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sublime新建一个helloProcess.c源文件，并输入以下代码。

   实验过程：

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

int main()
{

    pid_t pid,cid;

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

    cid = fork();

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

    return 0;
}
```

   保存退出sublime，使用gcc对源文件进行编译，然后运行，观察结果并解释原因。

```
imfengyuan@kevinz:~/code/exp2$ ./helloProcess
Before fork Process id : 51389
After fork Process id : 51389
imfengyuan@kevinz:~/code/exp2$ After fork Process id : 51390
```

   原因：Fork()将当前进程完整拷贝了一份，子进程从fork()语句后开始执行

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：加入两行语句

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

```c
#include "stdio.h"
#include "sys/types.h"
#include "unistd.h"

int main(int argc, char const *argv[])
{
    pid_t pid,cid;
    printf("Before fork Process id : %d \n", getpid() );
    cid = fork();

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

保存并退出，使用gcc编译helloProcess.c，然后执行

```
imfengyuan@kevinz:~/code/exp2$ ./helloProcess
Before fork Process id : 52288
After fork Process id : 52288
After fork Process id : 52289
```

另外打开一个终端，输入

```
ps -al
```

```
imfengyuan@kevinz:~$ ps -al
F S   UID    PID   PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000  52288  22067  0  80   0 -   570 wait_w pts/0    00:00:00 helloProcess
1 S  1000  52289  52288  0  80   0 -   570 n_tty_ pts/0    00:00:00 helloProcess
0 R  1000  52290  34414  0  80   0 -  2897 -      pts/1    00:00:00 ps
imfengyuan@kevinz:~$
```

原因：helloProcess进程因为scanf()函数调用，等待输入设备输入，helloProcess进程进入等待状态

3. 通过判断fork()的返回值让父子进程执行不同的语句。

   修改helloProcess.c，如下所示

```c
#include "stdio.h"
#include "sys/types.h"
#include "unistd.h"

int main(int argc, char const *argv[])
{
    pid_t pid,cid;
    printf("Before fork Process id : %d \n", getpid() );
    cid = fork();
    if (cid == 0)
    {
        printf("Child process id (my parent pid is %d) : %d \n", getpid(),getpid() );
        for (int i = 0; i < 3; ++i)
        {
            printf("Hello\n");
        }
    }
    else
    {
        printf("Parent process id :%d\n", getpid());
        for (int i = 0; i < 3; ++i)
        {
            printf("World\n");
        }
    }

    return 0;
}
```

保存并退出，使用gcc编译helloProcess.c，然后执行

```
imfengyuan@kevinz:~/code/exp2$ ./helloProcess
Before fork Process id : 52671
Parent process id :52671
World
World
World
imfengyuan@kevinz:~/code/exp2$ Child process id (my parent pid is 52672) : 52672
Hello
Hello
Hello
```

修改循环的次数至300，如图所示

```
World
World
World
Child process id (my parent pid is 52780) : 52780
World
World
World
World
World
World
World
World
World
Hello
Hello
Hello
World
World
Hello
World
World
World
World
World
imfengyuan@kevinz:~/code/exp2$ Hello
Hello
```

原因：1. fork()的返回值在父进程和子进程中不同；

父进程中fork()的返回值为子进程的pid号

如果正常，子进程中fork()的返回值为0。错误则返回为一个负值。

2. “Hello和World交替出现”说明父进程和子进程并发执行

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subl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。

实验过程：使用subl命令新建一文件helloProcess2.c，输入下面的代码

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

int main(int argc, char const *argv[])
{
    pid_t cid;
    int x;

    cid = fork();

    if (cid == 0)
    {
        x++;
        printf("In child : x = %d\n", x);
    }
    else
    {
        x++;
        printf("In parent : x =%d\n", x);
    }
    return 0;
}
```

保存并退出，使用gcc编译helloProcess.c，然后执行

```
imfengyuan@kevinz:~/code/exp2$ ./helloProcess2
In parent : x =1
imfengyuan@kevinz:~/code/exp2$ In child : x = 1
```

原因：1. fork()将当前进程完整的拷贝给了子进程，子进程被分配了一个完整的新的内存空间。

2. 父进程和子进程不共享同一块内存空间

5. 在上一步的代码的return 0 前，添加如下语句，同时代码最顶端要包含一个新的头文件

```
#include "sys/wait.h"
wait(NULL);
```

保存并退出，使用gcc编译helloProcess.c，然后执行

```
imfengyuan@kevinz:~/code/exp2$ ./helloProcess2
In parent : x =1
In child : x = 1
```

将wait语句移入if的父进程执行的语句中，并在if的子进程执行的语句中加上sleep(3);

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

int main(int argc, char const *argv[])
{
    pid_t cid;
    int x;

    cid = fork();

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

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，subl 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。

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

void* threadFunc(void* arg){

        printf("In NEW thread\n");

}

int main()
{
    pthread_t tid;

    pthread_create(&tid, NULL, threadFunc, NULL);

    //pthread_join(tid, NULL);

    printf("In main thread\n");

    return 0;
```

```
}
```

保存并退出，使用gcc编译helloThread.c，然后执行,注意gcc要加入新的参数

```
imfengyuan@kevinz:~/code/exp2$ gcc helloThread.c -o helloThread -pthread
imfengyuan@kevinz:~/code/exp2$ ./helloThread
In main thread
imfengyuan@kevinz:~/code/exp2$
```

现象：创建了线程，但是线程中的语句没有被执行

将上面代码的注释去掉

```
imfengyuan@kevinz:~/code/exp2$ gcc helloThread.c -o helloThread -pthread
imfengyuan@kevinz:~/code/exp2$ ./helloThread
In NEW thread
In main thread
imfengyuan@kevinz:~/code/exp2$
```

现象：线程中的语句被成功执行

原因：此处父线程和子线程相互独立，父线程执行完毕后，子线程就会终止。

修改helloThread.c，如下图所示

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

void* threadFunc(void* arg){

        printf("In NEW thread\n");
        for (int i = 0; i < 3; ++i)
      {
        printf("World\n");
      }
}

int main()
{
    pthread_t tid;

    pthread_create(&tid, NULL, threadFunc, NULL);



    printf("In main thread\n");

    for (int i = 0; i < 3; ++i)
    {
      printf("Hello\n");
    }
    pthread_join(tid, NULL);
    return 0;
}
```

保存并退出，使用gcc编译helloThread.c，执行如下所示

```
imfengyuan@kevinz:~/code/exp2$ gcc helloThread.c -o helloThread -pthread
imfengyuan@kevinz:~/code/exp2$ ./helloThread
In main thread
Hello
Hello
Hello
In NEW thread
World
World
World
imfengyuan@kevinz:~/code/exp2$ ./helloThread
```

修改循环次数，并增加sleep(3)，增加线程间隔

```
Hello
World
World
Hello
World
Hello
World
World
Hello
World
Hello
World
World
Hello
```

结果：在主线程和新线程里加入循环输出，输出的效果和并发父子进程的执行效果相似

四、总结

1. fork()函数创建子进程的使用以及其返回值的特点
2. 进程进入等待状态的时候，OS调度其他进程进入CPU
3. 进程是并发执行的
4. 线程是CPU使用的基本单位，线程之间是并发的
5. 子线程依赖其进程，如果进程退出，所有线程也就会退出。