

操作系统 -- 实验二实验报告

实验二：Linux进程管理

一、实验题目

子进程执行新任务

任务要求：编写一段程序，使用系统调用 `fork()` 创建一个子进程。子进程通过系统调用 `exec` 更换自己原有的执行代码，转去执行 Linux 命令 `/bin/lis` (显示当前目录的列表)，然后调用 `exit()` 函数结束。父进程则调用 `waitpid()` 等待子进程结束，并在子进程结束后显示子进程的标识符，然后正常结束。程序执行过程如图 2-1 所示。

进程的创建

任务要求：编写一段程序，使用系统调用 `fork()` 创建两个子进程。当此程序运行时，在系统中有一个父进程和两个子进程活动。让每一个进程在屏幕上显示一个字符：父进程显示字符“a”；两子进程分别显示字符“b”和字符“c”。

实现一个简单的shell（命令行解释器）(此任务有一些难度，可选做)。

要设计的 shell 类似于 `sh`, `bash`, `csh` 等，必须支持以下内部命令：

cd <目录>更改当前的工作目录到另一个<目录>。如果<目录>未指定，输出当前工作目录。如果<目录>不存在，应当有适当的错误信息提示。这个命令应该也能改变 **PWD** 的环境变量。

environ 列出所有环境变量字符串的设置（类似于 Unix 系统下的 **env** 命令）。

echo <内容> 显示 echo 后的内容且换行

help 简短概要的输出你的 shell 的使用方法和基本功能。

jobs 输出 shell 当前的一系列子进程，必须提供子进程的命名和 PID 号。

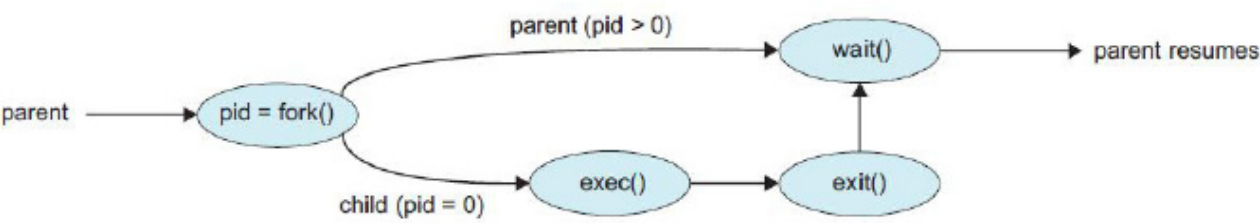
quit,exit,bye 退出 shell。

二、实验目的

通过进程的创建、撤销和运行加深对进程概念和进程并发执行的理解，明确进程和程序之间的区别。

三、总体设计

进程创建部分程序流程图如下：



Shell程序设计：

shell 的主体就是反复下面的循环过程

```
while(1){

//接收用户输入的命令行；
//解析命令行；

if(用户命令为内部命令)

//直接处理；

else if(用户命令为外部命令)

//创建子进程执行命令；      //参考清单 2-2

else

//提示错误的命令；

}
```

四、详细设计

(1) 进程的创建

任务要求：编写一段程序，使用系统调用 fork（）创建两个子进程。当此程序运行时，在系统中有一个父进程和两个子进程活动。让每一个进程在屏幕上显示一个字符：父进程显示字符“a”；两子进程分别显示字符“b”和字符“c”。

步骤1：使用 vi 或 gedit 新建一个 fork_demo.c 程序，然后拷贝清单* 2-1 中的程序，使用 cc** 或者 gcc 编译成可执行文件 fork_demo。例如，可以使用 gcc -o fork_demo fork_demo.c 完成编译。

步骤 2：在命令行输入 ./fork_demo 运行该程序。

(2) 子进程执行新任务

任务要求：编写一段程序，使用系统调用 fork（）创建一个子进程。子进程通过系统调用 exec 更换自己原有的执行代码，转去执行 Linux 命令/bin/lis (显示当前目录的列表)，然后调用 exit（）函数结束。父进程则调用 waitpid() 等待子进程结束，并在子进程结束后显示子进程的标识符，然后正常结束。程序执行过程如图2-1 所示。

步骤 1：使用*vi** 或 gedit 新建一个 exec_demo.c 程序，然后拷贝清单 2-2 中的程序（该程序的执行如图 2-1 所示），使用 cc 或者 gcc 编译成可执行文件 exec_demo。例如，可以使用 gcc -o exec_demo exec_demo.c 完成编译。

步骤 2：在命令行输入 ./exec_demo 运行该程序。

步骤 3：观察该程序在屏幕上的显示结果，并分析。

(3) 实现一个简单的shell（命令行解释器）(此任务有一些难度，可选做)。

要设计的 shell 类似于 sh,bash,csh 等，必须支持以下内部命令：

cd<目录>更改当前的工作目录到另一个<目录>。如果<目录>未指定，输出当前工作目录。如果<目录>不存在，应当有适当的错误信息提示。这个命令应该也能改变 **PWD** 的环境变量。

environ 列出所有环境变量字符串的设置（类似于** Unix 系统下的env **命令）。

echo <内容>显示 echo后的内容且换行

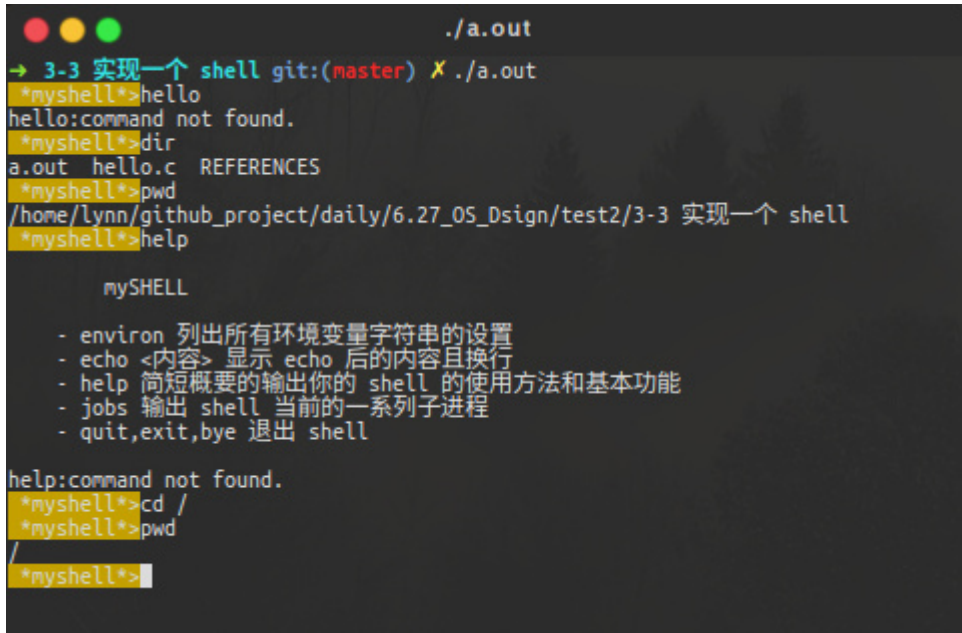
help简短概要的输出你的 **shell** **的使用方法和基本功能。

jobs输出 **shell** 当前的一系列子进程，必须提供子进程的命名和 **PID****号。

quit,exit,bye退出shell。

五、实验结果与分析

实验结果图：



实验结果分析：

整体框架为：一个死循环,一直在等待这用户输入命令.主要的工作都在eval函数里面。

eval函数实现

Linux中命令分为内置命令和外置命令,内置命令不需要开进程就可以直接执行,比如cd,pwd.外置命令需要重开个进程才能执行,比如vim,/bin/ls,/bin/echo. eval函数中有两个外调函数就是parseline,解析命令行cmdstring,储存在argv数组中.bulidin_command函数判断命令是否是内置命令,如果是,则执行.不是则返回false.

parseline函数实现

就是一个字符串操作,根据空格分割,写的比较屎.读者可以自己实现,不需要模仿我的.记得最后的argv数组要以NULL结尾.

六、小结与心得体会

- 关于fork()函数:

fork () 函数调用时, 在调用的地方立马创建一个子进程, 子进程同时立马开始执行代码 (全部代码)。对于以下代码:

```
12 int main()
13 {
14     int x;
15     printf("1111");
16     x=fork();
17     printf("222");
18     printf("x=%d\n",x);
19     printf("333\n");
20     int i=0;
21     int b=9;
22     while(i<9)
23     {
24         i++;
25         b=i;
26     }
27     printf("finally\n");
28 }
```

当**22**行为 `while(i<999999)` 时, 运行结果为:

```
1111222x=22093
333
1111222x=0
333
finally
finally
```

当**22**行为 `while(i<9)` 时, 运行结果为:

```
1111222x=22093
333
finally
1111222x=0
333
finally
```

由此可见, 执行父进程执行 `fork()` 函数之后, 当即产生一个子进程, 并且同时子进程开始运行(相同的代码)。