java当中的线程和操作系统的线程是什么关系?

关于操作系统的线程 linux操作系统的线程控制原语

```
int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
            void *(*start_routine) (void *), void *arg);
```

可以在linux系统下面通过man手册查看该函数的定义

根据man配置的信息可以得出pthread_create会创建一个线程，这个函数是linux系统的函数，可以用C或者C++直接调用，上面信息也告诉程序员这个函数在pthread.h， 这个函数有四个参数

参数名字	参数定义	参数解释
pthread_t *thread	传出参数，调用之后会传出被创建线程的id	定义 pthread_t pid; 继而 取地址 &pid
const pthread_attr_t *attr	线程属性，关于线程属性是linux的知识	一般传NULL，保持默认属性
void (start_routine) (void *)	线程的启动后的主体函数	需要你定义一个函数，然后传函数名即可
void *arg	主体函数的参数	没有可以传nulll

linux上启动一个线程的代码：

```
//头文件
#include <pthread.h>
#include <stdio.h>
//定义一个变量，接受创建线程后的线程id
pthread_t pid;
//定义线程的主体函数
```

```c
void* thread_entity(void* arg) {
    printf("i am new Thread! from c");
}
//main方法，程序入口，main和java的main一样会产生一个进程，继而产生一个main线程
int main() {
    //调用操作系统的函数创建线程，注意四个参数
    pthread_create(&pid,NULL,thread_entity,NULL);
    //usleep是睡眠的意思，那么这里的睡眠是让谁睡眠呢？
    //为什么需要睡眠？如果不睡眠会出现什么情况
    usleep(100);
    printf("main\n");
}
```

假设有了上面知识的铺垫，那么可以试想一下java的线程模型到底是什么情况呢?

在java代码里启动一个线程的代码

```java
public class Example4Start {
    public static void main(String[] args) {
        Thread thread = new Thread(){
            @Override
            public void run() {
                System.out.println("i am new Thread! from java ");
            }
        };
        thread.start();
    }
}
```

这里启动的线程和上面我们通过linux的pthread_create函数启动的线程有什么关系呢？只能去可以查看start()的源码了,看看java的start()到底干了什么事才能对比出来。start方法的源码的部分截图

可以看到这个方法最核心的就是调用了一个start0方法，而start0方法又是一个native方法，故而如果要搞明白start0我们需要查看Hotspot的源码，好吧那我们就来看一下Hotspot的源码吧，Hotspot的源码怎么看么？一般直接看openjdk的源码，openjdk的源码如何查看、编译调试？openjdk的编译我们后面会讨论，在没有openjdk的情况下，我们做一个大胆的猜测，java级别的线程其实就是操作系统级别的线程，什么意思呢？说白了我们大胆猜想 star----->start0----------->ptherad_create

我们鉴于这个猜想来模拟实现一下java启动线程

```
public class EnjoyThread {
    public static void main(String[] args) {
        //自己定义的一个类
        EnjoyThread enjoythread =new EnjoyThread();
        enjoythread.start0();
    }
    //本地方法
    private native void start0();
}
```

这里我们让自己写的start0调用一个本地方法，在本地方法里面去启动一个系统线程，好吧我们写一个c程序来启动本地线程

本地方法的代码编写

```
#include <pthread.h>
#include <stdio.h>
//定义变量接受线程id
pthread_t pid;
//线程的主体方法相当于 java当中的run
void* thread_entity(void* arg) {
    //子线程死循环
    while(1){
        //睡眠100毫秒
        usleep(100);
        //打印
        printf("I am new Thread\n");
    }
}
//c语言的主方法入口方法，相当于java的main
int main() {
    //调用linux的系统的函数创建一个线程
    pthread_create(&pid,NULL,thread_entity,NULL);
    //主线程死循环
    while(1){
        //睡眠100毫秒
        usleep(100);
        //打印
        printf("I am  main\n");
    }
}
```

在linux上编译、运行上述C程序

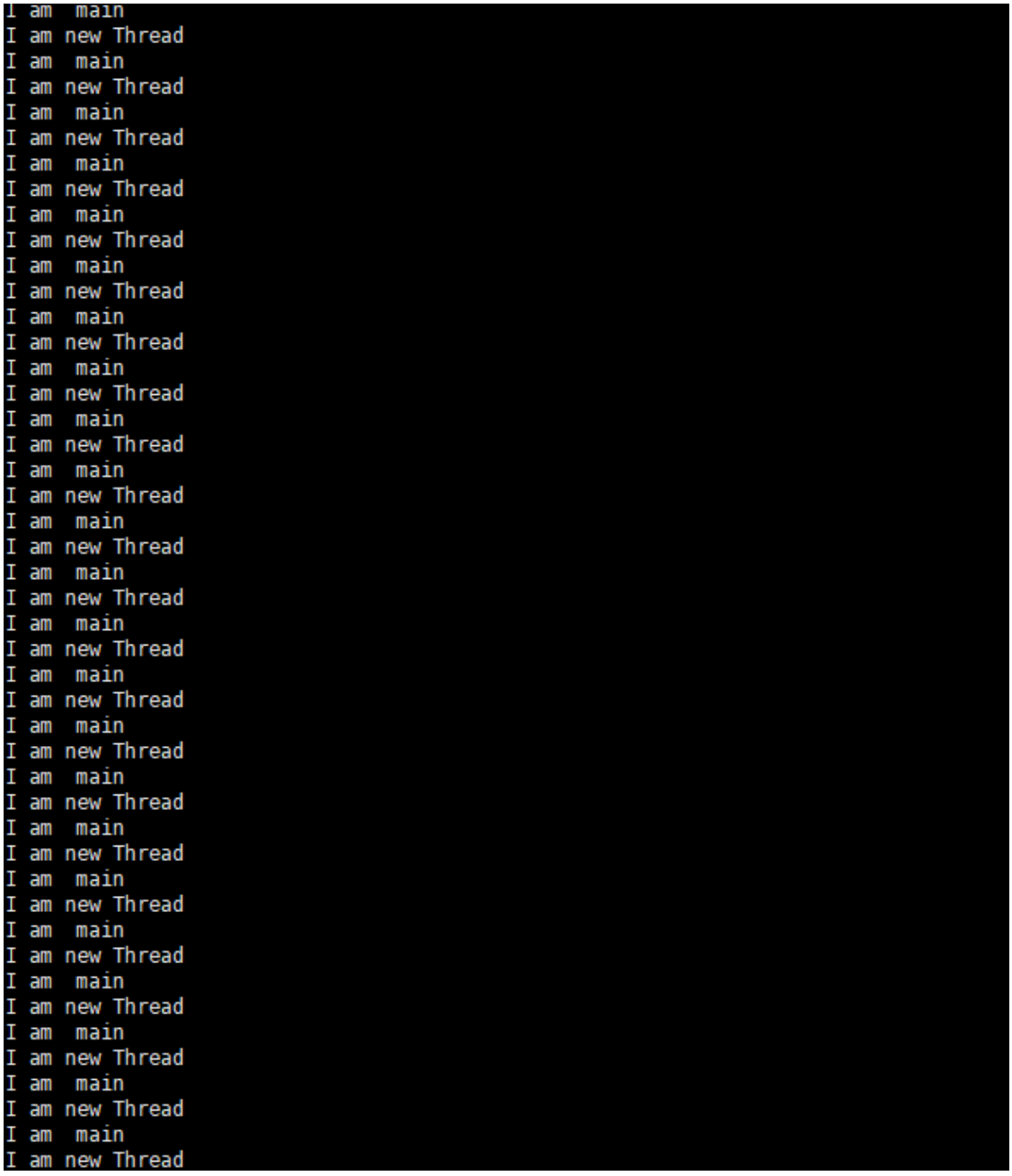
编译这个程序

`gcc thread.c -o thread.out -pthread`

运行这个程序

`./thread.out`

结果如图所示

- 

结果是两个线程一直在交替执行，得到我们预期的结果。现在的问题就是我们如何通过start0调用这个c程序，这里就要用到JNI了

java利用JNI调用本地方法

```
package com.enjoy.concurrency;

public class EnjoyThread {
    //装载库，保证JVM在启动的时候就会装载，故而一般是也给static
   static {
      System.loadLibrary( "EnjoyThreadNative" );
   }
   public static void main(String[] args) {
      EnjoyThread enjoyThread =new EnjoyThread();
      enjoyThread.start0();
   }
   private native void start0();
}
```

大家可以提前预习一下JNI的知识，这样我们就可以完全还原java当中调用JVM源码启动线程的场景，继而可以系统的了解java线程的模型和本质