# 实验四 进程的创建和简单控制

**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：**

**(一)系统调用**

系统功能调用（system call）是操作系统提供给程序设计人员的一种服务。程序设计人员在编写程序时，可以利用系统调用来请求操作系统的服务。

**(二)本实验涉及的系统调用**

1. 创建一个新进程：pid\_t fork( void);

函数说明：

pid\_t是一个[宏定义](https://baike.baidu.com/item/%E5%AE%8F%E5%AE%9A%E4%B9%89" \t "_blank)，其实质是int，被定义在#include<sys/types.h>中。

系统调用fork用于创建一个新进程。调用者称为父进程，生成的新进程称为子进程。创建新进程后，父子两个进程将执行fork（）系统调用之后的下一条指令。子进程使用相同的PC（程序计数器），相同的CPU寄存器，以及相同的打开文件。

fork调用的一个奇妙之处就是它仅仅被调用一次，却能够返回两次，它可能有三种不同的返回值：

1）在父进程中，fork返回新创建子进程的进程ID；

2）在子进程中，fork返回0；

3）如果出现错误，fork返回一个负值。

需要的头文件：

1）#include<unistd.h>

2）#include<sys/types.h>

1. 获取进程标识号：pid\_t getpid(void); pid\_t getpid(void);

函数说明：

系统调用getpid（）用来取得当前进程的进程ID，系统调用getppid（）用来取得当前进程的父进程ID。

返回值：

getpid（）返回当前进程的进程ID；getppid（）返回当前进程的父进程ID。

1. 等待子进程终止：pid\_t wait(int \*status);

函数说明：

系统调用wait( )用于使父进程阻塞，直到一个子进程结束或者该进程接收到了一个指定的信号为止。如果该父进程没有子进程或者它的子进程已经结束，则wait( )就会立即返回。

返回值：

成功返回已运行结束的子进程号；失败返回-1。

需要的头文件：

1）#include<sys/wait.h>

2）#include<sys/types.h>

1. 终止进程：void exit(int status);

函数说明：

使调用本函数的进程正常终止，然后把形参的值status&0377（八进制）返回给父进程，父进程可以通过wait函数族来获取这个返回值。

exit(status)函数执行之后，形参的值会被传递到父进程，这时有3种情况：

1）如果父进程设置了SA\_NOCLDWAIT标志，或者 把SIGCHLD信号的处理函数设置为SIG\_IGN，那么子进程的返回值被丢弃，子进程立即消亡；

2）如果父进程在等待子进程，他将被通知子进程的退出状态；子进程立即消亡；

3）如果父进程没有被设置为“忽略子进程的退出值”，这时父进程应当使用wait或者waitpid等待子进程的结束，如果父进程不等待，那么子进程结束之后会变成僵尸进程。

**(三)孤儿进程和僵尸进程**

孤儿进程：一个父进程退出，而它的一个或多个子进程还在运行，那么那些子进程将成为孤儿进程。孤儿进程将被init/systemd进程(进程号为1)所收养，并由init进程对它们完成状态收集工作。孤儿进程不会浪费资源。

僵尸进程：一个进程使用fork创建子进程，如果子进程退出，而父进程并没有调用wait或waitpid获取子进程的状态信息，那么子进程的进程描述符仍然保存在系统中。这种进程称之为僵尸进程。僵尸进程浪费系统资源（进程描述符task\_struct存在）。

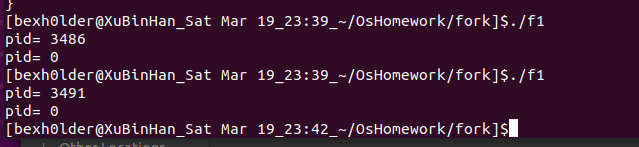
**(四)例程，使用fork( )创建进程**

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);  } |

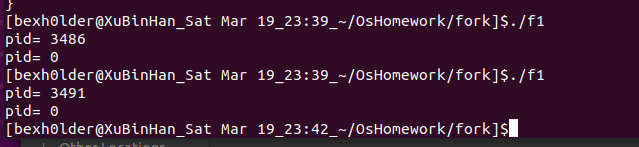
编译链接通过后，多次运行例程，观察进程并发执行结果，并思考下述问题：

1. 为什么会有两行输出？理解fork（）的作用；



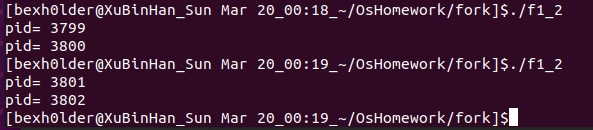
调用fork函数后原本进程会生成一个子进程如何从父进程同样执行之后的代码，所以执行了两次printf函数

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；



每个进程有唯一的PID编号。它是进程运行时系统随机分配的，并不代表专门的进程。在运行时PID是不会改变标识符的，但是你终止程序后再运行PID标识符就会被系统回收，就可能会被继续分配给新运行的程序。

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。



父进程调用fork()函数返回了子进程的pid，子进程调用fork()函数返回0且不生成子进程

调用getpid()函数会返回当前进程的pid

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？

第一个fork()生成的子进程输出b，第二个fork()生成的子进程输出c，父进程输出a

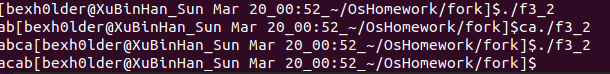
1. ./f1 运行结果为什么不一样？每种结果的产生原因。

进程争取cpu资源,争取到就先输出.

不同的争取,输出不同结果

1. 删除语句➀或➁，观察输出的内容，体会fork的使用。

删除了➁



fork函数作用生成子进程重复执行父进程之后代码

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

环境变量设置的运行格式

使命令能在任何地方执行

1. . 和 .. 什么含义？理解Linux当前目录和父目录的概念。

在Linux中.代表你的当前目录即"pwd"命令所返回的目录，

在Linux中..代表你的当前目录的上一级目录，及当前目录的父目录(即上一级目录)

1. shell 提示为什么不换行，而是紧接着输出内容显示？

Shell是执行进程的父进程，和子进程同时进行，输出的是已经到缓冲区的数据

1. 输出字母为什么和提示交错？

父子进程同时执行，进程争取cpu资源,争取到就先输出.

不同的争取,输出不同结果

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

运行f1进程并将命令以树状图显示，并且只显示与f1有关的进程

管理命令的输出说明：操作符是：”|”,它只能处理经由前面一个指令传出的正 确输出信息，对错误信息信息没有直接处理能力。然后，传递给下一个命令， 作为标准的输入

1. 第8问中有时组合命令没有输出，请分析原因？

在非交互模式下，大多数的UNIX命令行程序都会缓冲它们的输出，这就意味 着程序会缓冲一定数量（通常是4kilobytes）的字符再进行输出，而不是直接 输出它的每个字符

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。

使用重定向输出的方法将输出保存到文件中

./f1 > f1.txt

|  |  |
| --- | --- |
| command > file | 将输出重定向到 file。 |
| command < file | 将输入重定向到 file。 |
| command >> file | 将输出以追加的方式重定向到 file。 |
| n > file | 将文件描述符为 n 的文件重定向到 file。 |
| n >> file | 将文件描述符为 n 的文件以追加的方式重定向到 file。 |
| n >& m | 将输出文件 m 和 n 合并。 |
| n <& m | 将输入文件 m 和 n 合并。 |
| << tag | 将开始标记 tag 和结束标记 tag 之间的内容作为输入。 |

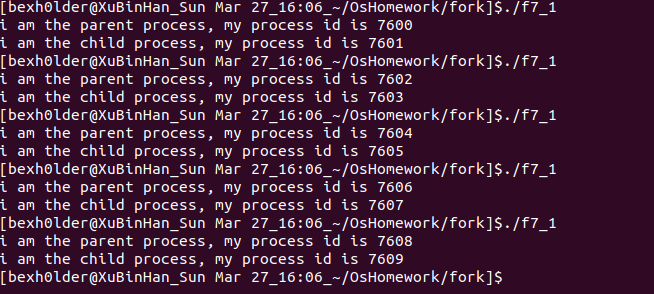
**(五)例程，使用getpid（）和getppid（）查看进程号。**

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：

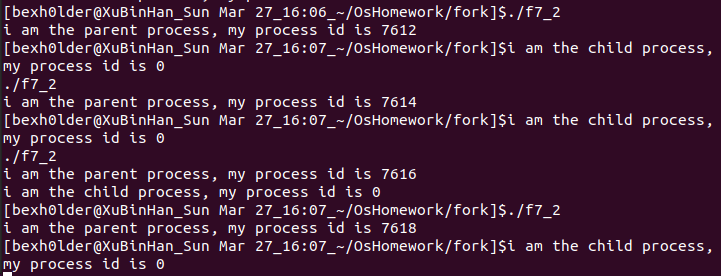
1. 请分析父子进程输出内容交替的原因。



进程争取cpu资源,争取到就先输出.

不同的争取,输出不同结果

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。



每个进程有唯一的PID编号。它是进程运行时系统随机分配的，并不代表专门的进程。在运行时PID是不会改变标识符的，但是你终止程序后再运行PID标识符就会被系统回收，就可能会被继续分配给新运行的程序。

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。

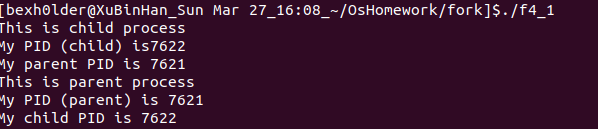
孤儿进程：在父进程完成或终止后仍在继续运行的子进程

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；

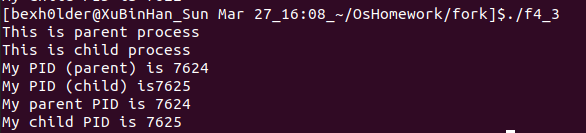


父进程调用wait函数后，会在子进程结束之前一直阻塞，所以父进程会等子进程全部输出完之后再输出

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；

使线程挂起，此时若有其他进程可执行其他进程

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。



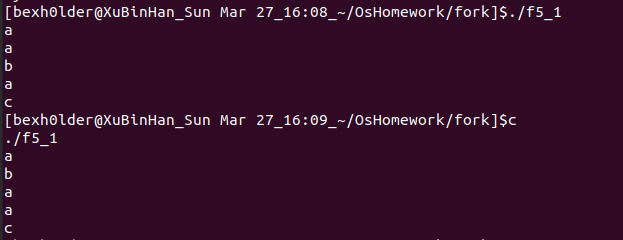
父进程调用wait函数后，会在子进程结束之前一直阻塞，所以父进程会等子进程全部输出完之后再输出

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。



产生了4个进程

1个父进程

2个子进程

1个孙子进程

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？

输出5个字符

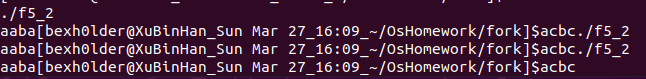
父进程输出两个a

由a\_pid=fork()生成的子进程输出一个b和一个c

由b\_pid=fork()生成的子进程输出一个c

由b\_pid=fork()生成的孙子进程输出一个c

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。



printf()的缓冲区的特点：当为标准输出(STDOUT)时，为行缓冲(遇\n打印一句)；当重定向(非标准输出)后，为全缓冲(缓冲区满打印，一般缓冲区不是几个字符能充满的)。

子进程输出的时候会带上父进程在缓冲区一起输出

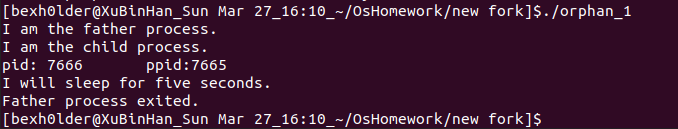
**(六)例程，观察僵尸进程和孤儿进程。**

1. 编辑运行下述程序，观察孤儿进程：

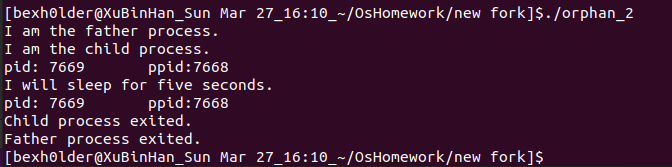
|  |
| --- |
| #include <stdio.h>#include <stdlib.h>#include <errno.h>#include <unistd.h>  int main(){ pid\_t pid;  pid = fork(); //创建一个进程 if (pid == 0) //子进程 { printf("child:I am the child process.\n"); printf("child:pid: %d\tppid:%d\n", getpid(), getppid());//输出进程ID和父进程ID printf("child:I will sleep 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；



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。



在子进程休眠的时间内，父进程执行完成并结束，使子进程成为孤儿进程，之后孤儿进程被bash程序”认养”，由bash程序结束子进程

1. 编辑运行下述例程，观察孤儿进程：

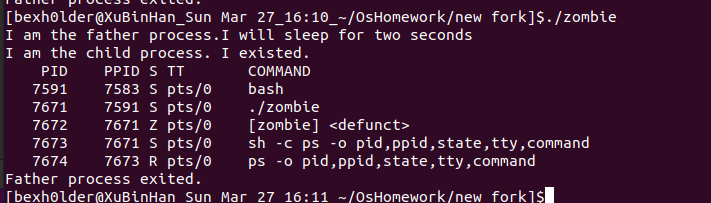
|  |
| --- |
| #include <stdio.h>#include <unistd.h>#include <errno.h>#include <stdlib.h>  int main(){ pid\_t pid; pid = fork(); if (pid == 0) { printf("I am the child process. I existed.\n");  exit(0); } printf("I am the father process.I will sleep for two seconds\n"); //等待子进程先退出 sleep(2); //输出进程信息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

调用system函数会生成一个新的子进程执行system()中的命令

1. 观察输出内容，查看僵尸进程。



子进程退出并没有调用wait或waitpid获取子进程的状态信息，那么子进程的进程描述符仍然保存在系统中。这种进程称之为僵尸进程

**(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。
6. #include <stdio.h>
7. #include <unistd.h>
8. #include <errno.h>
9. #include <stdlib.h>
10. #include <sys/types.h>
11. #include <sys/wait.h>
12. #include <sys/stat.h>
13. #include <fcntl.h>
14. int main()
15. {
16. pid\_t pid;
17. pid = fork();
18. if (pid < 0)
19. {
20. perror("fork error:");
21. exit(1);
22. }
23. else if (pid == 0)
24. {
25. printf("I am the child process.My pid is %d,My ppid is %d\n",getpid(),getppid());
26. for(int i = 0; i < 5; ++i){
27. system("/bin/ps");
28. sleep(3);
29. }
30. }
31. else
32. {
33. printf("I am the parent process.My pid is %d,My ppid is %d\n",getpid(),getppid());
34. for(int i = 0; i < 5; ++i){
35. system("/bin/ps");
36. sleep(3);
37. }
38. wait(0);
39. exit(0);
40. }
41. }

