

Operating System

Lab01:Unix Shell and History Feature

|  |  |
| --- | --- |
| 学 院： | 计算机与信息技术学院 |
| 学生姓名： | 刘嘉鹏 |
| 学 号： | 20281319 |

**北京交通大学**

2022年10月

目录

[1. Linux系统进程控制方法概述 3](#_Toc116675630)

[2. 程序实现 4](#_Toc116675631)

[3. 运行结果及分析 5](#_Toc116675632)

[4. 问题分析和解决 7](#_Toc116675633)

[5. 总结与建议 8](#_Toc116675634)

[6. 参考材料 8](#_Toc116675635)

# Linux系统进程控制方法概述

1.1进程相关概念

一个进程映像（进程实体）由程序段、相关数据段和进程控制块（PCB）组成。进程有三种基本状态：就绪、执行和阻塞，且这三种基本状态存在四种转换关系。

|  |
| --- |
| 图1-1进程状态转换图 |

1.2Linux系统的进程控制方法

（1）进程创建

在Linux的C环境中，我们使用fork()函数来为已经存在的进程创建一个新进程。新创建的进程称为子进程，原进程被称为该进程的父进程。fork()函数的返回值是pid\_t类型（其实就是int），如果子进程创建失败，则返回-1，否则会返回“两个”pid\_t，在子进程中，该pid\_t为0，而在父进程中，该pid\_t为一个大于0的数。通过对fork()函数返回值与0关系的判断，可以得知当前进程是父进程还是子进程。

（2）进程终止

一般程序的退出方式有：main函数返回、调用exit()函数、程序异常退出等等。

（3）进程等待

进程等待的目的是防止僵尸进程的产生：如果子进程先于父进程退出，且父进程未回收子进程的资源，那么就会导致子进程成为僵尸进程。而进程等待就是让父进程通过该方式实现子进程资源的回收，并获取子进程的退出状态。

通过在父进程中调用wait()函数来实现进程等待。如语句 pid\_t wait(int\* status);中，pid\_t用于接收被等待进程的pid，失败会返回-1，status用于获取子进程的退出状态（可以设为NULL表示不关心）。

# 程序实现

我把程序分为了如下几个部分分别设计：输入函数、args赋值函数、历史命令记录函数、历史命令输出函数、main函数。

2.1输入函数

get\_input(char \*input)

函数读入一串以回车结尾的字符串，判断其长度是否满足要求，并将其存入待定input数组中以备后续使用。

2.2args赋值函数

get\_args(char \*args[], char \*input, int \*run\_together)

函数通过对input数组进行拆分（按空格或制表符拆分），将输入分割并存入args数组中，删除可能的“&”并在参数最后添加NULL以备后续execvp()函数的调用，其中run\_together参数是依据输入命令是否含有“&”进行赋值的，当含有“&”时表示父进程和子进程一同运行，反之则不是。该函数的返回值为有效命令行参数的个数（除去末尾可能的“&”）。

2.3历史命令记录函数

add\_history(char \*history[], char \*input, int \*his\_num)

当函数被调用时，input数组中的内容将被复制到history[his\_num]中，但是由于his\_num最大为10（最多存储10条历史记录），所以需要一直对history数组进行迭代更新（类似固定长度的队列）。

2.4历史命令输出函数

print\_history(char \*history[])

当函数调用时，可以对当前的所有history进行打印，最多打印10条历史记录，当不够十条时，仅打印存有的历史记录。

2.5 main函数

主要结合pid、run\_together等参数双进程功能进行处理，同时特判了输入“!!”、“!N”、“history”时，对应的处理方法。

# 运行结果及分析

我的运行环境是Windows 10内置Linux子系统wsl2，Ubuntu20.04，结合vscode的wsl扩展进行编程和运行。

|  |
| --- |
| 编译通过，简单测试如下  图3-1 程序简单编译运行结果 |
| 图中命令依次是ps,ls -l,cal,cat hello.c,date,history,!!,!3，运行结果正确。  图3-2 程序基本功能测试 |

测试父子进程同时运行情况：

|  |
| --- |
| 在命令含有“&”时，父进程和子进程会同时运行，终端会输出“Parent is still running”以提示。  图3-3测试包含&的命令 |

测试一些错误情况：

|  |
| --- |
| 通过execvp()函数返回值以及特判识别了一些命令，也是正确的。  图3-3 测试错误命令 |

# 问题分析和解决

在测试中遇到过一个问题，复现如下：

|  |
| --- |
| 图4-1一个可疑的错误 |

从图中可以看出，在父、子进程同步运行时，出现了输出顺序混乱的情况。初步分析可以得知，在执行含有“&”的命令时，可能是由于父、子进程的先后执行顺序导致输出乱序。由于多次实验结果均会产生如上情况，经查阅资料发现父、子进程执行顺序不定，我对此产生了怀疑。

再分析得知，每次运行含“&”指令时，程序给出的运行流程如下：原（父）进程接收命令、父进程运行、父进程进入第二次循环（等待接收命令）、子进程运行。原来是由于在父进程进入下一轮命令前，子进程尚未执行上一条命令，才导致了如上问题。

因此，我在新一轮循环的开始，添加了wait(NULL);语句，以此来优先等待尚未运行的子进程（如果有），完美解决了如上问题。

|  |
| --- |
| 图4-2 修改后的部分 |

# 总结与建议

这次实验极大地锻炼我自己解决问题的能力。从简单的fork()函数调用开始，到进程控制的相关概念、wait()函数的必要性等等，都是我在自己的探索过程中加深了印象。同时我也学习了一些github相关知识，并且把自己的项目上传到了[github](https://github.com/AmiliaSama/2022OS-Unix-shell-and-history-feature)中。

本次实验的实践性很强，让我明白了学习光靠理论是远远不够的，一定要动手和“折腾”才能弄明白一些东西，希望在后续的实验也能继续努力。

# 参考材料

主要参考了Linux进程相关的函数及其调用方法

[Linux进程控制](https://blog.csdn.net/X_Mrjw/article/details/110399885)

https://blog.csdn.net/X\_Mrjw/article/details/110399885

[fork()函数](https://en.wikipedia.org/wiki/Fork_(system_call))

https://en.wikipedia.org/wiki/Fork\_(system\_call)

[execvp()函数](https://www.digitalocean.com/community/tutorials/execvp-function-c-plus-plus)

https://www.digitalocean.com/community/tutorials/execvp-function-c-plus-plus