《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 jincheng.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。

```
root@lxd-PC:/home/lxd/2# gcc jincheng.c -o jincheng
root@lxd-PC:/home/lxd/2# ls
jincheng  jincheng.c
root@lxd-PC:/home/lxd/2# ls -l
总用量 24
-rwxr-xr-x 1 root root 16520 3月  26 09:53 jincheng
-rw-r--r-- 1 lxd  lxd    966 3月  26 09:52 jincheng.c
root@lxd-PC:/home/lxd/2#
```

```
root@lxd-PC:/home/lxd/2# gcc jincheng.c -o jincheng
root@lxd-PC:/home/lxd/2# ls
jincheng  jincheng.c
root@lxd-PC:/home/lxd/2# ls -l
总用量 24
-rwxr-xr-x 1 root root 16520 3月  26 09:53 jincheng
-rw-r--r-- 1 lxd  lxd    966 3月  26 09:52 jincheng.c
root@lxd-PC:/home/lxd/2# ./jincheng
Before fork Process id :64843
After fork, Process id :64843
root@lxd-PC:/home/lxd/2# After fork, Process id :64844
```

Fork（）将当期进程完整拷贝了一份，子进程从 fork（）语句后面开始执行

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。

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

```
root@lxd-PC:/home/lxd/2# gcc jincheng.c -o jincheng
root@lxd-PC:/home/lxd/2# ./jincheng
Before fork Process id :70044
After fork, Process id :70044
After fork, Process id :70045
```

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。

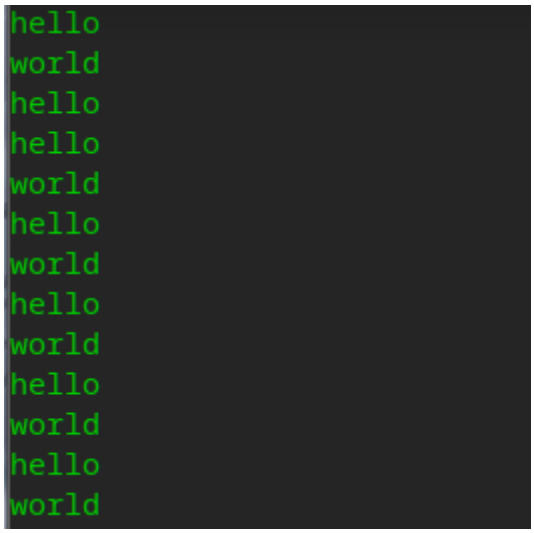
```
ps -al
```

```
lxd@lxd-PC:~/2$ ps -al
F S   UID    PID   PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S     0  34933  33403  0  80   0 - 21679 -      pts/1    00:00:00 su
4 S     0  34936  34933  0  80   0 -  2561 -      pts/1    00:00:00 bash
4 S     0  41419  34936  0  80   0 - 125260 -     pts/1    00:00:00 gedit
4 S     0  63965  62745  0  80   0 - 21679 -      pts/2    00:00:00 su
4 S     0  63968  63965  0  80   0 -  2502 -      pts/2    00:00:00 bash
0 S     0  70044  63968  0  80   0 -   570 -      pts/2    00:00:00 jincheng
1 S     0  70045  70044  0  80   0 -   570 -      pts/2    00:00:00 jincheng
4 R  1000  70135  64952  0  80   0 -  2896 -      pts/3    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。

```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<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;
}
```

```
root@lxd-PC:/home/lxd/2# gcc jincheng.c -o jincheng
root@lxd-PC:/home/lxd/2# ./jincheng
Before fork process id :70234
Parent process id :70234
world
world
world
root@lxd-PC:/home/lxd/2# Child process id (my parent pid is 1):70235
hello
hello
hello
```



循环次数 3000

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 子进程的 fork()的返回值为 0 错误返回一个负值

Hello 和 word 交替出现说明并发执行

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 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;
}
```

```
root@lxd-PC:/home/lxd/2# gcc jincheng2.c -o jincheng2
root@lxd-PC:/home/lxd/2# ./jincheng2
In parent: x=101
root@lxd-PC:/home/lxd/2# In child: x=101
```

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。
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 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);
```

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。

```
root@lxd-PC:/home/lxd/2# gcc jincheng2.c -o jincheng2
root@lxd-PC:/home/lxd/2# ./jincheng2
In child: x=101
In parent: x=101
root@lxd-PC:/home/lxd/2#
```

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 <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");
}
```

```c
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
```