

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();
```

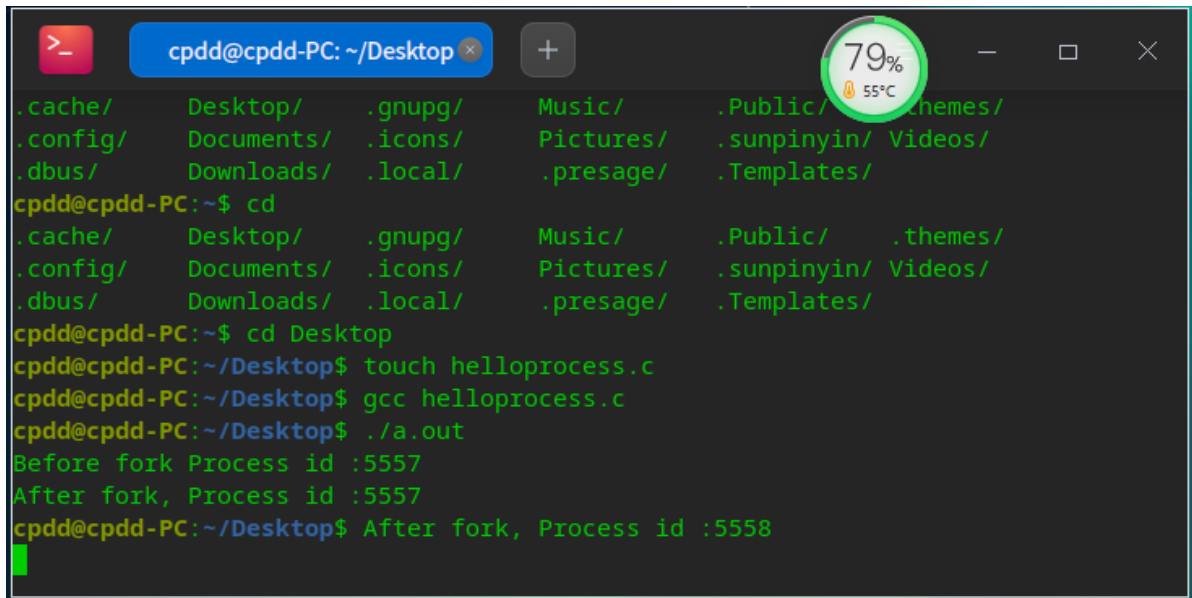
```
    printf("After fork, Process id :%d\n", getpid());
```

```

return 0;
}

```

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。



```

cpdd@cpdd-PC: ~/Desktop
.cache/ Desktop/ .gnupg/ Music/ .Public/ .themes/
.config/ Documents/ .icons/ Pictures/ .sunpinyin/ Videos/
.dbus/ Downloads/ .local/ .presage/ .Templates/
cpdd@cpdd-PC:~$ cd
.cache/ Desktop/ .gnupg/ Music/ .Public/ .themes/
.config/ Documents/ .icons/ Pictures/ .sunpinyin/ Videos/
.dbus/ Downloads/ .local/ .presage/ .Templates/
cpdd@cpdd-PC:~$ cd Desktop
cpdd@cpdd-PC:~/Desktop$ touch helloprocess.c
cpdd@cpdd-PC:~/Desktop$ gcc helloprocess.c
cpdd@cpdd-PC:~/Desktop$ ./a.out
Before fork Process id :5557
After fork, Process id :5557
cpdd@cpdd-PC:~/Desktop$ After fork, Process id :5558

```

结果：出现三行代码，第一行打印before fork的进程id，第二三行为after fork的进程id,且两个id不同。

原因：fork()函数创建了一个新的子进程，父进程与子进程都执行了一遍after fork的getpid(),并发执行了fork之后的所有代码。由此可知第三行为子进程id号，第二行为父进程id号。

1. 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。

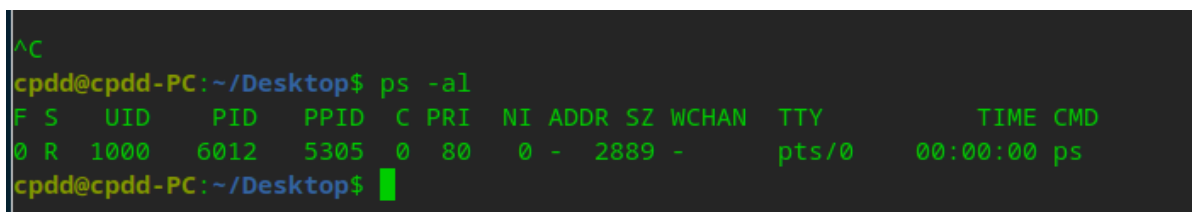
```

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
```



```

^C
cpdd@cpdd-PC:~/Desktop$ ps -al
F S  UID      PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 R   1000    6012   5305  0  80   0 -  2889 -      pts/0    00:00:00 ps
cpdd@cpdd-PC:~/Desktop$

```

1. 通过判断fork的返回值让父子进程执行不同的语句。

```

#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<3 ; 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<3 ; i++)

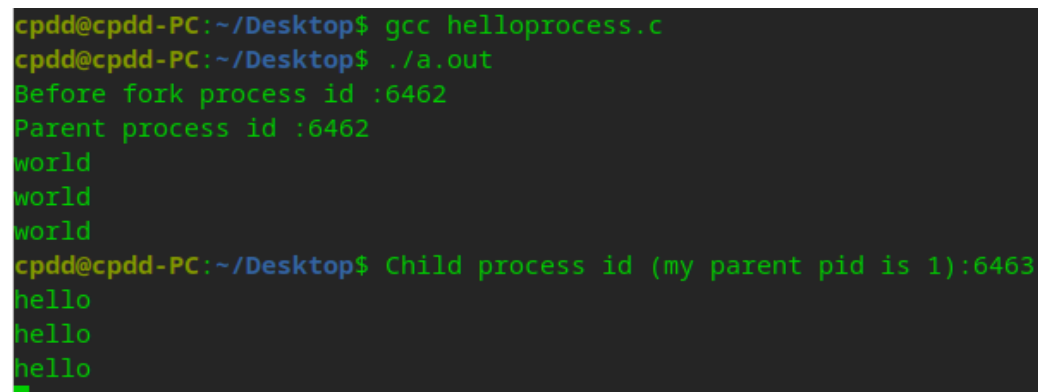
            printf("world\n");

    }

    return 0;
}

```

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。



```

cpdd@cpdd-PC:~/Desktop$ gcc helloprocess.c
cpdd@cpdd-PC:~/Desktop$ ./a.out
Before fork process id :6462
Parent process id :6462
world
world
world
cpdd@cpdd-PC:~/Desktop$ Child process id (my parent pid is 1):6463
hello
hello
hello

```

原因：父进程与子进程是顺序执行的

```

Parent process Gocid=fork()cid==0?Child process codeParent process ReturnParent process
codeyesno

```

上图解释了fork的工作流程，请大家参照代码仔细理解。

1. 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++;

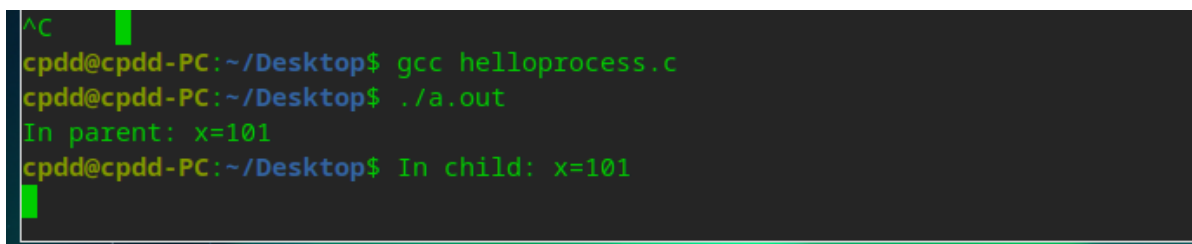
        printf("In child: x=%d\n",x);

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

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

    }

    return 0;
}
```



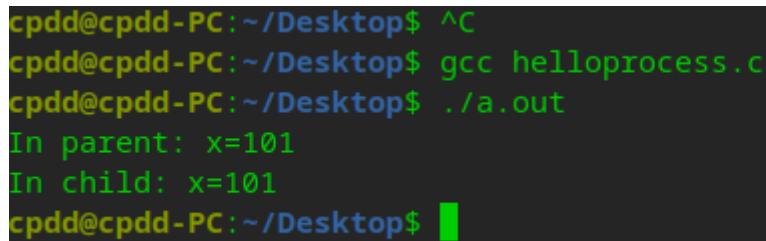
```
^C
cpdd@cpdd-PC:~/Desktop$ gcc helloprocess.c
cpdd@cpdd-PC:~/Desktop$ ./a.out
In parent: x=101
cpdd@cpdd-PC:~/Desktop$ In child: x=101
```

原因：子进程的内存空间是由父进程完全复制过来的，两个进程之间的内存空间是相互独立的，不共享同一个空间。

1. 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);
```



```
cpdd@cpdd-PC:~/Desktop$ ^C
cpdd@cpdd-PC:~/Desktop$ gcc helloprocess.c
cpdd@cpdd-PC:~/Desktop$ ./a.out
In parent: x=101
In child: x=101
cpdd@cpdd-PC:~/Desktop$
```

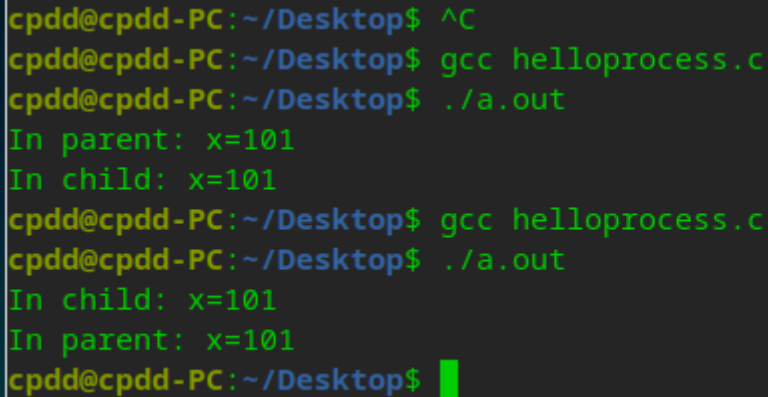
wait函数会让调用者陷入等待，直到子进程的状态变为可用（即子进程结束前父进程一直处于等待状态）。

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：

```
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。



```
cpdd@cpdd-PC:~/Desktop$ ^C
cpdd@cpdd-PC:~/Desktop$ gcc helloprocess.c
cpdd@cpdd-PC:~/Desktop$ ./a.out
In parent: x=101
In child: x=101
cpdd@cpdd-PC:~/Desktop$ gcc helloprocess.c
cpdd@cpdd-PC:~/Desktop$ ./a.out
In child: x=101
In parent: x=101
cpdd@cpdd-PC:~/Desktop$
```

1. 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 <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);

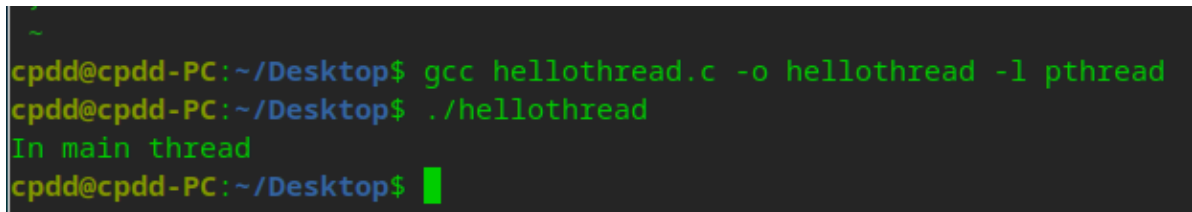
    printf("In main thread\n");

    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
```



```

~
cpdd@cpdd-PC:~/Desktop$ gcc hellothread.c -o hellothread -l pthread
cpdd@cpdd-PC:~/Desktop$ ./hellothread
In main thread
cpdd@cpdd-PC:~/Desktop$ █

```

现象：创建了线程但是线程中的语句未被执行

运行一下观察到什么现象了？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？

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。