实验一：进程创建、并发执行

一、 实验目的

加强对进程概念的理解

进一步了解并发执行的实质

二、 实验内容

1、 利用 fork()函数创建子进程。

2、 考察 fork()函数创建的子进程与父进程之间的同名变量是否为临界资源。

3、 利用 fork()函数编写一个程序，要求父进程创建两个子进程，父进程、子进程并发执行，输出并发执行的消息。

三、 实验环境

PC + Linux Red Hat 操作系统

GCC

四、 实验原理及实验思路

fork()

功能：创建一个新的进程

语法：#include <unistd.h>

#include <sys/types.h>

pid_t fork();

说明：本系统调用为调用进程（也称父进程）创建一子进程。事实上，子进程是父进程的一个“复制品”。父子进程为独立进程，平等调度，用户空间独立 。

返回值：调用成功，则返回两次。对子进程返回 0，对父进程返回子进程号，这也是最方便的区分父子进程的方法。调用失败则返回-1 给父进程，子进程不生成

kill()

功能：杀死执行中的进程

语法：#include <sys/types.h>

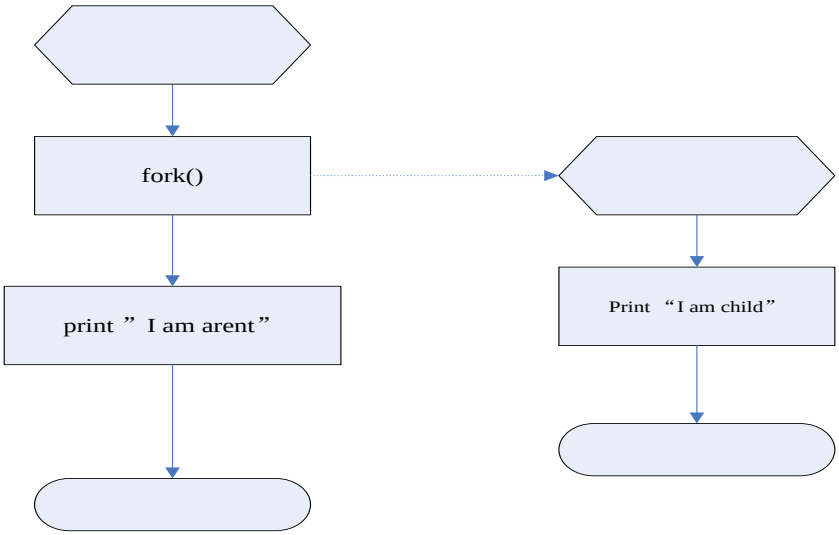
#include <signal.h>

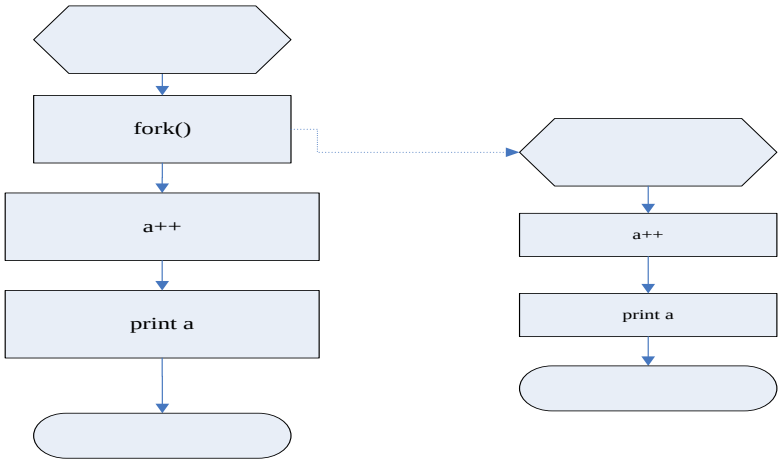
void kill(pid_t pid,int signo);

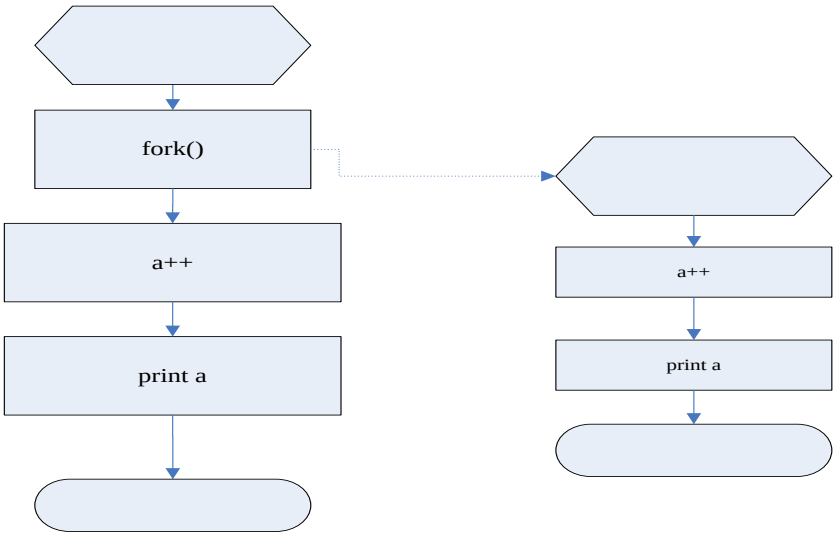
说明：pid 为要被杀死的进程 id,signo 可以置为 SIGINT 或 SIGTERM。

返回值：等待到一个子进程返回时，返回值为该子进程号，同时 stat_loc 带回子进程的返回状态信息（参考 exit）。若无子进程，则返回值为-1。

五、 流程图


fork()
print ” I am arent”
Print “I am child”


fork()
a++
print a
a++
print a


fork()
a++
print a
a++
print a

六、源代码

Lab1-1：

```
#include <unistd.h>
#include <sys/types.h>
#include <stdio.h>
main()
{
    pid_t child;
    printf(“Forking...\n");
child =fork();
    if (child<0){
        perror("Fork failed!\n");
        exit(1);
    }
    else if (child==0){
        printf("I'm the child!\n");
    }
    else{
        printf("I'm the parent!\n");
    }
    printf("Why I'm printed twice??\n");
}
```

Lab1-2：

```
#include <unistd.h>
#include <sys/types.h>
#include <stdio.h>
main()
{
    pid_t child;
    int a=0;
    printf("Forking...\n");
child=fork();
    if (child<0){
        perror("Fork failed!\n");
        exit(1);
    }
    else if (child==0){
        a++;
        printf("Child:a=%d\n",a);
    }
    else{
```

```
        a++;
        printf("Parent:a=%d\n",a);
    }
}
```

Lab1-3：

```
#include "unistd.h"
#include "sys/types.h"
#include "signal.h"
#include "stdio.h"
int main(int argc,char* argv[])
{
 pid_t child1_pid,child2_pid;
 int i = 15;
 /*fork*/
 printf("first fork\n");
 child1_pid = fork();
 if(child1_pid < 0)
 {
  printf("fork() fail!\n");
  return -1;
 }
 else if(child1_pid == 0)
 {
  printf("this is the first child process \n");
  while(1)
  {
   sleep(1);
   printf("the first child proc waiting to be killed\n");
  }
 }
 else
 {
  printf("this is farther process, after first fork\n");
 }
 /*fork*/
 printf("second fork\n");
 child2_pid = fork();
 if(child2_pid < 0)
 {
  printf("fork() fail!\n");
  return -1;
 }
 else if(child2_pid == 0)
```

```
 {
  printf("this is the second child process \n");
  while(1)
  {
    sleep(1);
    printf("the second child proc waiting to be killed\n");
  }
 }
 else
 {
  printf("this is farther process, after second fork\n");
 }
 while(i > 0)
 {
  printf("after %d second,all proc will be killed\n",i);
  sleep(2);
  i -= 2;
 }
 /*kill*/
 printf("kill the first child proc\n");
 kill(child1_pid,SIGINT);
 printf("kill the second child proc\n");
 kill(child2_pid,SIGINT);
 return 0;

}
```

七、 运行结果及其分析

Lab1-1：输出：Forking...
I'm the child!
Why I'm printed twice??

Lab1-2：输出：Forking...
Child a1

Lab1-3：输出：first fork
this is the first child process
the first child proc waiting to be killed
this is the second child process
this is farther process, after second fork

八、 实验总结

加强了对进程概念的理解，进一步了解了并发执行的实质