《Linux操作系统》实验2：进程及线程创建

网安1901 201904080141 李旭健

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.

```
y1ng@y1ng-PC:~/Desktop/code$ cat process.c
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>

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

        cid = fork(); //创建一个新的进程，为当前进程的子进程，创建失败返回-1
                      //创建成功，对于父进程fork会返回不同的值
                      //          对于子进程，返回值是0
        printf("after fork,process id :%d\n",cid);


        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。

```
y1ng@y1ng-PC:~/Desktop/code$ gcc process.c -o process
y1ng@y1ng-PC:~/Desktop/code$ ./process
Before fork process id :3959
after fork,process id :3959
after fork,process id :3960
```

父进程一直运行没有停止.

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。

```
y1ng@y1ng-PC:~/Desktop/code$ cat process.c
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>

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

        cid = fork(); //创建一个新的进程，为当前进程的子进程，创建失败返回-1
                      //创建成功，对于父进程fork会返回不同的值
                      //          对于子进程，返回值是0
        printf("after fork,process id :%d\n",getpid());

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

        return 0;
}
```

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。

```
y1ng@y1ng-PC:~/Desktop/code$ ps -al
F S   UID    PID   PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000   4131   4127  0  80   0 -   569 wait_w pts/1    00:00:00 process
1 S  1000   4132   4131  0  80   0 -   569 n_tty_ pts/1    00:00:00 process
0 R  1000   4136   4133  0  80   0 -  2129 -      pts/0    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。

```
y1ng@y1ng-PC:~/Desktop/code$ cat process_验证.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;
        printf("Before fork process id :%d\n",getpid());

        //int value = 10000;
        cid = fork();

        if(cid == 0){
                        printf("child process id(my parent pid is %d):%d\n", getppid(),getpid());
                        for(int i=0;i<=3;i++)
                                printf("hello!\n");
                                //printf("hello!%d\n",value--);
                        //sleep(3); //睡3秒钟
        }
        else{
                printf("parent process id :%d\n",getpid());
                for(int i=0;i<=3;i++)
                        printf("world!\n");
                        //printf("world!%d\n",value++);

                //wait(NULL);  //等待子进程结束再返回
                }

}
```

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。

```
y1ng@y1ng-PC:~/Desktop/code$ ./process_验证
Before fork process id :6344
parent process id :6344
world!
world!
world!
world!
world!
world!
world!
world!
world!
world!
world!
world!
world!
world!
child process id(my parent pid is 6344):6345
world!
world!
world!
world!
hello!
world!
hello!
world!
```

由此可见父子进程是并发进行.

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。

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

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

        int value = 100;
        cid = fork();

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


                }
        return 0;

}
```

```
y1ng@y1ng-PC:~/Desktop/code$ gcc heloProcess2.c  -o helloprocess2
y1ng@y1ng-PC:~/Desktop/code$ ./helloprocess2
In parent:value=101
In child: calue=101
```

父进程与子进程相互独立运行,互不干扰

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

        int value = 10000;
        cid = fork();

        if(cid == 0){
                        printf("child process id(my parent pid is %d):%d\n", getppid(),getpid());
                        for(int i=0;i<=40;i++)
                                printf("hello!%d\n",value--);
                        sleep(3); //睡3秒钟
        }
        else{
                printf("parent process id :%d\n",getpid());
                for(int i=0;i<=40;i++)
                        printf("world!%d\n",value++);

                wait(NULL);  //等待子进程结束再返回
                }
        return 0;

}


y1ng@y1ng-PC:~/Desktop/code$
```

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。

```
op/code$ cat hellothread.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);
        printf("In main thread\n");
        return 0;
}
y1ng@y1ng-PC:~/Desktop/code$
```

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

运行一下观察到什么现象了？

```
y1ng@y1ng-PC:~/Desktop/code$ gcc hellothread.c -o hellothread -l pthread
y1ng@y1ng-PC:~/Desktop/code$ ./hellothread
In main thread
```

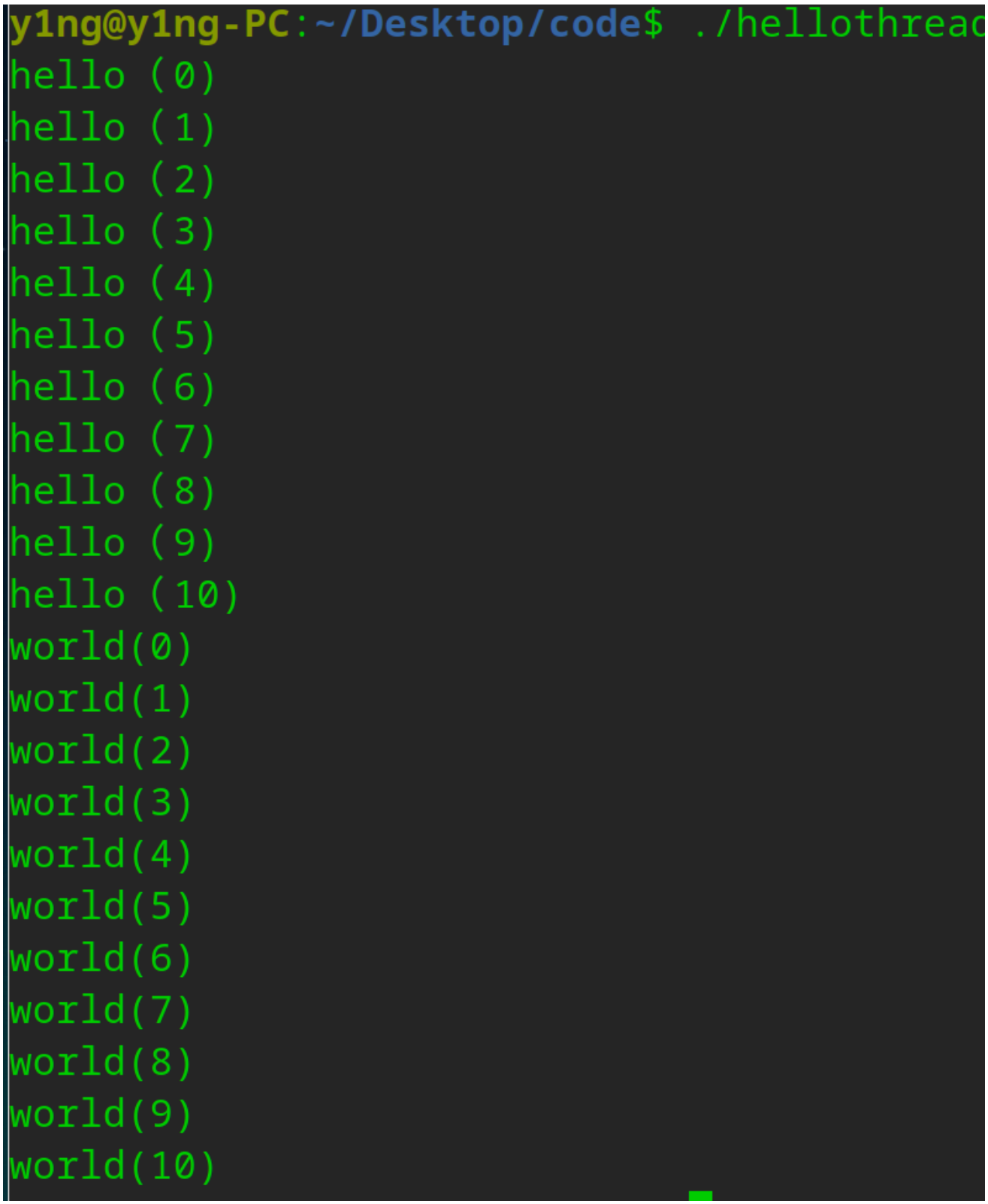
将上面第18行代码的注释去掉又观察到了什么现象？为什么？

```
y1ng@y1ng-PC:~/Desktop/code$ gcc hellothread.c -o hellothread -l pthread
y1ng@y1ng-PC:~/Desktop/code$ ./hellothread
In new thread
In main thread
```

主线程运行完没有等待新线程,导致进程结束.

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。

```
y1ng@y1ng-PC:~/Desktop/code$ cat hellothread.c
#include <sys/types.h>
#include<unistd.h>
#include<stdio.h>
#include<pthread.h>

void* threadfunc(void* arg)
{       int i;
        //printf("In new thread\n");
        for(i = 0;i<=10;i++)
        {printf("hello (%d)\n",i);
         sleep(2);}
}

int main(){
            int i;
        pthread_t tid;

        pthread_create(&tid,NULL,threadfunc,NULL);//1.thread id address
                                                  //2.thread attribute 线程属性
                                                  //3.thread function address 线程函数地址
                                                  //1.thread parameters address 参数地址
        pthread_join(tid,NULL);                   //相当于wait

        //printf("In main thread\n");

        for(i=0;i<=10;i++){
        printf("world(%d)\n",i);
        sleep(3);}
        return 0;
}
```



四、实验总结

1.子进程依赖父进程

2.wait（）与sleep（）与scanf（）的等效目的：便于观察实验结果