《Linux操作系统》实验2：进程及线程创建

班级：网安1901 姓名：董学智 学号：201904080138

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，对于父子进程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;
}
```

```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());
    pause();
    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。

```
zzz@zzz-PC:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Videos  wocao
zzz@zzz-PC:~$ cd Desktop
zzz@zzz-PC:~/Desktop$ gcc hh.c -o zz
zzz@zzz-PC:~/Desktop$ ./zz
before fork process id :5130
after fork,process id : 5130
after fork,process id : 5131
```

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。

```c
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
```

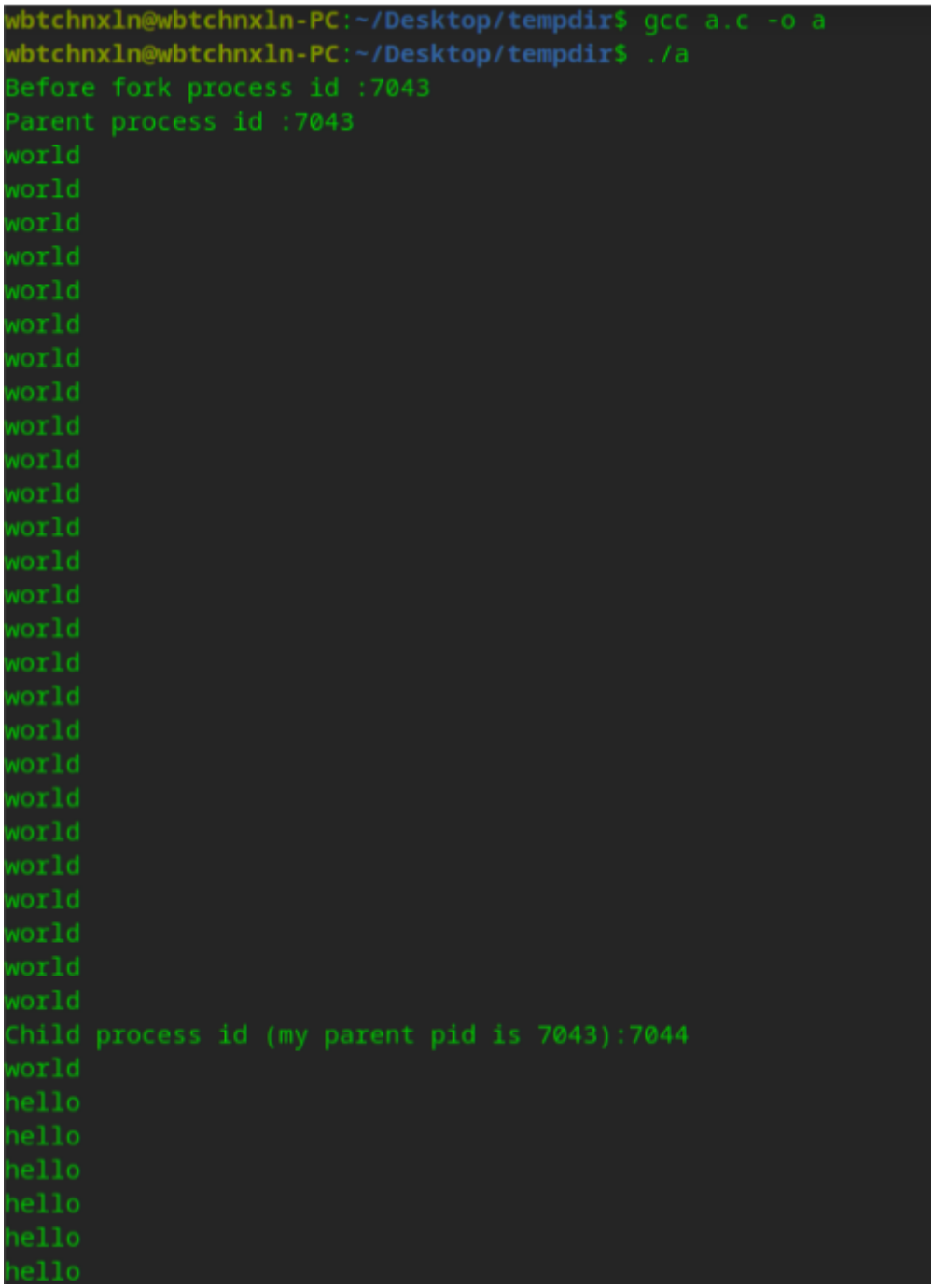
```
zzz@zzz-PC:~/Desktop$ ps -al
F S   UID    PID   PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 R  1000   5212   5207  0  80   0 -  2889 -      pts/0    00:00:00 ps
zzz@zzz-PC:~/Desktop$
```

3. 通过判断fork的返回值让父子进程执行不同的语句。

```c
#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){ //该分支是子进程执行的代码

       printf("Child process id (my parent pid is %d):%d\n",
getppid(),getpid());
       for(int i=0; i<5 ; i++)
            printf("hello\n");

    }else{ //该分支是父进程执行的代码

        printf("Parent process id :%d\n", getpid());
        for(int i=0; i<5 ; i++)
            printf("world\n");
    }

    return 0;
}
```

```
zzz@zzz-PC:~/Desktop$ gcc hhh.c -o zzz
zzz@zzz-PC:~/Desktop$ ./zzz
before fork process id :5686
parent process id :5686
world
world
world
world
world
zzz@zzz-PC:~/Desktop$ child process id (my parent pid is 5687):5687
hello
hello
hello
hello
hello
```

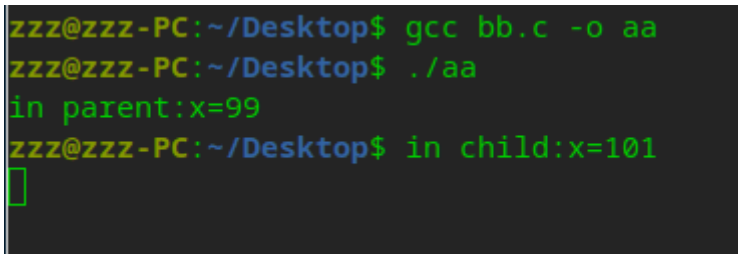


"hello""world"交替出现说明是并发过程

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>

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;
}
```



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 <sys/wait.h>
wait(NULL);
```

```
sleep(3);
```

```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--;
        printf("in parent:x=%d\n", x);
        wait(NULL);
    }

    return 0;
}
```

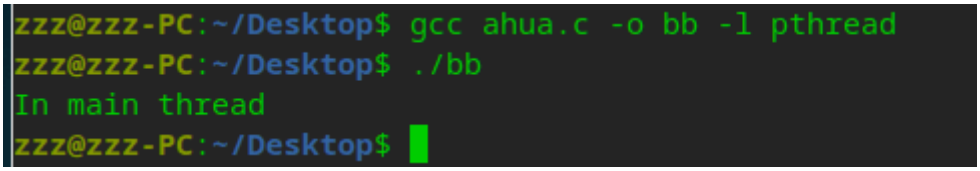
6. 创建线程。先关闭先前的文件，gedit helloThread.c以创建一个新的C语言源文件，将下面的代码拷贝进编辑器。

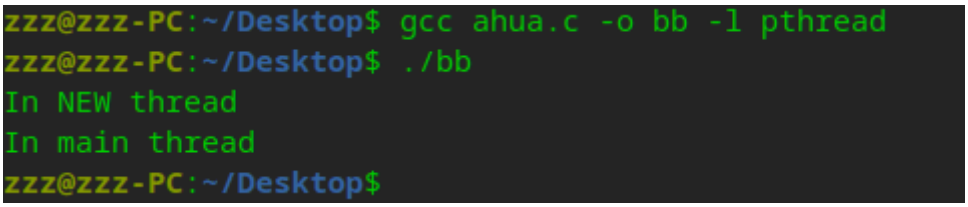
```c
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <pthread.h>

void* threadFunc(void* arg){ //线程函数

        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;
}
```

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
```





四、实验总结

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