基于 Linux 的 C++

第十五讲　网络编程

提　纲

Internet网络协议

- TCP/IP协议
- HTTP协议

套接字

编程实践：Web服务器开发

Internet网络协议

TCP/IP协议

- Internet主流协议族
- 分层、多协议的通信体系

HTTP协议

TCP/IP协议

数据链路层

- 网卡接口的网络驱动程序，处理数据在物理媒介上的传输；不同的物理网络具有不同的电气特性，网络驱动程序隐藏实现细节，为上层协议提供一致的接口
- 数据链路层常用协议：地址解析协议（ARP）和反向地址解析协议（RARP），实现IP地址与机器物理地址（通常为MAC地址）之间的相互转换

网络层

- 实现数据包的路由和转发
- 常用协议：IP、ICMP

TCP/IP协议

网络层

- IP协议：逐跳发送模式；根据数据包的目的地IP地址决定数据如何发送；如果数据包不能直接发送至目的地，IP协议负责寻找一个合适的下一跳路由器，并将数据包交付给该路由器转发
- ICMP协议：因特网控制报文协议，用于检测网络连接

传输层

- 为两台主机的应用程序提供端到端通信
- 传输层使用的主要协议：TCP、UDP

TCP/IP协议

传输层

- TCP：传输控制协议，为应用层提供可靠的、面向连接的、基于流的可靠服务；使用超时重发、数据确认等方式确保数据被正确发送至目的地
- UDP：用户数据报协议，为应用层提供不可靠的、无连接的、基于数据报的服务；不保证数据能正确发送

应用层

- 应用程序逻辑实现
- 常用协议：ping、telnet、DNS、HTTP、FTP、DHCP等

HTTP协议

超文本传输协议：应用层协议

主要特点

- **支持客户/服务器模式**
- **简单快速：客户向服务器请求服务时，只需传送请求方法和路径；请求方法常用GET、HEAD、POST等，每种方法规定了客户与服务器联系的不同类型；HTTP协议简单，服务器程序规模小，通信速度较快**
- **灵活：HTTP允许传输任意类型的数据对象；正在传输的类型由Content-Type加以标记**

HTTP协议

主要特点

- 无连接：无连接是指每次连接只处理一个请求；服务器处理完客户请求，并收到客户应答后，即断开连接，节省传输时间
- 无状态：无状态是指协议对于事务处理没有记忆能力；应答较快，但传输数据量较大

HTTP URL：定位网络资源

- http://host[:port][abs_path]

HTTP协议

HTTP请求

- 由三部分组成：请求行、消息报头、请求正文
- 格式：Method Request-URI HTTP-Version CRLF
- Method：请求方法，GET、POST等
- Request-URI：统一资源标识符
- HTTP-Version：请求的HTTP协议版本
- CRLF：回车换行

HTTP协议

HTTP响应

- 由三部分组成：状态行、消息报头、响应正文
- 状态行格式：HTTP-Version Status-Code Reason-Phrase CRLF
- HTTP-Version：服务器HTTP协议版本
- Status-Code：服务器返回的响应状态码
- Reason-Phrase：状态码的文本描述

HTTP协议

HTTP状态码

- 状态代码有三位数字组成，首数字定义响应类别
  - 1xx：指示信息，表示请求已接收，继续处理；2xx：成功；3xx：重定向，要完成请求必须进行更进一步的操作；4xx：客户端错误，请求有语法错误或请求无法实现；5xx：服务器端错误，服务器未能实现合法的请求
- 常见状态代码
  - 200：OK，请求成功；400：Bad Request，请求有语法错误，不能被服务器所理解；401：Unauthorized，请求未经授权；403：Forbidden，服务器收到请求，但是拒绝提供服务；404：Not Found，请求资源不存在；500：Internal Server Error，服务器发生不可预期的错误；503：Server Unavailable，服务器不能处理客户请求

套接字

套接字的基本概念

套接字函数：“sys/socket.h”

服务器

本地套接字

网络套接字

套接字的基本概念

通信类型：控制套接字如何传输和处理数据，数据以包的形式传输

- 连接（connection）类型：确保所有包依序传输，如果丢包，则请求重传
- 数据报（datagram）类型：不保证包的到达顺序，包可能丢失

名空间：指定套接字地址格式

- 本地名空间：套接字地址为普通文件名
- Internet名空间：套接字地址由Internet地址和端口号（用于区分一台主机上的多个套接字）确定

协议：确定数据如何传输

套接字函数

socket()函数：创建套接字

- 原型：int socket(int domain, int type, int protocol);
- 参数：名空间、通信类型和协议
- 名空间：PF_LOCAL（本地）或PF_INET（Internet）
- 通信类型：SOCK_STREAM（连接类型）或SOCK_DGRAM（数据报类型）
- 协议：传递0，让系统自动选择协议（通常为最佳协议）
- 返回值：套接字描述符

套接字函数

close()函数：释放套接字

- 原型：int close(int fd);

connect()函数：创建两个套接字之间的连接

- 客户发起此系统调用，试图与服务器建立套接字连接
- 原型：int connect(int sockfd, const struct sockaddr * addr, socklen_t addrlen);
- 参数： sockfd为套接字文件描述符； addr为指向套接字地址结构体的指针（服务器地址）； addrlen为服务器地址字符串的长度
- 返回值：0表示连接成功，-1表示连接失败

套接字函数

send()函数：发送数据

- **原型：ssize_t send(int sockfd, const void * buf, size_t len, int flags);**
- **原型：ssize_t sendto(int sockfd, const void * buf, size_t len, int flags, const struct sockaddr * dest_addr, socklen_t addrlen);**
- **原型：ssize_t sendmsg(int sockfd, const struct msghdr * msg, int flags);**
- **只有在套接字处于连接状态时才可调用**

套接字函数

bind()函数：绑定服务器套接字与其地址

- 原型：int bind(int sockfd, const struct sockaddr * addr, socklen_t addrlen);

listen()函数：侦听客户连接

- 原型：int listen(int sockfd, int backlog);
- 参数：backlog指定有多少个挂起连接可以进入队列，超出该值的连接将被抛弃

套接字函数

accept()函数：接受连接，为该连接创建一个新的套接字

- **原型：int accept(int sockfd, struct sockaddr * addr, socklen_t * addrlen);**
- **参数：addr为指向套接字地址结构体（客户地址）的指针**
- **返回值：创建一个新的套接字，以接受客户连接，返回值为新的套接字文件描述符**
- **原先套接字文件描述符可以继续接受新连接**

本地套接字示例：服务器端

```
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <sys/socket.h>
#include <sys/un.h>
#include <unistd.h>
// 持续读取消息，直到套接字关闭或接收到客户发送的“quit”消息
// 前者返回true，后者返回false，服务器随后将停止服务
bool  Serve( int client_socket )
{
  while( true )
  {
    int  length;
    char *  msg;
    // 从套接字中读取文本消息的长度，返回值为0表示客户连接已关闭
    if( read( client_socket, &length, sizeof(length) ) == 0 )
      return true;
    msg = new char[length];
```

本地套接字示例：服务器端

```
    read( client_socket, msg, length );
    std::cout << msg << std::endl;
    if( !strcmp( msg, "quit" ) ) { delete[] msg, msg = NULL; return false; }
    else delete[] msg, msg = NULL;
  }
}
int main( int argc, char * const argv[] )
{
  const char * const socket_name = argv[1];
  int socket_fd;
  struct sockaddr_un name;
  bool serving = true;
  // 创建套接字
  socket_fd = socket( PF_LOCAL, SOCK_STREAM, 0 );
  // 设定服务器性质
  name.sun_family = AF_LOCAL;
  strcpy( name.sun_path, socket_name );
  // 绑定套接字
  bind( socket_fd, (struct sockaddr *)&name, SUN_LEN( &name ) );
```

本地套接字示例：服务器端

```
// 侦听客户连接
listen( socket_fd, 5 );
// 重复接受连接，直到某个客户发出“quit”消息
while( serving )
{
  struct sockaddr_un  client_name;
  socklen_t  client_name_len;
  int  client_socket_fd;
  // 接受客户连接请求
  client_socket_fd = accept( socket_fd,
      (struct sockaddr *)&client_name, &client_name_len );
  serving = Serve( client_socket_fd );    // 服务连接请求
  close( client_socket_fd );    // 关闭客户连接
}
close( socket_fd );
unlink( socket_name );    // 删除套接字文件
return 0;
}
```

本地套接字示例：客户端

```
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <sys/socket.h>
#include <sys/un.h>
#include <unistd.h>

void SendMsg( int socket_fd, const char * msg )
{
  int length = strlen( msg ) + 1;
  write( socket_fd, &length, sizeof( length ) );
  write( socket_fd, msg, length );
}
```

本地套接字示例：客户端

```
int  main( int argc, char * const argv[] )
{
  const char * const  socket_name = argv[1];
  const char * const  msg = argv[2];
  int  socket_fd;
  struct sockaddr_un  name;
  //  创建套接字
  socket_fd = socket( PF_LOCAL, SOCK_STREAM, 0 );
  //  在套接字地址中存储服务器名称
  name.sun_family = AF_LOCAL;
  strcpy( name.sun_path, socket_name );
  //  连接
  connect( socket_fd, (struct sockaddr *)&name, SUN_LEN( &name ) );
  //  发送消息
  SendMsg( socket_fd, msg );
  close( socket_fd );
  return 0;
}
```

本地套接字示例：运行

程序测试运行

- 编译链接服务器端程序和客户端程序
- 进入服务器端程序目录，在终端中输入：./server /tmp/socket；./server为服务器端程序名，/tmp/socket为本服务器启动后的套接字文件名
- 进入客户端程序目录，在新终端中输入：./client /tmp/socket "Hello World!"；./client为客户端程序名
- 停止服务器，在客户端输入命令：./client /tmp/socket "quit"

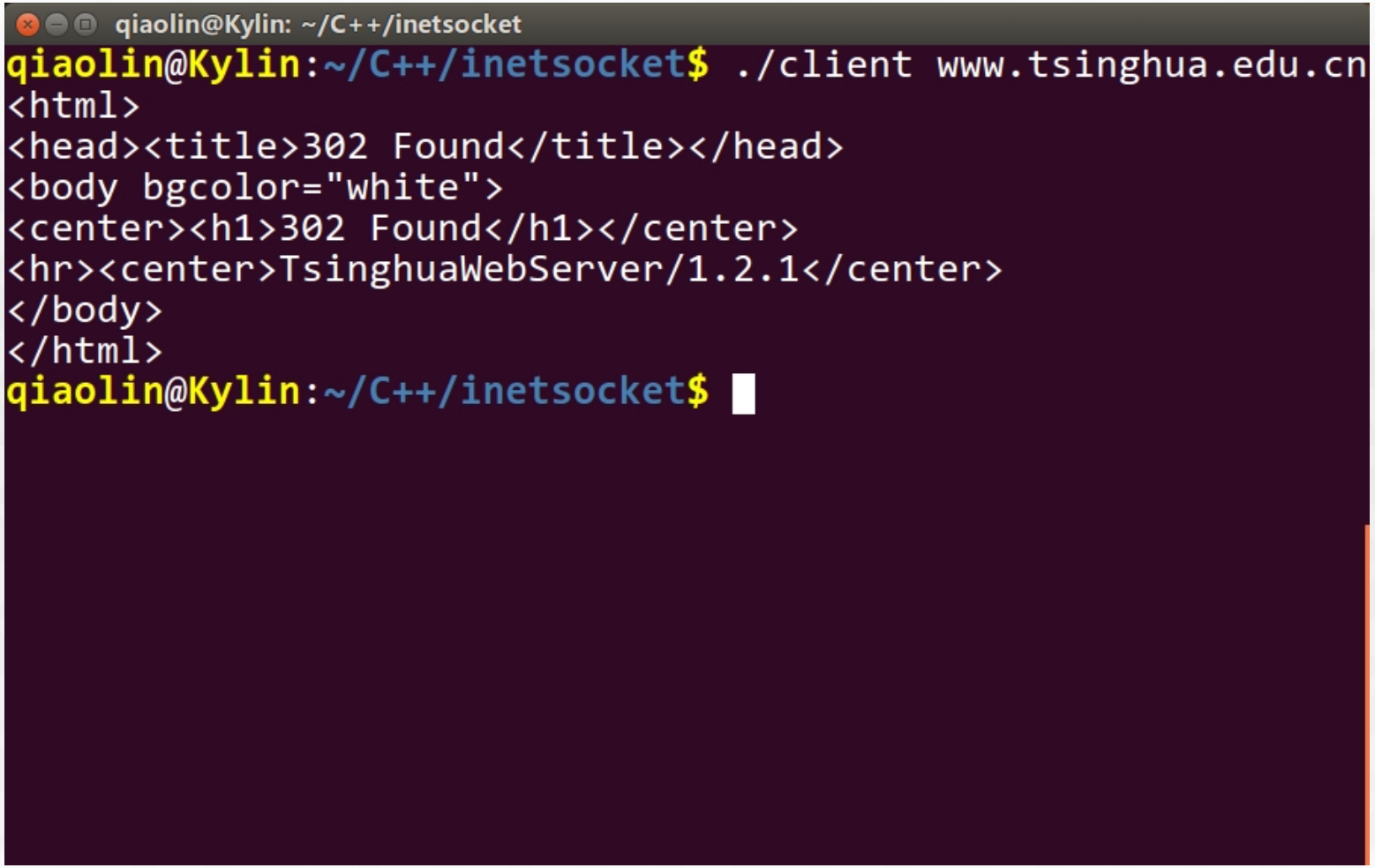
网络套接字示例：客户端

```
#include <stdlib.h>
#include <stdio.h>
#include <netinet/in.h>
#include <netdb.h>
#include <sys/socket.h>
#include <unistd.h>
#include <string.h>
//  请求Web服务器的主页
void GetHomepage( int socket_fd )
{
  char  buffer[8192];
  sprintf( buffer, "GET /\n" );
  write( socket_fd, buffer, strlen( buffer ) );
  while( true )  {
    ssize_t  count = read( socket_fd, buffer, 8192 );
    if( count == 0 )    return;
    fwrite( buffer, sizeof( char ), count, stdout );
  }
}
```

网络套接字示例：客户端

```
int main( int argc, char * const argv[] )
{
  int  socket_fd;
  struct sockaddr_in  name;
  struct hostent *  hostinfo;
  socket_fd = socket( PF_INET, SOCK_STREAM, 0 );
  name.sin_family = AF_INET;
  hostinfo = gethostbyname( argv[1] );
  if( hostinfo == NULL )    return 1;
  else    name.sin_addr = *( (struct in_addr *)hostinfo->h_addr );
  name.sin_port = htons( 80 );
  if( connect( socket_fd, (struct sockaddr *)&name,
      sizeof(struct sockaddr_in) ) == -1 )  {
    perror( "Failure to connect the server." );
    return 2;
  }
  GetHomepage( socket_fd );
  return 0;
}
```

网络套接字示例：客户端


qiaolin@Kylin: ~/C++/inetsocket
qiaolin@Kylin:~/C++/inetsocket$ ./client www.tsinghua.edu.cn
<html>
<head><title>302 Found</title></head>
<body bgcolor="white">
<h1>302 Found</h1>
<hr> TsinghuaWebServer/1.2.1
</body>
</html>
qiaolin@Kylin:~/C++/inetsocket$

编程实践

远程系统管理Web服务器

- 允许本地或远程客户通过HTTP协议访问系统信息，例如显示时间、Linux发布版本、空闲磁盘空间、当前运行的进程等
- 使用模块架构针对Web请求生成动态HTML网页；各模块实现为共享目标库，允许动态装载模块，且可在服务器运行期间添加、删除和替换
- 在子进程或线程中服务Web请求，并设计进程池或线程池管理这些进程或线程
- 服务器不要求超级用户权限
- 不要求实现HTTP全部功能
- 使用面向对象架构