

《Linux操作系统》实验2：进程及线程创建

区队：网安1901 学号：201904080120 姓名：何芷萌

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。

```
1  #include <stdio.h>
2  #include <sys/types.h>
3  #include <unistd.h>
4
5  int main()
6  {
7      //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
8      pid_t pid,cid;
9      //getpid()函数返回当前进程的id号
10     printf("Before fork Process id :%d\n", getpid());
11
12     /*
13      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。
14      fork()的返回值：
15          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。
16          如果创建失败，返回值为-1。
17      */
18     cid = fork();
19
20     printf("After fork, Process id :%d\n", getpid());
21
22     return 0;
23 }
```

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。

```
hezhimeng@hezhimeng-PC: ~/Desktop/pr...
hezhimeng@hezhimeng-PC:~/Desktop$ cd Desktop
hezhimeng@hezhimeng-PC:~/Desktop$ cd practice2
hezhimeng@hezhimeng-PC:~/Desktop/practice2$ touch helloProcess.c
hezhimeng@hezhimeng-PC:~/Desktop/practice2$ gcc helloProcess.c -o helloProcess
hezhimeng@hezhimeng-PC:~/Desktop/practice2$ ./helloProcess
Before fork Process id :6453
After fork, Process id :6453
After fork, Process id :6454
hezhimeng@hezhimeng-PC:~/Desktop/practice2$
```

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。

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。

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

```
~/Desktop/practice2/helloProcess.c - Sublime Text 2 (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

helloProcess.c
1 #include <stdio.h>
2 #include <sys/types.h>
3 #include <unistd.h>
4
5 int main()
6 {
7     //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
8     pid_t pid,cid;
9     //getpid()函数返回当前进程的id号
10    printf("Before fork Process id :%d\n", getpid());
11
12    /*
13    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
14    fork()的返回值：
15    如果成功创建子进程，对于父进程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
16    如果创建失败，返回值为-1。
17    */
18    cid = fork();
19
20    printf("After fork, Process id :%d\n", getpid());
21
22    int i;
23    scanf("%d",&i);
24
25    return 0;
26 }
```

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。

```
1 ps -al
```

```
hezhimeng@hezhimeng-PC:~/Desktop/practice2$ gcc helloProcess.c -o helloProcess
hezhimeng@hezhimeng-PC:~/Desktop/practice2$ ./helloProcess
Before fork Process id :7788
After fork, Process id :7788
After fork, Process id :7789
```

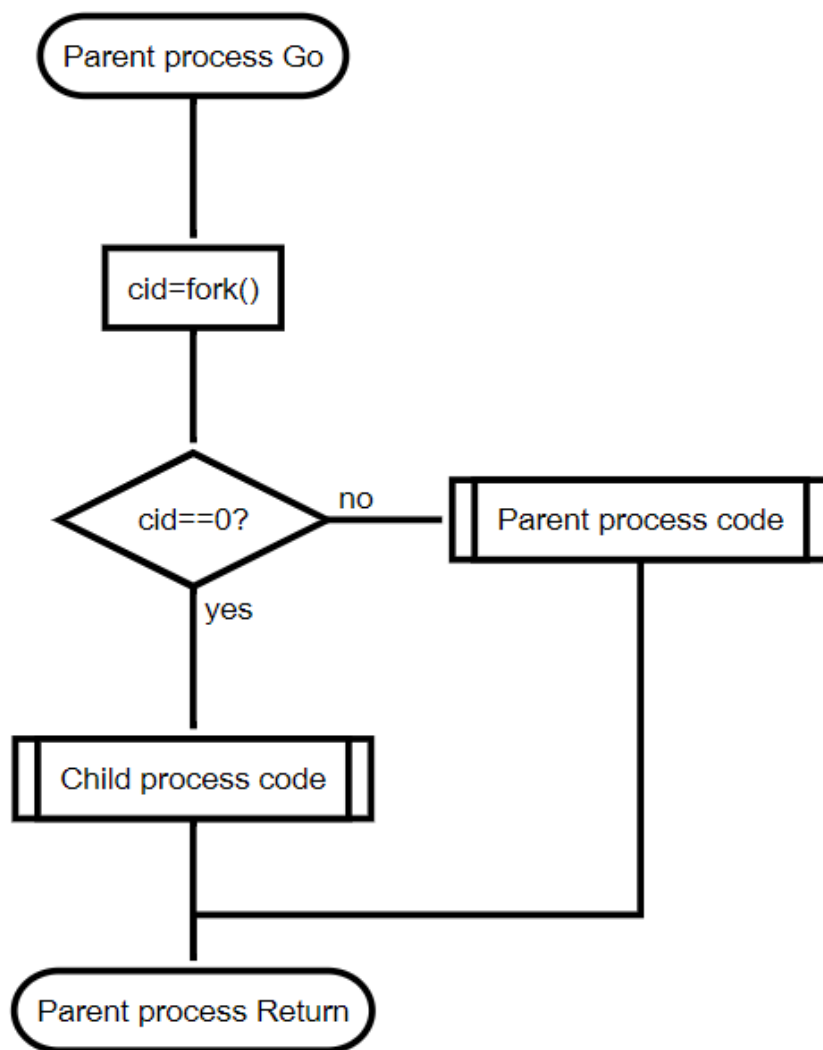
```
hezhimeng@hezhimeng-PC: ~/Desktop/pr... hezhimeng@hezhimeng-PC: ~/Desktop/pr...
hezhimeng@hezhimeng-PC:~/Desktop/practice2$ ps -al
F S  UID      PID   PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000    7788   6320  0  80   0  -   570 wait_w pts/0    00:00:00 helloProcess
1 S  1000    7789   7788  0  80   0  -   570 n_tty_ pts/0    00:00:00 helloProcess
0 R  1000    7843   6635  0  80   0  -  2897 -      pts/1    00:00:00 ps
```

结果：通过PID和PPID两列可知，第二个helloProcess是第一个helloProcess的子进程

3. 通过判断fork的返回值让父子进程执行不同的语句。

```
1  #include <stdio.h>
2  #include <sys/types.h>
3  #include <unistd.h>
4
5  int main()
6  {
7      pid_t cid;
8      printf("Before fork process id :%d\n", getpid());
9
10     cid = fork();
11
12     if(cid == 0){ //该分支是子进程执行的代码
13
14         printf("Child process id (my parent pid is %d):%d\n",
15             getppid(),getpid());
16         for(int i=0; i<3 ; i++)
17             printf("hello\n");
18     }else{ //该分支是父进程执行的代码
19
20         printf("Parent process id :%d\n", getpid());
21         for(int i=0; i<3 ; i++)
22             printf("world\n");
23     }
24
25     return 0;
26 }
```

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。



上图解释了fork的工作流程，请大家参照代码仔细理解。

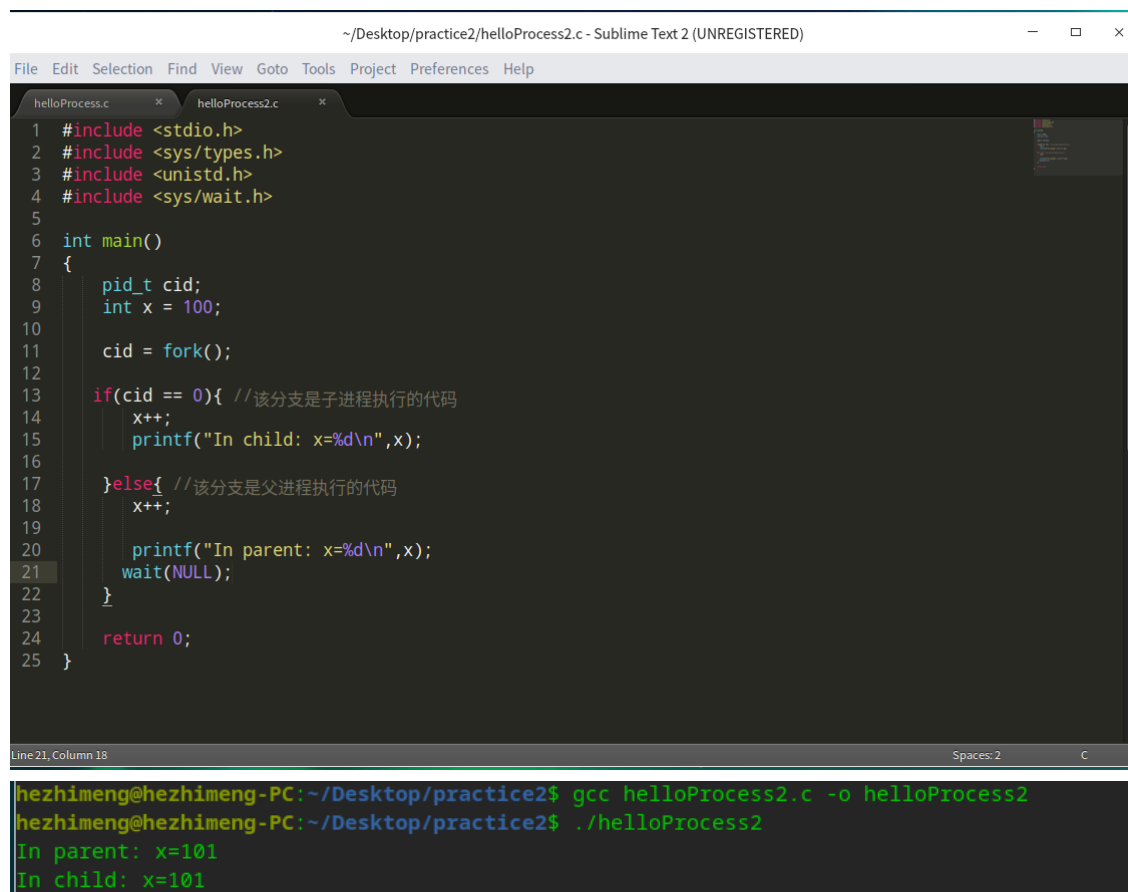
```
hezhimeng@hezhimeng-PC:~/Desktop/practice2$ gcc helloProcess.c -o helloProcess
hezhimeng@hezhimeng-PC:~/Desktop/practice2$ ./helloProcess
Before fork process id :3667
Parent process id :3667
world
world
world
Child process id (my parent pid is 3667):3668
hello
hello
hello
```

当执行300次时：

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。

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);
```



The screenshot shows a Sublime Text editor window titled "helloProcess2.c - Sublime Text 2 (UNREGISTERED)". The editor displays a C program in a dark theme. The code is as follows:

```
1 #include <stdio.h>
2 #include <sys/types.h>
3 #include <unistd.h>
4 #include <sys/wait.h>
5
6 int main()
7 {
8     pid_t cid;
9     int x = 100;
10
11     cid = fork();
12
13     if(cid == 0){ //该分支是子进程执行的代码
14         x++;
15         printf("In child: x=%d\n",x);
16     }else{ //该分支是父进程执行的代码
17         x++;
18         printf("In parent: x=%d\n",x);
19         wait(NULL);
20     }
21 }
22
23
24 return 0;
25 }
```

Below the editor, a terminal window shows the compilation and execution of the program:

```
hezhimeng@hezhimeng-PC:~/Desktop/practice2$ gcc helloProcess2.c -o helloProcess2
hezhimeng@hezhimeng-PC:~/Desktop/practice2$ ./helloProcess2
In parent: x=101
In child: x=101
```

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);
```

```
~/Desktop/practice2/helloProcess2.c - Sublime Text 2 (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

helloProcess.c x helloProcess2.c x
1 #include <stdio.h>
2 #include <sys/types.h>
3 #include <unistd.h>
4 #include <sys/wait.h>
5
6 int main()
7 {
8     pid_t cid;
9     int x = 100;
10
11     cid = fork();
12
13     if(cid == 0){ //该分支是子进程执行的代码
14         x++;
15         printf("In child: x=%d\n",x);
16         sleep(3);
17     }else{ //该分支是父进程执行的代码
18         x++;
19         wait(NULL);
20         printf("In parent: x=%d\n",x);
21     }
22
23     return 0;
24 }
```

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。

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。

```
1 #include <sys/types.h>
2 #include <unistd.h>
3 #include <stdio.h>
4 #include <pthread.h>
5
6 void* threadFunc(void* arg){ //线程函数
7
8     printf("In NEW thread\n");
9
10 }
11
12 int main()
13 {
14     pthread_t tid;
15
16     pthread_create(&tid, NULL, threadFunc, NULL);
17
18     //pthread_join(tid, NULL);
19
20     printf("In main thread\n");
21
22     return 0;
23 }
```

编译该段代码时，请注意gcc要加入新的参数，命令如下：

```
1 gcc helloThread.c -o helloThread -l pthread
```

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。

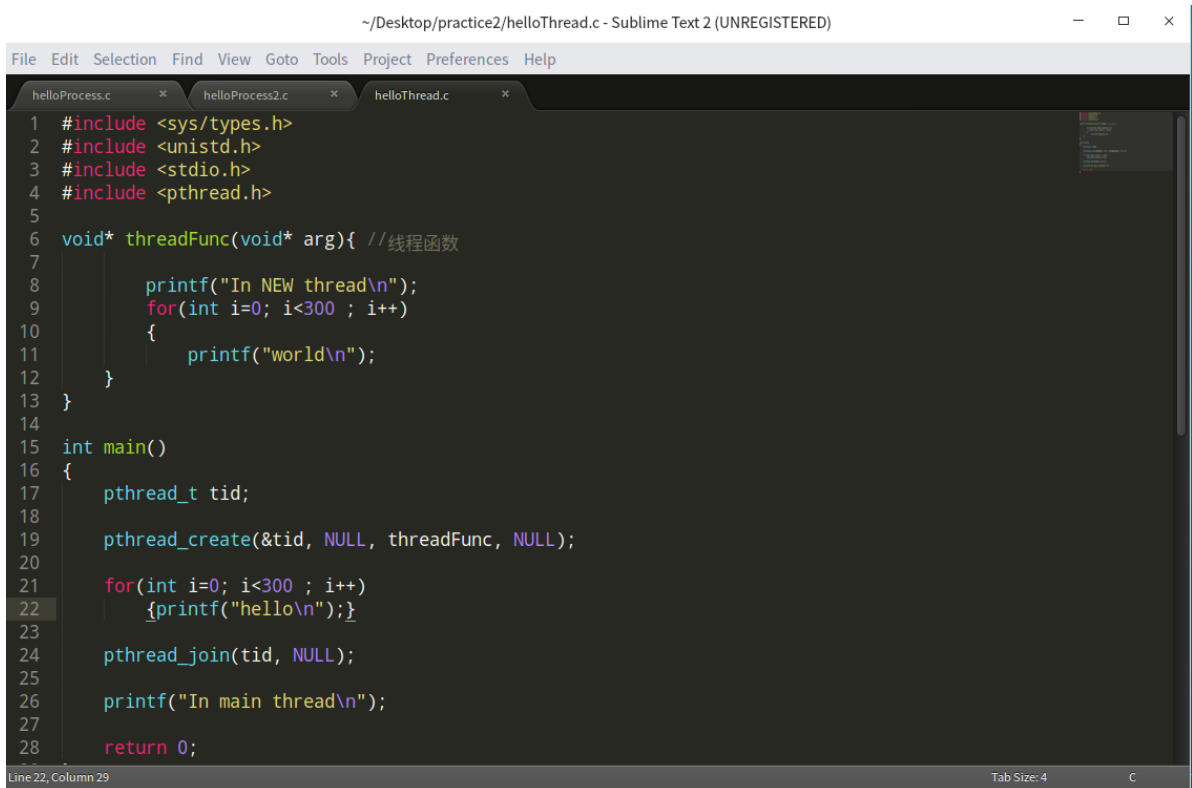
```
hezhimeng@hezhimeng-PC:~/Desktop/practice2$ touch helloThread.c
hezhimeng@hezhimeng-PC:~/Desktop/practice2$ gcc helloThread.c -o helloThread -l pthread
hezhimeng@hezhimeng-PC:~/Desktop/practice2$ ./helloThread
In main thread
hezhimeng@hezhimeng-PC:~/Desktop/practice2$
```

现象：创建了线程但是线程中的语句未被执行。

```
hezhimeng@hezhimeng-PC:~/Desktop/practice2$ gcc helloThread.c -o helloThread -l pthread
hezhimeng@hezhimeng-PC:~/Desktop/practice2$ ./helloThread
In NEW thread
In main thread
hezhimeng@hezhimeng-PC:~/Desktop/practice2$
```

现象：线程中的语句被成功执行。

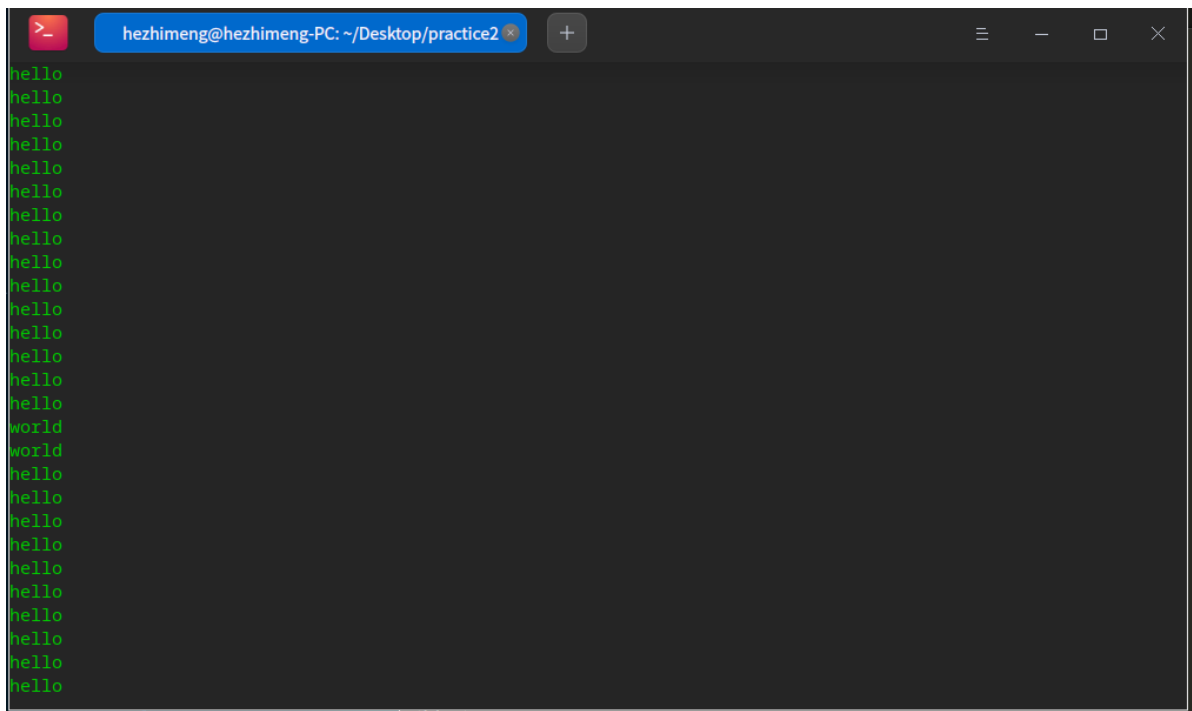
原因：pthread_join函数等待指定线程结束。父进程与子进程是相互独立的，如果在执行主函数时父进程不进行等待，父进程就会自行执行完毕，而相对应的子进程并没有被执行就直接终止。



```
~/Desktop/practice2/helloThread.c - Sublime Text 2 (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
helloProcess.c x helloProcess2.c x helloThread.c x
1 #include <sys/types.h>
2 #include <unistd.h>
3 #include <stdio.h>
4 #include <pthread.h>
5
6 void* threadFunc(void* arg){ //线程函数
7
8     printf("In NEW thread\n");
9     for(int i=0; i<300 ; i++)
10     {
11         printf("world\n");
12     }
13 }
14
15 int main()
16 {
17     pthread_t tid;
18
19     pthread_create(&tid, NULL, threadFunc, NULL);
20
21     for(int i=0; i<300 ; i++)
22     {printf("hello\n");}
23
24     pthread_join(tid, NULL);
25
26     printf("In main thread\n");
27
28     return 0;

```

Line 22, Column 29 Tab Size: 4 C

A terminal window with a dark background and green text. The window title bar shows the user 'hezhimeng' on a PC at the directory '~/Desktop/practice2'. The terminal output consists of 20 lines: 18 'hello' lines and 2 'world' lines. The lines are interleaved, with 'hello' appearing on odd-numbered lines (1, 3, 5, etc.) and 'world' appearing on even-numbered lines (2, 4, 6, etc.).

```
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
world
world
hello
hello
hello
hello
hello
hello
hello
hello
hello
```

结果：输出效果与并发父子进程的执行效果相似。

四、总结

无