

实验2：进程及线程创建

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。

实验过程:

(1)

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
    //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
    pid_t pid,cid;
    //getpid()函数返回当前进程的id号
    printf("Before fork Process id :%d\n", getpid());

    /*
        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。
        fork()的返回值：
            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。
            如果创建失败，返回值为-1。
    */
    cid = fork();

    printf("After fork, Process id :%d\n", getpid());

    return 0;
}
```

- (2) 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。

```
xlz@xlz-PC:~$ gcc helloProcess.c -o helloProcess -pthread
xlz@xlz-PC:~$ ./helloProcess
Before fork Process id :3962
After fork,Process id :3962
xlz@xlz-PC:~$ After fork,Process id :3963
```

现象：

第一条printf显示当前进程ID号：3962；

第二条printf显示两次，是不同的ID号。

分析：

由于fork（）使得创建一个子进程，第二个printf运行了两次，分别显示了父进程与子进程的进程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。

实验过程：

```
printf("After fork,Process id :%d\n",getpid());
int i;
scanf("%d",&i);
return 0;
}
```

```
This may indicate that pixbuf loaders or the mime database could not be found.
xlz@xlz-PC:~$ gcc helloProcess.c -o helloProcess -pthread
xlz@xlz-PC:~$ ./helloProcess
Before fork Process id :4109
After fork,Process id :4109
After fork,Process id :4110
^C
xlz@xlz-PC:~$
```

如图，直到输入CTRL+C才退出，说明父进程未停止

- (2) 查看当前进程

```
xlz@xlz-PC:~$ ps -al
F S  UID      PID   PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000    5609    3869  0  80   0 -   605 wait_w pts/0        00:00:00 helloProcess
1 S  1000    5610    5609  0  80   0 -   605 n_tty_ pts/0        00:00:00 helloProcess
0 R  1000    5611    5421  0  80   0 -  2896 -      pts/1        00:00:00 ps
xlz@xlz-PC:~$
```

如图，说明第二行为第一行子进程

3. 通过判断fork的返回值让父子进程执行不同的语句。

- (1) 如图

```

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    }else{ //该分支是父进程执行的代码
18
19        printf("Parent process id :%d\n", getpid());
20        for(int i=0; i<3 ; i++)
21            printf("world\n");
22    }
23
24    return 0;
25 }

```

运行

```

xlz@xlz-PC:~$ ./helloProcess
Before fork Process id :7217
Parent process id :7217
world
world
world
xlz@xlz-PC:~$ Child process id (my parent pid is 1):7218
hello
hello
hello

```

现象：第一行输出fork()之前的进程ID；接下来的条件语句分别执行如上图所示，说明先是父进程，然后是子进程。Parent process id为“7217”，Child process id为“7218”，判断正确。

(2) 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，分别尝试300、3000，观察运行结果并解释原因。

```

hello
hello
hello
hello
hello
hello
hello
hello
hello
world
hello
world
hello
world

```

重新编译观察结果，
但实验结果好像是顺

得出父子进程交替

- 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。

```

#include <stdio.h>
#include <sys/types.h>
#include <unistd.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);

    }else{ //该分支是父进程执行的代码
        x++;

        printf("In parent: x=%d\n",x);

    }

    return 0;
}

```

结果

```

xlz@xlz-PC:~$ gcc helloProcess2.c -o ll
xlz@xlz-PC:~$ ./ll
In child : x=101
In parent : x=101
xlz@xlz-PC:~$

```

现象：父进程与子进程先后执行，X值相同，说明子进程的初始值由父进程克隆而来

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

```

#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){
        x++;
        printf("In child:x=%d\n",x);
        sleep(3);
    }else{
        x++;
        wait(NULL);
        printf("In parent:x=%d\n",x);
    }
    return 0;
}

```

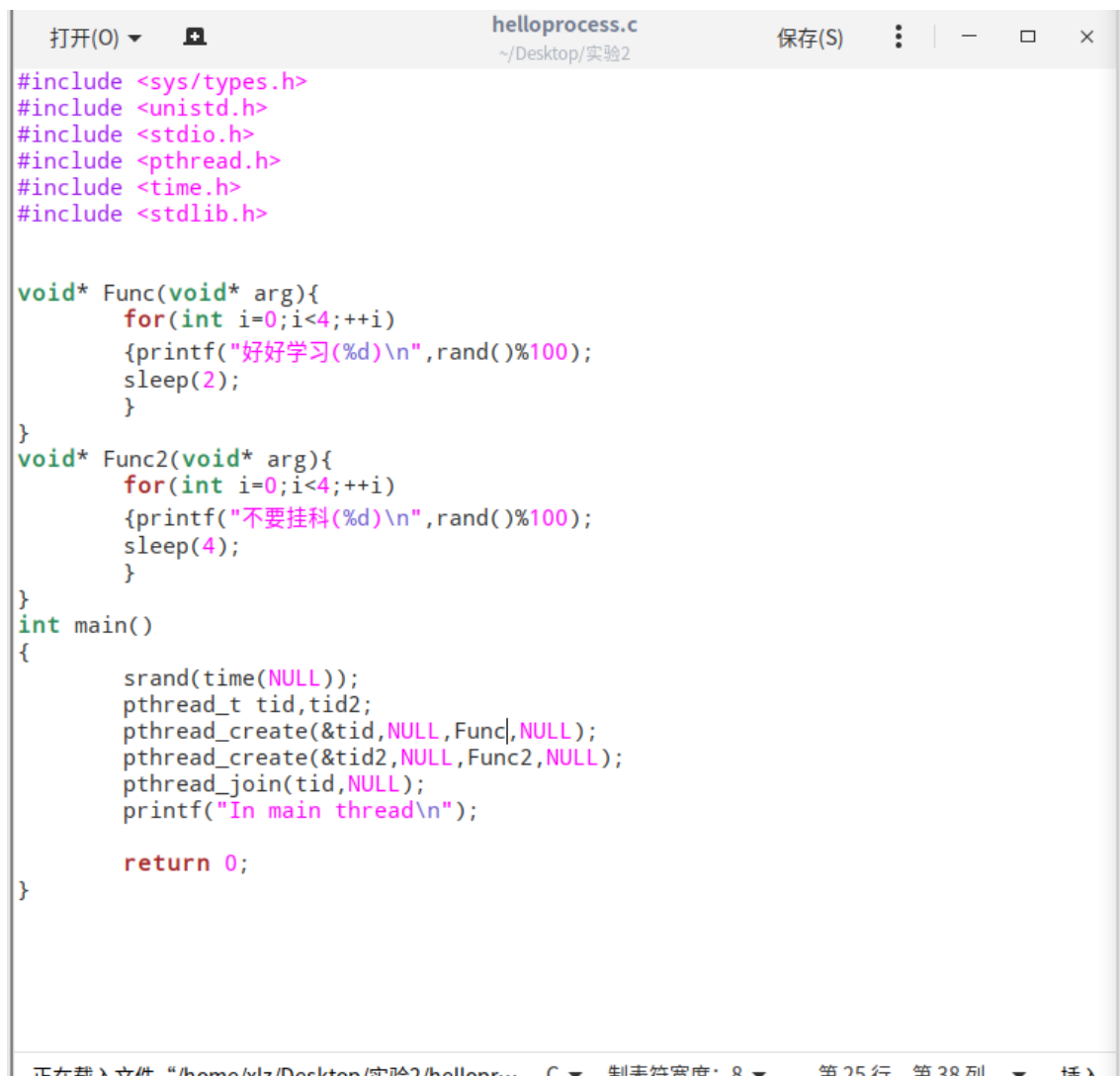
结果如下，相隔三秒后输出

```

xlz@xlz-PC:~$ gcc helloProcess2.c -o ll
xlz@xlz-PC:~$ ./ll
In child : x=101
In parent : x=101
xlz@xlz-PC:~$

```

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。



```
helloprocess.c
~/Desktop/实验2

打开(O) 保存(S)

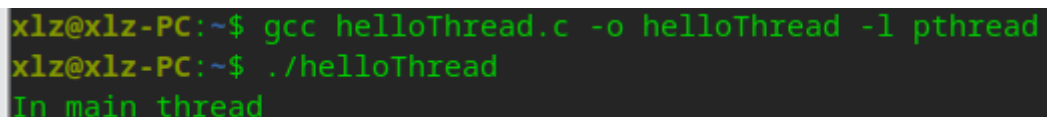
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <pthread.h>
#include <time.h>
#include <stdlib.h>

void* Func(void* arg){
    for(int i=0;i<4;++i)
    {printf("好好学习(%d)\n",rand()%100);
      sleep(2);
    }
}
void* Func2(void* arg){
    for(int i=0;i<4;++i)
    {printf("不要挂科(%d)\n",rand()%100);
      sleep(4);
    }
}
int main()
{
    srand(time(NULL));
    pthread_t tid,tid2;
    pthread_create(&tid,NULL,Func,NULL);
    pthread_create(&tid2,NULL,Func2,NULL);
    pthread_join(tid,NULL);
    printf("In main thread\n");

    return 0;
}
```

编译该段代码时，请注意gcc要加入新的参数，命令如下：

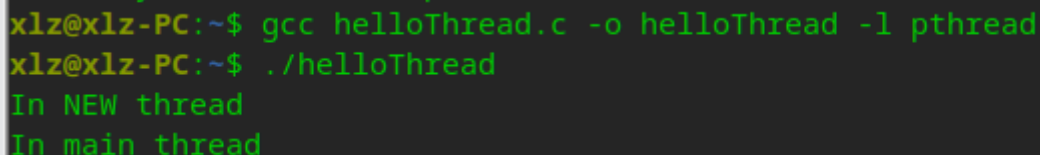
```
gcc helloThread.c -o helloThread -l pthread
```



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

(1)现象：只输出“In main thread”。由于main()中线程执行后没有等待子线程“threadFunc”就返回值，导致子线程不返回“In New thread”。

将第18行代码的注释去掉再次运行。



```
xlz@xlz-PC:~$ gcc helloThread.c -o helloThread -l pthread
xlz@xlz-PC:~$ ./helloThread
In NEW thread
In main thread
```

(2)现象:子/父线程的结果依次输出。说明父线程有等待子线程的运行结束

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

```
helloprocess.c
~/Desktop/实验2
保存(S)

#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <pthread.h>
#include <time.h>
#include <stdlib.h>

void* Func(void* arg){
    for(int i=0;i<4;++i)
    {printf("好好学习(%d)\n",rand()%100);
      sleep(2);
    }
}
void* Func2(void* arg){
    for(int i=0;i<4;++i)
    {printf("不要挂科(%d)\n",rand()%100);
      sleep(4);
    }
}
int main()
{
    srand(time(NULL));
    pthread_t tid,tid2;
    pthread_create(&tid,NULL,Func,NULL);
    pthread_create(&tid2,NULL,Func2,NULL);
    pthread_join(tid,NULL);
    printf("In main thread\n");

    return 0;
}
```

显示

```
xlz@xlz-PC:~/Desktop/实验2$ gcc helloprocess.c -o helloprocess -l pthread
xlz@xlz-PC:~/Desktop/实验2$ ./helloprocess
不要挂科 (95)
好好学习 (26)
好好学习 (53)
不要挂科 (48)
好好学习 (41)
好好学习 (14)
不要挂科 (15)
In main thread
xlz@xlz-PC:~/Desktop/实验2$
```

(3) 现象“好好学习”与“不要挂科”交替出现，说明父子进程并发执行

四、总结

父子进程在循环次数过少的情况下可能无法反映真实情况，要增加循环次数，才可以表现出理论值。