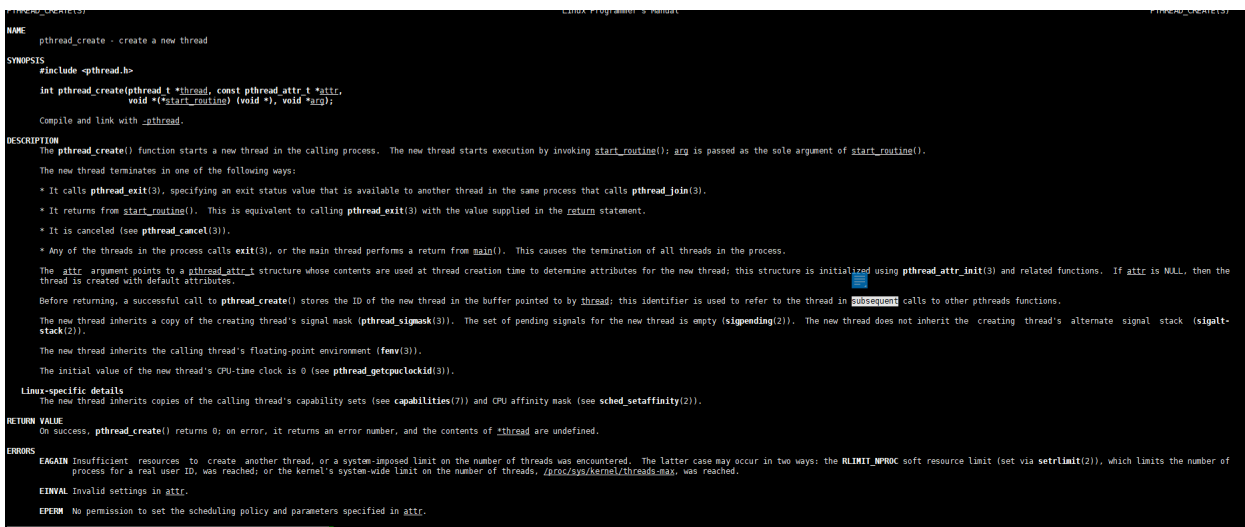


java当中的线程和操作系统的线程是什么关系？

关于操作系统的线程

linux操作系统的线程控制原语

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



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

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	在学习pthread_create函数的时候一般穿NULL，保持默认属性
void *(*start_routine) (void *)	线程的启动后的主体函数 相当于java当中的run	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	如果没有可以传NULL

在linux上启动一个线程的代码：

```
1 #include <pthread.h> //头文件  
2 #include <stdio.h>  
3 pthread_t pid; //定义一个变量，接受创建线程后的线程id  
4 //定义线程的主体函数  
5 void* thread_entity(void* arg)  
6 {  
7     printf("i am new Thread!");  
8 }  
9 //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  
10 int main()
```

```

11 {
12     //调用操作系统的函数创建线程，注意四个参数
13     pthread_create(&pid,NULL,thread_entity,NULL);
14     //usleep是睡眠的意思，那么这里的睡眠是让谁睡眠呢？
15     //为什么需要睡眠？如果不睡眠会出现什么情况
16     usleep(100);
17     printf("main\n");
18 }

```

假设有上面知识的铺垫，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

```

1 public class Example4Start {
2
3     public static void main(String[] args) {
4         Thread thread = new Thread(){
5             @Override
6             public void run() {
7                 System.out.println("i am new Thread!");
8             }
9         };
10        thread.start();
11    }
12 }

```

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 start-start0-pthead_create

我们鉴于这个猜想来模拟实现一下。

```

public synchronized void start() {
    /**
     * This method is not invoked for the main method thread or "system"
     * group threads created/set up by the VM. Any new functionality added
     * to this method in the future may have to also be added to the VM.
     *
     * A zero status value corresponds to state "NEW".
     */
    if (threadStatus != 0)
        throw new IllegalThreadStateException();

    /* Notify the group that this thread is about to be started
     * so that it can be added to the group's list of threads
     * and the group's unstarted count can be decremented. */
    group.add(this);

    boolean started = false;
    try {
        start0();
    }
}

```

```

    } catch (Throwable ignore) {
        /* do nothing. If start0 threw a Throwable then
         it will be passed up the call stack */
    }
}

private native void start0();

/**
 * If this thread was constructed using a separate
 * Runnable run object, then that
 * Runnable object's run method is called;
 * otherwise, this method does nothing and returns.
 *
 * 

* Subclasses of Thread should override this method.
 *
 * @see #start()
 */


```

如何来模拟实现呢？

```

1 public class LubanThread {
2     public static void main(String[] args) {
3         LubanThread lubanThread = new LubanThread();
4         lubanThread.start0();
5     }
}

```

```
6 private native void start0();
7 }
```

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

```
1 #include <pthread.h>
2 #include <stdio.h>
3 pthread_t pid;
4 void* thread_entity(void* arg)
5 {
6     while(1){
7         usleep(100);
8         printf("I am new Thread\n");
9     }
10 }
11 int main()
12 {
13     pthread_create(&pid,NULL,thread_entity,NULL);
14     while(1){
15         usleep(100);
16         printf("I am main\n");
17     }
18 }
```

在linux上编译、运行上述C程序

```
1 gcc thread.c -o thread.out -pthread
2 ./thread.out
```

[illegible]

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，如果你预习了epoll的课（当然epoll我没有讲完，看看大家的接受程度再决定要不要讲完）那么JNI调用就应该懂了

好吧你要是实在不懂，再说一遍

java代码如下:

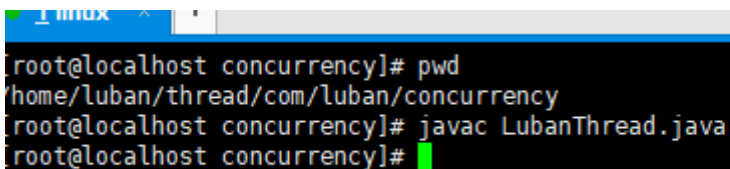
```
1 package com.luban.concurrency;
2
3 public class LubanThread {
4     static {
5         System.loadLibrary( "LubanThreadNative" );
6     }
7 }
```

```
7 public static void main(String[] args) {
8     LubanThread lubanThread =new LubanThread();
9     lubanThread.start0();
10 }
11 private native void start0();
12 }
```

装载库，保证JVM在启动的时候就会装载，故而一般是也给static

System.loadLibrary("LubanThreadNative");

编译成class文件

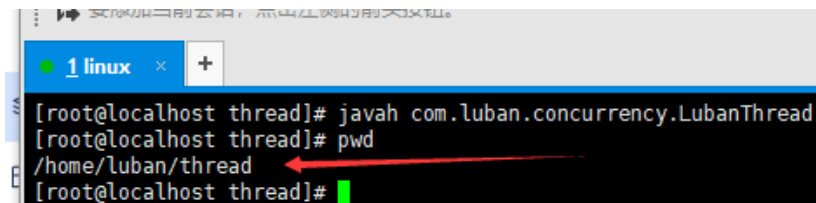


```
root@localhost concurrency]# pwd
/home/luban/thread/com/luban/concurrency
root@localhost concurrency]# javac LubanThread.java
root@localhost concurrency]#
```

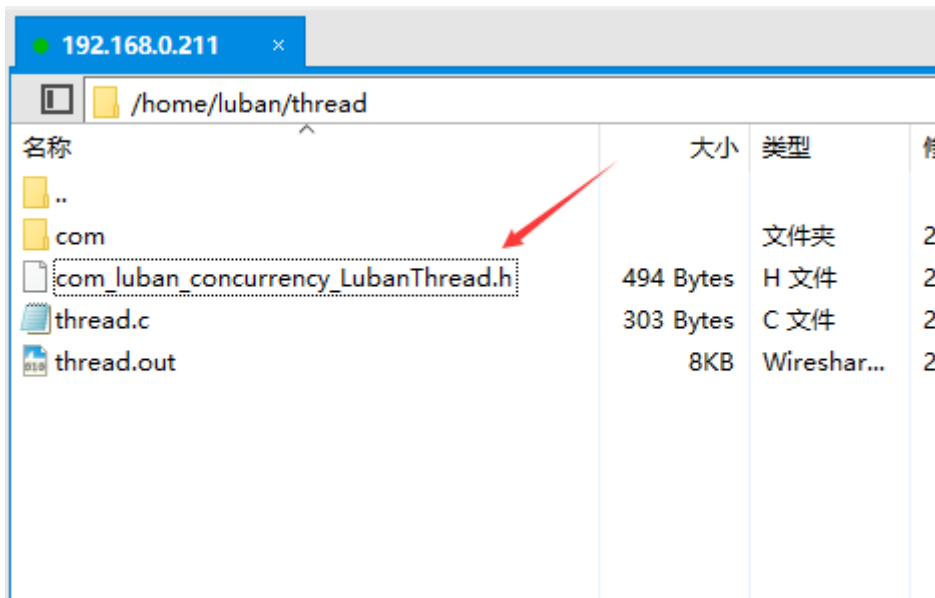
生成.h头文件

javah packageName.className

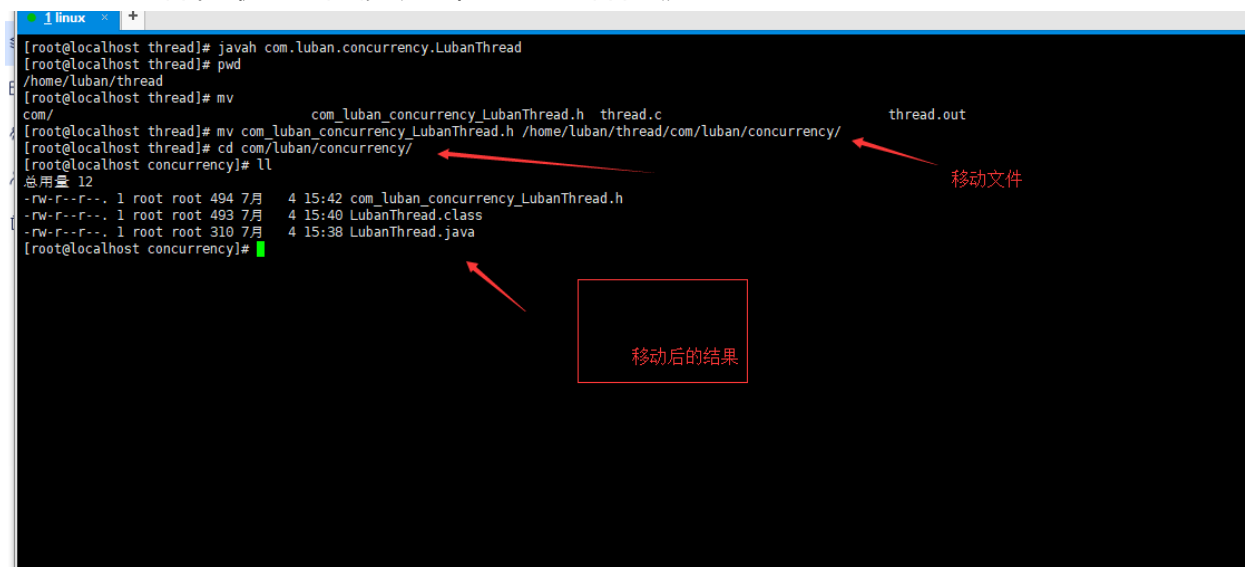
需要注意的运行javah命令得在包外面和编译不一样，编译运行javac得在包当中



```
[root@localhost thread]# javah com.luban.concurrency.LubanThread
[root@localhost thread]# pwd
/home/luban/thread
```



生成的.h文件，最好把他移动到和class文件同级目录吧



继而开始编写C程序，要上面那个thread.c程序上做一点修改，这里我复制一份出来修改，修改的时候需要参考那个这个.h文件，一下是.h文件的内容

```
1 /* DO NOT EDIT THIS FILE - it is machine generated */
2 #include <jni.h>
3 /* Header for class com_luban_concurrency_LubanThread */
4
5 #ifndef _Included_com_luban_concurrency_LubanThread
6 #define _Included_com_luban_concurrency_LubanThread
7 #ifdef __cplusplus
8 extern "C" {
9 #endif
10 /*
11  * Class: com_luban_concurrency_LubanThread
12  * Method: start0
```

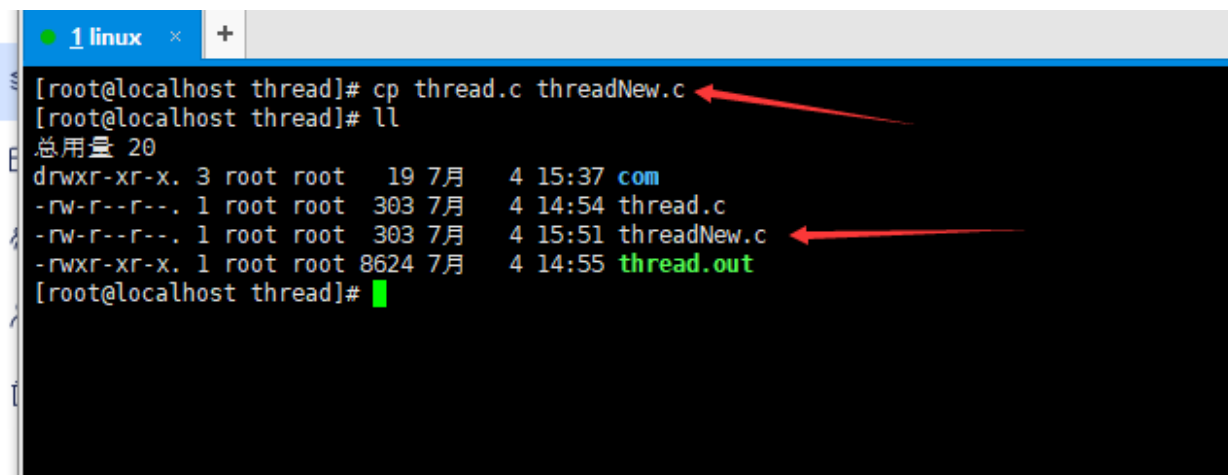
```

13  * Signature: ()V
14  */
15  JNIEXPORT void JNICALL Java_com_luban_concurrency_LubanThread_start0
16  (JNIEnv *, jobject);
17
18  #ifdef __cplusplus
19  }
20  #endif
21  #endif

```

上面第十五代码中的Java_com_luban_concurrency_LubanThread_start0方法就是你需要在C程序中定义的方法。

首先复制一份thread.c



```

[root@localhost thread]# cp thread.c threadNew.c
[root@localhost thread]# ll
总用量 20
drwxr-xr-x. 3 root root   19 7月  4 15:37 com
-rw-r--r--. 1 root root  303 7月  4 14:54 thread.c
-rw-r--r--. 1 root root  303 7月  4 15:51 threadNew.c
-rwxr-xr-x. 1 root root 8624 7月  4 14:55 thread.out
[root@localhost thread]#

```

修改threadNew.c, 定义一个方法Java_com_luban_concurrency_LubanThread_start0, 在方法中启动一个子线程, 代码如下

```

1  #include <pthread.h>
2  #include <stdio.h>
3  #include "com_luban_concurrency_LubanThread.h"//记得导入刚刚编译的那个.h文件
4  pthread_t pid;
5  void* thread_entity(void* arg)
6  {
7      while(1){
8          usleep(100);
9          printf("I am new Thread\n");
10     }
11 }
12
13 //这个方法要参考.h文件的15行代码, 这里的参数得注意, 你写死就行, 不用明白为什么
14 JNIEXPORT void JNICALL Java_com_luban_concurrency_LubanThread_start0
15 (JNIEnv *env, jobject c1){
16     pthread_create(&pid,NULL,thread_entity,NULL);

```



```

17  while(1){
18      usleep(100);
19      printf("I am main\n");
20  }
21  }
22
23  int main()
24  {
25      return 0;
26  }

```

解析类，把这个threadNew.c编译成为一个动态链接库，这样在java代码里会被load到内存libLubanThreadNative这个命名需要注意libxx，xx就等于你java那边写的字符串

```

1  gcc -fPIC -I /usr/lib/jvm/java-1.8.0-openjdk/include -I
   /usr/lib/jvm/java-1.8.0-openjdk/include/linux -shared -o libLubanThreadNative.so threadNew.c

```

做完这一系列事情之后需要把这个.so文件加入到path，这样java才能load到

```


1  export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:{libLubanThreadNative.so}所在的路径

```

```

[root@localhost thread]# export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/luban/thread/
[root@localhost thread]#

```



万事俱备，直接测试，运行我们自己写的那个java类直接测试看看结果能不能启动线程

```

[root@localhost thread]# java com.luban.concurrency.LubanThread

```

回车

[illegible]

牛逼！我们已经通过自己写的一个类，启动了一个线程，但是这个线程函数体是不是java的是C程序的，这个java线程的run方法不同。接下来我们来实现一下这个run
首先在java的代码里面提供一个run方法

```
1 package com.luban.concurrency;
2
3 public class LubanThread {
4     static {
5         System.loadLibrary( "LubanThreadNative" );
6     }
7     public static void main(String[] args) {
8         LubanThread lubanThread =new LubanThread();
9         lubanThread.start0();
10    }
11    //这个run方法，要让C程序员调用到，就完美了
12    public void run(){
13        System.out.println("I am java Thread !!");
14    }
```

```
15  private native void start0();  
16  }
```

所以现在的问题就是让C程序当中的thread_entity方法调用到java当中的run方法？思路同样是jni反调用java方法

```
gcc -o threadNew threadNew.c -I /usr/lib/jvm/java-1.8.0-openjdk/include -I  
/usr/lib/jvm/java-1.8.0-openjdk/include/linux -L /usr/lib/jvm/java-1.8.0-  
openjdk/jre/lib/amd64/server -ljvm -pthread
```

```
LD_LIBRARY_PATH=/usr/lib/jvm/java-1.8.0-openjdk/jre/lib/amd64/server ./a
```