

# 《linux操作系统》实验2：进程及线程创建

## 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。

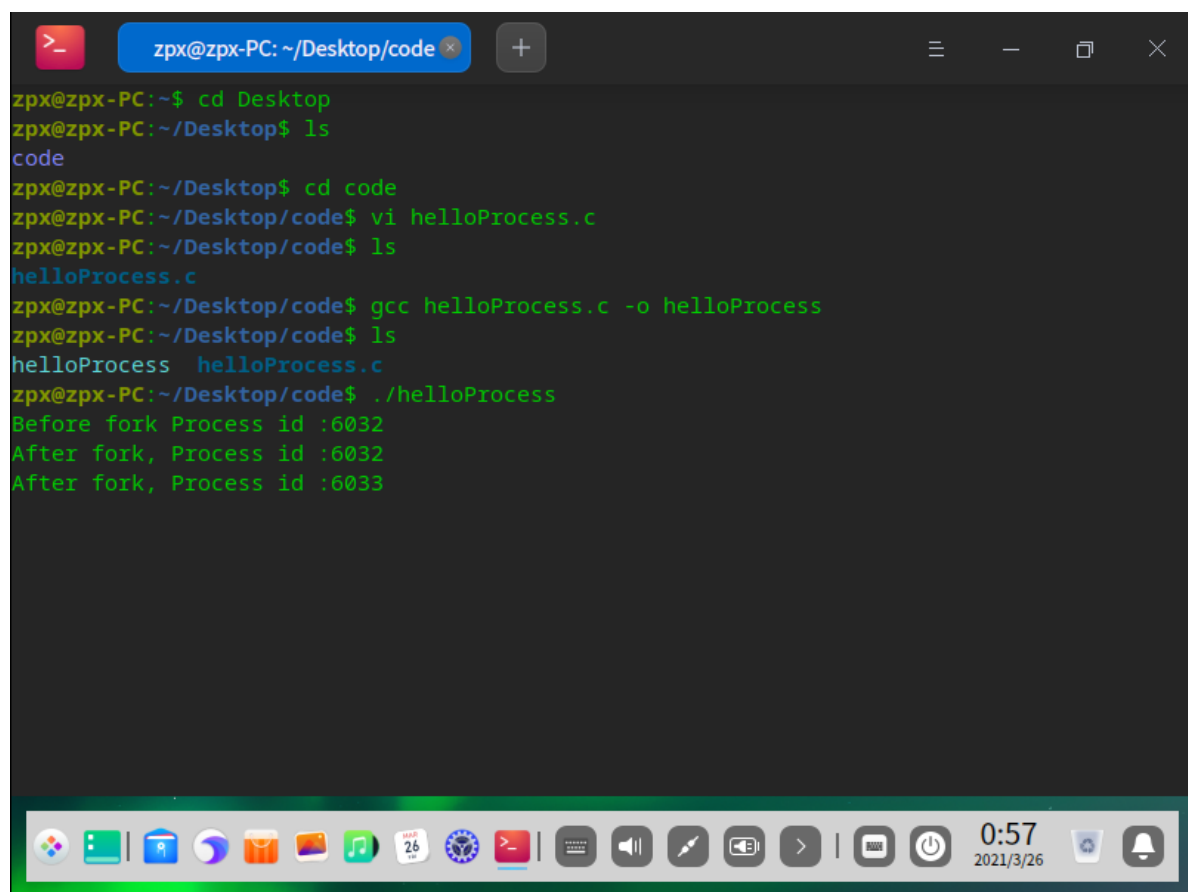
## 二、实验方法

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。

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。



```
zpx@zpx-PC: ~/Desktop/code
zpx@zpx-PC:~$ cd Desktop
zpx@zpx-PC:~/Desktop$ ls
code
zpx@zpx-PC:~/Desktop$ cd code
zpx@zpx-PC:~/Desktop/code$ vi helloProcess.c
zpx@zpx-PC:~/Desktop/code$ ls
helloProcess.c
zpx@zpx-PC:~/Desktop/code$ gcc helloProcess.c -o helloProcess
zpx@zpx-PC:~/Desktop/code$ ls
helloProcess  helloProcess.c
zpx@zpx-PC:~/Desktop/code$ ./helloProcess
Before fork Process id :6032
After fork, Process id :6032
After fork, Process id :6033
```

结果显示：父进程为6032，子进程6033，创建后子进程从cid=fork()处开始执行，打印了自己的pid。

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。

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。

```
zpx@zpx-PC:~$ ps -al
F S      UID      PID  PPID  C  PRI  NI ADDR  SZ  WCHAN  TTY          TIME CMD
0 S      1000     6032   5557  0   80   0  -    570  ia32_s pts/0        00:00:00 helloProcess
1 S      1000     6033   6032  0   80   0  -    570  ia32_s pts/0        00:00:00 helloProcess
0 R      1000     6377   6365  0   80   0  -   2896  -      pts/1        00:00:00 ps
zpx@zpx-PC:~$
```

3.通过判断fork的返回值让父子进程执行不同的语句。

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。

运行3次



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

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：

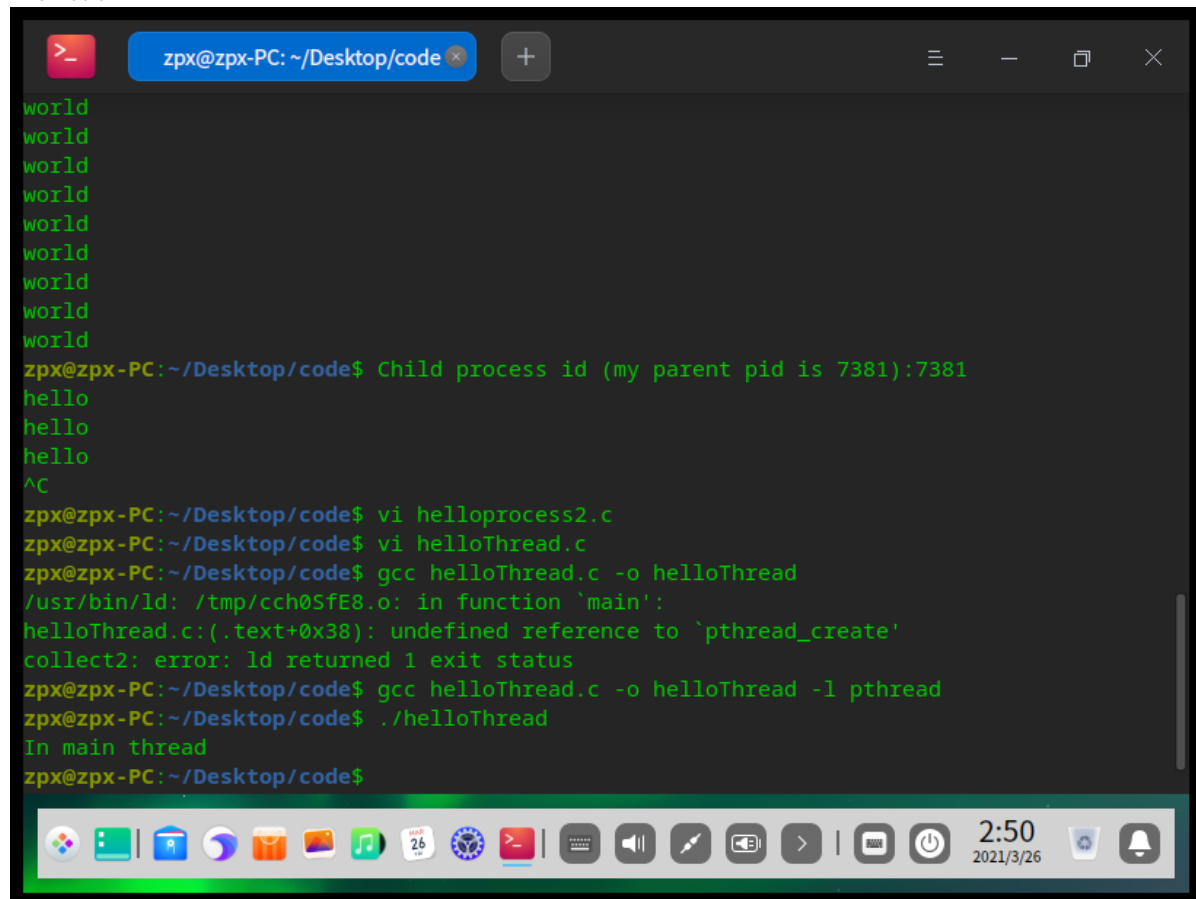
sleep该函数可以让调用进程睡上指定的时间长度（单位是second）。

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。

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。

编译该段代码时，请注意gcc要加入新的参数，命令如下：

运行结果



```
zpx@zpx-PC: ~/Desktop/code
world
world
world
world
world
world
world
world
world
world
world
zpx@zpx-PC:~/Desktop/code$ Child process id (my parent pid is 7381):7381
hello
hello
hello
^C
zpx@zpx-PC:~/Desktop/code$ vi helloprocess2.c
zpx@zpx-PC:~/Desktop/code$ vi helloThread.c
zpx@zpx-PC:~/Desktop/code$ gcc helloThread.c -o helloThread
/usr/bin/ld: /tmp/cch0SfE8.o: in function `main':
helloThread.c:(.text+0x38): undefined reference to `pthread_create'
collect2: error: ld returned 1 exit status
zpx@zpx-PC:~/Desktop/code$ gcc helloThread.c -o helloThread -l pthread
zpx@zpx-PC:~/Desktop/code$ ./helloThread
In main thread
zpx@zpx-PC:~/Desktop/code$
```

去掉注释

```
zpx@zpx-PC: ~/Desktop/code
world
world
zpx@zpx-PC:~/Desktop/code$ Child process id (my parent pid is 7381):7381
hello
hello
hello
^C
zpx@zpx-PC:~/Desktop/code$ vi helloprocess2.c
zpx@zpx-PC:~/Desktop/code$ vi helloThread.c
zpx@zpx-PC:~/Desktop/code$ gcc helloThread.c -o helloThread
/usr/bin/ld: /tmp/cch0SfE8.o: in function 'main':
helloThread.c:(.text+0x38): undefined reference to `pthread_create'
collect2: error: ld returned 1 exit status
zpx@zpx-PC:~/Desktop/code$ gcc helloThread.c -o helloThread -l pthread
zpx@zpx-PC:~/Desktop/code$ ./helloThread
In main thread
zpx@zpx-PC:~/Desktop/code$ vi helloThread.c
zpx@zpx-PC:~/Desktop/code$ gcc helloThread.c -o helloThread -l pthread
zpx@zpx-PC:~/Desktop/code$ ./
bash: ./: 是一个目录
zpx@zpx-PC:~/Desktop/code$ ./helloThread
In NEW thread
In main thread
zpx@zpx-PC:~/Desktop/code$
```

原因：主函数使用了pthread\_create创建了子线程，如果主进程不等待子线程，则进程结束，所有线程也结束，线程依赖进程。

## 四、总结

- 1.fork()函数的使用。
- 2.进程是并发执行的。
- 3.线程是CPU的基本单位，且线程的运行是并发的。
- 4.线程依赖于进程。