**浙大城市学院实验报告**

课程名称 操作系统原理实验

实验项目名称 实验三-进程的创建和简单控制

学生姓名 司晨旭 专业班级 计算2002 学号 32001019

**实验目的：**

1. 理解系统调用的概念；
2. 认识进程的并发执行，了解进程族之间各种标识及其存在的关系；
3. 熟悉进程的创建、阻塞、唤醒、撤销等控制方法。

**实验内容：**

1. 掌握进程创建的系统调用fork ( )；
2. 了解并发程序的不可确定性，进行简单并发程序设计。
3. 使用系统调用：进程的创建fork ( )、阻塞wait ( )、睡眠sleep ( )、终止exit( )等。

**实验步骤：**

**(四)例程，使用fork( )创建进程**

1. 编辑下述代码，详见教材P98。

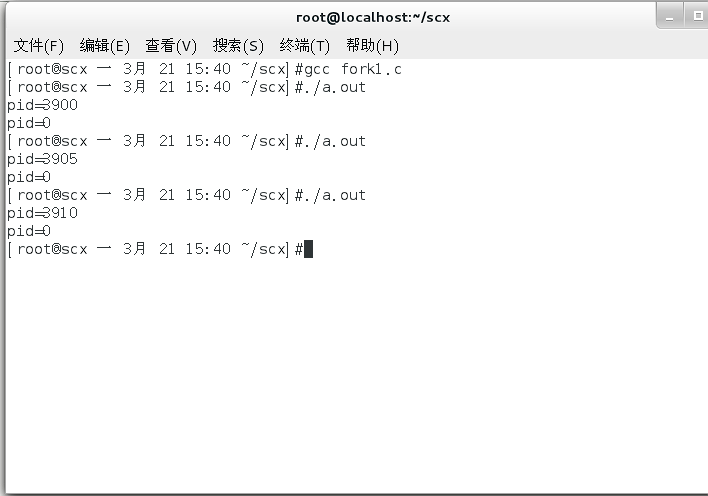
|  |
| --- |
| #include<stdio.h>  main()  {  int pid;  pid = fork();  printf("pid= %d\n",pid); ➀  //printf("pid= %d\n",getpid()); ➁  sleep(1);  } |

编译链接通过后，多次运行例程，观察进程并发执行结果，并思考下述问题：

1. 为什么会有两行输出？理解fork（）的作用；

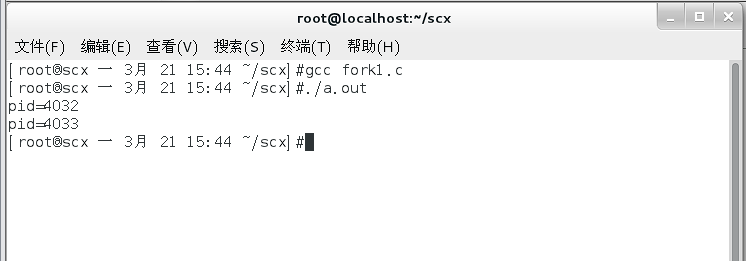
**因为fork（）创建了一个子进程，父进程和子进程均执行了printf（）函数**

1. 多次运行，观察输出内容的变化，理解系统给进程随机分配进程号；



**每次执行的pid的值都不一样，系统随机给进程分配进程号**

1. 将语句➀替换成语句➁，再次运行程序。观察输出的改变，理解fork（）的返回值。



**getpid()为获得当前进程的id**

**fork()有三种可能的返回值，在父进程中，返回子进程的进程id**

**在子进程中，返回0**

**出现错误返回负值**

1. 编辑下面的程序，要求实现父进程产生两个子进程，父进程显示字符“a”、两个子进程，分别显示字符“b”、“c”，如下所示。

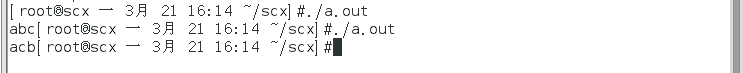
|  |
| --- |
| #include<stdio.h>  main( )  {  int p1,p2;  while ((p1 = fork( )) == -1); //父进程创建第一个进程，直到成功  if(p1 == 0) //0返回给子进程 1  {  putchar('b'); //P1的处理过程  }  else ➀  { //正数返回给父进程(子进程号)  while ((p2 = fork( )) == -1); //父进程创建第二个进程，直到成功  if(p2 == 0) //0返回给子进程2  {  putchar('c'); //P2的处理过程  }  else ➁  {  putchar('a'); //P2创建完成后，父进程的处理过程  }  }  } |

编译链接通过后，多次运行例程，观察进程并发执行结果，并思考下述问题：

1. 分析为何例程中三个分支都运行了？

**总过有三个进程，每个进程执行不同的分支。父进程的第一个子进程输出b，第二个子进程输出c，父进程输出a**

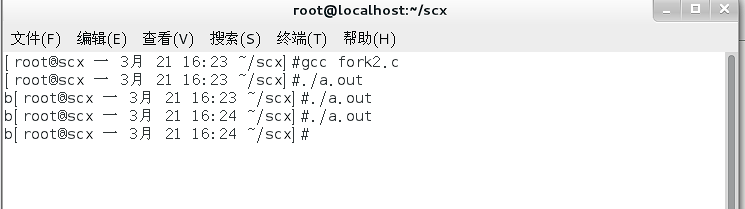
1. ./f1 运行结果为什么不一样？每种结果的产生原因。



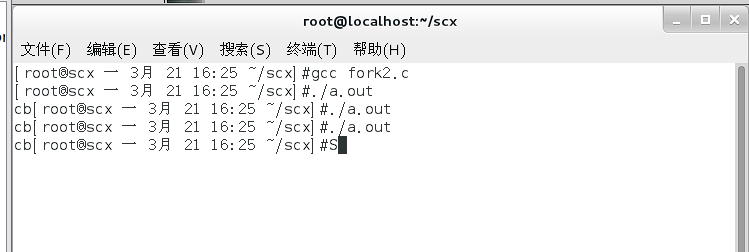
**因为进程的执行顺序并不固定，三个进程不一定谁先执行完putchar（）语句，和时间片有关。**

1. 删除语句➀或➁，观察输出的内容，体会fork的使用。

删除语句1之后的运行内容



删除语句2之后的运行内容



**则fork（）为创建一个进程的含义**

1. 运行命令为什么是“./command”？理解Linux的PATH环境变量的作用。

**./表示在当前目录下执行文件。环境变量的意思是，如果在当前目录下没有找到这个文件，就在设置好的环境变量中查找是否有这个文件。**

1. . 和 .. 什么含义？理解Linux当前目录和父目录的概念。

**.表示当前路径下，..表示上一层路径。**

1. shell 提示为什么不换行，而是紧接着输出内容显示？

**shell 进程也是一个并发的进程**

1. 输出字母为什么和提示交错？

**有些输出字母的进程还没有执行到输出语句，而有的进程已经执行到输出语句，shell进程的输出语句可能先执行到，因此先输出了出来**

1. ./f1|pstree|grep f1什么含义？理解命令中管道的作用和使用方法。

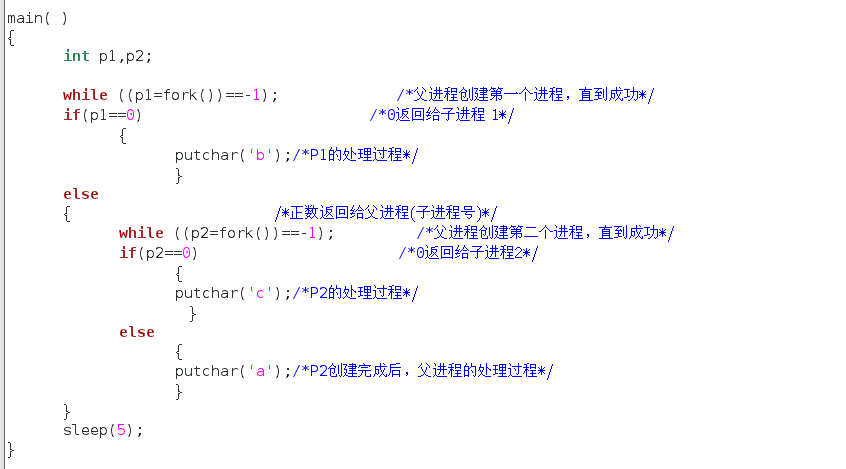
**指令中的|表示管道，即将|前面的指令的输出结果作为|后面的指令的结果的输入，同时grep的功能为分析一行信息，若当中有感兴趣的信息，就将该行拿出来。**

1. 第8问中有时组合命令没有输出，请分析原因？

**进程所存在的时间太短了，无法被捕捉到**

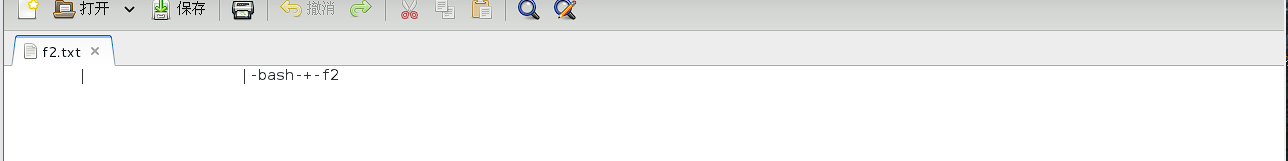
1. 如果想保留第8问的./f1的输出内容，该如何操作？理解Linux文件重定向的概念和使用方法。

**可以在代码中添加一行sleep（）。使得进程存在的时间长一些。**





**使用重定位的输出来保存输出的结果**



**扩展编程：修改代码，产生祖孙三代的进程。**

**代码如下**

|  |
| --- |
| #include <unistd.h>  #include <stdio.h>  #include <sys/types.h>  main( )  {  int p1,p2,p3;  while ((p1=fork())==-1);  if(p1==0)  {  while ((p2=fork())==-1);  if(p2==0)  {  printf("I am the grandchild,my pid is %d\n",getpid());  }  else  {  printf("I am son,my pid is %d\n",getpid());  }    }  else  {  printf("I am grandfather,my pid is%d\n",getpid());  }  } |

**运行结果如下**



**(五)例程，使用getpid（）和getppid（）查看进程号。**

1. 理解系统调用fork（）的两个返回值，理解获取进程号的系统调用getpid( )和getppid( )。

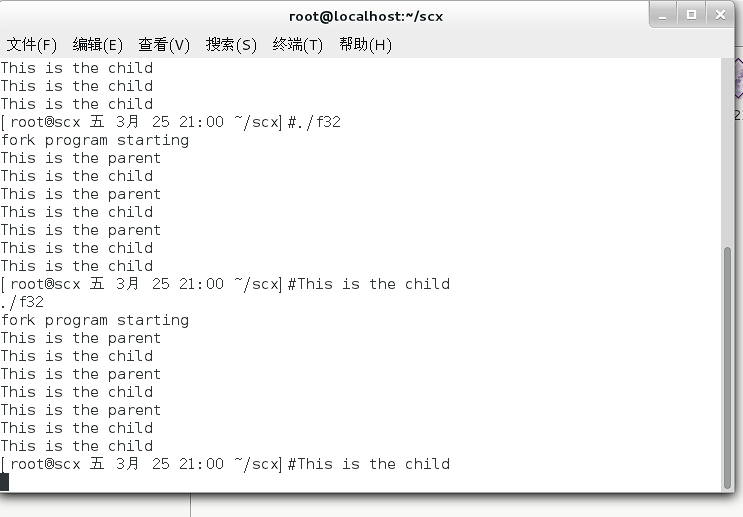
|  |
| --- |
| #include <unistd.h>  #include <stdio.h>  #include <sys/types.h>  main ()  {  pid\_t pid;  pid=fork();  if (pid < 0)  printf("error in fork!");  else if (pid == 0)  {  printf("i am the child process, my process id is %d\n",getpid());  }  else  {  printf("i am the parent process, my process id is %d\n",getpid());  }  } |

编译链接通过后，多次运行例程，观察进程并发执行结果，并思考下述问题：

**讲义中给的代码我在运行中没有遇到父子进程交替的现象，因此我接下来使用的代码是老师给的fork1.zip里的**

1. 请分析父子进程输出内容交替的原因。

**进程是并发运行的，一个进程在它的时间片到了之后，cpu转去执行另一个进程**



1. 改写原程序，用变量pid替换getpid()，再次观察运行情况，理解fork()在父子进程中有不同的返回值。

**fork()在父进程中返回新创建的子进程的进程ID**

**在子进程中，fork()返回0**

**如果出现错误，fork()返回一个负值**

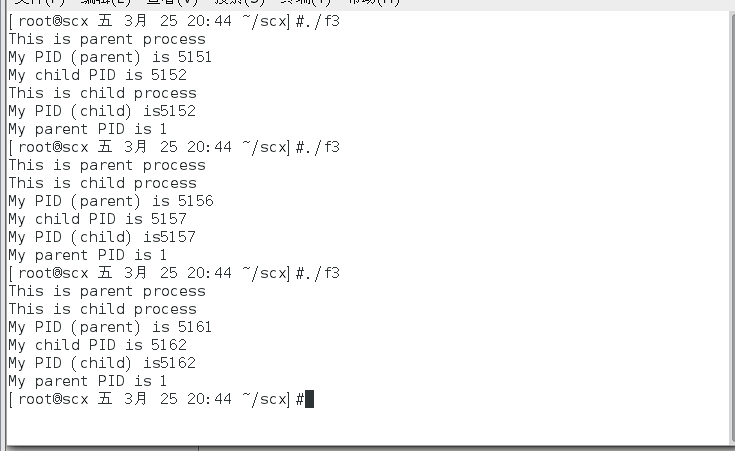
1. 理解上图中两次运行后，子进程输出的差异，理解孤儿进程的概念。

**当一个进程的父进程提前结束时，这个儿子进程没有了父亲，于是系统让它 的父进程变成了pid为1的进程（init）。这个儿子进程也就变成了孤儿进程**

1. 理解系统调用wait( )、getpid( )和getppid( )的使用。程序代码如下所示。

|  |
| --- |
| #include <stdio.h>  #include <sys/types.h>  #include <sys/wait.h>  #include <sys/stat.h>  #include <fcntl.h>  #include <unistd.h>  #include <errno.h>  int main()  {  char buf[100];  pid\_t cld\_pid;  int fd;  if((fd=open("temp",O\_CREAT|O\_TRUNC|O\_RDWR,S\_IRWXU))==-1)  {  printf("open error%d",errno);  exit(1);  }  strcpy(buf,"This is parent process write\n");  if((cld\_pid=fork())==0)  { //这里是子进程执行的代码  strcpy(buf,"This is child process write\n");  printf("This is child process\n");  sleep(1);  printf("My PID (child) is%d\n",getpid()); //打印出本进程的ID  sleep(1);  printf("My parent PID is %d\n",getppid()); //打印出父进程的ID  sleep(1);  write(fd,buf,strlen(buf));  close(fd);  exit(0);  }  else  { //这里是父进程执行的代码  wait(0); //如果此处没有这一句会如何？  printf("This is parent process\n");  sleep(1);  printf("My PID (parent) is %d\n",getpid()); //打印出本进程的ID  sleep(1);  printf("My child PID is %d\n",cld\_pid); //打印出子进程的ID  sleep(1);  write(fd,buf,strlen(buf));  close(fd);  }  return 0;  } |

编译链接通过后，多次运行例程，观察进程并发执行结果，并思考下述问题：



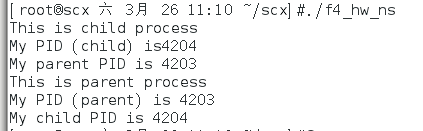
1. 分析父子进程输出内容交替的原因；

**进程是并发运行的，一个进程在它的时间片到了之后，cpu转去执行另一个进程**

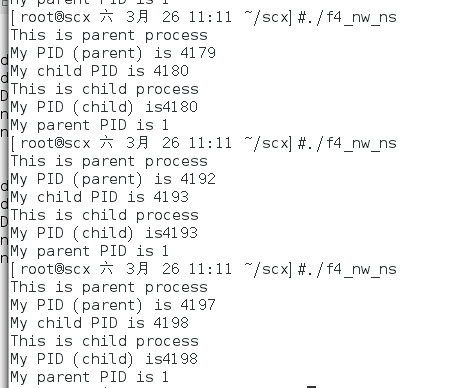
**也就是说，cpu在执行父进程时，并没有执行完，转去执行子进程，然后子进程也没有执行完，再去执行父进程**

1. 语句sleep(1);起什么作用？删除所有sleep(1);语句，并观察运行结果；

**单独删除sleep（1），不删除wait（）。运行结果仍为子进程先运行结束，然后父进程再运行。**



**sleep()和wait()全删除完之后，运行结果如下**

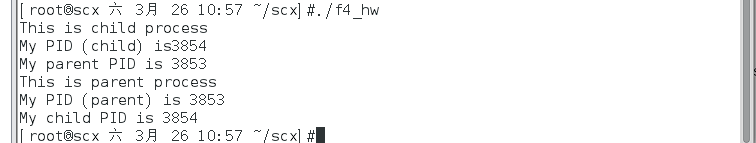


**可以看到，删除sleep()后父进程结束的太快，导致儿子进程成为了孤儿进程。**

**因此sleep(1)的作用为当前进程休眠1s**

1. 删除wait(0);语句，并观察运行结果，并请分析两次结果不同的原因，理解wait的作用。





**可以看到两次运行结果不一样，一个是父子进程交替出现，一个是子进程结束，父进程再运行。**

**wait()用于使父进程阻塞，直到当前进程的一个子进程结束或者收到了一个指定信号为止。如果该进程没有子进程或者它的子进程已结束，则wait（）就会立刻返回。**

1. 扩充：关于父子进程各自又再生成子进程的例子。

|  |
| --- |
| #include <unistd.h>  #include <stdio.h>  #include <sys/types.h>  main()  {  pid\_t a\_pid,b\_pid;  if((a\_pid=fork())<0)  printf("error!");  else  if(a\_pid==0)  printf("b\n");  else  printf("a\n");    if((b\_pid=fork())<0)  printf("error!");  else  if(b\_pid==0)  printf("c\n");  else  printf("a\n");  } |

编译链接通过后，多次运行例程，观察进程并发执行结果，并思考下述问题：

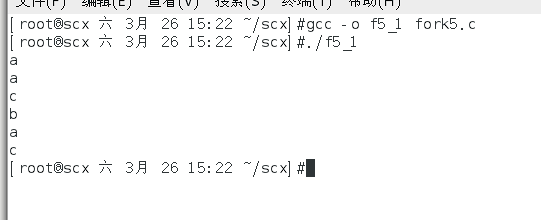
1. 例程运行后，共产生了几个进程？请分析出它们的宗族关系。

**一共产生了四个进程，分别设为1，2，3，4**

**1进程产生了2，3两个进程。2进程产生了4进程**

1. 例程运行后，共输出几个字符？分别是什么字符？分别由哪个进程输出的？

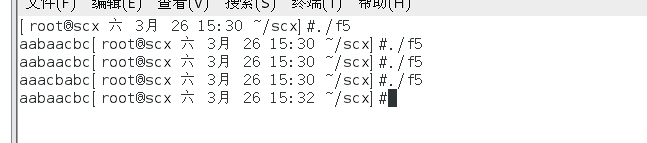
如图所示



**一共输出了6个字母，其中前两个aa由1进程输出，然后下面的c由3进程输出，ba由2进程输出，最后的c由4进程输出**

1. 删除输出语句中的回车符，输出结果有何改变？试分析原因，理解输出缓冲的概念。

**输出结果如下**



**在执行printf()语句时是将缓冲区的内容一并输出，当有\n时，实际上将前面的缓冲区里的内容清零。因此删掉\n后，输出里多了一个a，一个b**

**(六)例程，观察僵尸进程和孤儿进程。**

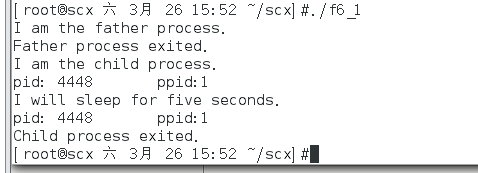
1. 编辑运行下述程序，观察孤儿进程：

|  |
| --- |
| #include <stdio.h>#include <stdlib.h>#include <errno.h>#include <unistd.h>  int main(){ pid\_t pid;  pid = fork(); //创建一个进程 if (pid == 0) //子进程 { printf("child:I am the child process.\n"); printf("child:pid: %d\tppid:%d\n", getpid(), getppid());//输出进程ID和父进程ID printf("child:I will sleep for five seconds.\n"); sleep(5);//睡眠5s，保证父进程先退出 printf("child:pid: %d\tppid:%d\n", getpid(), getppid()); printf("Child process exited.\n"); } else//父进程 { printf("I am the father process.\n"); sleep(1);//父进程睡眠1s，保证子进程输出进程id printf("Father process exited.\n"); } exit(0);} |

编译链接通过后，多次运行例程，观察进程并发执行结果，并思考下述问题：

1. 删除所有sleep（），观察父子进程的宗族关系及各自的进程号；

**删除所有sleep（）后运行结果如下**



**删除所有sleep（）后，所提供代码只能看到子进程的进程号为4448**

1. 恢复所有sleep（），观察父进程提前结束后，子进程成为孤儿进程转交给1号进程。

**运行结果如下**

**刚开始父进程为退出，因此父进程号为4288，子进程号为4299**



**父进程先结束运行，因此子进程成为了孤儿进程，转交给1号进程**

1. 编辑运行下述例程，观察孤儿进程：

|  |
| --- |
| #include <stdio.h>#include <unistd.h>#include <errno.h>#include <stdlib.h>  int main(){ pid\_t pid; pid = fork(); if (pid == 0) { printf("I am the child process. I existed.\n");  exit(0); } printf("I am the father process.I will sleep for two seconds\n"); //等待子进程先退出 sleep(2); //输出进程信息 system("ps -o pid,ppid,state,tty,command"); printf("Father process exited.\n"); exit(0);  } |

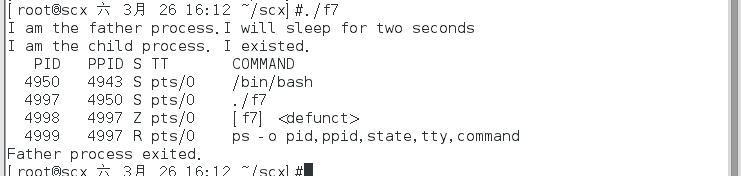
编译链接通过后，多次运行例程，观察进程并发执行结果，并思考下述问题：

1. 理解system（），在程序中运行shell命令；

**system()为在程序中运行shell命令，在本程序中运行了**

**ps -o pid,ppid,state,tty,command**

1. 观察输出内容，查看僵尸进程。



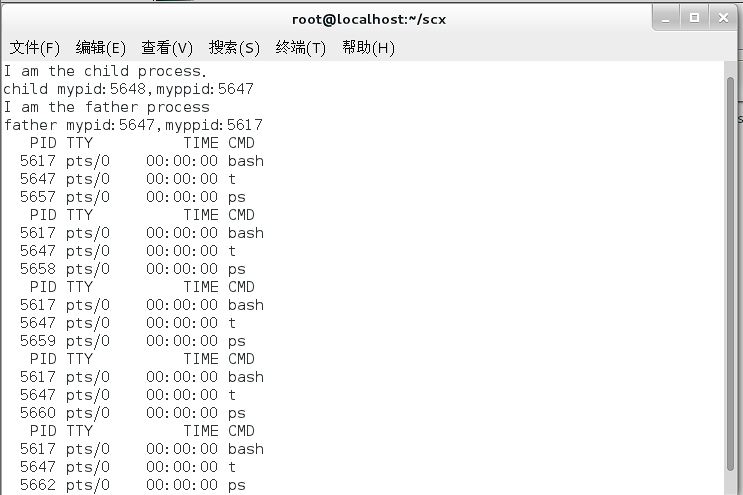
**运行结果如图，S为Z的进程为zombie进程**

**(七)编程题：理解前述例程后，按要求完成程序编写。**

编写程序创建子进程。父子进程分别打印自己和父进程的进程ID，要求每3秒钟打印系统进程信息，重复5次后退出。父进程待子进程结束后退出。提示：

1. 用系统调用getpid和getppid获取进程ID；
2. 用系统调用fork进程创建；
3. 用系统调用wait控制父子进程同步；
4. 用库函数system实现在一个进程内部运行另一个进程，即创建一个新进程；
5. Shell命令＂/bin/ps＂作为system的字符串参数，实现打印系统进程信息。

**运行结果如下**



**代码如下**

|  |
| --- |
| **#include <stdio.h>**  **#include <sys/types.h>**  **#include <sys/wait.h>**  **#include <sys/stat.h>**  **#include <fcntl.h>**  **#include <unistd.h>**  **#include <errno.h>**  **#include <stdlib.h>**  **int main()**  **{**  **int pid;**  **pid = fork();**  **if (pid < 0)**  **{**  **perror("fork error:");**  **exit(1);**  **}**  **else if (pid == 0)**  **{**  **printf("I am the child process.\n");**  **printf("child mypid:%d,myppid:%d\n",getpid(),getppid());**  **exit(0);**  **}**    **{**  **wait(0);**  **printf("I am the father process\n");**  **printf("father mypid:%d,myppid:%d\n",getpid(),getppid());**  **}**  **int i = 0;**  **for (i = 0; i < 5; i++)**  **{**  **sleep(3);**  **system("/bin/ps");**  **}**  **}** |