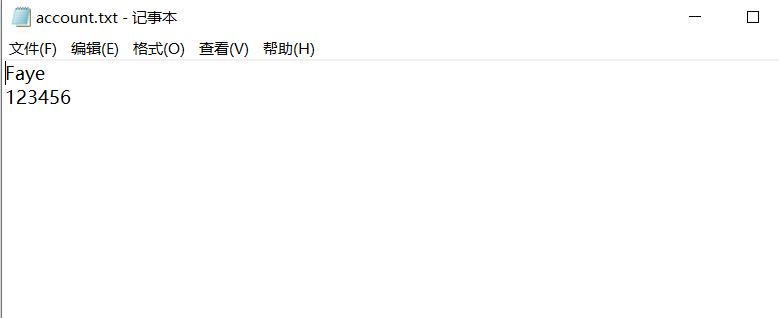


图5 account.txt文件内容

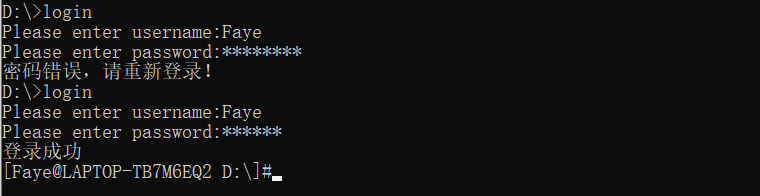


图6 用户登录界面

**7.4 修改密码指令passwd**

在没有登录的情况下修改密码提示先登录，登录成功后会提示修改密码，在这里我把密码从123456修改成了123，然后在查看account.txt里的文件内容：

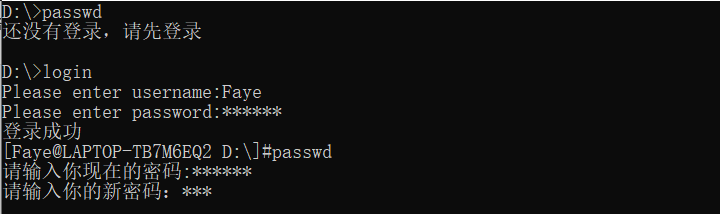


图7 用户修改密码

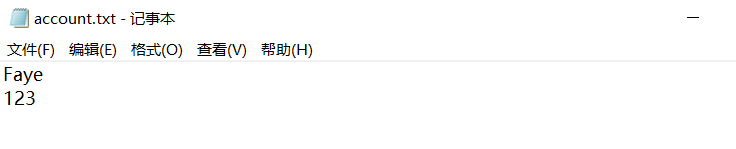


图8 修改后的account.txt文件内容

这个文件的密码选项修改成了123。

**7.5 more命令**

查看当前目录下某个文件的内容：

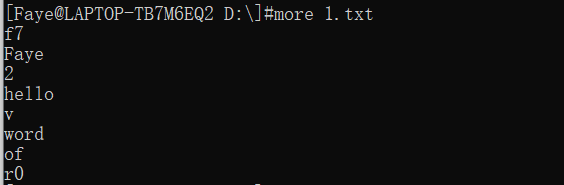
****

图9 more命令

**7.6 sort命令**

查看磁盘D盘有一个文件1.txt里的内容，然后我将这个文件里的内容进行排序，这是排序前的内容：

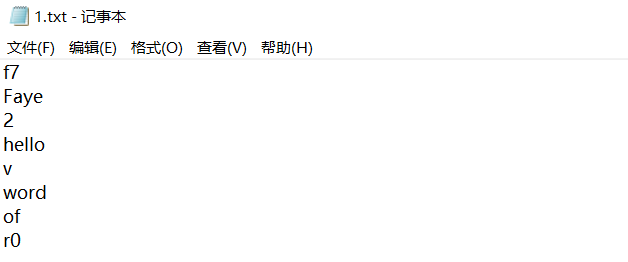


图10 1.txt文件内容

调用操作系统命令sort 1.txt：

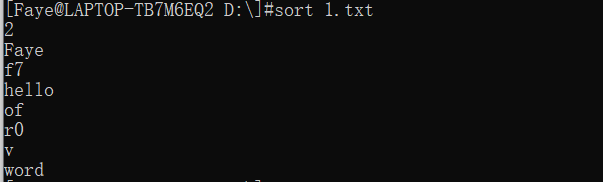


图11 sort命令

然后查看排序后的文件内容，发现文件内容被排序:

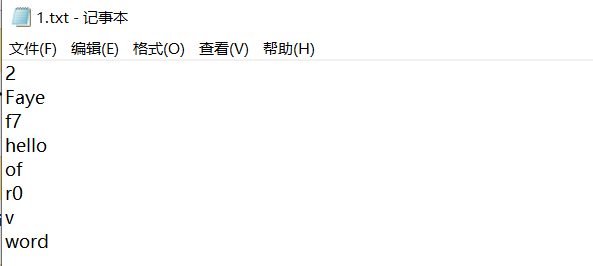


图12 执行sort命令修改后的1.txt文件内容

**7.7 print命令**

这里展示了普通输出和三种格式化输出：

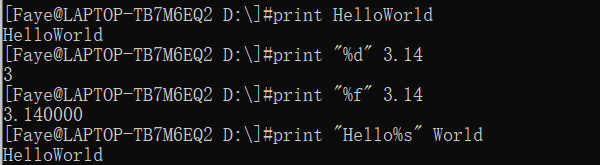


图13 print命令

**7.8 输出重定向命令>>、>**

执行测试第一种输出重定向命令>：



图14 第一种输出重定向命令

结果会覆盖目标文件原内容：



图15 2.txt中的原内容被覆盖

执行测试第二种输出重定向命令>>：



图16 第二种输出重定向命令

会在文件末尾增加追加内容：



图17 2.txt文件内容未被覆盖

从上图中可以看出ls中的输出内容被重定向到了2.txt中，执行第一种重定向命令会将原来2.txt文件中的内容覆盖，而第二种重定向命令则是在原文件内容后增加追写。

**7.9 输入重定向 <<、<**

（1）<为把文件内容作为输入重定向到指令中，这里我创建了2.txt，文件内容为1.txt；

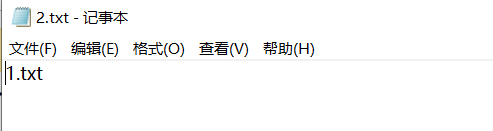


图18 2.txt文件内容

现在我调用命令对它进行重定向，是把2.txt文件内容里的1.txt，作为输入重定向到more里，相当于执行了more 1.txt这条指令。

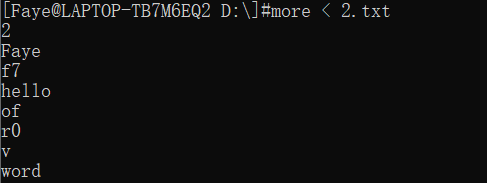


图19 第一种输入重定向命令

1. <<为从键盘输入的内容作为输入重定向到指令中，并且它的参数是作为分隔符的。

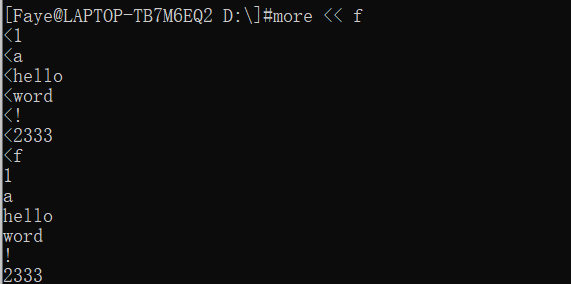


图20 第二种输入重定向命令

**7.10 管道命令 |**

管道命令为把前一个命令的输出内容作为后面一个命令的输入。这个过程中相当于执行了两条命令，print 1.txt 输出1.txt这个字符串，然后把1.txt这个输出内容作为sort命令的输入，执行sort 1.txt。

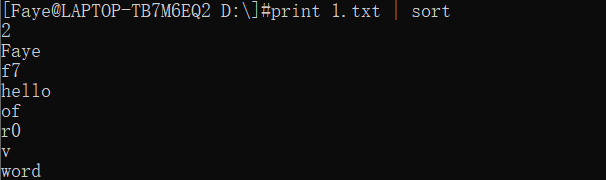


图21 管道命令

**8.总结和感想体会**

在开始编程之前，我需要深入理解每一个命令的工作原理。例如，对于sort命令，我需要明白它是如何对输入的行进行排序的。对于more命令，我需要了解它是如何逐页显示文本的。为了解决这个问题，我查阅了相关的资料和文档，阅读了Linux/Unix命令手册，并尝试在Linux终端中直接使用这些命令，观察它们的输出和行为。最终我选择了C++作为编程语言。虽然我对C++比较熟悉，但在实现一些特定功能时，我还是遇到了困难。例如，为了处理文本数据，我需要使用字符串和字符数组。在这个过程中，我遇到了一些关于字符编码和字符串操作的问题。为了解决这些问题，我查阅了C++的字符串处理函数和相关的编程书籍和资料。

我先制作了一张逻辑图，先判断用户输入的命令是否为管道命令或是重定向命令，若不是则为一般命令，直接调用相应的函数处理即可；若为重定向，则还需判断其是输出重定向还是输入重定向，再进行相应处理；若为管道命令，则将管道标识符前一个命令的结果作为后一项命令的输入。关于login、passwd、logout命令我通过读取并修改系统磁盘上的用户文件来实现这些命令，这让我对操作系统用户登录有了一个更加深入的理解；关于sort、print、more命令，它们主要为Linux/Unix系统的基本常用命令，不管是对文件的操作还是对字符串的处理，都让我学会了利用C++语言的一些功能库进行处理；最后对于重定向和管道命令，我利用了输入输出数组存储的方式对输入输出内容进行处理，使用了较为简单的方式实现了命令相应的功能。

在实现这些命令的过程中，我遇到了一些挑战。例如，为了实现sort命令，我需要了解排序算法，并实现一个稳定的排序函数。在这个过程中，我学习了快速排序、归并排序、冒泡排序等排序算法的原理，并掌握了C++中的数组和指针操作。同时，我还发现了一些我之前忽视的细节问题，比如函数参数的传递方式、数组越界等。在实现这些命令时，我遇到了许多错误和异常。为了解决这些问题，我使用了调试工具来跟踪程序的执行过程。我还编写了一些测试用例来验证我的程序的正确性。通过不断的调试和测试，我逐渐找到了解决问题的办法。

我用了一周多的时间完成了这次操作系统课设，这次课设让我学会了很多，让我更加熟练地使用C++语言进行编程。在开始编写程序前，画一个逻辑架构图非常有利于之后的编写过程，也容易调式发现问题。在遇到问题时让我学会了坚持，一步一步通过实现各个命令小部分来完成总体的实现，尽管有些命令实现起来有些困难，但我学会了由难到易的过程，把一个较为复杂的问题转换为一个个小的问题，并把他们解决掉。总的来说，实现这些命令的过程充满了挑战和收获。通过这次经历，我不仅提高了自己的编程技能，还学到了很多关于Linux系统和系统调用的知识。在未来的学习和工作中，我将继续努力克服困难，不断提高自己的技术能力和解决问题的能力。最后，我非常感觉老师提供的这次课设的机会，让我受益颇多。

**参考文献**

[1] 汤小丹等编.计算机操作系统 ，西安电子科技大学出版社.2014年5月

**附录1：程序清单(部分)**

**login/logout命令**

int check\_Login(char \*array,char\* fileNamePath,char \*username,char \* passwd)

{

char data[100];

//打开文件

ifstream infile;

infile.open(fileNamePath);

if (infile.fail()) {

return -1;

} else {

while (!infile.eof()) {

infile.getline(username,100);

infile.getline(passwd,100);

if(strstr(username,array))

{

return 1;

}

}

}

return 0;

infile.close();

}

//登出命令

void logout(int& is\_login){

//把登录的标志置为0

is\_login=0;

}

//登录命令

else if(str=="login"){

char buff[100] = { 0 };

printf("Please enter username:");

cin>>buff;

//如果用户名不准确

int result=check\_Login(buff,fileNamePath,username,passwd);

if(result==-1)

{

cout << "文件" << fileNamePath << "打开失败或者不存在！" << endl;

continue;

// return -1;

}

else if(result==0){

cout<<"用户名错误！"<<endl;

continue;

}

// wchar\_t\* G\_LOGIN\_NAME = char2wchar\_t(buff);

printf("Please enter password:");

char buff1[100] = { 0 };

char c;

int i = 0;

while (TRUE)

{

c = getch(); //从控制台读取一个字符但是不显示

if (c == '\n' || c == '\r') //windows下getch输入回车会返回\r\n

{

printf("\n");

break;

}

putchar('\*');

buff1[i++] = c;

}

if(!(strcmp(passwd,buff1)==0)){

cout<<"密码错误，请重新登录！"<<endl;

continue ;

}

cout<<"登录成功"<<endl;

// wchar\_t \* G\_LOGIN\_PWD = char2wchar\_t(buff1);

is\_login=1;

continue;

}

**passwd命令**

else if (str == "password") {

if (is\_login) {

cout << "请输入你现在的密码:";

char buff1[100] = { 0 };

char c;

int i = 0;

while (TRUE)

{

c = \_getch(); //从控制台读取一个字符但是不显示

if (c == '\n' || c == '\r') //windows下getch输入回车会返回\r\n

{

printf("\n");

break;

}

//用 \* 来掩盖当前的密码

putchar('\*');

buff1[i++] = c;

}

//当密码和用户文件的密码一致时

if (strstr(buff1, passwd)) {

cout << "请输入你的新密码：";

char buff2[100] = { 0 };

char ch;

int j = 0;

while (TRUE)

{

ch = \_getch(); //从控制台读取一个字符但是不显示

if (ch == '\n' || ch == '\r') //windows下getch输入回车会返回\r\n

{

printf("\n");

break;

}

putchar('\*');

buff2[j++] = ch;

}

char data[100];

fstream infile(fileNamePath, ios::out);

//重新把新密码写到文件里面

infile << username << endl << buff2;

infile.close();

}

else {

cout << "密码错误！" << endl;

continue;

}

}

cout << "还没有登录，请先登录" << endl;

continue;

}

**ls命令**

int ls\_cmd(char \*array,char output[],int flag)

{

char path1[100]={0};

// char path2[100]={0};

HANDLE handle;

int filenum=0;

FILETIME lpCreationTime,lpLastAccessTime,lpLastWriteTime;

SYSTEMTIME stime;

//调用windows接口

WIN32\_FIND\_DATA\* path2 = new WIN32\_FIND\_DATA();

if(!GetCurrentDirectory(50,path1))

{

// cout<<"路径不存在!"<<endl;

return -1;

}

strcat(path1,"\\\*.\*"); //DOS通配符

if((handle = FindFirstFile(path1, path2)) != INVALID\_HANDLE\_VALUE)

{

int j=0;

for(int y=0;y<1000;y++){

output[y]=0;

}

do{

//调用操作系统句柄

HANDLE hDir = CreateFile(path2->cFileName,GENERIC\_READ,FILE\_SHARE\_READ | FILE\_SHARE\_DELETE,NULL,OPEN\_EXISTING,FILE\_FLAG\_BACKUP\_SEMANTICS, NULL);

if(GetFileTime(hDir, &lpCreationTime, &lpLastAccessTime, &lpLastWriteTime)){

FILETIME ftime;

FileTimeToLocalFileTime(&lpLastWriteTime, &ftime); // 转换成本地时间

FileTimeToSystemTime(&ftime, &stime); // 转换成系统时间格式

}

//cout<<setfill('0');

if(strcmp(array,"ls")==0)

{

for(int p=0;path2->cFileName[p]!=0;p++){

output[j]=path2->cFileName[p];

j++;

}

//设置格式化标志来控制输出格式

cout.setf(ios::left);

//每隔三个文件进行一次输出

if(filenum%3!=0){

if(flag==0){

cout<<setw(20)<<path2->cFileName<<" ";

}

output[j]='\t';

j++;

}

else{

if(flag==0){

cout<<setw(20)<<path2->cFileName<<endl;

}

output[j]='\n';

j++;

}

filenum++;

}

else

{

cout<<stime.wYear<<"-"<<setw(2)<<stime.wMonth<<"-"<<setw(2)<<stime.wDay<<" "<<setfill('0')<<setw(2)<<stime.wHour<<":"<<setw(2)<<stime.wMinute;

cout<<" "<<path2->cFileName<<endl;

}

CloseHandle(hDir);

}while (FindNextFile(handle, path2));

}

FindClose(handle);

cout<<endl;

//设置当前路径

GetCurrentDirectory(100,path1);

}

**more命令**

char cmd\_more(char \*array,char output[],char \* path,int flag)

{

char ch[1000];

//char tmp[1000];

//cout << array<<endl;

GetCurrentDirectory(100,path);

//对路径进行拼接

strcat(path,"\\");

strcat(path,array);

//cout << path<<endl;

ifstream infile(path);

if(!infile)

{

cout<<"无法打开文件"<<path<<endl;

GetCurrentDirectory(100,path);

}

else

{

int count1=0,count2=0,l;//count1用于显示文件总行数，count2用于显示当前读到的文件行数

while(!infile.eof())

{

infile.getline(ch,1000,'\n');

count1++;

}

infile.clear();//不用clear，之前的文件状态属性不会变化

infile.seekg(0,ios::beg);//把流指针重新定位到文件头

int j=0;

//在每次输出前，把输出内容数组置为零

for(int y=0;y<1000;y++){

output[y]=0;

}

if(count1<24)

{

for(l=0;l<count1;l++)

{

//一边输出，一边把输出内容保存到数组

infile.getline(ch,1000,'\n');

if(flag==0){

cout<<ch<<endl;

}

for(int x=0;x<1000;x++){

output[x]=0;

}

for(int p=0;ch[p]!=0;p++){

output[j]=ch[p];

j++;

}

//output[j]='\0';

//j++;

//cout <<output;

//strcpy(output, ch);

//return output;

}

output[j]='\0';

j++;

//cout <<output;

//获取当前的路径到Path数组里

//GetCurrentDirectory(100,path);

//tmp=output;

//return tmp;

}

else

{

for(l=0;l<24;l++)

{

infile.getline(ch,1000,'\n');

cout<<ch<<endl;

count2++;

}

cout<<"--More--"<<int((double)count2/count1\*100)<<"%";

while(!infile.eof())

{

if(cin.get()=='\n')

{

infile.getline(ch,1000,'\n');

cout<<ch<<endl;

count2++;

cout<<"--More--"<<int((double)count2/count1\*100)<<"%";

}

else if(getchar()==32)//输入两次空格，一次回车

{

if(cin.get()=='\n')//不加此判断条件，会把回车符解释成逐行显示文件的命令

{

if((count1-count2)>0&&(count1-count2)<24)

{

for(l=0;l<(count1-count2);l++)

{

infile.getline(ch,1000,'\n');

cout<<ch<<endl;

count2++;

}

}

else if((count1-count2)>24)

{

for(l=0;l<24;l++)

{

infile.getline(ch,1000,'\n');

cout<<ch<<endl;

count2++;

}

}

cout<<"--More--"<<int((double)count2/count1\*100)<<"%"<<'\n';

}

}

if(int((double)count2/count1\*100)==100)

{

cout<<endl;

GetCurrentDirectory(100,path);

break;

}

}

}

}

infile.close();

}

**sort命令**

int cmd\_sort(char \*array,char output[],char \*path,int flag)

{

int lnum=0,min;

char line[100][100],tmp[100];//行数、每行字符数

string s[100];

GetCurrentDirectory(100,path);

strcat(path,"\\");

strcat(path,array);

ifstream infile(path);

//判断文件是否存在

if(!infile)

{

// cout<<"无法打开文件"<<array<<endl;

return -1;

GetCurrentDirectory(100,path);

}

else

{

while(!infile.eof())

{

infile.getline(line[lnum],100,'\n');

lnum++;//记录总行数

}

infile.close();

//按照ascii码从小到大比较每行字符串

for(int x=0;x<lnum-1;x++)

{

min=x;

for(int y=x+1;y<lnum;y++)

if(strcmp(line[y],line[min])<0)

min=y;

//min改变，就将两行字符进行交换

{

if(min!=x)

for(int a=0;a<100;a++)

{

tmp[a]=line[x][a];

line[x][a]=line[min][a];

line[min][a]=tmp[a];

}

}

}

ofstream outfile(path,ios::out);

if(!outfile)

{

// cout<<"文件打开失败"<<endl;

return -1;

}

else

{

// line[0][0]=NULL;

int j=0;

for(int y=0;y<1000;y++){

output[y]=0;

}

//把结果写到文件里面

for(int c=0;c<lnum;c++){

outfile<<line[c]<<endl;

if(flag==0){

cout<<line[c]<<endl;

}

for(int h=0;line[c][h]!=0;h++){

output[j]=line[c][h];

j++;

}

output[j]='\n';

j++;

}

outfile.close();

}

GetCurrentDirectory(100,path);

}

}

**8.9 print命令**

int cmd\_print(char \*p2,char \*p3,char output[],int flag){

if(p3==NULL){

if(flag==0){

//没有格式化字符串，直接打印P2

cout<<p2<<endl;

}

//把输出内容数组置为0

for(int i=0;i<1000;i++){

output[i]=0;

}

for(int i=0;p2[i]!=0;i++){

output[i]=p2[i];

}

}

else{

//格式化输出

string str1=p2;

char str2[100]={0};

int j=0;

for(int i=0;i<str1.length();i++){

//去掉格式化输出的 " 字符

if(str1[i]!='\"'){

str2[j]=str1[i];

j++;

}

}

for(int i=0;i<1000;i++){

output[i]=0;

}

p2=str2;

//整型格式化输出

if(strstr(p2,"%d")){

if(flag==0){

printf(p2,atoi(p3));

cout<<endl;

}

sprintf(output,p2,atof(p3));

}

//浮点型格式化输出

else if(strstr(p2,"%f")){

if(flag==0){

printf(p2,atof(p3));

cout<<endl;

}

sprintf(output,p2,atof(p3));

}

//字符串型格式化输出

else if(strstr(p2,"%s")){

if(flag==0){

printf(p2,p3);

cout<<endl;

}

//并同时那格式化输出到字符输出中用于重定向

sprintf(output,p2,atof(p3));

}

else{

// cout<<"指令格式错误，请重新输入！"<<endl;

return -1;

}

}

return 1;

}

**重定向命令**

int cmd\_right(char \*data,char \*filename,int sign,char \*path){

GetCurrentDirectory(100,path);

char\* path1=path;

strcat(path1,"\\");

strcat(path1,filename);

//cout <<path1<<endl;

//cout << data<<endl;

//string d=data;

if(sign==0){ //表示重定向是>

//文件输出为覆盖模式

fstream fst(path1,ios::out);

if(!fst)

{

cout<<"文件打开失败"<<endl;

return -1;

}

fst<<data;

fst.close();

}else{ //表示重定向是>>

//文件输出为末尾增加模式

fstream fst(path1,ios::app);

if(!fst)

{

// cout<<"文件打开失败"<<endl;

return -1;

}

fst<<data;

fst.close();

}

}

**管道命令**

if(strstr(str1,"|")){

//用空格分开输入的字符串

const char \*d = " ";

string p1;

char\* p2;

char\* p3;

char\* p4;

//并把分隔的字符串用P1、P2、P3、P4表示

p1 = strtok(str1,d);

p2 = strtok(NULL,d);

p3 = strtok(NULL,d);

p4 = strtok(NULL,d);

string p\_1=p1;

string p\_2;

string p\_4;

string p\_3;

if(p3!=NULL){

p\_3=p3;

}

if(p2!=NULL){

p\_2=p2;

}

if(p4!=NULL){

p\_4=p4;

}

if(p\_2=="|"){

// 当P2为"|"时 ,判断是否为ls命令

if(p\_1=="ls"){

flag=1;

ls\_cmd(str1,output,flag);

}

else{

cout<<"命令错误!"<<endl;

}

}

//当P3为"|"时,判断是否为"sort,print,more"命令

if(p\_3=="|"){

//收集前项命令的输出

if(p\_1=="sort"){

flag=1;

cmd\_sort(p2,output,path,flag);

}

else if(p\_1=="print"){

flag=1;

cmd\_print(p2,NULL,output,flag);

}

else if(p\_1=="more"){

flag=1;

cmd\_more(p2,output,path,flag);

}else{

cout<<"指令不支持！"<<endl;

continue;

}

//转换为后项命令的输入

if(p\_4=="sort"){

flag=0;

cmd\_sort(output,output,path,flag);

}

else if(p\_4=="print"){

flag=0;

cmd\_print(output,NULL,output,flag);

}

else if(p\_4=="more"){

flag=0;

cmd\_more(output,output,path,flag);

}else{

cout<<"指令不支持！"<<endl;

continue;

}

}else{

cout<<"指令错误！"<<endl;

continue;

}

}