

《Linux操作系统》实验2：进程及线程创建

网安1901 201904080105 孙航

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。

```
sun@sun-PC:~/Desktop/1.code$ gcc helloProcess.c -o helloProcess
sun@sun-PC:~/Desktop/1.code$ ./helloProcess
Before fork Process id :13329
After fork, Process id :13329
sun@sun-PC:~/Desktop/1.code$ After fork, Process id :13330
```

理由：由于fork（）使得创建一个子进程，第二个printf运行了两次，分别显示了父进程和子进程的进程ID

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);
```

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。

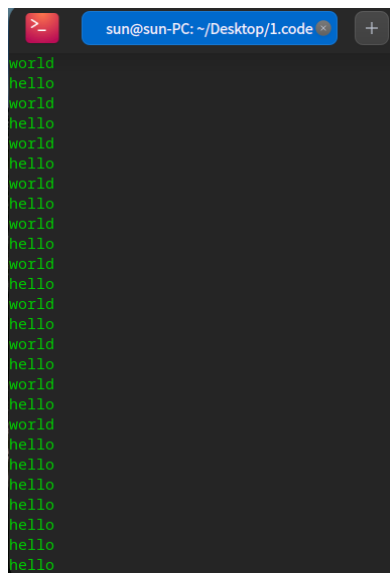
```
ps -al
```

```
sun@sun-PC:~/Desktop/1.code$ gcc helloProcess.c -o helloProcess
sun@sun-PC:~/Desktop/1.code$ ./helloProcess
Before fork Process id :13434
After fork, Process id :13434
After fork, Process id :13435
j
sun@sun-PC:~/Desktop/1.code$ ps -al
F S  UID    PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 R   1000   13439  13190  0  80   0 -  2889 -          pts/1        00:00:00 ps
sun@sun-PC:~/Desktop/1.code$
```

3.通过判断fork的返回值让父子进程执行不同的语句。

```
sun@sun-PC:~/Desktop/1.code$ gcc helloProcess.c -o helloProcess
sun@sun-PC:~/Desktop/1.code$ ./helloProcess
Before fork process id :13568
Parent process id :13568
world
world
world
Child process id (my parent pid is 13568):13569
hello
hello
hello
```

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。

A terminal window titled 'sun@sun-PC: ~/Desktop/1.code' showing the output of the program. The output consists of 15 lines, alternating between 'world' and 'hello'. The sequence is: world, hello, world, hello, world, hello, world, hello, world, hello, world, hello, world, hello, world, hello. This interleaving demonstrates that the parent and child processes are executing concurrently.

原因：在进程中，父子进程并发执行，父进程先执行，子进程再执行。

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。

```
sun@sun-PC:~/Desktop/1.code$ ./helloProcess2
In parent: x=101
In child: x=101
sun@sun-PC:~/Desktop/1.code$
```

父子进程间的内存是相互独立的，父子进程中的局部变量的值不会相互影响

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);
```

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。

5.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：

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

运行一下观察到什么现象了？

```
sun@sun-PC:~/Desktop/1.code$ gcc helloThread.c -o helloThread -l pthread
sun@sun-PC:~/Desktop/1.code$ ./helloThread
In NEW thread
In main thread
```

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？

```
In main thread
sun@sun-PC:~/Desktop/1.code$ gcc helloThread.c -o helloThread1 -l pthread
sun@sun-PC:~/Desktop/1.code$ ./helloThread1
In main thread
```

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。

```
1 #include <sys/types.h>
2 #include <unistd.h>
3 #include <stdio.h>
4 #include <pthread.h>
5
6 void* threadFunc(void* arg){ //线程函数
7     for(int i=0;i<25;i++)
8     {
9         printf("In NEW thread\n");
10    }
11 }
12
13 int main()
14 {
15     pthread_t tid;
16
17     pthread_create(&tid, NULL, threadFunc, NULL);
18
19     for(int i=0;i<25;i++)
20     {
21         printf("In main thread\n");
22     }
23     return 0;
24 }
```

```
In main thread  
In main thread  
In main thread  
In main thread  
In main thread  
In main thread  
In main thread  
In main thread  
In main thread  
In main thread  
In main thread  
In main thread  
In main thread  
In main thread  
In main thread  
In main thread  
In main thread  
In main thread  
In main thread  
In main thread  
In main thread  
In main thread  
In NEW thread  
In NEW thread  
In NEW thread  
In NEW thread  
In NEW thread  
In NEW thread  
In NEW thread  
In NEW thread  
In NEW thread  
In NEW thread  
In NEW thread
```

父子线程的执行和父子进程的执行相同均为并发执行