

# 《linux操作系统》实验2：进程及线程创建

区队：网安1901 姓名：王实 学号：201904080126

## 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。

```
"helloProcess" +
1 #include<stdio.h>
2 #include <sys/types.h>
3 #include <unistd.h>
4
5 int main()
6 {
7     //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
8     pid_t pid,cid;
9     //getpid()函数返回当前进程的id号
10    printf("Before fork Process id :%d\n", getpid());
11
12    /*
13    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。
14    fork()的返回值：
15    如果成功创建子进程，对于父进程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。
16    如果创建失败，返回值为-1。
17    */
18    cid = fork();
19
20    printf("After fork, Process id :%d\n", getpid());
21
22    return 0;
23 }
```

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。

```
wangshi@wangshi-PC: ~/Desktop/tempdir
wangshi@wangshi-PC:~$ cd Desktop/
wangshi@wangshi-PC:~/Desktop$ cd tempdir/
wangshi@wangshi-PC:~/Desktop/tempdir$ vi helloProcess.c
wangshi@wangshi-PC:~/Desktop/tempdir$ gcc helloProcess.c -o helloProcess
wangshi@wangshi-PC:~/Desktop/tempdir$ ls
helloProcess  helloProcess.c  process
wangshi@wangshi-PC:~/Desktop/tempdir$ ./helloProcess
Before fork Process id :7786
After fork, Process id :7786
After fork, Process id :7787
wangshi@wangshi-PC:~/Desktop/tempdir$
```

结果显示：父进程为7786，子进程7787，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。

```
1 int i;
2 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。

```
wangshi@wangshi-PC:~/Desktop/tempdir$ ps -al
F S  UID      PID     PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 R  1000      8211     7993  0  80   0  -  2890 -        pts/0        00:00:00 ps
wangshi@wangshi-PC:~/Desktop/tempdir$
```

3.通过判断fork的返回值让父子进程执行不同的语句。

```
1
2 #include <stdio.h>
3 #include <sys/types.h>
4 #include <unistd.h>
5
6 int main()
7 {
8     pid_t cid;
9     printf("Before fork process id :%d\n", getpid());
10
11     cid = fork();
12
13     if(cid == 0){ //该分支是子进程执行的代码
14
15         printf("Child process id (my parent pid is %d):%d\n", getppid(),getpid());
16         for(int i=0; i<3 ; i++)
17             printf("hello\n");
18     }else{ //该分支是父进程执行的代码
19
20         printf("Parent process id :%d\n", getpid());
21         for(int i=0; i<3 ; i++)
22             printf("world\n");
23     }
24 }
25
26 return 0;
27 }
```

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

```
wangshi@wangshi-PC:~/Desktop/tempdir$ ./fork_instance
Before fork process id :8393
Parent process id :8393
world
world
world
Child process id (my parent pid is 1):8394
hello
hello
hello
wangshi@wangshi-PC:~/Desktop/tempdir$
```

30次

```
hello
hello
world
hello
hello
world
hello
```

原因：父子进程是并发执行的，两个进程交替使用CPU。

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。

```
1 #include <stdio.h>
2 #include <sys/types.h>
3 #include <unistd.h>
4
5 int main()
6 {
7     pid_t cid;
8     int x = 100;
9
10    cid = fork();
11
12    if(cid == 0){ //该分支是子进程执行的代码
13        x++;
14        printf("In child: x=%d\n",x);
15
16    }else{ //该分支是父进程执行的代码
17        x++;
18
19        printf("In parent: x=%d\n",x);
20    }
21
22    return 0;
23 }
24 }
```

```
wangshi@wangshi-PC:~/Desktop/tempdir$ gcc helloProcess2.c -o helloProcess2
wangshi@wangshi-PC:~/Desktop/tempdir$ ./helloProcess2
In parent: x=101
In child: x=101
```

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

```
1 #include <sys/wait.h>
2 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：

```
1 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。

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。

```

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
8     printf("In NEW thread\n");
9
10 }
11
12 int main()
13 {
14     pthread_t tid;
15
16     pthread_create(&tid, NULL, threadFunc, NULL);
17
18     //pthread_join(tid, NULL);
19
20     printf("In main thread\n");
21
22     return 0;
23 }

```

编译该段代码时，请注意gcc要加入新的参数，命令如下：

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

运行结果

```

wangshi@wangshi-PC:~/Desktop/tempdir$ gcc helloThread.c -o helloThread -l pthread
wangshi@wangshi-PC:~/Desktop/tempdir$ ./helloThread
In main thread

```

去掉注释

```

wangshi@wangshi-PC:~/Desktop/tempdir$ gcc helloThread.c -o helloThread -l pthread
wangshi@wangshi-PC:~/Desktop/tempdir$ ./helloThread
In NEW thread
In main thread

```

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。

