

《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 pid,cid;
    printf("Before fork Process id :%d\n",getpid());
    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。

```
sui Fengdang@suifengdang-PC:~/Desktop$ vi helloProcess.c
sui Fengdang@suifengdang-PC:~/Desktop$ gcc helloProcess.c -o helloProcess
sui Fengdang@suifengdang-PC:~/Desktop$ ./helloProcess
Before fork Process id :8675
After fork,Process id :8675
After fork,Process id :8676
```

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。

```
sui Fengdang@sui Fengdang-PC:~/Desktop$ ./helloProcess
Before fork Process id :9076
After fork,Process id :9076
After fork,Process id :9077
```

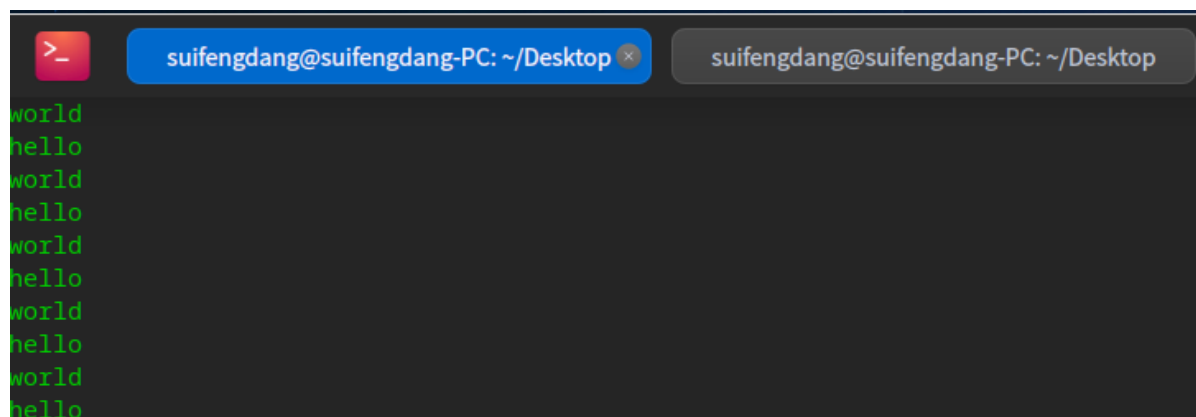
```
sui Fengdang@sui Fengdang-PC:~/Desktop$ ps -al
  S      UID      PID     PPID    C  PRI   NI   ADDR  SZ  WCHAN   TTY          TIME CM
0  S    1000     9076     8079    0   80    0    -    570 wait_w pts/0      00:00:00 he
lloProcess
1  S    1000     9077     9076    0   80    0    -    570 n_tty_ pts/0      00:00:00 he
lloProcess
0  R    1000     9090     9078    0   80    0    -    2897 -      pts/1      00:00:00 ps
```

ps -al

1. 通过判断fork的返回值让父子进程执行不同的语句。

```
sui Fengdang@sui Fengdang-PC:~/Desktop$ touch li.c
sui Fengdang@sui Fengdang-PC:~/Desktop$ vi li.c
sui Fengdang@sui Fengdang-PC:~/Desktop$ gcc li.c -o li
sui Fengdang@sui Fengdang-PC:~/Desktop$ ./li
Before fork process id :9290
Parent process id :9290
world
world
world
Child process id (my parent pid is 1):9291
hello
hello
hello
```

改循环为3000



```
>_ sui Fengdang@sui Fengdang-PC: ~/Desktop sui Fengdang@sui Fengdang-PC: ~/Desktop
world
hello
world
hello
world
hello
world
hello
world
hello
```

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。

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。

```
sui Fengdang@sui Fengdang-PC:~/Desktop$ vi zh.c
sui Fengdang@sui Fengdang-PC:~/Desktop$ gcc zh.c -o zh
sui Fengdang@sui Fengdang-PC:~/Desktop$ ./zh
In parent: x=101
sui Fengdang@sui Fengdang-PC:~/Desktop$ 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

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);
```

```
#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
sleep(3);    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。

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;
}
```

```
#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");
}
```

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
```

```
-
sui Fengdang@suifengdang-PC:~/Desktop$ gcc helloThread.c -o helloThread -l pthread
sui Fengdang@suifengdang-PC:~/Desktop$ ./helloThread
In main thread
sui Fengdang@suifengdang-PC:~/Desktop$
```

去注释后

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

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。

四、实验报告

1. 将本次实验内容的全过程写在报告中，每一个步骤的输出结果均要截图；
2. 报告在word中编辑，输出成pdf格式，**文件名：学号-姓名-实验X.pdf** [注：X为实验的编号，本次实验为1，以此类推。**文件名命名错误的最后成绩下降一个等级**]
3. 本次报告的最后提交时间如下，请学委在此时间后立即把班级作业和实验压缩好通过QQ传给我：
 - a. 3班：10月17日 18：00
 - b. 2班：10月18日 18：00
 - c. 1班：10月19日 18：00