用 C 语言创建 Linux 进程

要用到的系统调用

1、int fork( )创建一个新进程

头文件：#include <sys/types.h>
#include <unistd.h>

返回值的意义： 0，创建子进程，从子进程返回的 id 值
-1，创建失败
大于 0，从父进程返回的子进程 id 值

实验内容

0、 创建进程。

输入并运行下面的程序。思考为什么会输出两行信息。

```
int main(){
        int pid;
        pid = fork();
        switch(pid){
                case -1:
                        printf("fail to create process\n");
                        return 1;
                case 0: //对于子进程来讲，变量 pid 的值为 0
                        printf("I'm son, my pid is %d, my father's pid is %d\n", getpid(), getppid());
                        break;
                default: //对于父进程来讲，pid 记录了返回的子进程的 ID，必然大于 0
                        printf("I'm father, my pid is %d, my son's pid is %d\n", getpid(), pid);
        }
        return 0;
}
```

输出结果并非是一行，而是 2 行。比如，结果可能是：

I'm son, my pid is 1860, my father's pid is 1859
I'm father, my pid is 1859, my son's pid is 1860

解释：

fork 函数被调用一次但返回两次。两次返回的唯一区别：子进程中返回 0 值，父进程中返回子进程 ID。

为什么返回两次？原因：linux 将父进程的地址空间、数据空间、堆、栈等资源复制一份给子进程，子进程是父进程的副本，只有代码段是共享的。注意：由于在复制时复制了父进程的堆栈段，所以两个进程都停留在 fork 函数中，等待返回。不同的是，fork()返回给父进程的是新建子进程的 ID，返回给子进程的是 0。

1、验证进程并发执行

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。让每个进程在屏幕上显示一个字符串：父进程显示“parent is working”；两个子进程分别显示“son is working”和“daughter is working”。多运行几次该程序，观察屏幕上的显示结果，并分析原因。

<程序 1 如下> 文件名为 a21.c

```
#include <stdio.h>
main(){
        int p1,p2;
        while((p1=fork( ))= = -1);                    /*创建子进程，将其 ID 号给 p1，直到成功*/
        if(p1= =0) printf(“son is working\n”);/*子进程返回的 p1 才为 0，说明此时子进程执行*/
        else{/*p1>0，说明此 p1 是父进程函数调用返回的子进程的 ID 号*/
                while((p2=fork( ))= = -1);        /*创建子进程 p2，将其 ID 号给 p2，直到成功*/
                if(p2= =0) printf(“dauthter is working\n”); /*子进程 p2 执行*/
                else printf(“parent\n”);                    /*父进程执行*/
        }
        printf(“share\n”);/*3 个进程共享的语句*/
}
```

执行：用 gcc a21.c –o a21.out 编译，产生 a21.out 文件。在终端里输入./a21.out，并回车执行。看结果。再按向上箭头，并回车重复执行。看结果。如此多执行几次。

分析：可以认为，父进程和两个子进程有相同的代码。相当于，上述程序有 3 份在内存。它们并发执行，但它们返回的 ID 不同，因此执行不同的分支语句。从进程并发执行来看，输出有多种情况。可能是 son 在前，也可能是 parent 或 daughter 在前。大多数情况下，son 先输出。

原因：fork()创建进程所需的时间可能要多于输出一个字符串的时间，因此在主进程创建进程 p2 的这段时间可能有中断，进程 p1 就输出了 son is working。

<程序 2 如下>（对程序 1 稍做修改，修改处用红色标注）文件名为 a22.c

```
#include <stdio.h>
main(){
        int p1, p2, i;
```

```
            while((p1=fork( ))= = -1);          /*创建子进程 p1*/
            if(p1= =0)    for(i=0; i<50; i++)    printf("son %d\n", i); /*子进程 p1 执行
*/
            else{
                        while((p2=fork( ))= = -1);          /*创建子进程 p2*/
                        if(p1= =0)    for(i=0; i<50; i++)    printf("dauthter%d\n", i);
/*子进程 p2 执行*/
                        else    for(i=0; i<50; i++)    printf("parent%d\n", i); /*父进程
执行*/
            }
}
```

执行程序并分析结果：进程中的循环会由于进程被中断而中断，但字符串内部的字符顺序保持不变。同程序 1 一样，多进程是随机的，故，输出的字符串的顺序会有不同结果。