

**Linux开发环境及应用**



**Shell管道和重定向功能的实现**

**姓 名 鄭毓恒**

**学 院 计算机学院**

**专 业 计算机科学与技术**

**班 级 2020211302**

**学 号 2020211262**

**任课教师 张雪松**

**2023年 5 月**

**作业要求**

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);

**程序设计思路**

在编程之前，先分析要实现的shell命令。首先，grep -v usr < /etc/passwd 部分进行输入重定向，以/etc/passwd作为输入文件，再根据-v选项，从输入文件中选出不包含usr的行，输出至管道。

然后，wc -l > r.txt部分计算指令的上一部分输出到管道的结果的行数，使用输出重定向，将结果输出至r.txt文件。最后cat r.txt命令将r.txt文件打印到标准输出设备上。

接着，分析将要使用的系统调用函数的功能。

|  |  |
| --- | --- |
| 函数名称 | 功能 |
| fork() | 创建新进程。  原来的进程称为“父进程”，新创建进程称为“子进程”。  它不需要参数并返回一个整数值。当返回负值时，表示创建进程失败。创建成功时，返回0到子进程，返回正值，也就是子进程的PID到父进程。 |
| execlp() | 用一个指定的程序文件，重新初始化一个进程。  可指定新的命令行参数和环境参数。  不创建新进程，只是将当前进程重新初始化了指令段和用户数据段，堆栈段以及CPU的PC指针。 |
| dup2() | 复制文件描述符。 |
| pipe() | 创建一个管道。管道两端分别为文件描述符，分别为读端和写端。 |
| open() | 打开文件。  成功则返回文件描述符，否则返回-1。 |
| close() | 关闭文件或管道。 |
| wait() | 等待进程的子进程终止。  如果有子进程终止，则立即返回。 |

定义了两个宏，表示要使用的输入和输出文件的路径。

|  |
| --- |
| #define INPUT\_FILE "/etc/passwd"  #define OUTPUT\_FILE "r.txt" |

程序首先创建管道。

|  |
| --- |
| int fd[2];  pipe(fd); |

创建子进程执行grep -v usr < /etc/passwd 部分。

|  |
| --- |
| if (fork() == 0)  {  int inputFd = open(INPUT\_FILE, O\_RDONLY);  //如果文件开启成功，重定向输入，更改进程输入端口为/etc/passwd  if (inputFd != -1)  {  dup2(inputFd, 0);  close(inputFd);  }  //更改进程输出端口为管道写  dup2(fd[1], 1);  close(fd[0]);  execlp("grep", "grep", "-v", "usr", NULL);  } |

创建子进程执行wc -l > r.txt。

|  |
| --- |
| if (fork() == 0)  {  int outputFd = open(OUTPUT\_FILE, O\_RDWR | O\_CREAT, S\_IRUSR | S\_IWUSR | S\_IRGRP | S\_IROTH);  //如果文件开启成功，重定向输出，更改进程输出端口为r.txt  if (outputFd != -1)  {  dup2(outputFd, 1);  close(outputFd);  }  //更改进程输入端口为管道读  dup2(fd[0], 0);  close(fd[1]);  execlp("wc", "wc", "-l", NULL);  } |

关闭管道和等待两个子进程终止。

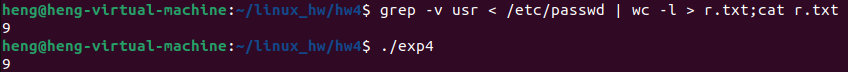
|  |
| --- |
| close(fd[1]);  close(fd[0]);  wait(&sv);  wait(&sv); |

执行cat r.txt。

|  |
| --- |
| execlp("cat", "cat", OUTPUT\_FILE, NULL); |

**运行结果**

直接执行shell命令和执行实验程序的结果一致。



**实验心得**

本次实验，通过分析shell命令和系统调用函数，设计并编写C语言程序，实现了shell命令。实验加深了我对于shell管道和重定向的理解，并且能够更加熟练运用系统调用函数，了解它们的原理和作用。在本次实验中我也遇到了一些问题，主要由于对管道的了解不够，在理解进程之间利用管道进行数据的输入输出花费了较长时间。

**源代码**

|  |
| --- |
| #include <stdio.h>  #include <unistd.h>  #include <errno.h>  #include <sys/wait.h>  #include <fcntl.h>  #include <string.h>  #define INPUT\_FILE "/etc/passwd"  #define OUTPUT\_FILE "r.txt"  int main(void)  {  int sv;  //创建管道  int fd[2];  pipe(fd);  if (fork() == 0)  {  int inputFd = open(INPUT\_FILE, O\_RDONLY);  //如果文件开启成功，重定向输入，更改进程输入端口为/etc/passwd  if (inputFd != -1)  {  dup2(inputFd, 0);  close(inputFd);  }  //更改进程输出端口为管道写  dup2(fd[1], 1);  close(fd[0]);  execlp("grep", "grep", "-v", "usr", NULL);  }  if (fork() == 0)  {  int outputFd = open(OUTPUT\_FILE, O\_RDWR | O\_CREAT, S\_IRUSR | S\_IWUSR | S\_IRGRP | S\_IROTH);  //如果文件开启成功，重定向输出，更改进程输出端口为r.txt  if (outputFd != -1)  {  dup2(outputFd, 1);  close(outputFd);  }  //更改进程输入端口为管道读  dup2(fd[0], 0);  close(fd[1]);  execlp("wc", "wc", "-l", NULL);  }  close(fd[1]);  close(fd[0]);  wait(&sv);  wait(&sv);  execlp("cat", "cat", OUTPUT\_FILE, NULL);  return 0;  } |