

# 实验2:进程及线程创建

网安1901 201904080142 刘润东

## 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

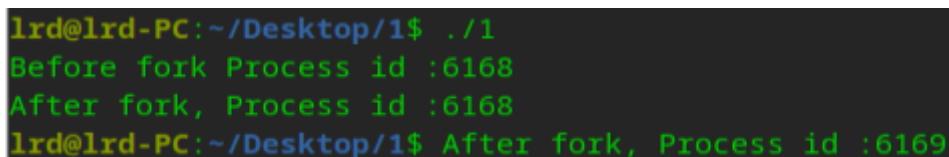
本次实验属于验证型实验，按照实验内容的指导完成所有实验步骤，并记录下实验结果，遇到不懂的问题或是在某一步骤上卡壳，先尝试在搜索引擎上寻找解决方法，积极与老师、同学沟通，务必亲自将实验完成。

## 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 pid,cid;
    printf("Before fork Process id :%d\n",getpid());
    cid = fork();
    printf("After fork, Process id :%d\n",getpid());
    return 0;
}
```



```
lrd@lrd-PC:~/Desktop/1$ ./1
Before fork Process id :6168
After fork, Process id :6168
lrd@lrd-PC:~/Desktop/1$ After fork, Process id :6169
```

fork()会创建一个子进程，子进程会复制fork()后的所有语句，父进程和子进程会并发执行相同的代码

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。

```
#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());
    int i;
    scanf("%d",&i);
    return 0;
}

```

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。

```

lrd@lrd-PC:~/Desktop/1$ ./1
Before fork Process id :6269
After fork, Process id :6269
After fork, Process id :6270

```

```

lrd@lrd-PC:~/Desktop/1$ ps -al
F S  UID    PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000    6269   6247  0  80   0 -   569 wait_w pts/3        00:00:00 1
1 S  1000    6270   6269  0  80   0 -   569 n_tty_ pts/3        00:00:00 1
0 R  1000    6274   6271  0  80   0 -  2896 -      pts/4        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。

```

#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){

        printf("Child process id (my parent pid is %d):%d\n",
getppid(),getpid());
        for(int i=0; i<3; i++)
            printf("hello\n");

    }else{

        printf("Parent process id :%d\n", getpid());
        for(int i=0; i<3; i++)
            printf("world\n");
        wait(NULL);
    }

    return 0;
}

```

```
lrd@lrd-PC:~/Desktop/1$ gcc 2.c -o 2
lrd@lrd-PC:~/Desktop/1$ ./2
Before fork process id :6480
Parent process id :6480
world
world
world
lrd@lrd-PC:~/Desktop/1$ Child process id (my parent pid is 1):6481
hello
hello
hello
```

父进程先于子进程运行完毕时，会将子进程移交给pid=1的进程，在else语句中加入wait(NULL)让父进程等待子进程，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）。

```
lrd@lrd-PC:~/Desktop/1$ gcc 2.c -o 2
2.c: In function 'main':
2.c:23:9: warning: implicit declaration of function 'wait'; did you mean 'main'? [-Wimplicit-function-declaration]
    wait(NULL);
    ^~~~
    main
lrd@lrd-PC:~/Desktop/1$ ./2
Before fork process id :6828
Parent process id :6828
world
world
world
Child process id (my parent pid is 6828):6829
hello
hello
hello
```

将循环次数改成30000，观察

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

说明父进程和子进程是并发执行的

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++;
    }
```

```

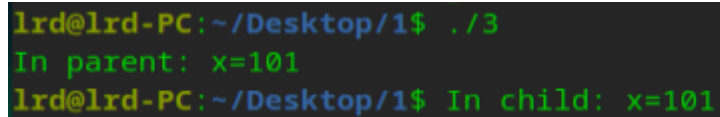
        printf("In child: x=%d\n",x);
    }else{ //该分支是父进程执行的代码
        x++;

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

    }

    return 0;
}

```



```

lrd@lrd-PC:~/Desktop/1$ ./3
In parent: x=101
lrd@lrd-PC:~/Desktop/1$ In child: x=101

```

说明父进程与子进程的存储空间相互独立，子进程是复制了一份父进程

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 <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){ //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);
        wait(NULL);
    }

    return 0;
}

```

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：

```

#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){ //该分支是子进程执行的代码
        x++;

        printf("In child: x=%d\n",x);
        sleep(3);
    }else{ //该分支是父进程执行的代码
        x++;

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

    }

    return 0;
}

```

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。

```

#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
```

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？

```
lrd@lrd-PC:~/Desktop/1$ ./4  
In main thread
```

```
lrd@lrd-PC:~/Desktop/1$ ./4  
In NEW thread  
In main thread
```

主线程没有等待子线程的进行。

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。

是并发执行，但线程中全局函数是共享参数。

## 四、总结

---

无