# 实验一 进程控制实验

## 实验目的

1、掌握进程的概念，了解进程的结构、状态，认识进程并发执行的实质。

2、熟悉进程控制相关的命令。

3、能够使用系统调用完成进程的创建，形成多进程并发执行的环境.

4、了解进程控制的系统调用，可实现对进程的有效控制

## 实验内容

**第4题：**编写程序，使用fork( )创建两个子进程。观察在程序运行过程中的进程状态变化，分析原因。

(1)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串,其中,每个进程显示其pid值，及其父进程的pid值。类似如下显示:父进程显示'"I am parent.pid:...ppid:.... "，子进程分别显示'"I am child1.pid:..., ppid:.... "和'"I am child2.pid:..., ppid:...."。试观察并分析屏幕上的显示结果。

（2）改写以上程序，使得父进程在创建所有子进程之前执行一次system(“ps -af”),创建子进程之后,每个进程都执行一次system(“ps -af”)。通过在多进程执行过程中执行命令“ps -af”,显示当前进程状态。试观察并分析屏幕上的显示结果。

从进程并发执行来看，上面的三个进程没有同步措施，只要进程就绪就可能执行，因此各种执行顺序都有可能，所以三个进程的输出次序带有随机性。并且，每当一个进程执行了一段时间，其它就绪进程可能抢占处理机，因此，多个进程可能交错执行。不过，操作系统实现函数printf( )时，保证了进程每次调用该函数输出一个字符串时不会被中断。

程序代码：

#include<stdio.h>

#include<unistd.h>

#include<stdlib.h>

#include<wait.h>

int main()

{

pid\_t pid1;

pid\_t pid2;

system("ps -f");

pid1 = fork();

if (pid1 == -1)

{

printf("fork fail!\n");

}

else if(pid1 == 0)

{

printf("I am child: PID: %d PPID: %d\n", getpid(), getppid());

system("ps -f");

exit(0);

}

else

{

printf("I am parent: PID: %d PPID: %d\n", getpid(), getppid());

pid2 = fork();

if(pid2 < 0)

{

printf("fork fail!\n");

}

if(pid2 == 0)

{

printf("I am child: PID: %d PPID: %d\n", getpid(), getppid());

system("ps -f");

exit(0);

}

wait(0);

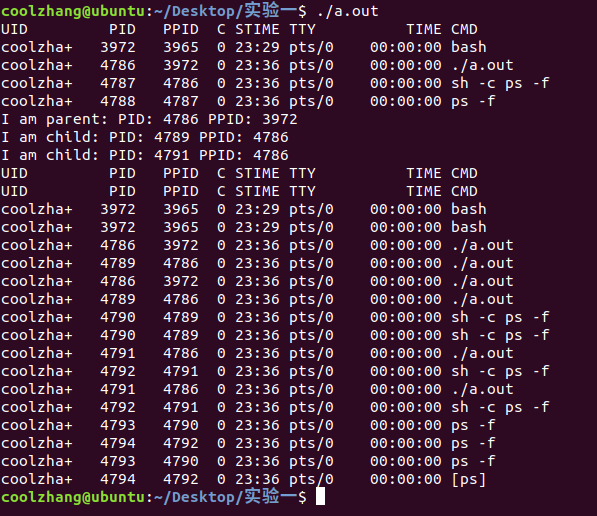
wait(0);

}

return 0;

}

运行结果：

分析：

**第11题：**编写简单的shell：父进程获取用户输入，创建子进程来执行用户输入命令，父进程等待子进程结束后输出用户提示符等待用户继续输入命令。

程序代码：

#include<stdio.h>

#include<unistd.h>

#include<stdlib.h>

#include<wait.h>

int main()

{

while(1)

{

char buf[100] = {0};

printf("输入命令：");

fflush(stdin);

scanf("%s",buf);

pid\_t pid = fork();

if (pid == -1)

{

printf("子进程创建失败");

}

else if (pid == 0)

{

system(buf);

exit(0);

}

else

{

wait(NULL);

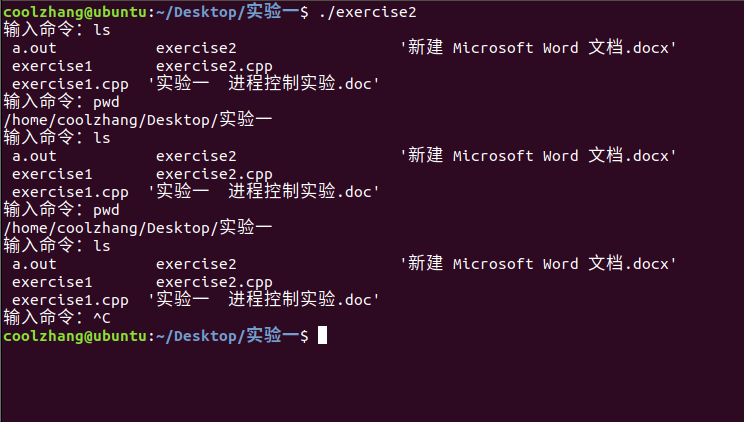
}

}

return 0;

}

运行结果：



分析：