

实验2：进程及线程创建

班级：网安1901 学号：201904080127 姓名：申雨淇

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。

```
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。
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。

```

shen@ubuntu:~/Desktop/1$ gedit helloprocess.c
shen@ubuntu:~/Desktop/1$ gcc helloprocess.c -o hello
shen@ubuntu:~/Desktop/1$ ./hello
Before fork Process id :5733
After fork, Process id :5733
shen@ubuntu:~/Desktop/1$ After fork, Process id :5734

shen@ubuntu:~/Desktop/1$ ./hello
Before fork Process id :11898
After fork, Process id :11898
shen@ubuntu:~/Desktop/1$ After fork, Process id :11899

shen@ubuntu:~/Desktop/1$

```

原因：fork()函数会创建一个新的进程，该进程为当前进程的子进程，所以创建的子进程的pid为5734，父进程pid为5733（上图我试了两次，取第一次的输出结果）；并且fork会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，于是print语句会在父进程、子进程都执行一次。

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。

```

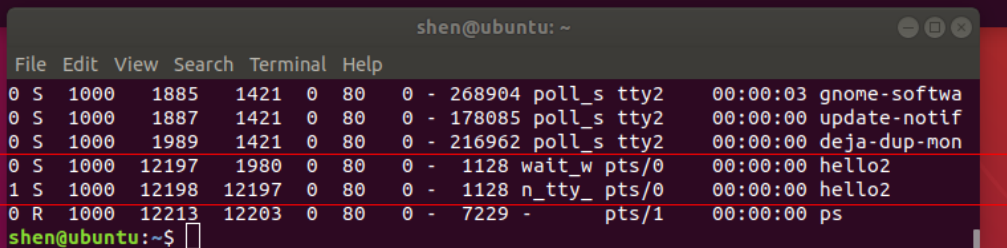
1  ps -al

```

```

shen@ubuntu:~/Desktop/1$ gedit helloprocess.c
shen@ubuntu:~/Desktop/1$ gcc helloprocess.c -o hello2
shen@ubuntu:~/Desktop/1$ ./hello2
Before fork Process id :12197
After fork, Process id :12197
After fork, Process id :12198

```



File	Edit	View	Search	Terminal	Help
0	S	1000	1885	1421	0 80 0 - 268904 poll_s tty2 00:00:03 gnome-softwa
0	S	1000	1887	1421	0 80 0 - 178085 poll_s tty2 00:00:00 update-notif
0	S	1000	1989	1421	0 80 0 - 216962 poll_s tty2 00:00:00 deja-dup-mon
0	S	1000	12197	1980	0 80 0 - 1128 wait_w pts/0 00:00:00 hello2
1	S	1000	12198	12197	0 80 0 - 1128 n_tty_ pts/0 00:00:00 hello2
0	R	1000	12213	12203	0 80 0 - 7229 - pts/1 00:00:00 ps

```

shen@ubuntu:~$

```

3. 通过判断fork的返回值让父子进程执行不同的语句。

```

1  #include <stdio.h>
2  #include <sys/types.h>
3  #include <unistd.h>
4
5  int main()
6  {
7      pid_t cid;
8      printf("Before fork process id :%d\n", getpid());

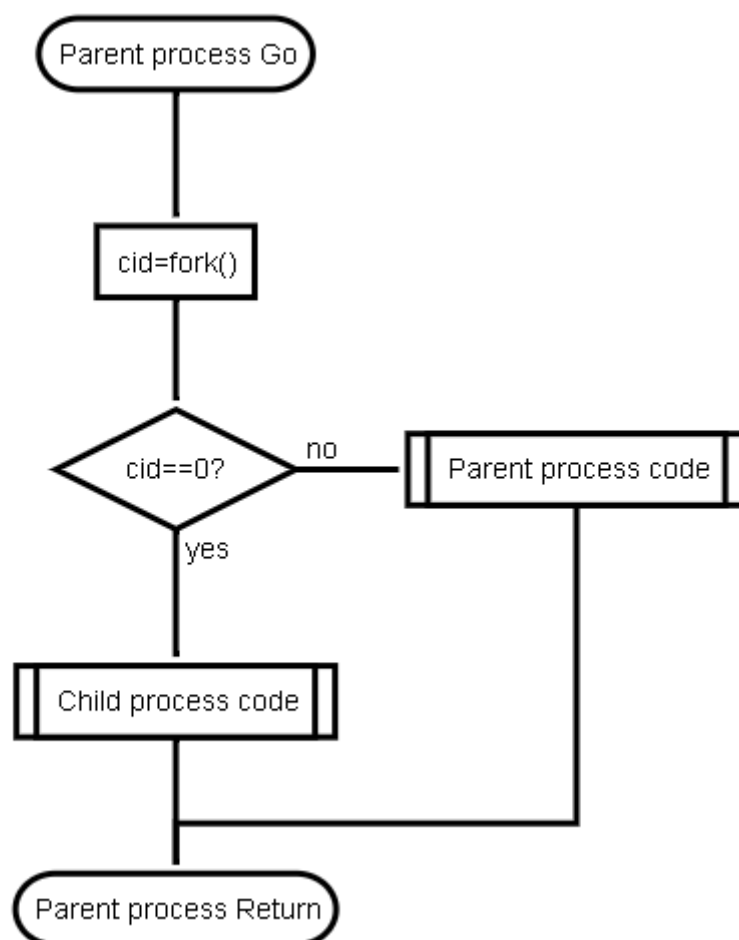
```

```

9
10     cid = fork();
11
12     if(cid == 0){ //该分支是子进程执行的代码
13
14         printf("Child process id (my parent pid is %d):%d\n",
getppid(),getpid());
15         for(int i=0; i<3 ; i++)
16             printf("hello\n");
17
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     return 0;
26 }

```

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。



上图解释了fork的工作流程，请大家参照代码仔细理解。

3次时：


```

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
23    return 0;
24 }

```

```

shen@ubuntu:~/Desktop/1$ gedit helloprocess2.c
shen@ubuntu:~/Desktop/1$ gcc helloprocess2.c -o hp
shen@ubuntu:~/Desktop/1$ ./hp
In parent: x=101
shen@ubuntu:~/Desktop/1$ In child: x=101
shen@ubuntu:~/Desktop/1$

```

原因：fork()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，所以父子进程初始x都为100。而父子进程间的内存空间是相互独立的，所以父子进程分别执行x++，最终x值为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。

```

shen@ubuntu:~/Desktop/1$ gedit helloprocess2.c
shen@ubuntu:~/Desktop/1$ gcc helloprocess2.c -o hp2
shen@ubuntu:~/Desktop/1$ ./hp2
In child: x=101
In parent: x=101
shen@ubuntu:~/Desktop/1$

```

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
```

运行一下观察到什么现象了？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？

```
shen@ubuntu:~/Desktop/1$ gcc hellothread.c -o het -l pthread
shen@ubuntu:~/Desktop/1$ ./het
In main thread
```

第18行代码的注释去掉

```
shen@ubuntu:~/Desktop/1$ gedit hellothread.c
shen@ubuntu:~/Desktop/1$ gcc hellothread.c -o het -l pthread
shen@ubuntu:~/Desktop/1$ ./het
In NEW thread
In main thread
```

原因：两次执行的区别在于是否有"In NEW thread"，即新线程是否得到执行。因为执行了pthread_join(tid,

NULL)，所以调用这个函数的父进程会等待子进程结束后一起结束，等子进程printf后，父进程printf，所以多了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  #include <time.h>
6  #include <stdlib.h>
7
8  int value = 100; //Shared data section belongs to
9
10 void* hello(void* arg){ //线程函数
11     for(int i=0;i<3;++i)
12     {
13         printf( "hello(%d)\n",value++);
14         sleep(1);
15     }
16 }
17
18 void* world(void* arg){ //线程函数
19     for(int i=0;i<3;++i)
20     {
21         printf( "world(%d)\n",value++);
22         sleep(2);
23     }
24 }
25
26 int main( )
27 {
28     //rand(time(NULL));
29     pthread_t tid,tid2;
30
31     //线程创建函数
32     pthread_create(&tid,NULL,hello,NULL);
33     pthread_create(&tid2,NULL,world,NULL);
34
35     //等待指定的线程结束
36     pthread_join(tid,NULL);
37     pthread_join(tid2,NULL);
38     printf("In main thread(%d)\n",value);
39
40     return 0;
41 }
```

```
shen@ubuntu:~/Desktop/1$ gedit hellothread.c
shen@ubuntu:~/Desktop/1$ gcc hellothread.c -o het2 -lpthread
shen@ubuntu:~/Desktop/1$ ./het2
world(100)
hello(101)
hello(102)
world(103)
hello(104)
world(105)
In main thread(106)
```

四、总结

无