# 互联网络程序设计实验

# 实验指导书

电子科技大学计算机学院

二○一二年十月

目录

[互联网络程序设计实验 1](#_Toc353968311)

[实验指导书 1](#_Toc353968312)

[一、环境准备 3](#_Toc353968313)

[1. VmWare虚拟机配置 3](#_Toc353968314)

[2. 单元测试概述 5](#_Toc353968315)

[3. boost::unit\_test单元测试框架 5](#_Toc353968316)

[二、socket封装实验 7](#_Toc353968317)

[三、线程与线程池 15](#_Toc353968318)

[四、epoll与reactor模式 24](#_Toc353968319)

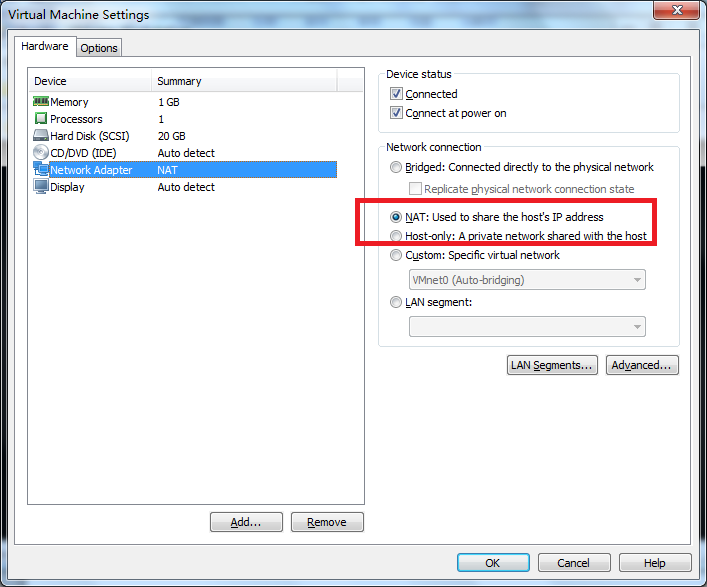
[五、定时器 29](#_Toc353968320)

# 一、环境准备

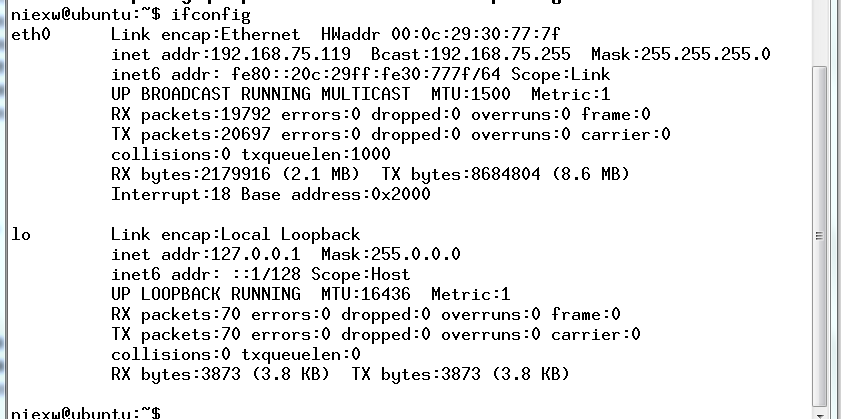
本实验课程的编程环境是Linux，要求使用VmWare虚拟机。

## VmWare虚拟机配置

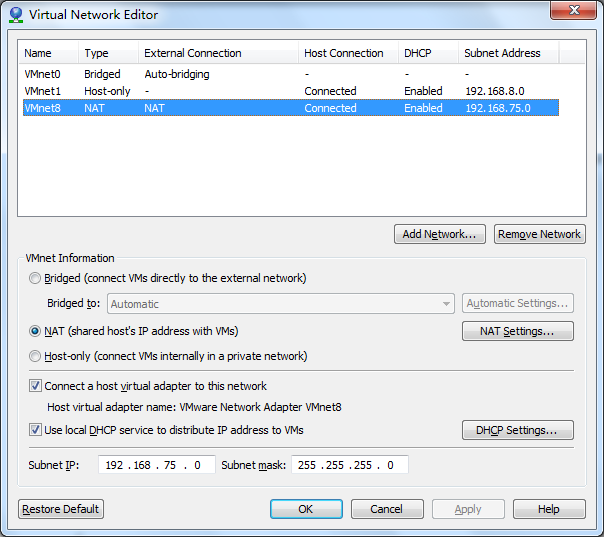
实验课程将下发VmWare虚拟机，该虚拟机已经配置好，下面给出虚拟机的配置情况，以防实验环境发生变化。



VmWare虚拟机的网络配置成NAT。Linux的网卡配置为静态：



这个网段必须与VmWare的VMnet8网卡相匹配：



## 单元测试概述

1）什么是单元测试？

单元测试是开发者编写的一小段代码，用于检验被测代码的一个很小的、很明确的功能是否正确。通常而言，一个单元测试是用于判断某个特定条件（或者场景）下某个特定函数的行为。例如，你可能把一个很大的值放入一个有序list 中去，然后确认该值出现在list 的尾部。或者，你可能会从字符串中删除匹配某种模式的字符，然后确认字符串确实不再包含这些字符了。

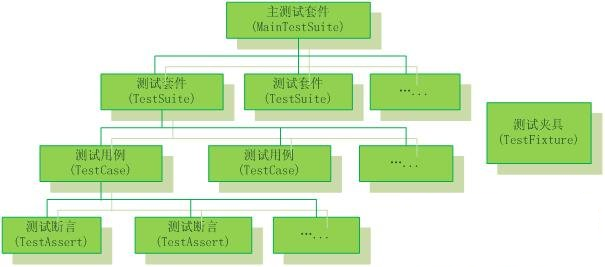
执行单元测试，是为了证明某段代码的行为确实和开发者所期望的一致。

2）为什么需要单元测试？

当编写项目的时刻，如果我们假设底层的代码是正确无误的，那么先是高层代码中使用了底层代码；然后这些高层代码又被更高层的代码所使用，如此往复。当基本的底层代码不再可靠时，那么必需的改动就无法只局限在底层。虽然你可以修正底层的问题，但是这些对底层代码的修改必然会影响到高层代码。于是，一个对底层代码的修正，可能会导致对几乎所有代码的一连串改动，从而使修改越来越多，也越来越复杂。从而使整个项目也以失败告终。

而单元测试的核心内涵：这个简单有效的技术就是为了令代码变得更加完美。

3）测试用例的组织



一般将测试用例按照树型进行组织：测试集，称为test suite；单个测试用例，称为test cases。suite包含多个cases，并且suite可以多个suite。这样，一个项目的所有测试用例以suite方式组织成树。

## boost::unit\_test单元测试框架

Boost 有一整套测试工具，基本上可以说它们是用于验证表达式的宏。测试工具的三个主要类别是 BOOST\_WARN、BOOST\_CHECK 和BOOST\_REQUIRE。BOOST\_CHECK 和 BOOST\_REQUIRE 之间的差异在于：对于前者，即使断言失败，测试仍然继续执行；而对于后者，认为这是严重的错误，测试会停止。下面的代码[清单](http://www.ibm.com/developerworks/cn/aix/library/au-ctools1_boost/#list4) 使用一个简单的 C++ 片段展示了这些工具类别之间的差异。

**使用 Boost 测试工具的三个变体**

|  |
| --- |
| #define BOOST\_TEST\_MODULE enumtest  #include <boost/test/included/unit\_test.hpp>  BOOST\_AUTO\_TEST\_SUITE (enum-test)  BOOST\_AUTO\_TEST\_CASE (test1)  {  typedef enum {red = 8, blue, green = 1, yellow, black } color;  color c = green;  BOOST\_WARN(sizeof(green) > sizeof(char));  BOOST\_CHECK(c == 2);  BOOST\_REQUIRE(yellow > red);  BOOST\_CHECK(black != 4);  }  BOOST\_AUTO\_TEST\_SUITE\_END( ) |

第一个 BOOST\_CHECK 会失败，第一个 BOOST\_REQUIRE 也是如此。但是，当 BOOST\_REQUIRE 失败时，代码退出，所以不会到达第二个BOOST\_CHECK。

**理解 BOOST\_REQUIRE 和 BOOST\_CHECK 之间的差异**

|  |
| --- |
| [arpan@tintin] ./a.out  Running 1 test case...  e2.cpp(11): error in "test1": check c == 2 failed  e2.cpp(12): fatal error in "test1": critical check yellow > red failed  \*\*\* 2 failures detected in test suite "enumtest" |

# 二、socket封装实验

本次课程的实验任务是对socket进行封装，首先给出address.h/file.h代码，这两个文件封装IPv4的地址与文件描述符。

Socket是用于进程之间通信的一种机制，是网络编程的核心，本实验主要关心三类socket，TCP、UDP和UNIX域。

本实验在linux上对以上三种类型的socket进行封装，使其更加简单易用，socket编程根据当前程序是客户端还是服务器流程不太相同，以下是具体步骤：

服务器：

首先使用socket函数创建一个套接口文件描述符，文件描述符是linux中通用的文件句柄，用于描述一个文件（unix把大部分IO资源统称为文件），原型如下：

*#include <sys/types.h>*

*#include <sys/socket.h>*

*int socket(int domain, int type, int protocol);*

之后使用bind函数绑定该套接口用于监听的IP地址和端口，之后，客户端可以使用该IP地址和端口进行连接，已达到通信的目的，原型如下：

*#include <sys/types.h>*

*#include <sys/socket.h>*

*int bind(int sockfd, const struct sockaddr \*addr, socklen\_t addrlen);*

bind完成之后，程序使用lsten函数使socket开始监听来自外部的连接，使用listen之后，外部的程序就可以连接到该程序了：

*#include <sys/types.h>*

*#include <sys/socket.h>*

*int listen(int sockfd, int backlog);*

当有连接到达时，监听的socket变为可读状态，这是使用accept函数读取该socket，即可得到一个新的socket描述符，该描述符代表了新进来的连接，之后就可以对该描述符进行读写操作了，这时的读写操作其实是调用网络设备进行网络收发：

*#include <sys/types.h>*

*#include <sys/socket.h>*

*int accept(int sockfd, struct sockaddr \*addr, socklen\_t \*addrlen);*

之后便是调用read和write函数进行网络IO了：

*#include <unistd.h>*

*ssize\_t read(int fildes, void \*buf, size\_t nbyte);*

*ssize\_t write(int fildes, const void \*buf, size\_t nbyte);*

在使用完socket时候，我们需要断开连接，这时可以调用close或者shutdown函数：

*#include <unistd.h>*

*int close(int fd);*

*#include <sys/socket.h>*

*int shutdown(int socket, int how);*

客户端：

首先使用socket函数创建一个套接口文件描述符，文件描述符是linux中通用的文件句柄，用于描述一个文件（unix把大部分IO资源统称为文件），原型如下：

*#include <sys/types.h>*

*#include <sys/socket.h>*

*int socket(int domain, int type, int protocol);*

客户端只需要使用临时的端口进行通信即可，因为它没有必要让用户知道该临时端口是多少，所以没有必要使用bind函数，在得到socket描述符之后，可以直接使用connect函数连接服务器，connect函数发现之前的套接口没有bind端口时会自动选择一个临时端口号：

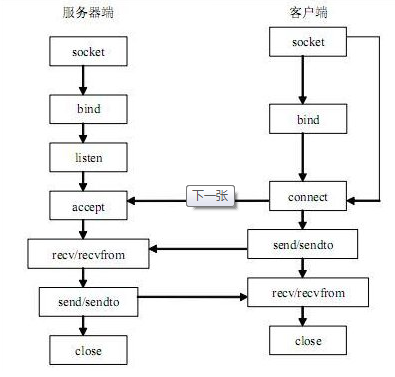
*#include <sys/types.h>*

*#include <sys/socket.h>*

*int connect(int sockfd, const struct sockaddr \*serv\_addr, socklen\_t addrlen);*

连接成功之后，就可以调用上面说过的read和write函数进行和服务器的通信了。并在通信完成之后关闭socket。

总体过程如下图：



以上对socket编程做了大体介绍，由于步骤较多且面向过程，所以使用不太方便，这里我们使用C++面向对象的方法对这个编程模式进行封装，以达到简化编程的目的。

*/\*\**

*\* @file Address.h*

*\* @comment*

*\* wrap of struct sockaddr\_in*

*\**

*\* @author niexw*

*\*/*

*#ifndef \_XCOM\_ADDRESS\_H\_*

*#define \_XCOM\_ADDRESS\_H\_*

*#include <netinet/in.h>*

*#include <arpa/inet.h>*

*#include <strings.h>*

*#include <string>*

*#include <assert.h>*

*#include <ostream>*

*#include <stdio.h>*

*#include <sys/un.h>*

*namespace xcom*

*{*

*/\*\**

*\* @class Address4*

*\* @comment*

*\* Address4 is wrap of sockaddr\_in*

*\*/*

*struct Address4 : public sockaddr\_in*

*{*

*typedef int value\_type;*

*static int const value = AF\_INET;*

*Address4()*

*{*

*bzero(this, sizeof(Address4));*

*sin\_family = AF\_INET;*

*}*

*explicit Address4(sockaddr\_in &sa)*

*{*

*bzero(this, sizeof(Address4));*

*sin\_family = AF\_INET;*

*if (sa.sin\_family != AF\_INET)*

*return;*

*sin\_port = sa.sin\_port;*

*sin\_addr.s\_addr = sa.sin\_addr.s\_addr;*

*}*

*Address4(const Address4 &addr)*

*{*

*bzero(this, sizeof(Address4));*

*sin\_family = AF\_INET;*

*sin\_port = addr.sin\_port;*

*sin\_addr.s\_addr = addr.sin\_addr.s\_addr;*

*}*

*Address4(const unsigned int addr, const unsigned short port)*

*{*

*bzero(this, sizeof(Address4));*

*sin\_family = AF\_INET;*

*sin\_port = htons(port);*

*sin\_addr.s\_addr = htonl(addr);*

*}*

*Address4(const unsigned short port)*

*{*

*bzero(this, sizeof(Address4));*

*sin\_family = AF\_INET;*

*sin\_port = htons(port);*

*sin\_addr.s\_addr = INADDR\_ANY;*

*}*

*Address4(const char \*p, unsigned short port)*

*{*

*bzero(this, sizeof(Address4));*

*sin\_family = AF\_INET;*

*inet\_pton(p);*

*sin\_port = htons(port);*

*}*

*Address4(const std::string &addr, unsigned short port)*

*{*

*bzero(this, sizeof(Address4));*

*sin\_family = AF\_INET;*

*inet\_pton(addr.c\_str());*

*sin\_port = htons(port);*

*}*

*~Address4() { }*

*inline size\_t size() const*

*{ return sizeof(Address4); }*

*//*

*// presentation to number and so on*

*//*

*bool inet\_pton(const char \*p)*

*{ return ::inet\_pton(AF\_INET, p, (void \*)&sin\_addr.s\_addr); }*

*bool inet\_pton(std::string &s)*

*{ return ::inet\_pton(AF\_INET, s.c\_str(), (void \*)&sin\_addr.s\_addr); }*

*bool inet\_ntop(char \*buf, size\_t size)*

*{*

*return buf == NULL || ::inet\_ntop(AF\_INET,*

*(const void \*)&sin\_addr.s\_addr, buf, size) == NULL ?*

*false : true;*

*}*

*bool inet\_ntop(std::string &s)*

*{*

*s.resize(16);*

*const char\* ret = ::inet\_ntop(AF\_INET, (const void \*)&sin\_addr.s\_addr,*

*(char \*)s.c\_str(), 16);*

*if (ret == NULL)*

*return false;*

*s.resize(strlen(s.c\_str()));*

*return true;*

*}*

*void setAnyAddress() { sin\_addr.s\_addr = INADDR\_ANY; }*

*void hostPort(unsigned short p) { sin\_port = htons(p); }*

*unsigned short hostPort() { return ntohs(sin\_port); }*

*void clear()*

*{*

*bzero(this, sizeof(Address4));*

*sin\_family = AF\_INET;*

*}*

*//*

*// operators*

*//*

*bool operator==(const Address4 &rhs)*

*{*

*return sin\_port == rhs.sin\_port*

*&& sin\_addr.s\_addr == rhs.sin\_addr.s\_addr;*

*}*

*bool operator==(const sockaddr\_in &rhs)*

*{*

*return sin\_family == rhs.sin\_family && sin\_port == rhs.sin\_port*

*&& sin\_addr.s\_addr == rhs.sin\_addr.s\_addr;*

*}*

*Address4& operator=(const sockaddr\_in &addr)*

*{*

*assert(addr.sin\_family == AF\_INET);*

*sin\_port = addr.sin\_port;*

*sin\_addr.s\_addr = addr.sin\_addr.s\_addr;*

*return \*this;*

*}*

*friend std::ostream& operator<<(std::ostream &os, Address4 &addr)*

*{*

*char p[16];*

*::inet\_ntop(AF\_INET, (const void \*)&addr.sin\_addr.s\_addr, p, 16);*

*os << p << ":" << ntohs(addr.sin\_port);*

*return os;*

*}*

*};*

*struct AddressUn : public sockaddr\_un*

*{*

*typedef int value\_type;*

*static int const value = AF\_LOCAL;*

*AddressUn()*

*{*

*bzero(this, sizeof(AddressUn));*

*sun\_family = AF\_LOCAL;*

*}*

*AddressUn(AddressUn &addr)*

*{*

*assert(addr.sun\_family == AF\_LOCAL);*

*bzero(this, sizeof(AddressUn));*

*sun\_family = AF\_LOCAL;*

*bcopy(addr.sun\_path, sun\_path, strlen(addr.sun\_path));*

*}*

*AddressUn(char const \*p)*

*{*

*bzero(this, sizeof(AddressUn));*

*sun\_family = AF\_LOCAL;*

*bcopy(p, sun\_path, strlen(p));*

*}*

*AddressUn(std::string const &str)*

*{*

*bzero(this, sizeof(AddressUn));*

*sun\_family = AF\_LOCAL;*

*bcopy(str.c\_str(), sun\_path, str.size());*

*}*

*size\_t size() const*

*{ return (long)&sun\_path - (long)this + strlen(sun\_path); }*

*void unlink() const*

*{ ::unlink(sun\_path); }*

*bool operator==(const AddressUn &rhs)*

*{ return bcmp(this, &rhs, sizeof(AddressUn)) == 0; }*

*bool operator==(const sockaddr\_un &rhs)*

*{ return bcmp(this, &rhs, sizeof(AddressUn)) == 0; }*

*AddressUn& operator=(const sockaddr\_un &addr)*

*{*

*assert(addr.sun\_family == AF\_LOCAL);*

*bzero(this, sizeof(AddressUn));*

*sun\_family = AF\_LOCAL;*

*bcopy(addr.sun\_path, sun\_path, strlen(addr.sun\_path));*

*return \*this;*

*}*

*friend std::ostream& operator<<(std::ostream &os, AddressUn &addr)*

*{*

*os << addr.sun\_path;*

*return os;*

*}*

*};*

*}; // namespace xcom*

*#endif // \_XCOM\_ADDRESS\_H\_*

要求阅读这两个文件，然后完成addresstest.h/filetest.h定义的测试用例。

# 三、线程与线程池

线程是程序执行的基本单位，或者说其是调度的基本单位。在日常的编程中，总会遇到为了充分利用资源，或者为了充分发挥程序高性能而让程序产生线程，已达到在一个进程多有多个控制流，并发或者并行执行的情况。

线程池是使用线程的一种常用方式，使用它可以避免线程大量创建与消亡带来的性能损失，以及将阻塞的事务交给线程池完成，而使主线程不产生阻塞来达到高性能的目的。

*/\*\**

*\* @file Thread.h*

*\* @comment*

*\**

*\* @author niexw*

*\*/*

*#ifndef \_XCOM\_THREAD\_H\_*

*#define \_XCOM\_THREAD\_H\_*

*#include <pthread.h>*

*#include <list>*

*#include <boost/typeof/typeof.hpp>*

*#include <boost/utility.hpp>*

*#include "Exception.h"*

*#include <boost/ref.hpp>*

*#include <boost/function.hpp>*

*#include <boost/bind.hpp>*

*#include <boost/lambda/lambda.hpp>*

*#define CALLBACK\_PARAMS(w) &w.run, &w*

*namespace xcom*

*{*

*typedef void\* (\*CallbackFunc)(void \*);*

*/\*\**

*\* @class Thread*

*\* @comment*

*\**

*\*/*

*class Thread*

*{*

*public:*

*enum Attribute { StackSize = 0, Affinity, Priority };*

*protected:*

*pthread\_t tid\_;*

*pthread\_attr\_t attr\_;*

*void exit()*

*{ pthread\_exit(NULL); }*

*void exit(void \*value)*

*{ pthread\_exit(value); }*

*public:*

*Thread()*

*{ pthread\_attr\_init(&attr\_); }*

*~Thread()*

*{*

*if (isRunning()) throw EXCEPTION();*

*pthread\_attr\_destroy(&attr\_);*

*}*

*// @TODO*

*void setAttribute(Attribute &a, void \*value)*

*{*

*switch(a)*

*{*

*case StackSize:*

*pthread\_attr\_setstacksize(&attr\_, (size\_t)value);*

*break;*

*case Priority:*

*pthread\_attr\_setschedparam(&attr\_, (const struct sched\_param \*)value);*

*break;*

*default:*

*break;*

*}*

*}*

*void getAttribute(Attribute &a, void \*value)*

*{*

*switch(a)*

*{*

*case StackSize:*

*pthread\_attr\_getstacksize(&attr\_, (size\_t \*)value);*

*break;*

*case Priority:*

*pthread\_attr\_getschedparam(&attr\_, (struct sched\_param \*)value);*

*break;*

*default:*

*break;*

*}*

*}*

*bool isRunning()*

*{ return pthread\_kill(tid\_, 0) == 0 ? true : false; }*

*void detach()*

*{ if (pthread\_detach(tid\_) != 0) throw EXCEPTION(); }*

*void detach(pthread\_t tid)*

*{ if (pthread\_detach(tid) != 0) throw EXCEPTION(); }*

*void cancel()*

*{*

*if (!isRunning()) {*

*tid\_ = 0;*

*return;*

*}*

*pthread\_cancel(tid\_);*

*join(); // force the thread to be canceled.*

*}*

*void kill(int sig)*

*{ if (pthread\_kill(tid\_, sig) != 0) throw EXCEPTION(); }*

*void join()*

*{ pthread\_join(tid\_, NULL); }*

*pthread\_t run(CallbackFunc f, void \*p)*

*{*

*pthread\_create(&tid\_, &attr\_, f, p);*

*}*

*};*

*template <typename T, bool>*

*struct CallbackWrapperImp*

*{*

*static void\* run(void \*p)*

*{*

*T \*self = (T\*)p;*

*return (void\*)self->func\_();*

*}*

*};*

*template <typename T>*

*struct CallbackWrapperImp<T, true>*

*{*

*static void\* run(void \*p)*

*{*

*T \*self = (T\*)p;*

*self->func\_();*

*return NULL;*

*}*

*};*

*template <typename R>*

*struct CallbackWrapper*

*: CallbackWrapperImp<CallbackWrapper<R>, boost::is\_void<R>::value>*

*{*

*boost::function<R ()> func\_;*

*CallbackWrapper()*

*{ }*

*CallbackWrapper(boost::function<R ()> f) : func\_(f)*

*{ }*

*template <typename T>*

*void set(T t)*

*{ func\_ = t; }*

*bool enabled()*

*{ return func\_; }*

*};*

*struct PThreadMutex*

*{*

*pthread\_mutex\_t mutex\_;*

*PThreadMutex()*

*{ pthread\_mutex\_init(&mutex\_, NULL); }*

*~PThreadMutex()*

*{ pthread\_mutex\_destroy(&mutex\_); }*

*void lock()*

*{ if (pthread\_mutex\_lock(&mutex\_) != 0) throw EXCEPTION(); }*

*void unlock()*

*{ if (pthread\_mutex\_unlock(&mutex\_) != 0) throw EXCEPTION(); }*

*bool trylock()*

*{*

*if(pthread\_mutex\_trylock(&mutex\_))*

*return false;*

*else*

*return true;*

*}*

*};*

*struct PThreadCond*

*{*

*pthread\_cond\_t cond\_;*

*timespec time\_;*

*PThreadCond()*

*{*

*time\_.tv\_sec = -1;*

*time\_.tv\_nsec = -1;*

*pthread\_cond\_init(&cond\_, NULL);*

*}*

*~PThreadCond()*

*{ pthread\_cond\_destroy(&cond\_); }*

*void setTimeout(long sec, long nsec)*

*{*

*time\_.tv\_sec = sec;*

*time\_.tv\_nsec = nsec;*

*}*

*void wait(PThreadMutex &m)*

*{ if (pthread\_cond\_wait(&cond\_, &m.mutex\_) != 0) throw EXCEPTION(); }*

*void signal()*

*{ if (pthread\_cond\_signal(&cond\_) != 0) throw EXCEPTION(); }*

*bool timedWait(PThreadMutex &m)*

*{*

*if (pthread\_cond\_timedwait(&cond\_, &m.mutex\_, &time\_) != 0) throw EXCEPTION();*

*}*

*};*

*class ThreadPool : private boost::noncopyable*

*{*

*public:*

*struct Task*

*{*

*CallbackFunc func;*

*void \*param;*

*PThreadCond &cond;*

*PThreadMutex &mutex;*

*int &status;*

*Task(CallbackFunc f, void \*p,*

*PThreadCond &c, PThreadMutex &m, int &s)*

*: func(f), param(p), cond(c), mutex(m), status(s)*

*{ }*

*void wait()*

*{*

*mutex.lock();*

*while(status <= 0)*

*cond.wait(mutex);*

*status--;*

*mutex.unlock();*

*}*

*};*

*protected:*

*enum { IDLE = 0, BUSY };*

*class Item : public Thread*

*{*

*protected:*

*std::list<Task\*> &handlers\_;*

*PThreadMutex &mutex\_;*

*PThreadCond &cond\_;*

*Task \*handler\_;*

*// pthread\_cond\_wait will lock the mutex, so release it*

*static void cleanup\_(void \*p)*

*{ ((Item\*)p)->mutex\_.unlock(); }*

*static void\* run\_(void \*self)*

*{*

*while(1)*

*{*

*Item \*pthis = (Item \*)self;*

*pthis->mutex\_.lock();*

*while(pthis->handlers\_.size()==0)*

*pthis->cond\_.wait(pthis->mutex\_);*

*pthis->handler\_ = pthis->handlers\_.back();*

*pthis->handlers\_.pop\_back();*

*pthis->mutex\_.unlock();*

*pthis->handler\_->param = pthis->handler\_->func(pthis->handler\_->param);*

*pthis->handler\_->mutex.lock();*

*pthis->handler\_->status++;*

*pthis->handler\_->cond.signal();*

*pthis->handler\_->mutex.unlock();*

*}*

*}*

*public:*

*Item(std::list<Task\*> &h, PThreadMutex &m, PThreadCond &c)*

*: handlers\_(h), mutex\_(m), cond\_(c), handler\_(NULL), status\_(IDLE)*

*{ }*

*~Item()*

*{ cancel(); }*

*int status\_; // 0 idle; 1 busy*

*pthread\_t run()*

*{*

*if (pthread\_create(&tid\_, &attr\_, &Item::run\_, this) == 0)*

*return tid\_;*

*else*

*throw EXCEPTION(); // EAGAIN means no memory*

*}*

*};*

*PThreadMutex mutex\_;*

*PThreadCond cond\_;*

*std::list<Task\*> handlers\_;*

*std::list<Item\*> threads\_;*

*public:*

*explicit ThreadPool(size\_t size)*

*{ add(size); }*

*~ThreadPool()*

*{ close(); }*

*void add(size\_t size)*

*{*

*for(int i = 0; i < size; i++)*

*{*

*Item \*p = new Item(handlers\_, mutex\_, cond\_);*

*threads\_.push\_back(p);*

*p->run();*

*}*

*}*

*size\_t remove(size\_t size)*

*{*

*for(int i = 0; i < size; i++)*

*{*

*delete threads\_.back();*

*threads\_.pop\_back();*

*}*

*}*

*void close()*

*{*

*while (!threads\_.empty())*

*{*

*delete threads\_.front();*

*threads\_.pop\_front();*

*}*

*}*

*size\_t size()*

*{ return threads\_.size(); }*

*void addTask(Task &handler)*

*{*

*mutex\_.lock();*

*if(handlers\_.size()==0)*

*cond\_.signal();*

*handlers\_.push\_back(&handler);*

*mutex\_.unlock();*

*}*

*};*

*}; // namespace xcom*

*#endif /\* \_XCOM\_THREAD\_H\_ \*/*

# 四、epoll与reactor模式

Epoll是linux上提供的一种io复用机制，其使用一种机制，等待大量的读写事件，和以往的机制完全不同，传统的io方式是每个IO请求都是必须阻塞，等待其完成为止，或者非阻塞，IO不能立刻成功就马上返回。

Epoll使用了完全不同的方式，其让需要等待IO的时间向其注册，并在没有其他操作时阻塞，知道注册的至少一个时间可用时为止或者时间到为止。

通过这样的方式编程，可以使一个线程同时启用大量的IO，同时不用带来轮训的开销。

这种方式替代了传统的多进程或者多线程的编程模型，减少了进城或线程切换的开销，能够达到很好的性能要求。但随之而来的是代码量的增加。

Reactor是一种事件驱动机制。和普通函数调用的不同之处在于：应用程序不是主动的调用某个API完成处理，而是恰恰相反，Reactor逆置了事件处理流程，应用程序需要提供相应的接口并注册到Reactor上，如果相应的时间发生，Reactor将主动调用应用程序注册的接口，这些接口又称为“回调函数”。

Reactor模式的优点

Reactor模式是编写高性能网络服务器的必备技术之一，它具有如下的优点：

1）响应快，不必为单个同步时间所阻塞，虽然Reactor本身依然是同步的；

2）编程相对简单，可以最大程度的避免复杂的多线程及同步问题，并且避免了多线程/进程的切换开销；

3）可扩展性，可以方便的通过增加Reactor实例个数来充分利用CPU资源；

4）可复用性，reactor框架本身与具体事件处理逻辑无关，具有很高的复用性；

*/\*\**

*\* @file Epoll.h*

*\* @comment*

*\**

*\* @author niexw*

*\*/*

*#ifndef \_XCOM\_EPOLL\_H\_*

*#define \_XCOM\_EPOLL\_H\_*

*#include <sys/epoll.h>*

*#include "File.h"*

*#include "Thread.h"*

*namespace xcom*

*{*

*/\*\**

*\* @class Epoll*

*\* @comment*

*\* wrap of epoll*

*\*/*

*class Epoll : public BasicFile*

*{*

*public:*

*Epoll() {}*

*~Epoll() {}*

*struct Event : public epoll\_event*

*{*

*Event() { events = EPOLLERR; data.u64 = 0; }*

*Event(unsigned int type, void \*magic)*

*{ data.u64 = 0; events = type; data.ptr = magic; }*

*};*

*int create()*

*{*

*if ((descriptor\_ = epoll\_create(1)) == -1)*

*throw EXCEPTION();*

*return descriptor\_;*

*}*

*void registerEvent(int fd, Event &event)*

*{*

*if (epoll\_ctl(descriptor\_, EPOLL\_CTL\_ADD, fd,*

*(epoll\_event\*)&event) == -1)*

*throw EXCEPTION();*

*}*

*void modifyEvent(int fd, Event &event)*

*{*

*if (epoll\_ctl(descriptor\_, EPOLL\_CTL\_ADD, fd,*

*(epoll\_event\*)&event) == -1)*

*{*

*if (errno == EEXIST)*

*errno = 0;*

*else*

*throw EXCEPTION();*

*}*

*}*

*void unregisterEvent(int fd)*

*{*

*if (epoll\_ctl(descriptor\_, EPOLL\_CTL\_DEL, fd, NULL) == -1)*

*throw EXCEPTION();*

*}*

*int waitEvent(Event \*events, int size, int msec)*

*{*

*return epoll\_wait(descriptor\_, events, size, msec);*

*}*

*};*

*class IoReactor*

*{*

*public:*

*static const ulong READ = 0x01UL;*

*static const ulong WRITE = 0x02UL;*

*static const ulong ERR = 0x04UL;*

*static const ulong TIMER = 0x08UL;*

*static const ulong SIGNAL = 0x10UL;*

*static const ulong CUSTOM = 0x20UL;*

*static const ulong TRIGGER = 0x40UL;*

*static const ulong ONESHOT = 0x80UL;*

*struct Task*

*{*

*BasicFile &file; // file*

*CallbackFunc func; // callback function*

*void \*param; // the parameter of callback*

*void \*result; // the result*

*Task(BasicFile &f, CallbackFunc fu, void \*p)*

*: file(f), func(fu), param(p), result(NULL)*

*{ }*

*};*

*typedef void \* Handle;*

*protected:*

*std::list<Task\*> tasks\_;*

*Epoll epoll\_;*

*inline ulong epollOp(ulong option)*

*{*

*return (option & READ ? EPOLLIN : 0)*

*| (option & WRITE ? EPOLLOUT : 0)*

*| (option & ERR ? EPOLLERR : 0)*

*| (option & TIMER ? EPOLLIN : 0)*

*| (option & SIGNAL ? EPOLLERR : 0)*

*| (option & CUSTOM ? EPOLLERR : 0)*

*| (option & TRIGGER ? EPOLLET : 0)*

*| (option & ONESHOT ? EPOLLONESHOT : 0);*

*}*

*public:*

*IoReactor()*

*{ epoll\_.create(); }*

*~IoReactor()*

*{ }*

*size\_t size()*

*{ return tasks\_.size(); }*

*Handle insert(Task &task, ulong option)*

*{*

*Handle p;*

*p = &task;*

*Epoll::Event ev(epollOp(option), &task);*

*epoll\_.registerEvent(task.file.descriptor(), ev);*

*tasks\_.push\_back(&task);*

*return p;*

*}*

*bool remove(Handle p)*

*{*

*epoll\_.unregisterEvent(((Task \*)p)->file.descriptor());*

*tasks\_.remove((Task \*)p);*

*return true;*

*}*

*bool option(Handle p, ulong option)*

*{*

*Epoll::Event ev(epollOp(option), p);*

*epoll\_.modifyEvent(((Task \*)p)->file.descriptor(), ev);*

*return true;*

*}*

*void run()*

*{*

*Epoll::Event ev[1024];*

*//while(1)*

*{*

*int nr = epoll\_.waitEvent(ev, 1024, -1);*

*for(int i = 0; i < nr; i++)*

*{*

*Task \*pt = (Task \*)ev[i].data.ptr;*

*pt->result = pt->func(pt->param);*

*}*

*}*

*}*

*};*

*class IoProactor*

*{*

*};*

*}; // namespace xcom*

*#endif /\* \_XCOM\_EPOLL\_H\_ \*/*

# 五、定时器

设想有个时间钟，每隔一段时间（如100ms）有个滴答，每个滴答处对应一个list，存放了一些定时任务。但时间走到一个滴答处时就触发其对应的任务list。每个任务根据其超时时间放到相应滴答的list中，但一个任务list被滴答查看时，list中的所有任务的超时时间都被更新，并重新放入到时间轮盘中合适的位置。

首先为了方便讨论，约定一些术语。

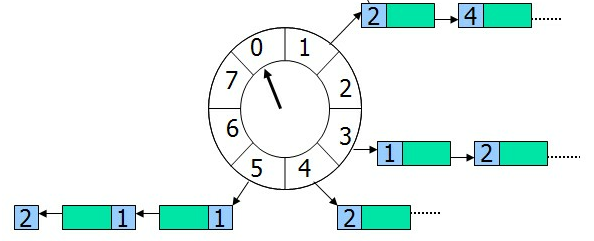
时间轮有两个属性， frequence, wheelSize。

(1)时间轮每隔一段时间有个滴答, 每个滴答， 轮子的指针会顺时针转动一次。这段时间的长度为TimeWheel的时间粒度, 我们称之为frequence, 也就是每过frequence ms 轮子的指针会转动一次。

(2)每个时间轮都有一个轮子大小， 我们称之为wheelSize.

时间轮每一项都对应一个list， 这个list中的每一项都是一个定时器， 称为Timer。

下面用图示来讲解timeWheel.



图一

如上图所示， 我们维护了一个wheelSize为8的TimeWheel。 我们设置 frequence为 100ms， 当前指针指向第0项， 每过100ms， 指针顺时针转动1格。轮子转动一圈的时间为 frequence \* wheelSize = 8 \* 100ms = 800ms

Timewheel的每一项维护一个timer队列。假设指针指向在第0项时， 用户申请一个时间间隔为300ms的timer,时， timerwheel 会将这个timer放入到第3项(图中的两个timer会变成3个timer), 当指针经过300ms后指向第3项时， 会将第3项对应的list （3个timer)全部删除。

所以timeWheel 可以在o(1)常数时间内实现维护定时器。

*/\*\**

*\* @file Timer.h*

*\* @comment*

*\**

*\* @author niexw*

*\*/*

*#ifndef \_XCOM\_TIMER\_H\_*

*#define \_XCOM\_TIMER\_H\_*

*#include <sys/timerfd.h>*

*#include <sys/time.h>*

*#include <set>*

*#include "File.h"*

*#include "Thread.h"*

*namespace xcom*

*{*

*class Timer : public BasicFile*

*{*

*protected:*

*struct Item*

*{*

*timespec time;*

*CallbackFunc func;*

*void \*param;*

*Item(timespec &t, CallbackFunc f, void \*p) : func(f), param(p)*

*{ time.tv\_sec = t.tv\_sec; time.tv\_nsec = t.tv\_nsec; }*

*};*

*struct Comp*

*{*

*bool operator()(Item const &i1, Item const &i2)*

*{*

*return i1.time.tv\_sec == i2.time.tv\_sec ?*

*i1.time.tv\_nsec < i2.time.tv\_nsec :*

*i1.time.tv\_sec < i2.time.tv\_sec;*

*}*

*};*

*std::multiset<Item, Comp> queue\_;*

*public:*

*typedef std::multiset<Item, Comp>::iterator Handle;*

*Timer()*

*{ create(); }*

*~Timer()*

*{ close(); }*

*size\_t size()*

*{ return queue\_.size(); }*

*Handle add(timespec &tv, CallbackFunc f, void \*p)*

*{*

*timespec tvc;*

*clock\_gettime(CLOCK\_MONOTONIC, &tvc);*

*tvc.tv\_sec += tv.tv\_sec;*

*tvc.tv\_nsec += tv.tv\_nsec;*

*Item item(tvc, f, p);*

*Handle h = queue\_.insert(item);*

*if (h == queue\_.begin())*

*{*

*itimerspec t1;*

*t1.it\_interval = {0, 0};*

*t1.it\_value = {h->time.tv\_sec, h->time.tv\_nsec};*

*if (timerfd\_settime(descriptor\_,*

*TFD\_TIMER\_ABSTIME, &t1, NULL) != 0)*

*throw EXCEPTION();*

*}*

*return h;*

*}*

*void remove(Handle h)*

*{*

*itimerspec t1;*

*if (h == queue\_.begin())*

*{*

*if (queue\_.size() == 1) {*

*queue\_.erase(h);*

*t1.it\_interval = {0, 0};*

*t1.it\_value = {0, 0};*

*timerfd\_settime(descriptor\_, TFD\_TIMER\_ABSTIME, &t1, NULL);*

*} else {*

*queue\_.erase(h);*

*h = queue\_.begin();*

*t1.it\_interval = {0, 0};*

*t1.it\_value = {h->time.tv\_sec, h->time.tv\_nsec};*

*timerfd\_settime(descriptor\_, TFD\_TIMER\_ABSTIME, &t1, NULL);*

*}*

*return;*

*}*

*else*

*queue\_.erase(h);*

*}*

*bool first(timespec &tv, CallbackFunc &f, void \*&p)*

*{*

*if (queue\_.size() > 0) {*

*Handle h = queue\_.begin();*

*tv.tv\_sec = h->time.tv\_sec;*

*tv.tv\_nsec = h->time.tv\_nsec;*

*f = h->func;*

*p = h->param;*

*return true;*

*} else*

*return false;*

*}*

*void create()*

*{*

*descriptor\_ = timerfd\_create(CLOCK\_MONOTONIC, 0);*

*if (descriptor\_ == -1)*

*throw EXCEPTION();*

*}*

*static void\* callback(void \*p)*

*{*

*Timer \*This = (Timer\*)p;*

*Handle h = This->queue\_.begin();*

*CallbackFunc f = h->func;*

*void \*param = h->param;*

*This->queue\_.erase(h);*

*return f(param);*

*}*

*};*

*}; // namespace xcom*

*#endif /\* \_XCOM\_TIMER\_H\_ \*/*