

# Linux操作系统》实验2：进程及线程创建

---

网安1901      李毅      201904080136

## 一、实验目的

理解创建子进程函数的fork()的用法，通过观察运行结果理解进程的基本特征；通过代码及运行结果理解线程的概念，能够理解进程与线程之间的关联。

## 二、实验方法

本次实验属于验证型实验，按照实验内容的指导完成所有实验步骤，并记录下实验结果，遇到不懂的问题或是在某一步骤上卡壳，先尝试在搜索引擎上寻找解决方法，积极与老师、同学沟通，务必亲自将实验完成。

## 三、实验内容

1. 使用编辑器gedit新建一个helloProcess.c源文件，并输入后面的范例代码。

```
#include <stdio.h>
```

```
#include <sys/types.h>
```

```
#include <unistd.h>
```

```
int main()

{

    //pid_t是数据类型，实际上是一个整型，通过
    typedef重新定义了一个名字，用于存储进程id

    pid_t pid,cid;

    //getpid()函数返回当前进程的id号

    printf("Before fork Process id :%d\n", getpid());
```

```
/*
```

fork()函数用于创建一个新的进程，该进程为当前进程的子进程，创建的方法是：将当前进程的内存内容完整拷贝一份到内存的另一个区域，两个进程为父子关系，他们会同时（并发）执行fork()语句后面的所有语句。

fork()的返回值：

如果成功创建子进程，对于父子进程fork会返回不同的值，对于父进程它的返回值是子进程的进程id值，对于子进程它的返回值是0.

如果创建失败，返回值为-1.

```
*/
```

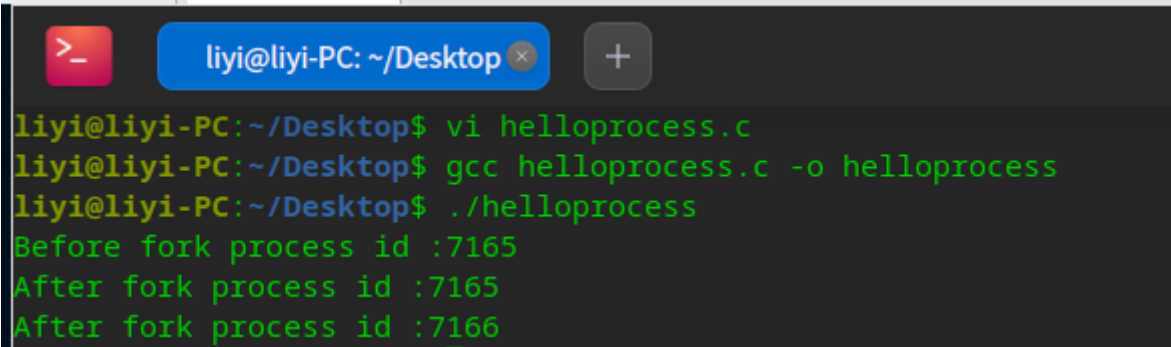
```
cid = fork();
```

```
printf("After fork, Process id :%d\n", getpid());
```

```
return 0;
```

```
}
```

保存退出gedit，使用gcc对源文件进行编译，然后运行，观察结果并解释原因。



```
liyi@liyi-PC: ~/Desktop
liyi@liyi-PC:~/Desktop$ vi helloprocess.c
liyi@liyi-PC:~/Desktop$ gcc helloprocess.c -o helloprocess
liyi@liyi-PC:~/Desktop$ ./helloprocess
Before fork process id :7165
After fork process id :7165
After fork process id :7166
```

2.练习ps命令，该命令可以列出系统中当前运行的进程状态，我们在上面代码的21行处加入下面两行语句，目的是让父子进程暂停下来，否则我们无法观测到他们运行时的状态。

```
int i;
```

```
scanf("%d",&i);
```

```
liyi@liyi-PC: ~/Desktop
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>
int main ()
{
    pid_t pid,cid;
    printf("Before fork process id :%d\n", getpid());
    cid=fork();
    printf("After fork process id :%d\n",getpid());
    int i;
    scanf("%d",&i);
    return 0;
}
```

重新编译运行程序，开启一个新的终端窗口输入下面的命令并观察运行结果。

ps -al

```
liyi@liyi-PC: ~/Desktop
liyi@liyi-PC: ~/Desktop$ ps -al
F S  UID    PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000    8255   7060  0  80   0 -   569 wait_w pts/0    00:00:00 helloprocess
1 S  1000    8256   8255  0  80   0 -   569 n_tty_ pts/0    00:00:00 helloprocess
0 R  1000    8260   8257  0  80   0 -  2896 -      pts/1    00:00:00 ps
liyi@liyi-PC: ~/Desktop$
```

原因：进程因为scanf调用等待外源输入进入waiting状态

3.通过判断fork的返回值让父子进程执行不同的语句。

```
#include <stdio.h>
```

```
#include <sys/types.h>
```

```
#include <unistd.h>
```

```
int main()
```

```
{
```

```
    pid_t cid;
```

```
    printf("Before fork process id :%d\n", getpid());
```

```
    cid = fork();
```

```
    if(cid == 0){ //该分支是子进程执行的代码
```

```
        printf("Child process id (my parent pid is  
%d):%d\n", getppid(),getpid());
```

```
        for(int i=0; i<3 ; i++)
```

```
            printf("hello\n");
```

```

}else{ //该分支是父进程执行的代码

    printf("Parent process id :%d\n", getpid());

    for(int i=0; i<3 ; i++)

        printf("world\n");

}

return 0;

}

```

重新编译观察结果，重点观察父子进程是否判断正确（通过比较进程id）。父子进程其实是**并发**执行的，但实验结果好像是顺序执行的，多执行几遍看看有无变化，如果没有变化试着将两个循环的次数调整高一些，比如30、300，然后再观察运行结果并解释原因。

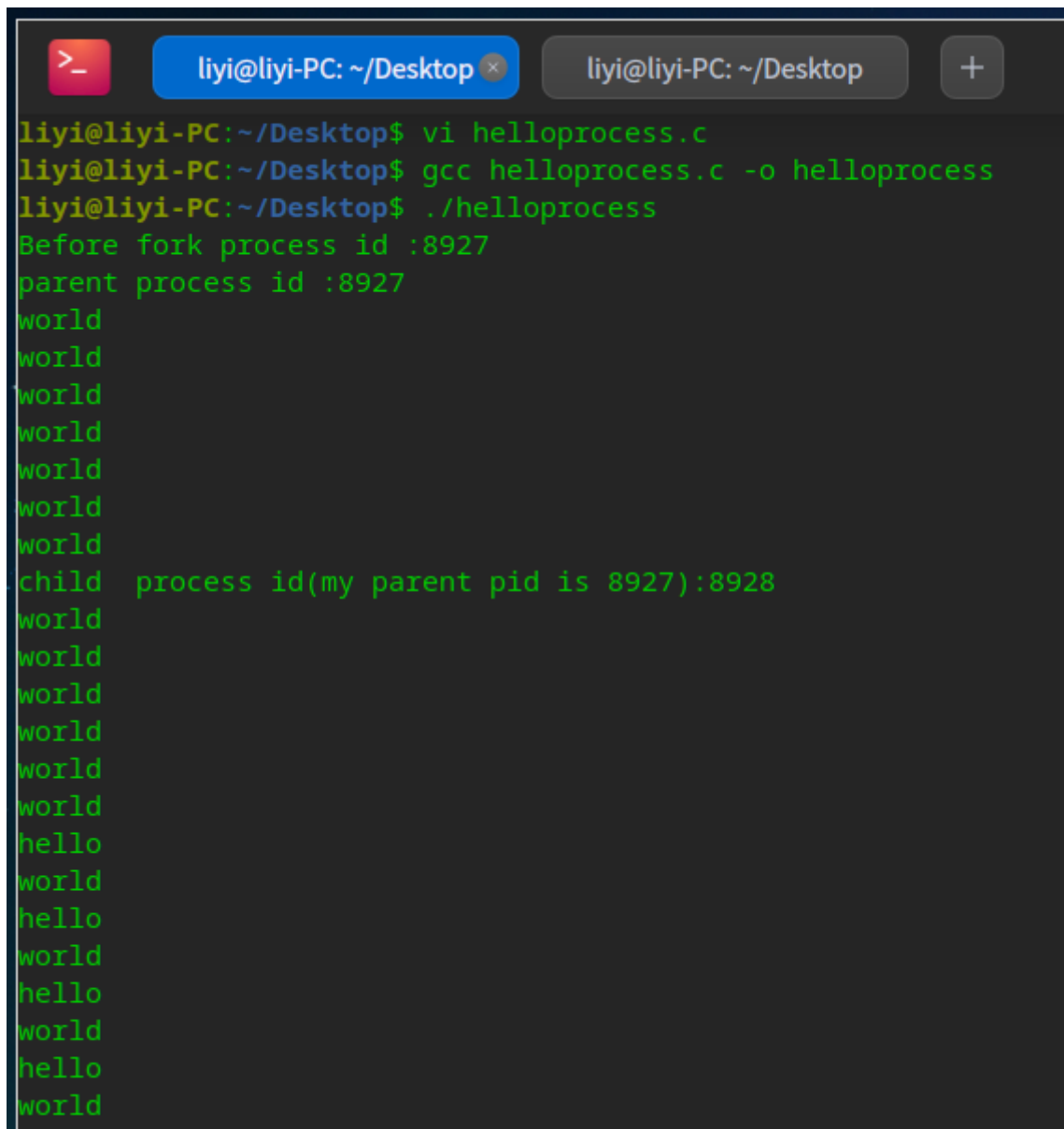
Created with Raphaël 2.1.2  
 Parent process  
 Gocid=fork()  
 cid==0? Child process code  
 Parent process Return  
 Parent process code  
 yesno

上图解释了fork的工作流程，请大家参照代码仔细理解。

循环3次

```
liyi@liyi-PC:~/Desktop$ vi helloprocess.c
liyi@liyi-PC:~/Desktop$ gcc helloprocess.c -o helloprocess
liyi@liyi-PC:~/Desktop$ ./helloprocess
Before fork process id :8796
parent process id :8796
world
world
world
child process id(my parent pid is 8796):8797
hello
hello
hello
```

30次



```
liyi@liyi-PC: ~/Desktop
liyi@liyi-PC:~/Desktop$ vi helloprocess.c
liyi@liyi-PC:~/Desktop$ gcc helloprocess.c -o helloprocess
liyi@liyi-PC:~/Desktop$ ./helloprocess
Before fork process id :8927
parent process id :8927
world
world
world
world
world
world
world
child process id(my parent pid is 8927):8928
world
world
world
world
world
world
world
hello
world
hello
world
hello
world
hello
world
```

答:

fork()返回值在父子进程中是不同的

30次时候hello和World已经交替出现证明其实并发进行的

4.验证父子进程间的内存空间是相互独立的。在终端中进入自己的主目录，使用gedit命令新建一文件helloProcess2.c，输入下面的代码，然后编译运行，解释其原因。

```
#include <stdio.h>
```

```
#include <sys/types.h>
```

```
#include <unistd.h>
```

```
int main()
```

```
{
```

```
    pid_t cid;
```

```
    int x = 100;
```

```
    cid = fork();
```

```
    if(cid == 0){ //该分支是子进程执行的代码
```



```
x++;

printf("In child: x=%d\n",x);


}else{ //该分支是父进程执行的代码

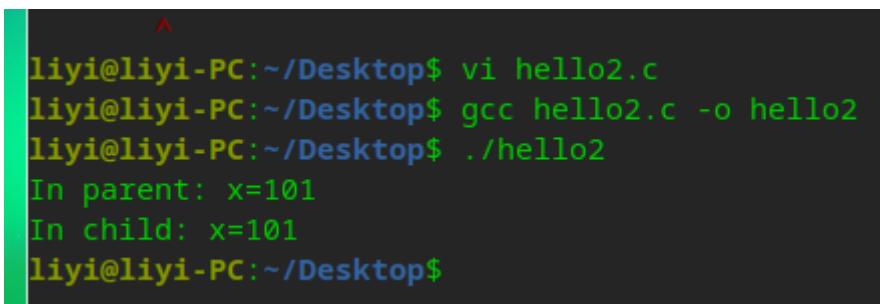
    x++;

    printf("In parent: x=%d\n",x);

}

return 0;

}
```



```
liyi@liyi-PC:~/Desktop$ vi hello2.c
liyi@liyi-PC:~/Desktop$ gcc hello2.c -o hello2
liyi@liyi-PC:~/Desktop$ ./hello2
In parent: x=101
In child: x=101
liyi@liyi-PC:~/Desktop$
```

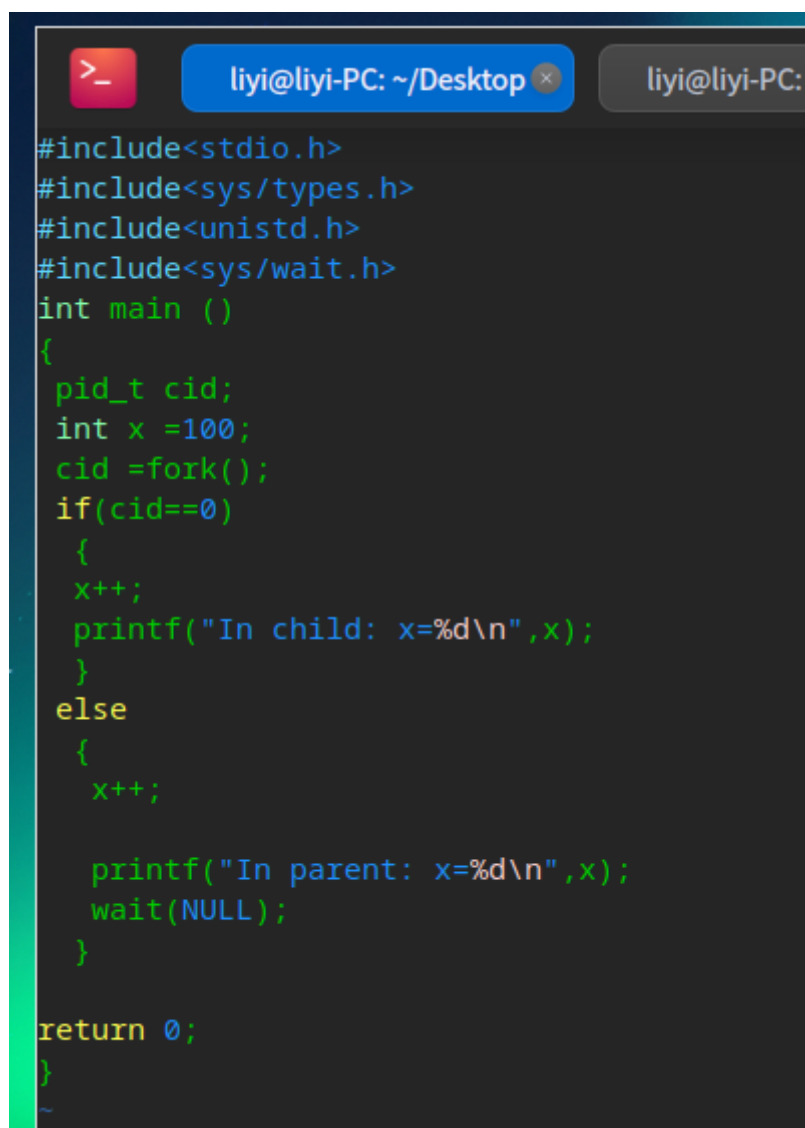
答:

fork是完全复制的但是父子两个都拥有独立的空间互相不影响彼此

5.在上一步的代码的20行添加如下语句，同时代码最顶端要包含一个新的头文件

```
#include <sys/wait.h>
```

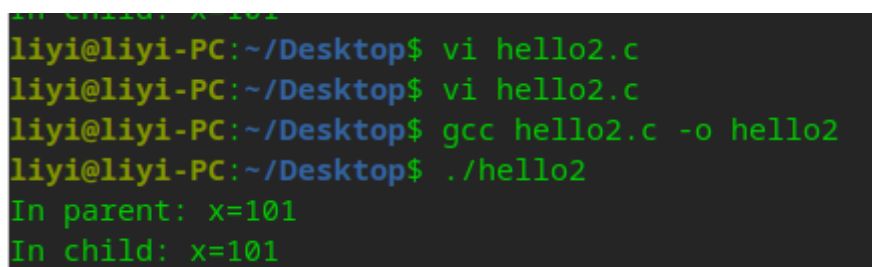
```
wait(NULL);
```

A screenshot of a code editor window with a dark background. The window has a title bar with a red icon, a blue tab labeled 'liyi@liyi-PC: ~/Desktop', and a grey tab labeled 'liyi@liyi-PC:'. The code is written in C and includes headers for stdio, sys/types, unistd, and sys/wait. It defines a main function that forks a child process. Both parent and child increment a variable x and print its value. The parent then calls wait(NULL) to wait for the child to finish. The code ends with a return 0 statement.

```
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>
#include<sys/wait.h>
int main ()
{
    pid_t cid;
    int x =100;
    cid =fork();
    if(cid==0)
    {
        x++;
        printf("In child: x=%d\n",x);
    }
    else
    {
        x++;

        printf("In parent: x=%d\n",x);
        wait(NULL);
    }

    return 0;
}
```

A screenshot of a terminal window with a dark background. It shows the commands used to compile and run the C program. The output shows the parent process printing 'In parent: x=101' and the child process printing 'In child: x=101'.

```
liyi@liyi-PC:~/Desktop$ vi hello2.c
liyi@liyi-PC:~/Desktop$ vi hello2.c
liyi@liyi-PC:~/Desktop$ gcc hello2.c -o hello2
liyi@liyi-PC:~/Desktop$ ./hello2
In parent: x=101
In child: x=101
```

wait函数会让调用者陷入等待，直到子进程的状态变为可用（即子进程结束前父进程一直处于等待状态）。

为了让效果更清楚，请将wait语句从20行移到18行，并在15行加上如下语句：

```
sleep(3);
```

sleep该函数可以让调用进程睡上指定的时间长度（单位是second）。

重新编译代码运行，我们特意让子进程输出完毕后睡了3秒，在这期间父进程什么事也没有做一直在wait，直到子进程结束后父进程才执行printf语句。

```
liyi@liyi-PC:~/Desktop$ vi hello2.c
liyi@liyi-PC:~/Desktop$ gcc hello2.c -o hello2
liyi@liyi-PC:~/Desktop$ ./hello2
In child: x=101
In parent: x=101
liyi@liyi-PC:~/Desktop$
```

6.创建线程。先关闭先前的文件，gedit helloThread.c以创建一个新的C语言源文件，将下面的代码拷贝进编辑器。

```
#include <sys/types.h>
```

```
#include <unistd.h>
```

```
#include <stdio.h>
```

```
#include <pthread.h>
```

```
void* threadFunc(void* arg){ //线程函数
```

```
    printf("In NEW thread\n");
```

```
}
```

```
int main()
```

```
{
```

```
    pthread_t tid;
```

```
    pthread_create(&tid, NULL, threadFunc, NULL);
```

```
    //pthread_join(tid, NULL);
```

```
    printf("In main thread\n");
```

```
    return 0;
}
```

编译该段代码时，请注意gcc要加入新的参数，命令如下：

```
gcc helloThread.c -o helloThread -l pthread
```

运行一下观察到什么现象了？

```
liyi@liyi-PC:~/Desktop$ vi hellothread.c
liyi@liyi-PC:~/Desktop$ gcc hellothread.c -o hellothread -l pthread
/usr/bin/ld: 找不到 -lpthrea
collect2: error: ld returned 1 exit status
liyi@liyi-PC:~/Desktop$ gcc hellothread.c -o hellothread -l pthread
liyi@liyi-PC:~/Desktop$ ./hellothread
In main thread
liyi@liyi-PC:~/Desktop$
```

现象：成功执行、

将上面第18行代码的注释去掉又观察到了什么现象？为什么？

```
liyi@liyi-PC:~/Desktop$ vi hellothread.c
liyi@liyi-PC:~/Desktop$ gcc hellothread.c -o hellothread -l pthread
liyi@liyi-PC:~/Desktop$ ./hellothread
In New therad
In main thread
liyi@liyi-PC:~/Desktop$
```

答：

主线程运行完直接结束

试着在主线程和新线程里加入循环输出，观察一下输出的效果和并发父子进程的执行效果是否相似。

```
liyi@liyi-PC: ~/Desktop
#include<sys/types.h>
#include<unistd.h>
#include<stdio.h>
#include<pthread.h>
void* threadFunc(void* arg){
    printf("In New therad\n");
    for(int i=0;i<3;i++)
        printf("hello\n");
}
int main()
{
    pthread_t tid;
    pthread_create(&tid, NULL, threadFunc, NULL);
    pthread_join(tid,NULL);
    printf("In main thread\n");
    for(int i=0;i<3;i++)
        printf("world\n");

    return 0;
}
```

```
liyi@liyi-PC:~/Desktop$ vi hellothread.c
liyi@liyi-PC:~/Desktop$ gcc hellothread.c -o hellothread -l pthread
liyi@liyi-PC:~/Desktop$ ./hellothread
In New therad
hello
hello
hello
In main thread
world
world
world
liyi@liyi-PC:~/Desktop$
```

答：不相似 此不是交替进行

## 四、实验报告

无