

实验2：进程及线程创建

班级：网安1901 学号：201904080125 姓名：汪弋猛

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.使用编辑器新建一个.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;
    //getpid()函数返回当前进程的id号
    printf("Before fork Process id :%d\n", getpid());
    /*
        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。
        fork()的返回值：
            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。
            如果创建失败，返回值为-1。
    */
    cid = fork();
    printf("After fork, Process id :%d\n", getpid());
    return 0;
}
```

```
wym@wym-PC: ~/Desktop
wym@wym-PC:~$ cd Desktop
wym@wym-PC:~/Desktop$ gcc ex1.c -o ex1
wym@wym-PC:~/Desktop$ ./ex1
Before fork Process id :9223
After fork, Process id :9223
wym@wym-PC:~/Desktop$ After fork, Process id :9224
```

出现上面结果的原因是fork函数产生了一个子进程，和父进程并发执行了，其pid是父进程的pid+1

2.加入pause函数,练习ps命令

```
ps -al
```

```
wym@wym-PC: ~/Desktop
wym@wym-PC:~/Desktop$ ps -al
 F S      UID      PID   PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
 0 S      1000     9405    9214  0  80   0 -   569 ia32_s pts/0        00:00:00 ex1
 1 S      1000     9406    9405  0  80   0 -   569 ia32_s pts/0        00:00:00 ex1
 0 R      1000     9411    9407  0  80   0 -  2896 -          pts/1        00:00:00 ps
wym@wym-PC:~/Desktop$
```

可以看出ex1的父进程和子进程

3.通过判断fork的返回值让父子进程执行不同的语句

```
#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){ //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;
}
```

```
wym@wym-PC:~/Desktop$ gcc ex1.c -o ex1
wym@wym-PC:~/Desktop$ ./ex1
Before fork process id :9562
Parent process id :9562
world
world
world
wym@wym-PC:~/Desktop$ Child process id (my parent pid is 1):9563
hello
hello
hello
```

通过比较ID，父子进程判断正确，增加循环次数使得运行时间延长，可以看到父子进程交替输出，验证了并发执行

```
>_ wym@wym-PC: ~/D... wym@wym-PC: ~/D... + - □ ×
world
hello
hello
world
world
hello
world
hello
world
```

4.验证父子进程间的内存空间是相互独立的

```
#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;
}
```

```
>_ wym@wym-PC: ~/D... wym@wym-PC: ~/D... + - □ ×
wym@wym-PC:~/Desktop$ gcc ex1.c -o ex1
wym@wym-PC:~/Desktop$ ./ex1
In parent: x=99
wym@wym-PC:~/Desktop$ In child: x=101
```

父子进程运行相互独立，所以分别输出了自己代码的结果

5.在合适位置分别加入下列代码

```
#include <sys/wait.h>
wait(NULL);
sleep(3)
```

可以看到父进程等待到子进程结束才执行printf

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

```
#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");

}

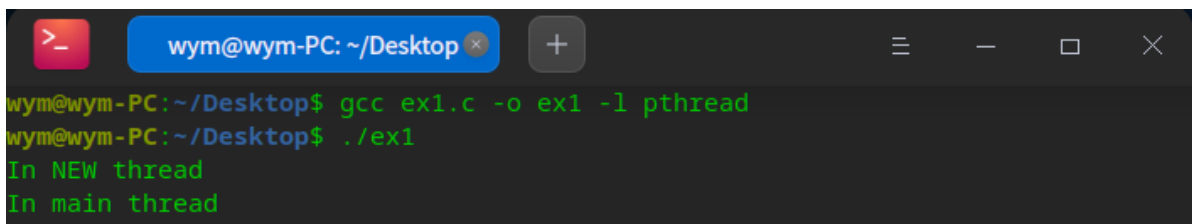
int main()
{
    pthread_t tid;

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

    pthread_join(tid, NULL);

    printf("In main thread\n");

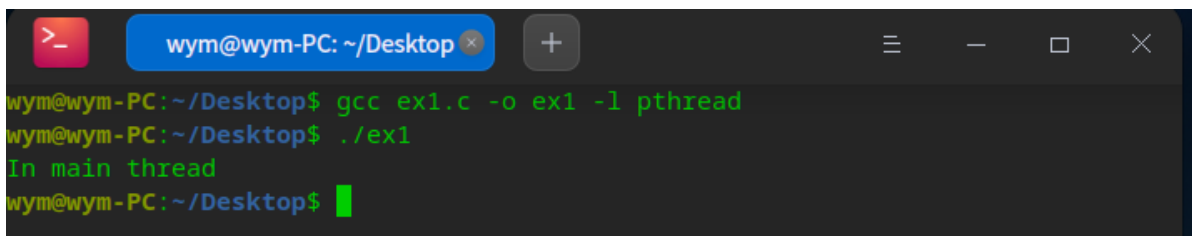
    return 0;
}
```

A terminal window titled 'wym@wym-PC: ~/Desktop' with a dark background and green text. It shows the compilation of 'ex1.c' with pthread support and its execution. The output shows 'In NEW thread' followed by 'In main thread' on the next line, indicating the child thread completed before the parent thread continued.

```
wym@wym-PC: ~/Desktop
wym@wym-PC:~/Desktop$ gcc ex1.c -o ex1 -l pthread
wym@wym-PC:~/Desktop$ ./ex1
In NEW thread
In main thread
```

可以看到线程创建运行成功

如果去掉pthread_join一行，可能会出现新创建的线程没有运行的情况

A terminal window titled 'wym@wym-PC: ~/Desktop' with a dark background and green text. It shows the compilation of 'ex1.c' with pthread support and its execution. The output shows 'In main thread' followed by a blank line, indicating the child thread was created but did not run before the parent thread continued.

```
wym@wym-PC:~/Desktop$ gcc ex1.c -o ex1 -l pthread
wym@wym-PC:~/Desktop$ ./ex1
In main thread
wym@wym-PC:~/Desktop$
```

这是因为主线程运行完毕后直接返回，整个程序结束了，此时新线程还没来得及运行

四、总结

无