

《操作系统》lab1

|  |  |
| --- | --- |
| 学 院： | 计算机与信息技术学院 |
| 专 业： | 计算机科学与技术 |
| 班 级： | 计科1602 |
| 姓 名： | 麻锦涛 |
| 学 号： | 16281262 |

**日期：2019.4.12**

1. **实验目的**

（1）充分了解操作系统shell和系统调用

（2）实践创建过程并协调父进程和子进程的运行。

1. **前期准备**

（1）安装下载VMware 14.1.1以及Ubuntu18.04 64位操作系统的镜像文件

（2）虚拟机硬件配置

（3）Ubuntu镜像安装之后选择重启，这里花费时间较久

（4）Ubuntu镜像安装，前四步参考了知乎专栏《手把手教你安装Linux虚拟机

》：<https://zhuanlan.zhihu.com/p/41940739>，一路顺利，没有任何问题。

（5）Ubuntu18安装gcc，在这里我遇见了一些问题，所以就专门截图了。

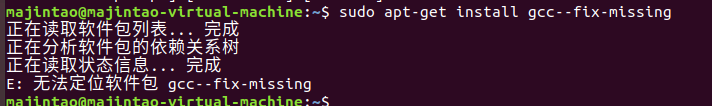
打开控制台终端，依次输入指令：

sudo add-apt-repository ppa:unbutu-toolchain-r/test

sudo apt-get update

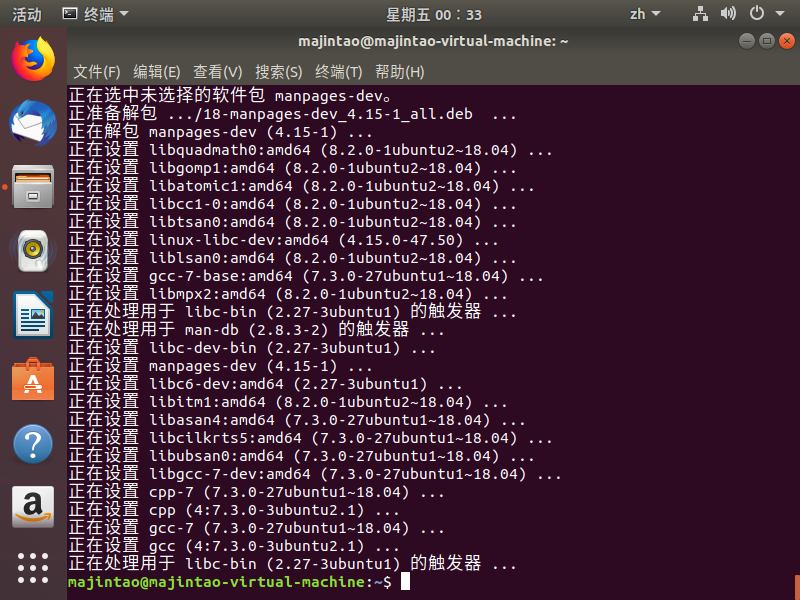
sudo apt-get install gcc有

出现问题：有几个软件包无法下载，要不运行apt-get update或者加上--fix missing

这里我的解决方法是输入：sudo apt-get install gcc --fix-missing 

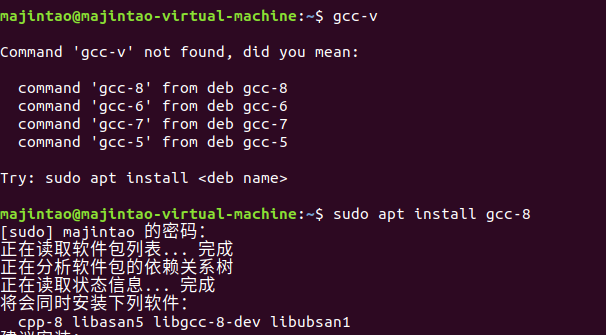
出现问题：无法定位软件包gcc --fix-missing，这里我参考了网站“Ubuntu出现“E: 无法定位软件包问题”解决方法”：<https://jingyan.baidu.com/article/f0e83a254e791622e59101aa.html>

输入：sudo apt-get update 解决问题



默认获取最新版本的gcc。安装之后，gcc还不是默认编译器，再次输入如下指令：

gcc –v



显示版本号，同时会将gcc设置成默认编译器。

1. **实验要求**

**第一部分 - 创建子进程**

第一个任务是修改上面代码中的main（）函数，以便子进程分叉并执行用户指定的命令。这将需要解析用户输入的单独令牌并将令牌存储在字符串数组中（上面代码中的args）。例如，如果用户在osh>提示符下输入命令ps -ael，则存储在args数组中的值为：

args [0] =“ps”args [1] =“ - ael”args [2] = NULL

这个args数组将传递给execvp（）函数，该函数具有以下原型：execvp（char \* command，char \* params []）;

这里，command表示要执行的命令，params将参数存储到该命令。对于这个项目，execvp（）函数应该作为execvp（args [0]，args）调用。请务必检查用户是否包含＆以确定父进程是否要等待子进程退出。

**第二部分 - 创建历史特征**

下一个任务是修改shell接口程序，以便它提供允许用户访问最近输入的命令的历史记录功能。用户将能够使用该功能访问最多10个命令。命令将从1开始连续编号，编号将继续超过10.例如，如果用户输入了35个命令，则最近的10个命令将编号为26到35。为

用户可以通过输入命令列出命令历史记录

在osh>提示的历史。例如，假设历史记录由命令组成（从最近到最近）：

ps，ls -l，top，cal，who，date

命令历史记录将输出：

6 ps

5 ls -l

4 top

3 cal

2 who

1 date

您的程序应该支持两种从命令历史记录中检索命令的技术：1。当用户输入!! ，行历史记录中的最新命令。

2.当用户输入单个！后跟一个整数N，执行历史记录中的第N个命令。

继续我们上面的例子，如果用户输入!! ，将执行ps命令;如果用户输入！3，将执行命令cal。以这种方式执行的任何命令都应该在用户的屏幕上回显。该命令也应作为下一个命令放在历史缓冲区中。

该程序还应该管理基本的错误处理。如果历史记录中没有命令，请输入!!应该会显示一条消息“历史记录中没有命令。 “如果没有与单个输入的数字相对应的命令！ ，程序应该输出“历史上没有这样的命令”。

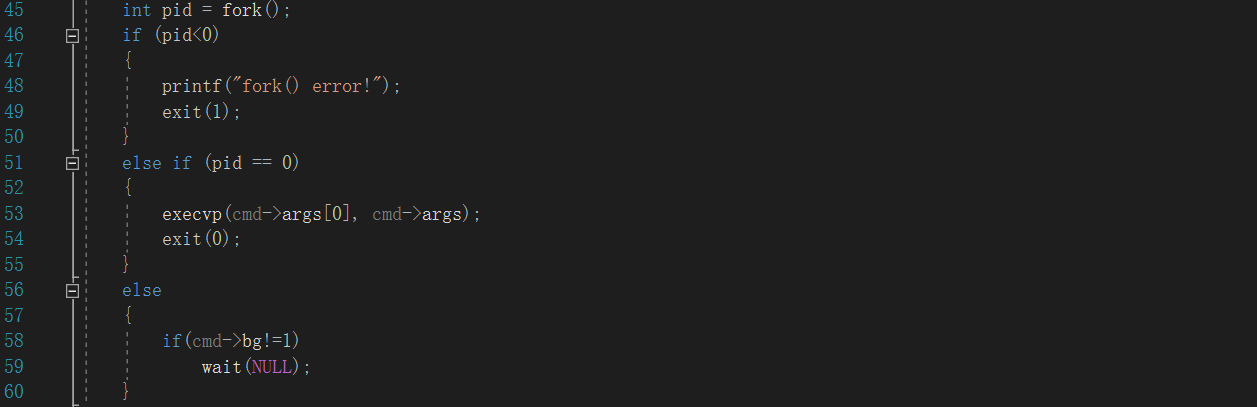
1. **运行截图**
2. **创建子进程：**

fork()函数是Unix中派生新进程的唯一方法

我们需要理解的是，调用一次fork()方法，该方法会返回两次。一次是在调用进程(也就是派生出的子进程的父进程)中返回一次，返回值是新派生的进程的进程ID。一次是在子进程中返回，返回值是0，代表当前进程为子进程。如果返回值为-1的话，则代表在派生新进程的过程中出错。

那么在程序中，我们就可以根据此返回值来判断当前进程是父进程还是子进程，来实现一些具体的操作。

C:\Users\dell\AppData\Local\Temp\1555202435(1).png



调用 fork 时，系统将创建一个与当前进程相同的新进程。通常将原有的进程称为父进程，把新创建的进程称为子进程。子进程是父进程的一个拷贝，子进程获得同父进程相同的数据，但是同父进程使用不同的数据段和堆栈段。子进程从父进程继承大多数的属性，但是也修改一些属性，下表对比了父子进程间的属性差异：

|  |  |
| --- | --- |
| 继承属性 | 差异 |
| uid,gid,euid,egid | 进程 ID |
| 进程组 ID | 父进程 ID |
| SESSION ID | 子进程运行时间记录 |
| 所打开文件及文件的偏移量 | 父进程对文件的锁定 |
| 控制终端 |  |
| 设置用户 ID 和 设置组 ID 标记位 |  |
| 根目录与当前目录 |  |
| 文件默认创建的权限掩码 |  |
| 可访问的内存区段 |  |
| 环境变量及其它资源分配 |  |

**fork 函数的特点是 "调用一次，返回两次"：在父进程中调用一次，在父进程和子进程中各返回一次。**

**（2）创建历史特征**

下一个任务是修改shell接口程序，以便它提供允许用户访问最近输入的命令的历史记录功能。用户将能够使用该功能访问最多10个命令。命令将从1开始连续编号，编号将继续超过10.通过输入命令列出命令历史记录

在osh>提示的历史。

例如，历史记录由命令组成有下面几条ps，ls -l，top，cal，who，date

date

C:\Users\dell\AppData\Local\Temp\1555199841(1).png

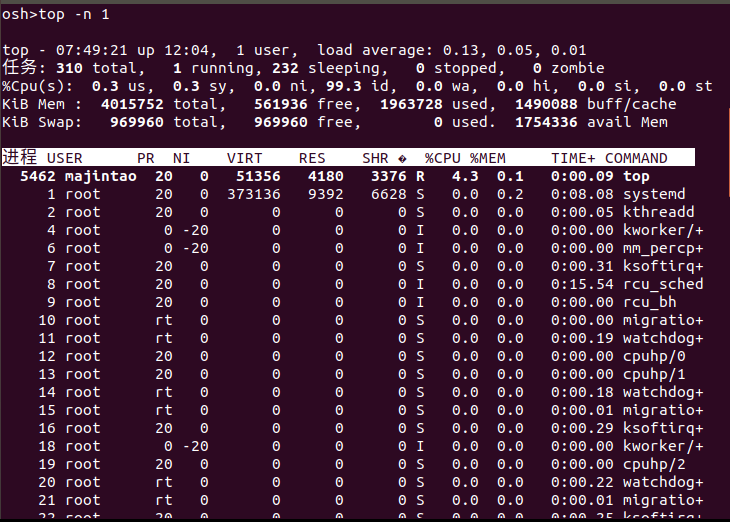
who

C:\Users\dell\AppData\Local\Temp\1555199868(1).png

cal



top –n 1



ls –l

ps



两种从命令历史记录中检索命令的方式：

1. 当用户输入!! ，执行历史记录中的最新命令。

2.当用户输入单个！后跟一个整数N，执行历史记录中的第N个命令。

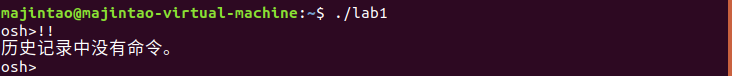
如果用户输入！3，将执行命令cal。

如果用户输入！1，将执行命令date。

C:\Users\dell\AppData\Local\Temp\1555201927(1).png

以这种方式执行的任何命令都应该在用户的屏幕上回显。该命令也应作为下一个命令放在历史缓冲区中。

该程序还管理了基本的错误处理。如果历史记录中没有命令，请输入!!应该会显示一条消息“历史记录中没有命令。



如果没有与单个输入的数字相对应的命令！x，程序应该输出“历史上没有这样的命令”。比如在这里输入“！15”C:\Users\dell\AppData\Local\Temp\1555200577(1).png

1. **总结**

Linux提供的流程控制方法：linux shell有一套自己的流程控制语句，其中包括条件语句(if)，循环语句(for,while)，选择语句(case)。下面我将通过例子介绍下，各个语句使用方法。

**一、shell条件语句（if用法）**

**if语句结构[if/then/elif/else/fi]**

if 条件测试语句

then

action

[elif 条件

action

else

action

]

fi

如果对于：条件测试语句不是很清楚，可以参考：[linux shell 逻辑运算符、逻辑表达式详解](http://www.cnblogs.com/chengmo/archive/2010/10/01/1839942.html)

shell命令，可以按照分号分割，也可以按照换行符分割。如果想一行写入多个命令，可以通过“';”分割。

如：

[chengmo@centos5 ~]$ a=5;if [[ a -gt 4 ]] ;then echo 'ok';fi;                          
ok

**二、循环语句(for,while,until用法）：**

**for循环使用方法(for/do/done)**

**语法结构：**

*1.for … in 语句*

for 变量 in seq字符串

do

action

done

说明：seq字符串 只要用空格字符分割，每次for…in 读取时候，就会按顺序将读到值，给前面的变量。

**while循环使用（while/do/done)**

**while语句结构**

while 条件语句

do

action

done;

**until循环语句**

**语法结构：**

until 条件

do

action

done

意思是：直到满足条件，就退出。否则执行action.

**三、shell选择语句（case、select用法）**

**case选择语句使用（case/esac)**

**语法结构**

case $arg in    
    pattern | sample) # arg in pattern or sample    
    ;;    
    pattern1) # arg in pattern1    
    ;;    
    \*) #default    
    ;;    
esac

说明：pattern1 是正则表达式,可以用下面字符：

                 \*       任意字串  
                 ?       任意字元  
                 [abc]   a, b, 或c三字元其中之一  
                 [a-n]   从a到n的任一字元  
                 |       多重选择

**select语句使用方法（产生菜单选择）**

**语法：**

select 变量name  in seq变量

do

    action

done

说明：select是循环选择，一般与case语句使用。

1. **实验感想与收获**

（1）前期略显漫长的准备工作非常考验一个人的耐心。在Ubuntu18安装gcc的时候我遇见了一些问题，在搜索引擎以及隔壁专业班阮同学的帮助下gcc安装成功。

（2）做实验过程中遇到了很多问题。这次对于操作系统命令解释程序的设计和实现实验使我对操作系统的 shell 有 了初步的理解，熟悉和掌握一些简单的命令，一开始，我通过查阅资料，了解了 linux 系统中提供的一系列的系统调用函数，从中选择了一些常见的命令来实现。 在仔细阅读课本并掌握相关的理论知识之后，才开始着完成满足实验要求的任务显然，在运用知识解决问题之前，一定要掌握理论，让理论指导实践，才能少走 弯路。但是“纸上得来终觉浅，绝知此事要躬行”，许多知识是在运用中得以理 解的，在这次试验中用到许多C语言的的知识，如果对于C语言的理解不够深刻， 是没有办法很好地将自己对相关思想的实现。因此，唯有踏踏实实认真听课才会 更的掌握这些，谢谢老师的悉心教学。

1. **附录**
2. #include<stdio.h>
3. #include<stdlib.h>
4. #include<unistd.h>
5. #include<string.h>
6. #define MAX\_LINE 80 /\* The maximum length command \*/
7. typedef struct
8. {
9. char buf[MAX\_LINE];
10. char \*args[MAX\_LINE / 2 + 1];
11. int bg;
12. int index;
13. }Command;
14. typedef struct
15. {
16. Command\* cmd[10];
17. }History;
18. History history;
19. int cmdmaxindex=0;
20. Command\* paraseCommand(char \*str)
21. {
22. char \*tmp;
23. int i=0;
24. Command\* cmd=(Command\*)malloc(sizeof(Command));
26. memset(cmd,0,sizeof(Command));
27. tmp=strstr(str,"&");
28. if(tmp!=NULL)
29. {
30. \*tmp=0;
31. cmd->bg=1;
32. }
33. strcpy(cmd->buf,str);
34. tmp=strtok(cmd->buf," ");
35. while(tmp!=NULL)
36. {
37. cmd->args[i++]=tmp;
38. tmp=strtok(NULL," ");
39. }
40. return cmd;
41. }
42. void runCommand(Command\* cmd)
43. {
44. int pid = fork();
45. if (pid<0)
46. {
47. printf("fork() error!");
48. exit(1);
49. }
50. else if (pid == 0)
51. {
52. execvp(cmd->args[0], cmd->args);
53. exit(0);
54. }
55. else
56. {
57. if(cmd->bg!=1)
58. wait(NULL);
59. }
60. }
61. void initHistory()
62. {
63. int i;
64. for(i=0;i<10;i++)
65. {
66. history.cmd[i]=NULL;
67. }
68. }
69. void addToHistory(Command\* cmd)
70. {
71. int i;
73. cmdmaxindex+=1;
74. cmd->index=cmdmaxindex;
75. if(history.cmd[9]==NULL)
76. {
77. for(i=0;i<10;i++)
78. if(history.cmd[i]==NULL)
79. {
80. history.cmd[i]=cmd;
81. break;
82. }
83. }
84. else
85. {
86. Command\* tmp=history.cmd[0];
87. for(i=0;i<9;i++)
88. history.cmd[i]=history.cmd[i+1];
89. history.cmd[9]=cmd;
90. if(tmp!=cmd)
91. {
92. free(tmp);
93. }
94. }
95. }
96. void printHistory()
97. {
98. int i;
99. printf("history:\n");
100. for(i=9;i>=0;i--)
101. if(history.cmd[i]!=NULL)
102. {
103. printf("%d %s\n",history.cmd[i]->index,history.cmd[i]->args[0]);
104. }
105. }
106. int main(void)
107. {
108. char \*args[MAX\_LINE / 2 + 1];
109. int should\_run = 1;
110. char readline[MAX\_LINE];
111. Command\* cmd;
113. initHistory();
114. while (should\_run)
115. {
116. printf("osh>");
117. fflush(stdout);
118. memset(readline,0,sizeof(readline));
119. gets(readline);
120. if(strcmp(readline,"exit")==0)
121. {
122. should\_run=0;
123. continue;
124. }
125. else if(strcmp(readline,"history")==0)
126. {
127. printHistory();
128. continue;
129. }
130. else if(readline[0]=='!')
131. {
132. int i,tmp;
133. if(readline[1]=='!')
134. {
135. tmp=cmdmaxindex;
136. }else
137. {
138. tmp=atoi(readline+1);
139. }
140. for(i=0;i<10;i++)
141. if(history.cmd[i]!=NULL&&history.cmd[i]->index==tmp)
142. {
143. break;
144. }
145. if(i>=10)
146. {
147. if(cmdmaxindex==0)
148. printf("历史记录中没有命令。\n");
149. else
150. printf("历史上没有这样的命令\n");
151. cmd=NULL;
152. }
153. else
154. {
155. cmd=(Command\*)malloc(sizeof(Command));
156. memcpy(cmd,history.cmd[i],sizeof(Command));
157. for(tmp=0;tmp<MAX\_LINE / 2 + 1;tmp++)
158. if(cmd->args[tmp]!=NULL)
159. {
160. cmd->args[tmp]=(history.cmd[i]->args[tmp]-history.cmd[i]->buf)+cmd->buf;
161. }
162. }
163. }
164. else
165. cmd=paraseCommand(readline);
166. if(cmd!=NULL)
167. {
168. runCommand(cmd);
169. addToHistory(cmd);
170. }
171. }
172. return 0;

**8.参考资料**

手把手教你安装Linux虚拟机：https://zhuanlan.zhihu.com/p/41940739

Ubuntu 18 安装gcc：https://blog.csdn.net/zhouyingge1104/article/details/80669995

Ubuntu出现“E: 无法定位软件包问题”解决方法：https://jingyan.baidu.com/article/f0e83a254e791622e59101aa.html

linux流程控制：https://blog.csdn.net/gqv2009/article/details/76851026

linux shell流程控制：https://www.cnblogs.com/linuxprobe/p/5590387.html

linux创建子进程--fork()方法：https://www.cnblogs.com/xingzc/p/5988069.html

Linux 创建子进程执行任务：https://www.cnblogs.com/sparkdev/p/8214455.html

fork()创建子进程步骤、函数用法及常见考点（内附fork()过程图）：https://blog.csdn.net/yueyansheng2/article/details/78860040

linux创建子进程：https://blog.csdn.net/zhang\_referee/article/details/82558642