

《Linux操作系统》实验2：进程及线程创建

班级：网安1901 学号：201904080130 姓名：杨城昂

一、实验目的

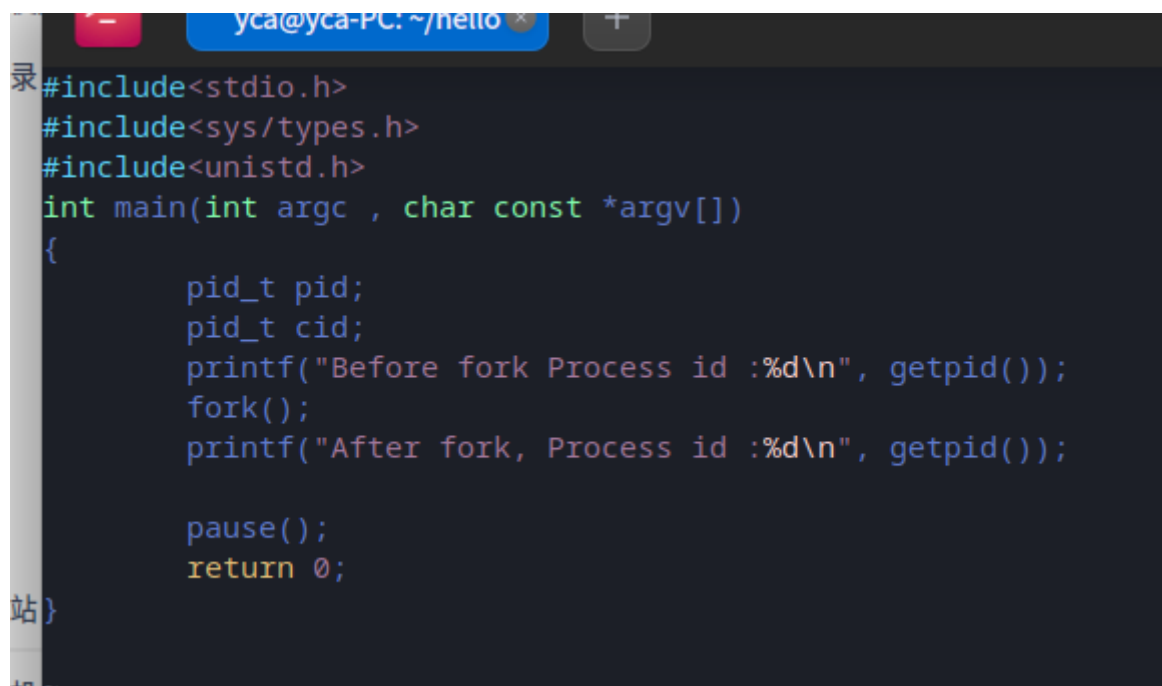
理解创建子进程函数的fork()的用法，通过观察运行结果理解进程的基本特征；通过代码及运行结果理解线程的概念，能够理解进程与线程之间的关联。

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的步骤，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，想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

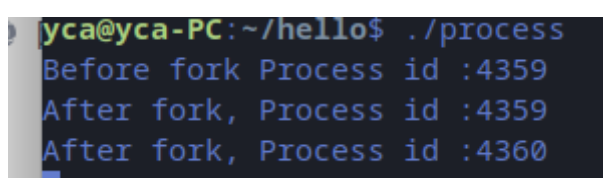
1. 使用编辑器vi新建一个process.c源文件，并输入后面的范例代码。



```
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>
int main(int argc , char const *argv[])
{
    pid_t pid;
    pid_t cid;
    printf("Before fork Process id :%d\n", getpid());
    fork();
    printf("After fork, Process id :%d\n", getpid());

    pause();
    return 0;
}
```

保存退出vi，使用gcc对源文件进行编译，然后运行，观察结果并解释原因。



```
yca@yca-PC:~/hello$ ./process
Before fork Process id :4359
After fork, Process id :4359
After fork, Process id :4360
```

Before fork 输出的是第一个进程id。

第一个After fork输出的是父进程的id，第二个输出的则是子进程的id，两者是并发进行的。

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pause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。

```
pause();
```

重新编译运行程序，开启一个新的终端窗口输入ps -al并观察运行结果。

```
yca@yca-PC:~/hello$ ps -al
F S  UID      PID   PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S   1000     4359    3850  0  80   0  -   569 ia32_s pts/0        00:00:00 process
1 S   1000     4360    4359  0  80   0  -   569 ia32_s pts/0        00:00:00 process
0 R   1000     4361    4199  0  80   0  -   2896 -      pts/1        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。

```
>_ yca@yca-PC: ~/hello +
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

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

"process.c" 25L, 614C
```

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。

3次，没出现

```
yca@yca-PC:~/hello$ gcc process.c -o process
yca@yca-PC:~/hello$ ./process
Before fork process id :5128
Parent process id :5128
world
world
world
Child process id (my parent pid is 5128):5129
hello
hello
hello
```



A screenshot of a terminal window with a dark background. The window title bar at the top shows the text "yca@yca-PC: ~/hello" and standard window control buttons. On the left side, there is a vertical sidebar with various icons. The main area of the terminal displays the word "hello" printed 20 times, one on each line. At the bottom of the terminal, the prompt "yca@yca-PC: ~/hello\$" is visible, indicating the current directory is ~/hello.



A screenshot of a terminal window with a dark background. On the left side, there is a vertical sidebar containing several icons: a hamburger menu, a list of dots, a musical note, a picture icon, and a document icon. The terminal output consists of a repeating sequence of the words "world" and "hello" on alternating lines. The text is rendered in a light blue or cyan monospaced font.

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

- ```
yca@yca-PC:~/hello$ gcc process2.c -o process2
yca@yca-PC:~/hello$./process2
In parent: x=101
In child: x=99
```

### 5. 当输入sleep () 函数后

出现了

```
Child process id (my parent pid is 1):4377
hello
```

父进程的id变为1 的情况这是因为，父进程先结束了返回了return，导致子进程变为孤儿进程，托管给了系统的进程，所以id为1

6. 解决方法就是在父进程的运行代码中加入wait () 函数，等待子进程先运算结束后在继续运行父进程。

```
}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");
 wait(NULL);
 //让自身变为wait状态，当子进程结束后，再进入ready队列
}
```

结果

```
world
Child process id (my parent pid is 4589):4590
hello
hello
```

7. 创建线程。先关闭先前的文件，vi 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;
}
```

运行后

```
yca@yca-PC:~/hello$ vi thread.c
+ yca@yca-PC:~/hello$ gcc thread.c -o thread -l pthread
yca@yca-PC:~/hello$./ thread
bash: ./: 是一个目录
yca@yca-PC:~/hello$./thread
In main thread
```

删除注释标记

```
yca@yca-PC:~/hello$ gcc thread.c -o thread -l pthread
yca@yca-PC:~/hello$./thread
In NEW thread
In main thread
```

原因是主线程结束了，子线程消失了。

尝试循环

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

void* threadFunc(void* arg){

 printf("In NEW thread\n");
 for (int i = 0; i < 3; ++i)
 {
 printf("World\n");
 }
}

int main()
{
 pthread_t tid;

 pthread_create(&tid, NULL, threadFunc, NULL);

 printf("In main thread\n");

 for (int i = 0; i < 3; ++i)
 {
 printf("Hello\n");
 }
 pthread_join(tid, NULL);
 return 0;
}
```

运行

```
imfengyuan@kevinz:~/code/exp2$ gcc helloThread.c -o helloThread -pthread
imfengyuan@kevinz:~/code/exp2$./helloThread
In main thread
Hello
Hello
Hello
In NEW thread
World
World
World
```

改变循环次数，加入sleep

```
Hello
World
World
Hello
World
Hello
World
World
Hello
World
Hello
World
World
Hello
```

效果与并发相似。

## 总结

---

1. fork () 产生的子进程和父进程是并发进行的，相互独立，不影响，父进程结束，子进程就不会运行
2. 线程是cpu的基本单位，线程之间是并发的
3. 子线程是基于进程的，进程如果退出，线程就会退出