

实验2：进程及线程创建

班级：网安1901 学号：201904080123 姓名：黄珏

一、实验目的

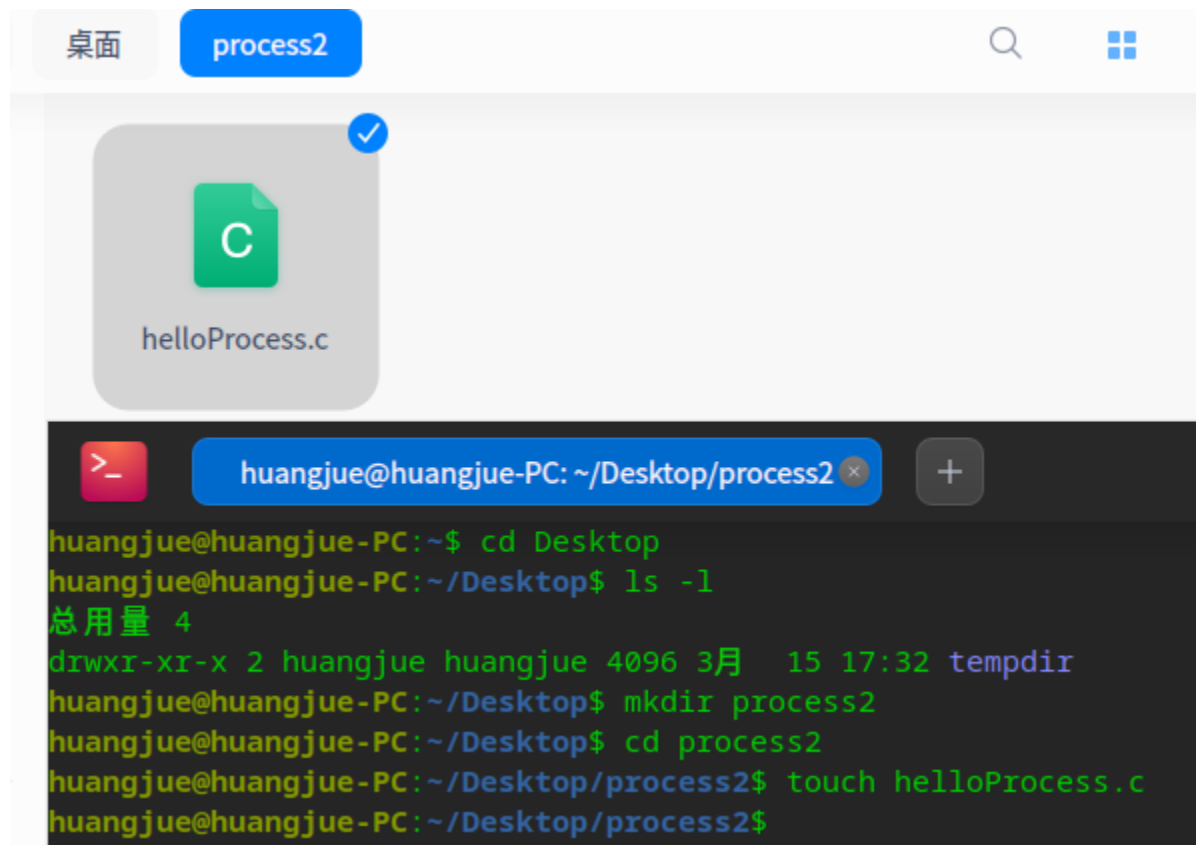
理解创建子进程函数的fork()的用法，通过观察运行结果理解进程的基本特征；通过代码及运行结果理解线程的概念，能够理解进程与线程之间的关联。

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。

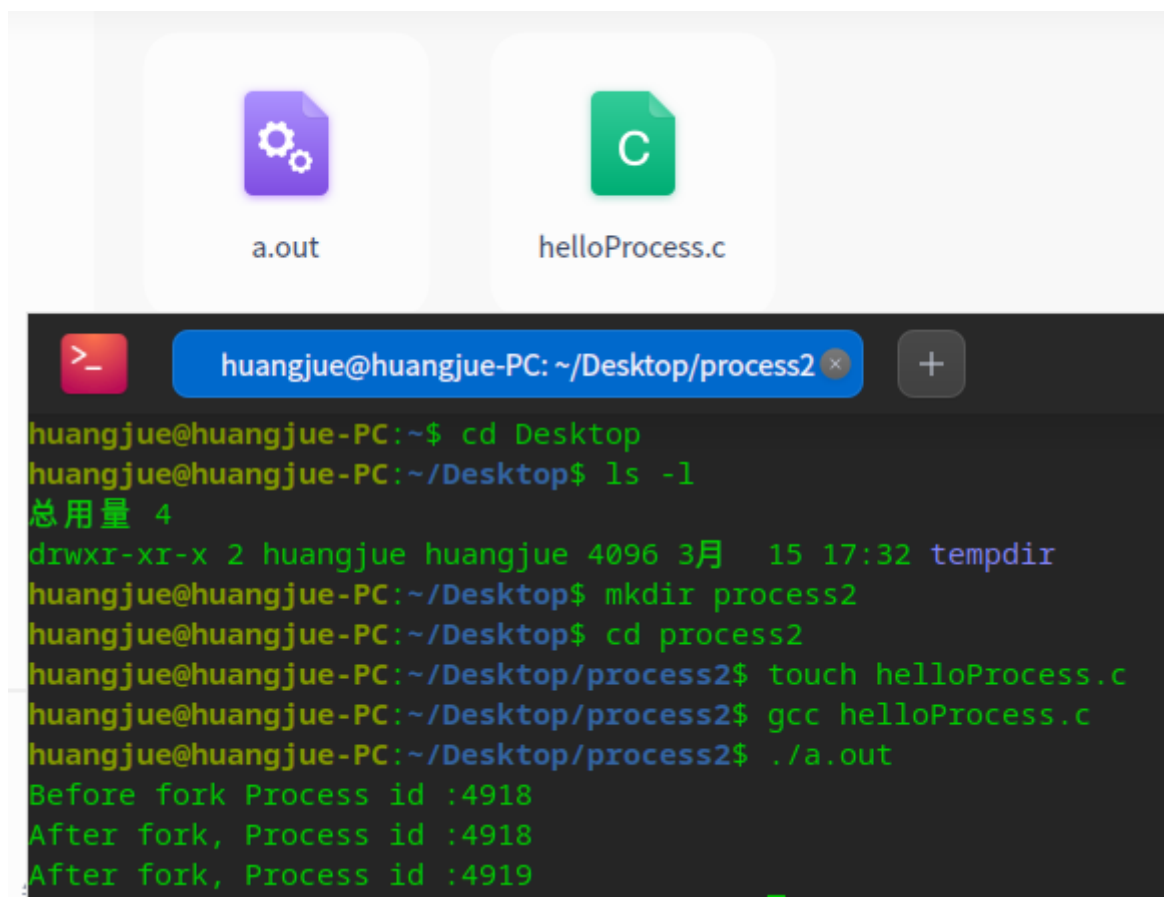
三、实验内容

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，
14     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。
15     fork()的返回值：
16         如果成功创建子进程，对于父进程fork会返回不同的值，
17     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。
18         如果创建失败，返回值为-1。
19     */
20    cid = fork();
21
22    printf("After fork, Process id :%d\n", getpid());
23
24    return 0;
25 }
```

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。



原因：

第一条输出结果，getpid()函数返回当前进程的id号，是4918；第二条输出结果，是父进程的id号，仍是4918；第三条输出结果，成功创建子进程，对于父进程它的返回值是子进程的进程id值，是4919。

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。

```

20     cid = fork();
21     int i;
22     scanf("%d",&i);
23     printf("After fork, Process id :%d\n", getpid());
24
25     return 0;
26 }

```

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。

```

huangjue@huangjue-PC: ~/Desktop/...
huangjue@huangjue-PC: ~/Desktop/process2$ ps -al
F S  UID    PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S   1000    5335   4445  0  80   0  -   570 wait_w pts/0      00:00:00 process2
1 S   1000    5336   5335  0  80   0  -   570 n_tty_ pts/0      00:00:00 process2
0 R   1000    5341   5338  0  80   0  -  2897 -      pts/1      00:00:00 ps

```

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

3:

```
huangjue@huangjue-PC:~/Desktop/process2$ gcc helloProcess.c -o process3
huangjue@huangjue-PC:~/Desktop/process2$ ./process3
Before fork process id :6096
Parent process id :6096
world
world
world
Child process id (my parent pid is 6096):6097
hello
hello
hello
```

300:

```
world
world
world
world
world
world
world
world
world
world
world
world
world
world
world
Child process id (my parent pid is 1):6262
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
```

world输出300次之后, hello输出300次

3000:

[illegible]

hello和world交替输出

原因：父子进程并发执行。

当运行次数少时，父进程会先执行完毕；当运行次数到3000次时，父子进程的printf语句交替执行，看出父子进程并发执行。

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
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
23     return 0;
24 }

```

```

huangjue@huangjue-PC:~/Desktop/process2$ touch helloProcess2.c
huangjue@huangjue-PC:~/Desktop/process2$ gcc helloProcess2.c -o process6
huangjue@huangjue-PC:~/Desktop/process2$ ./process6
In parent: x=101
In child: x=101

```

原因：fork()函数用于创建一个新进程，该进程为当前进程的子进程，父子进程内容完全一样，但是内存空间相互独立，父子进程都进行x++，最终输出两个结果都为101。

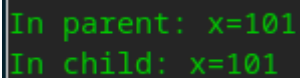
5.在上一步的代码的20行添加如下语句，同时代码最顶端要包含一个新的头文件

```

1 #include <stdio.h>
2 #include <sys/types.h>
3 #include <unistd.h>
4 #include <sys/wait.h>
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        wait(NULL);
21    }
22
23    return 0;
24 }

```

wait函数会让调用者陷入等待，直到子进程的状态变为可用（gcc即子进程结束前父进程一直处于等待状态）。



```

In parent: x=101
In child: x=101

```

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);

```

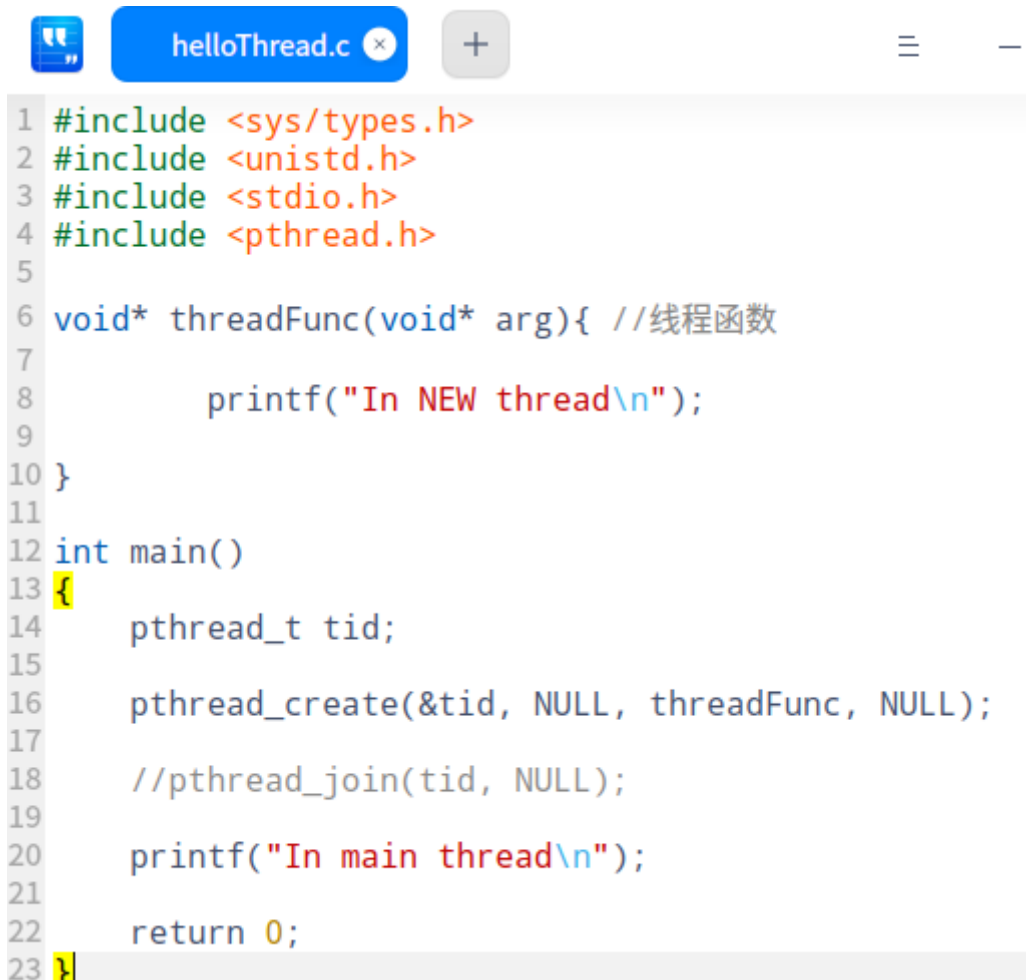
1 #include <stdio.h>
2 #include <sys/types.h>
3 #include <unistd.h>
4 #include <sys/wait.h>
5 int main()
6 {
7     pid_t cid;
8     int x = 100;
9
10    cid = fork();
11    sleep(3);
12    if(cid == 0){ //该分支是子进程执行的代码
13        x++;
14        printf("In child: x=%d\n",x);
15
16    }else{ //该分支是父进程执行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

```

sleep该函数可以让调用进程睡上指定的时间长度（单位是second）。

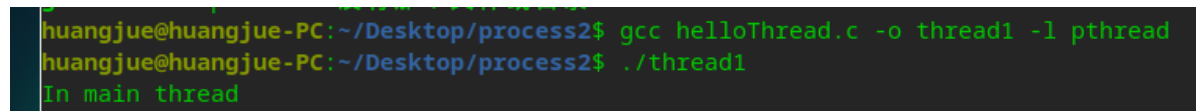
重新编译代码运行，我们特意让子进程输出完毕后睡了3秒，在这期间父进程什么事也没有做一直在wait，直到子进程结束后父进程才执行printf语句。

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。



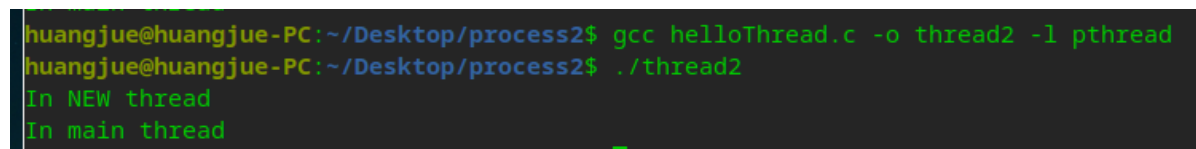
```
1 #include <sys/types.h>
2 #include <unistd.h>
3 #include <stdio.h>
4 #include <pthread.h>
5
6 void* threadFunc(void* arg){ //线程函数
7
8     printf("In NEW thread\n");
9
10 }
11
12 int main()
13 {
14     pthread_t tid;
15
16     pthread_create(&tid, NULL, threadFunc, NULL);
17
18     //pthread_join(tid, 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。运行一下观察到现象：



```
huangjue@huangjue-PC:~/Desktop/process2$ gcc helloThread.c -o thread1 -l pthread
huangjue@huangjue-PC:~/Desktop/process2$ ./thread1
In main thread
```

将上面第18行代码的注释去掉，观察到现象：



```
huangjue@huangjue-PC:~/Desktop/process2$ gcc helloThread.c -o thread2 -l pthread
huangjue@huangjue-PC:~/Desktop/process2$ ./thread2
In NEW thread
In main thread
```

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”因此输出。

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。

当循环达到300次时：

```
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 void* hello(void* arg){ //线程函数
9     for(int i=0;i<300;++i)
10    {
11        printf("hello(%d)\n",rand()%100);
12    }
13 }
14
15 void* world(void* arg){ //线程函数
16     for(int i=0;i<300;++i)
17    {
18        printf("world(%d)\n",rand()%100);
19    }
20 }
21
22 int main()
23 {
24     srand(time(NULL));
25     pthread_t tid,tid2;
26
27     pthread_create(&tid, NULL, hello, NULL);
28     pthread_create(&tid2, NULL, world, NULL);
29
30     pthread_join(tid,NULL);
31     pthread_join(tid2,NULL);
32
33     printf("In main thread\n");
34
35     return 0;
36 }
```

```
world(92)
world(49)
world(45)
world(57)
world(0)
world(32)
world(55)
hello(99)
hello(10)
hello(8)
world(83)
world(88)
world(87)
world(28)
world(49)
```

观察结果：输出的效果和并发父子进程的执行效果相似。