**实验四 Shell管道和重定向功能的实现**

**一、实验目的**

* 掌握Shell管道功能与重定向功能的内容及其实现原理；
* 熟悉Linux系统中的各类命令；

掌握Linux所提供的系统调用使用方法，能够通过C语言编程使用系统调用实现一定功能。

**二、实验要求**

使用**fork(), exec(), dup2(), pipe() ，open()，wait()**等系统调用编写C语言程序完成与下列shell命令等价的功能。

|  |
| --- |
| **grep -v usr < /etc/passwd | wc –l > r.txt; cat r.txt** |

（提示：为简化编程，不需要用**strtok**断词，直接用**execlp**实现能达到shell命令相同功能的程序即可）

例如：**execlp("grep", "grep", "-v", "usr", 0);**

**三、实验环境**

* Ubuntu 20.04.2 LTS(GNU/Linux 5.4.0-58-generic x86\_64)
* GNU bash, version 5.0.17(1)-release (x86\_64-pc-linux-gnu)
* xshell

**四、实验过程**

**1.shell命令分析**

本实验是为了完成与一个shell命令等价的功能，因此第一步需要分析该shell命令完成的功能：

* **grep -v：**该命令结合输入重定向**< /etc/passwd**，以**/etc/passwd**文件作为输入，由**-v**选项，从中筛选出不包含有usr的行，并将它们输出到管道
* **wc：**由grep输出到管道的结果，利用**wc -l**指令计算行数，使用输出重定向 **> r.txt**将结果输出到**r.txt**文件中，以;显示结束该命令
* **cat：**连接文件r.txt并打印到标准输出设备上。

**2.系统调度函数说明**

* **fork()**

fork系统调用用于创建一个新进程，称为子进程，它与进程（称为系统调用fork的进程）同时运行，此进程称为父进程。创建新的子进程后，两个进程将执行fork()系统调用之后的下一条指令。子进程使用相同的pc（程序计数器），相同的CPU寄存器，在父进程中使用的相同打开文件。

它不需要参数并返回一个整数值。下面是fork()返回的不同值。

* 负值：创建子进程失败。
* 零：返回到新创建的子进程。
* 正值：返回父进程或调用者。该值包含新创建的子进程的进程ID
* **exec()**

exec函数族提供了一个在进程中启动另一个程序执行的方法。它可以根

据指定的文件名或目录名找到可执行文件，并用它来取代原调用进程的数据段、代码段和堆栈段，在执行完之后，原调用进程的内容除了进程号外，其他全部被新的进程替换了。另外．这里的可执行文件既可以是二进制文件，也可以是Linux下任何可执行的脚本文件。

* **dup2()**

复制文件描述符

* **pipe()**

pipe函数可用于创建一个管道，以实现进程间的通信, pipe函数定义中的

fd参数是一个大小为2的一个数组类型的指针。该函数成功时返回0，并将一对打开的文件描述符值填入fd参数指向的数组。失败时返回 -1并设置errno。

通过pipe函数创建的这两个文件描述符 fd[0] 和 fd[1] 分别构成管道的两端，往 fd[1] 写入的数据可以从 fd[0] 读出。并且 fd[1] 一端只能进行写操作，fd[0] 一端只能进行读操作，不能反过来使用。要实现双向数据传输，可以使用两个管道。

* **open()**

open函数用来打开或创建一个文件，若成功返回文件描述符，否则返回-1。

* **wait()**

阻塞并等待子进程退出，回收子进程残留资源，获取子进程结束状态。

**3.数据结构定义**

* **int sv**

wait函数的参数，接收进程结束状态

* **int inFile，outFile**

用于存储open函数返回的文件描述符，分别代表需要输入重定向的文件和

需要输出重定向的文件

* **int fd[2]**

用于作为pipe函数的参数，建立管道两端

* **char \*inFileName，\*outFileName**

输入和输出重定向文件名称

**4.程序设计**

**4.1 使用open函数打开输入和输出文件**

|  |
| --- |
| inFileName = "/etc/passwd";  inFile = open(inFileName, O\_RDONLY);//O\_RDONLY  outFileName = "r.txt";  outFile = open(outFileName, O\_RDWR | O\_CREAT,S\_IRUSR | S\_IWUSR | S\_IRGRP | S\_IROTH); |

**4.2 建立管道**

|  |
| --- |
| pipe(fd);//fd[1]为管道写入端，fd[2]为管道输出端 |

**4.3 执行grep**

使用fork()函数创建子进程，在该子进程中执行grep命令

|  |
| --- |
| dup2(inFile, 0);  dup2(fd[1], 1);  close(fd[0]);  execlp("grep", "grep", "-v", "usr", NULL); |

**4.4 执行wc**

使用fork()函数创建子进程，在该子进程中执行wc命令

|  |
| --- |
| dup2(fd[0], 0);  dup2(outFile, 1);  close(fd[1]);  execlp("wc", "wc", "-l", NULL); |

**4.5 释放资源**

使用close()函数关闭管道，使用wait()函数等待并销毁僵尸进程

**4.6 执行cat**

使用fork()函数创建子进程，在该子进程中执行cat命令

|  |
| --- |
| execlp("cat", "cat", outFileName, NULL); |

**五、实验运行结果**

* 原命令输出结果：



* 实验四程序输出结果：



结果一致

**六、实验心得**

本次实验,使我初步掌握了shell脚本的编写方法，通过对于task4.c的编写与运行，加深了我对于shell管道和重定向的理解，在本次实验中我也遇到了一些问题。主要有以下一个问题：

* fork()函数执行问题

fork()函数创建子进程的过程一直让我有疑惑，由于对进程创建过程的

不熟悉，在一开始做实验时一直不清楚fork()函数该怎么用，后来在仔细了解了fork()函数后，才熟练的使用起来。

本次实验也让我复习了之前学过的知识，对于管道、重定向的实现本质以及如何使用C语言调用Linux当中的系统调用函数来实现管道与重定向的功能。充分熟悉了Linux当中所提供的fork()、execlp()等系统调用函数。

**附 录**

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <unistd.h>  #include <errno.h>  #include <fcntl.h>  #include <sys/wait.h>  #include <sys/types.h>  #include <sys/stat.h>  #include <fcntl.h>  #include <string.h>  int main(void)  {  int sv, fd[2], inFile, outFile;  char \*inFileName, \*outFileName;  pipe(fd);  inFileName = "/etc/passwd";  inFile = open(inFileName, O\_RDONLY);*//O\_RDONLY*  outFileName = "r.txt";  outFile = open(outFileName, O\_RDWR | O\_CREAT,  S\_IRUSR | S\_IWUSR | S\_IRGRP | S\_IROTH);  if (fork() == 0)  {  dup2(inFile, 0);  dup2(fd[1], 1);  close(fd[0]);  execlp("grep", "grep", "-v", "usr", NULL);  }  if (fork() == 0)  {  dup2(fd[0], 0);  dup2(outFile, 1);  close(fd[1]);  execlp("wc", "wc", "-l", NULL);  }  close(fd[1]);  close(fd[0]);  wait(&sv);  wait(&sv);  if (fork() == 0)  {  execlp("cat", "cat", outFileName, NULL);  }  wait(&sv);  return 0;  } |