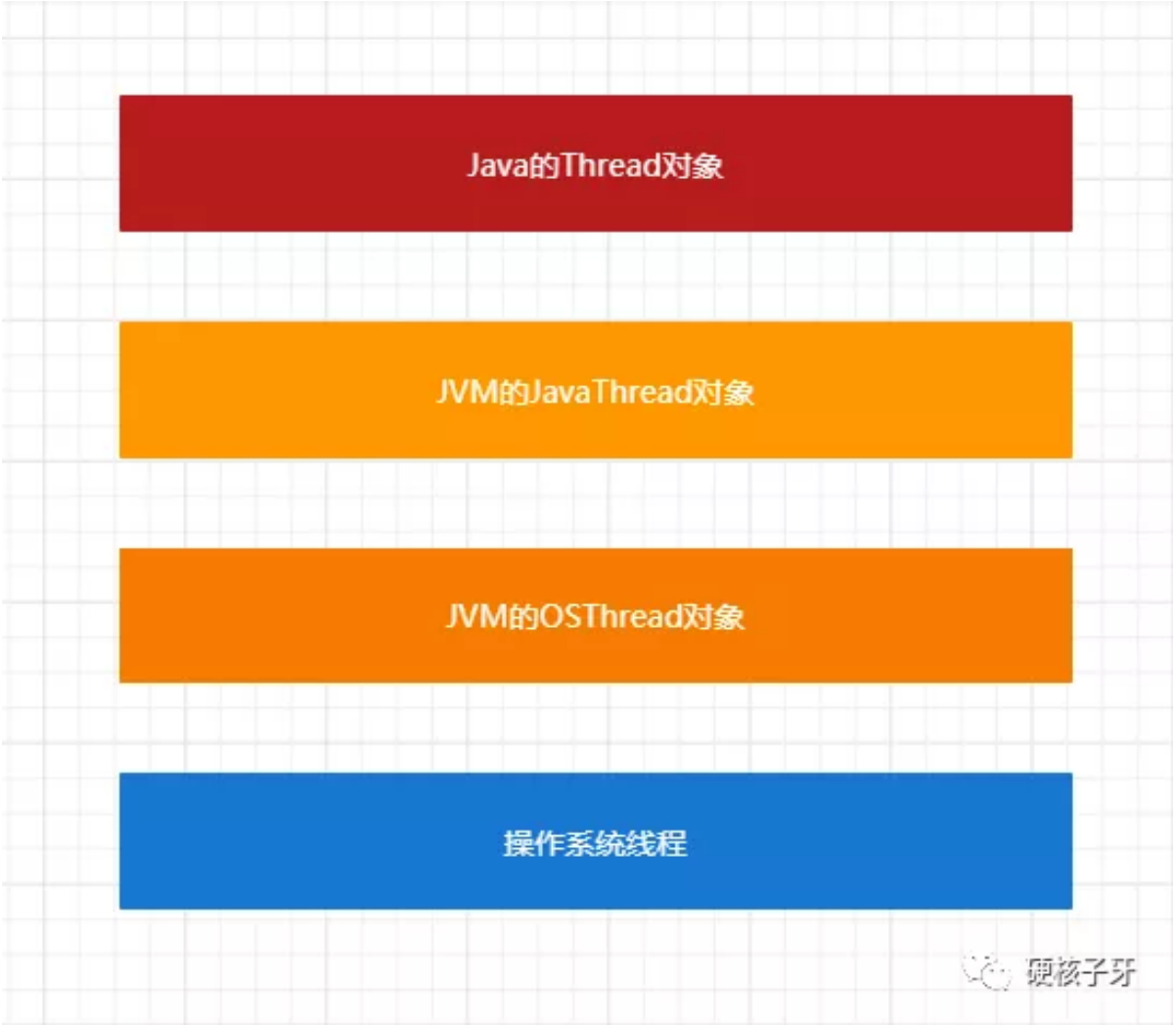
有道云链接：http://note.youdao.com/noteshare?id=8e0ad5bff9ba2a921481538e1af28fa1&sub=6A1808AECD8A42788B49A320E1E794F5

这节课是精通多线程系列第一节课
这节课的目标是：
1、掌握Linux系统提供的线程API
2、掌握Linux的线程状态切换图
3、基于面向过程的Linux线程API开发出面向对象的线程机制
想精通Java的多线程，必先精通Linux的多线程，这张图要在你的脑海中
我们的学习顺序是先学习Linux多线程，再向上讲Java的多线程



配置环境

CMakeLists.txt中增加：set(CMAKE_CXX_FLAGS "${CAMKE_CXX_FLAGS} -std=c++11 -pthread")

```
cmake_minimum_required(VERSION 3.19)
project(ziya_aqs_cpp)

set(CMAKE_CXX_FLAGS "${CAMKE_CXX_FLAGS} -std=c++11 -pthread")

set(CMAKE_CXX_STANDARD 14)

add_executable(ziya_aqs_cpp main.cpp core/Atomic.h core/Atomic.cpp
```

创建线程

```
#include <pthread.h>

int pthread_create(pthread_t *restrict tidp,
                   const pthread_attr_t *restrict attr,
                   void *(*start_rtn)(void *), void *restrict arg);
```

返回值：若成功，返回 0；否则，返回错误编号

参数介绍：

1. tidp，**必传**，获取线程ID，所有操作线程的API都需要这个参数
2. attr，非必传，可以通过该参数设置线程栈大小、分离线程属性
1. start_rtn，**必传**，线程执行函数，有点类似于注册回调函数
2. arg，非必传，通过该参数给线程执行函数传参

设置线程属性

```
#include <pthread.h>

int pthread_attr_init(pthread_attr_t *attr);

int pthread_attr_destroy(pthread_attr_t *attr);
```

两个函数的返回值：若成功，返回 0；否则，返回错误编号

可以设置的属性

名称	描述	FreeBSD 8.0	Linux 3.2.0	Mac OS X 10.6.8	Solaris 10
detachstate	线程的分离状态属性	•	•	•	•
guardsize	线程栈末尾的警戒缓冲区大小（字节数）	•	•	•	•
stackaddr	线程栈的最低地址	•	•	•	•
stacksize	线程栈的最小长度（字节数）	•	•	•	•

图 12-3　POSIX.1 线程属性

如果在创建线程时就知道不需要了解线程的终止状态，就可以修改 pthread_attr_t 结构中的 *detachstate* 线程属性，让线程一开始就处于分离状态。可以使用 pthread_attr_setdetachstate 函数把线程属性 *detachstate* 设置成以下两个合法值之一：PTHREAD_CREATE_DETACHED，以分离状态启动线程；或者 PTHREAD_CREATE_JOINABLE，正常启动线程，应用程序可以获取线程的终止状态。

```
#include <pthread.h>
int pthread_attr_getdetachstate(const pthread_attr_t *restrict attr,
                                int *detachstate);
int pthread_attr_setdetachstate(pthread_attr_t *attr, int *detachstate);
                                        两个函数的返回值：若成功，返回 0；否则，返回错误编号
```

可以调用 pthread_attr_getdetachstate 函数获取当前的 *detachstate* 线程属性。第二个参数所指向的整数要么设置成 PTHREAD_CREATE_DETACHED，要么设置成 PTHREAD_CREATE_JOINABLE，具体要取决于给定 pthread_attr_t 结构中的属性值。

JVM的虚拟机栈大小初始是1M是如何实现的，就是通过这两个API实现的

可以使用函数 pthread_attr_getstack 和 pthread_attr_setstack 对线程栈属性进行管理。

```
#include <pthread.h>
int pthread_attr_getstack(const pthread_attr_t *restrict attr,
                          void **restrict stackaddr,
                          size_t *restrict stacksize);
int pthread_attr_setstack(pthread_attr_t *attr,
                          void *stackaddr, size_t stacksize);
                                        两个函数的返回值：若成功，返回 0；否则，返回错误编号
```

线程中断机制

Linux多线程是没有中断机制的

JVM的中断机制是自己实现的，实现逻辑很简单，在线程运行函数中或线程运行这条路上有一处或多处这样的判断。意思就是通过改变中断标志告诉线程，你的任务已经执行完了，你可以死亡了。当然，这个标志可以被清理掉。所以看到说设置了有可能会被忽略掉。其实就是其他逻辑发现这个线程不能死，把这个状态位改掉了。

```
while (0 == taskpool.task_count()) {
    INFO_PRINT("[%s] 暂无任务执行，进入阻塞\n", Self->name().c_str());

    pthread_cond_wait(taskpool._cond, taskpool._lock);
}

// 唤醒以后检查自己是不是已被中断，如果是，就退出
if (Self->interrupted()) {
    INFO_PRINT("[%s] 已被中断，退出\n", Self->name().c_str());

    pthread_exit( retval: NULL);
}
```

线程结束

```
#include <pthread.h>
void pthread_exit(void *rval_ptr);
```

参数可以用来设置线程执行结束后的返回值

等待线程运行结束

```
#include <pthread.h>
int pthread_join(pthread_t thread, void **rval_ptr);
                                        返回值：若成功，返回 0；否则，返回错误编号
```

参数可以用来获取线程执行结果

传参

通过pthread_create的第四个参数传参

获取返回值

课上演示，需要考虑两种模型：

1. 阻塞线程模型
2. 分离线程模型

互斥锁

```
#include <pthread.h>
int pthread_mutex_init(pthread_mutex_t *restrict mutex,
                       const pthread_mutexattr_t *restrict attr);
int pthread_mutex_destroy(pthread_mutex_t *mutex);
                                        两个函数的返回值：若成功，返回 0；否则，返回错误编号
```

```
#include <pthread.h>
int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_trylock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);
                                        所有函数的返回值：若成功，返回 0；否则，返回错误编号
```

读写锁

```
#include <pthread.h>
int pthread_rwlock_init(pthread_rwlock_t *restrict rwlock,
                        const pthread_rwlockattr_t *restrict attr);
int pthread_rwlock_destroy(pthread_rwlock_t *rwlock);
                                        两个函数的返回值：若成功，返回 0；否则，返回错误编号
```

```
#include <pthread.h>
int pthread_rwlock_rdlock(pthread_rwlock_t  *rwlock);
int pthread_rwlock_wrlock(pthread_rwlock_t  *rwlock);
int pthread_rwlock_unlock(pthread_rwlock_t  *rwlock);
```

所有函数的返回值：若成功，返回 0；否则，返回错误编号

条件变量

```
#include <pthread.h>
int pthread_cond_init(pthread_cond_t *restrict cond,
                      const pthread_condattr_t *restrict attr);
int pthread_cond_destroy(pthread_cond_t *cond);
```

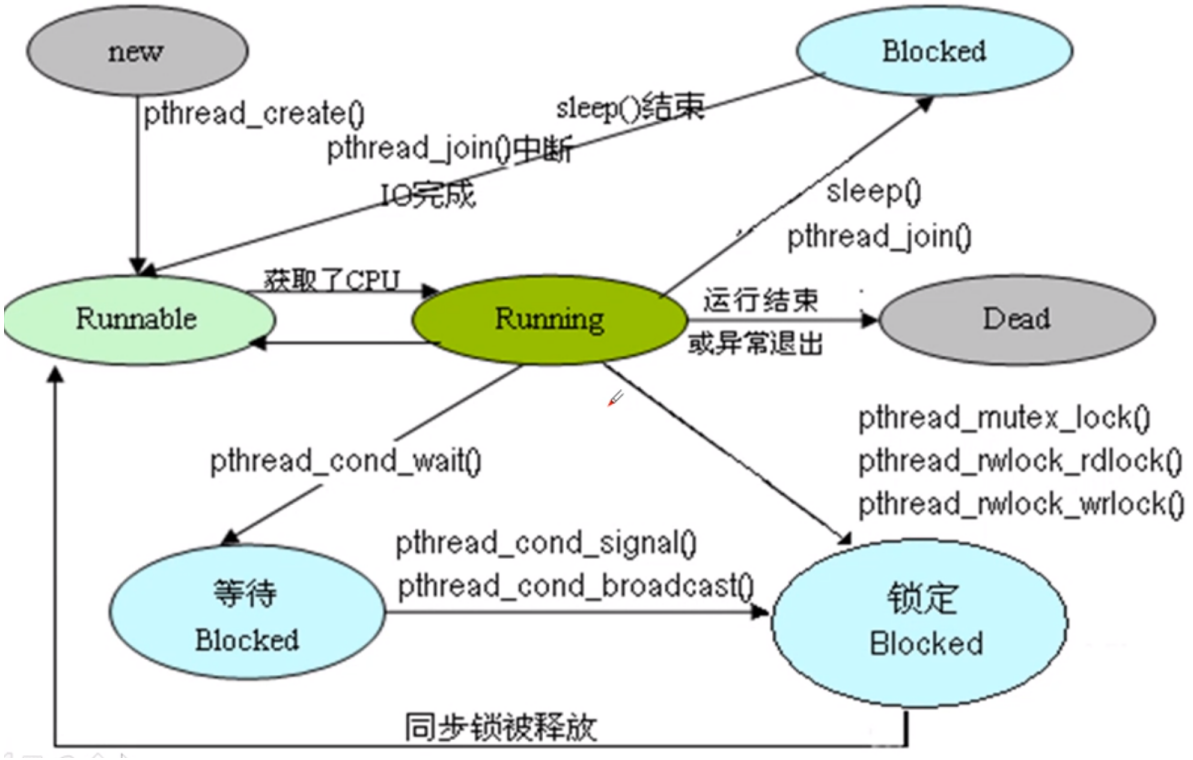
两个函数的返回值：若成功，返回 0；否则，返回错误编号

```
#include <pthread.h>
int pthread_cond_wait(pthread_cond_t *restrict cond,
                      pthread_mutex_t *restrict mutex);
int pthread_cond_timedwait(pthread_cond_t *restrict cond,
                           pthread_mutex_t *restrict mutex,
                           const struct timespec *restrict tsptr);
```

两个函数的返回值：若成功，返回 0；否则，返回错误编号

pthread_cond_wait课堂上细讲，很重要

线程状态转换图



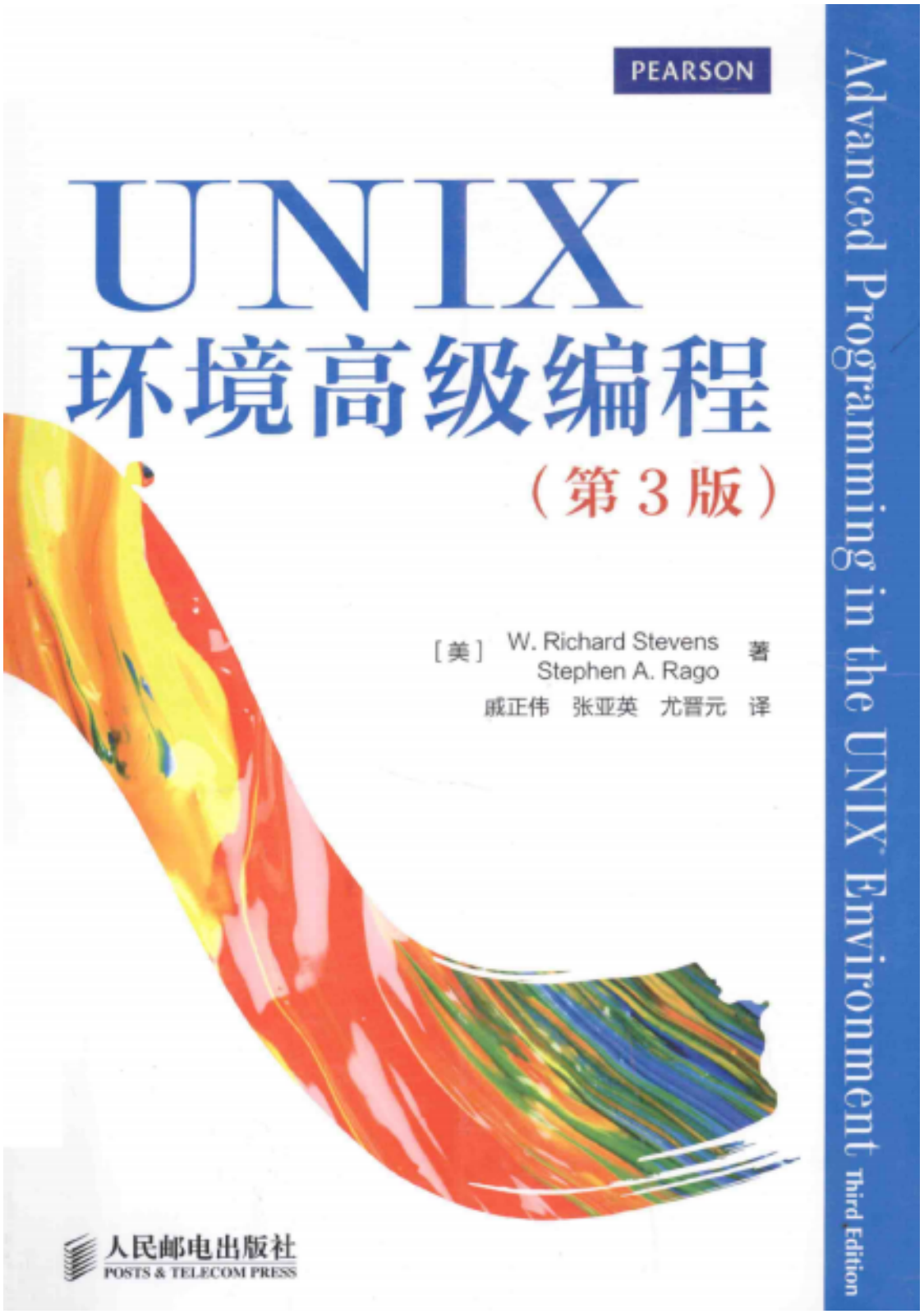
开发实现

记住一句话：线程是自己控制自己的，外界所有的操作只不过是在打标记

1、控制某个线程的阻塞、唤醒、死亡
2、子线程控制主线程的阻塞、唤醒、死亡
3、生产者消费者模型

推荐阅读

《Unix高级环境编程》第10、11两章
讲真，翻译得挺烂，读着来气，建议看英文版


PEARSON
UNIX
环境高级编程
（第3版）
[美] W. Richard Stevens 著
Stephen A. Rago
戚正伟 张亚英 尤晋元 译
人民邮电出版社
POSTS & TELECOM PRESS
Advanced Programming in the UNIX Environment
Third Edition

若有收获，就点个赞吧