《linux操作系统》实验2：进程及线程创建

网安1901 201904080121 刘步云

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sublime新建一个hellopocess.c源文件，并输入以下代码。

- 实验过程：

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

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

- 保存退出sublime，使用gcc对源文件进行编译，然后运行，观察结果并解释原因。

```
lll@lll-PC:~/Desktop$ gcc helloprocess.c -o ll
lll@lll-PC:~/Desktop$ ./ll
Before fork Process id:36615
After fork, Process id:36615
lll@lll-PC:~/Desktop$ After fork, Process id:36616
```

- 原因：Fork()将当前进程完整拷贝了一份，子进程从fork()语句后开始执行

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);
```

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

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

- 保存并退出，使用gcc编译helloProcess.c，然后执行

```
lll@lll-PC:~/Desktop$ gcc helloprocess.c -o ll
lll@lll-PC:~/Desktop$ ./ll
Before fork Process id:36745
After fork, Process id:36745
After fork, Process id:36746
```

- 另外打开一个终端，输入

  ps -al

```
lll@lll-PC:~$ ps -al
F S   UID    PID   PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CM
D
0 S  1000  36745  36495  0  80   0 -   569 wait_w pts/0    00:00:00 ll
1 S  1000  36746  36745  0  80   0 -   569 n_tty_ pts/0    00:00:00 ll
0 R  1000  36962  36958  0  80   0 -  2896 -      pts/1    00:00:00 ps
lll@lll-PC:~$
```

- 原因：helloprocess进程因为scanf()函数调用，等待输入设备输入，helloprocess进程进入等待状态

3. 通过判断fork()的返回值让父子进程执行不同的语句。

- 修改helloprocess.c，如下所示

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

int main(int argc,char const *argv[])
{
    pid_t pid,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",getpid().getpid());
        for (int i = 0; i < 3; ++i)
        {
            printf("Hello\n");
            /* code */
        }
        /* code */
    }
    else
    {
        printf("Parent process id:%d\n",getpid());
        for (int i = 0; i < 3; ++i)
        {
            printf("World\n");
            /* code */
        }
    }
    return 0;
}
```

- 保存并退出，使用gcc编译helloprocess 然后执行

```
lll@lll-PC:~/Desktop$ gcc helloprocess.c -o ll
lll@lll-PC:~/Desktop$ ./ll
Before fork Process id:39164
Parent process id:39164
脑瘫
脑瘫
脑瘫
lll@lll-PC:~/Desktop$ Child process id (my parent pid is 39165) : 39165
阿花
阿花
阿花
```

- 修改循环次数至300 如图所示

```
脑瘫
脑瘫
脑瘫
脑瘫
脑瘫
lll@lll-PC:~/Desktop$ Child process id (my parent pid is 39197) : 39197
阿花
阿花
阿花
阿花
阿花
阿花
阿花
阿花
阿花
```

- 原因：父子进程是并发执行的，两个进程交替使用CPU

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。

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

int main()
{
    pid_t cid;
    int x=100;
    cid=fork();

    if(cid==0){
        x++;
        printf("In child:x=%d\n",x );
    }else{
        x++;
        printf("In parent:x=%d\n",x );
    }
    /* code */
    return 0;
}
```

```
lll@lll-PC:~/Desktop$ gcc helloprocess2.c -o ll
lll@lll-PC:~/Desktop$ ./ll
In parent:x=101
lll@lll-PC:~/Desktop$ In child:x=101
```

5. 在上一步中添加如下语句入图

#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语句从当前位置移到printf语句前，并在else语句前加上如下语句

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。

6. 创建线程。先关闭先前的文件，以c.c创建一个新的C语言源文件，将下面的代码拷贝进编辑器

c.c

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

void* threadFunc(void* arg){

    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 c.c -o c -l pthread

- 运行结果

```
lll@lll-PC:~/Desktop/a$ gcc c.c -o c -l pthread
lll@lll-PC:~/Desktop/a$ ./c
In main thread
```

- 去掉注释

```
lll@lll-PC:~/Desktop/a$ gcc c.c -o c -l pthread
lll@lll-PC:~/Desktop/a$ ./c
In NEW thread
In main thread
```

- 原因：主函数使用了pthread_create创建了子线程，如果主进程不等待子线程，则进程结束，所有线程也结束，线程依赖进程。

四、总结

1.fork()函数的使用。

2.进程是并发执行的。

3.线程是CPU的基本单位，且线程的运行是并发的。

4.线程依赖于进程。