

# 实验2：进程及线程创建

班级： 网安1901 学号： 201904080108 姓名： 刘薇

## 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；通过代码及运算结果理解线程的概念，能够理解进程和线程之间的关联。

## 二、实验方法

本次实验属于验证实验，按照实验内容的知道完成所有步骤，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sublime新建一个helloProcess.c源文件，并输入代码。

```
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    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
13    fork()的返回值：
14    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
15    如果创建失败，返回值为-1。
16    */
17    cid = fork();
18    printf("After fork, Process id :%d\n", getpid());
19    return 0; |
20
21 }
```

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。

```
liuwei@liuwei-PC:~/Desktop/code$ gcc helloProcess.c -o helloProcess
liuwei@liuwei-PC:~/Desktop/code$ ./helloProcess
Before fork Process id :24789
After fork, Process id :24789
After fork, Process id :24790
liuwei@liuwei-PC:~/Desktop/code$
```

结果：打印了before fork的进程代码的id，其余的两行After fork的进程代码不同

原因：fork()函数创建了一个新的子进程，父进程和子进程都执行了一遍after fork的getpid(),并发执行fork之后的所有代码。最后一行的为子进程id，第二行为父进程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。

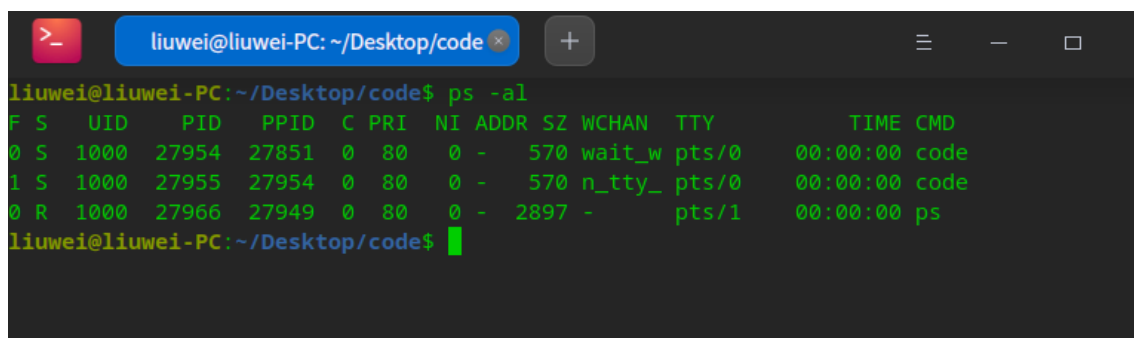
```
int i;
scanf("%d",&i);
```

```

4
5 int main()
6 {
7     //pid_t是数据类型，实际上是一个整型，通过typedef定义
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    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
13    fork()的返回值：
14    如果成功创建子进程，对于父子进程fork会返回不同的值，对
15    如果创建失败，返回值为-1。
16    */
17    cid = fork();
18    printf("After fork, Process id :%d\n", getpid())
19    int i;
20    scanf("%d",&i);
21    return 0;
22
23 }

```

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



```

liuwei@liuwei-PC: ~/Desktop/code$ ps -al
F S  UID    PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S   1000    27954   27851  0  80   0  -   570 wait_w  pts/0        00:00:00 code
1 S   1000    27955   27954  0  80   0  -   570 n_tty_  pts/0        00:00:00 code
0 R   1000    27966   27949  0  80   0  -  2897 -      pts/1        00:00:00 ps
liuwei@liuwei-PC: ~/Desktop/code$

```

### 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。

```

4
5 int main()
6 {
7     pid_t cid;
8     printf("Before fork process id :%d\n", getpid());
9     cid = fork();
10    if(cid == 0){ //该分支是子进程执行的代码
11        printf("Child process id (my parent pid is %d):%d\n", ge
12        for(int i=0; i<3 ; i++)
13            printf("hello\n");
14    }else{ //该分支是父进程执行的代码
15        printf("Parent process id :%d\n", getpid());
16        for(int i=0; i<3 ; i++)
17            printf("world\n");
18    }
19    return 0;
20
21 }

```

3:

```

Before fork process id :28556
Parent process id :28556
world
world
world
Child process id (my parent pid is 28556):28557
hello
hello
hello
liuwei@liuwei-PC:~/Desktop/code$

```

300:

```
world
world
world
world
world
world
world
world
world
world
world
world
world
world
world
world
Child process id (my parent pid is 28745):28746
world
hello
world
hello
world
hello
world
hello
world
hello
world
hello
```

hello和world交替出现

原因：当运行次数少时，父进程会先执行完毕；当运行次数到达300次时，父子进程的printf语句交替执行，可以看出来父子进程并发执行。

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。

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

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

原因：子进程的内存空间是由父进程完全复制过来的，两个进程之间的内存空间是相互独立的，不共享同一个空间。

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(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）。

```
#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){ //该分支是子进程执行的代码
        x++;
        printf("In child: x=%d\n",x);
        sleep(3);
    }else{ //该分支是父进程执行的代码
        x++;
        wait(NULL);
        printf("In parent: x=%d\n",x);
    }
    return 0;
}
```

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 void* threadFunc(void* arg){ //线程函数
6 printf("In NEW thread\n");
7 }
8 int main()
9 {
10 pthread_t tid;
11 pthread_create(&tid, NULL, threadFunc, NULL);
12 //pthread_join(tid, NULL);
13 printf("In main thread\n");
14 return 0;
15 }
```




```
helloThread.c
1 #include <sys/types.h>
2 #include <unistd.h>
3 #include <stdio.h>
4 #include <pthread.h>
5 void* threadFunc(void* arg){ //线程函数
6 printf("In NEW thread\n");
7 }
8 int main()
9 {
10 pthread_t tid;
11 pthread_create(&tid, NULL, threadFunc, NULL);
12 //pthread_join(tid, NULL);
13 printf("In main thread\n");
14 return 0;
15 }
```

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
```

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

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？



A terminal window with a dark background. The prompt is `liuwei@liuwei-PC: ~/Desktop/code$`. The command `./helloThread` has been executed, resulting in two lines of output: `In main thread` and `In NEW thread`. The prompt is now `liuwei@liuwei-PC: ~/Desktop/code$`.

```
liuwei@liuwei-PC: ~/Desktop/code$ ./helloThread
In main thread
In NEW thread
liuwei@liuwei-PC: ~/Desktop/code$
```

原因：因为第二次执行了 `/pthread_join(tid, NULL);` 所以调用这个函数的父进程会等子进程结束后再一起结束，子进程 `printf` 后，父进程再进行 `printf`，子进程里的 `("In NEW thread\n")` 会输出。

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。

[illegible]

均为并发执行

#### 四、总结

无