《Linux操作系统》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。

```
#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());
int i;
scanf("%d",&i);
    return 0;
}
```

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。

```
alenswet@alenswet-PC:~/Desktop$ ./helloProcess
Before fork Process id :5106
After fork, Process id :5106
alenswet@alenswet-PC:~/Desktop$ After fork, Process id :5107
```

fork()创建了一个子进程，第二个print输出了两次，显示的分别是父进程和子进程的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);
```

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。

```
ps -al
```

```
alenswet@alenswet-PC:~/Desktop$ ./helloProcess
Before fork Process id :5679
After fork, Process id :5679
After fork, Process id :5680
```

```
alenswet@alenswet-PC:~/Desktop$ ps -al
F S   UID    PID   PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000   5679   5088  0  80   0 -   570 wait_w pts/0    00:00:00 helloProcess
1 S  1000   5680   5679  0  80   0 -   570 n_tty_ pts/0    00:00:00 helloProcess
0 R  1000   5685   5682  0  80   0 -  2897 -      pts/1    00:00:00 ps
```

如上图所示，运行后并未退出，需键入ctrl+C退出，说明父子进程并未结束

3. 通过判断fork的返回值让父子进程执行不同的语句。

（1） 输入如下代码并保存运行。

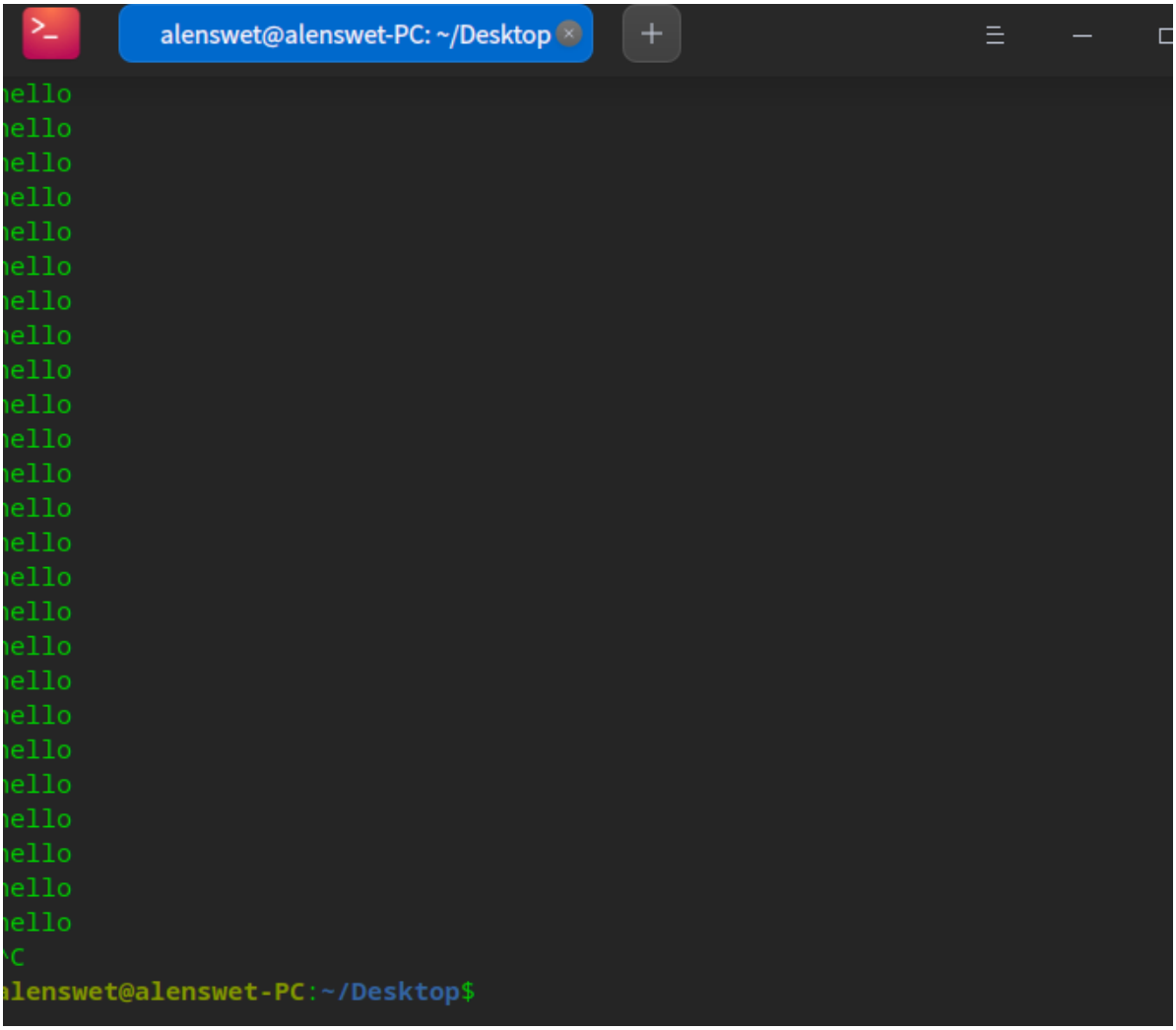
```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
    pid_t 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<300;i++)
            printf("hello\n");
}else{
    printf("Parent process id :%d\n",getpid() );
    for(int i=0;i<300; i++)
        printf("world\n");
}

 return 0;
}
```

结果如下图所示：

```
alenswet@alenswet-PC:~/Desktop$ ./a.out
Before fork process id :7289
Parent process id :7289
world
world
world
alenswet@alenswet-PC:~/Desktop$ Child process id (my parent pid is 7290):7290
hello
hello
hello
```

第一行输出fork()之前的进程ID；由图可知，先是父进程进行，然后是子进程。Parent process id=7289，Child process id=7290 子进程ID在父进程后面。

……分别尝试300、3000，然后再观察运行结果并解释原因。



调整后输出结果并未发生变化，可能是因为进程运行依然很快

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

```c
#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;
}
```

```
alenswet@alenswet-PC:~/Desktop$ ./helloProcess2
In parent:x=101
alenswet@alenswet-PC:~/Desktop$ In child;x=101
```

父进程和子进程先后执行，x均为101，说明子进程是父进程克隆而来的。

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。

两次输出间隔三秒：

```
alenswet@alenswet-PC:~/Desktop$ ./helloProcess2
In child;x=101
In parent:x=101
```

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。

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

void* threadFunc(void* arg)
{
    printf("In NEW thread(%d)\n",rand()%100);
}

int main()
{
    pthread_t tid,tid2;

    pthread_create(&tid, NULL, threadFunc,NULL);
    pthread_create(&tid2, NULL, main,NULL);
    pthread_join(tid,NULL);
    printf("In main thread(%d)\n",rand()%100);

    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
```

```
alenswet@alenswet-PC:~/Desktop$ ./helloThread
In main thread
```

只输出了*In main thread*。由于main()中线程执行后没有等待子线程，子线程不返回*In New thread*。

（1）将上面第18行代码的注释去掉又观察到了什么现象？

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

父子线程依次输出结果，说明父线程在等待子线程的运行结束

（2）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。

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

void* threadFunc(void* arg)
{
    printf("In NEW thread(%d)\n",rand()%100);
}

int main()
{
    pthread_t tid,tid2;

    pthread_create(&tid, NULL, threadFunc,NULL);
    pthread_create(&tid2, NULL, main,NULL);
    pthread_join(tid,NULL);
    printf("In main thread(%d)\n",rand()%100);

    return 0;
}
```

四、总结