William Kelley
Sockets and Router(An Exercise in Echo Servers)

```
We received this response from the server:
"7)A good friendship is often more important than a passionate romance." We
received this response from the server:
"9)A lifetime friend shall soon be made."
We received this response from the server:
"3)A friend asks only for your time not your money."
We received this response from the server:
"8)A hunch is creativity trying to tell you something."
We received this response from the server:
"0)A beautiful, smart, and loving person will be coming into your life."
We received this response from the server:
"2)A fresh start will put you on your way."
We received this response from the server:
"4)A friend is a present you give yourself."
We received this response from the server:
"8)A hunch is creativity trying to tell you something."
We received this response from the server:
"3)A friend asks only for your time not your money."
We received this response from the server:
"9)A lifetime friend shall soon be made."

//
// File:     ClientSocket.h
// Author:   Adam.Lewis@athens.edu
// Purpose:
// Header file for the client portion of our C++ Sockets interface.   This code
// is based on code from an 2002 article in Linux Gazette
#ifndef ClientSocket_class
#define ClientSocket_class
#include "Socket.h"
class ClientSocket : private Socket
{
 public:

  ClientSocket ( std::string host, int port );
  virtual ~ClientSocket(){};

  const ClientSocket& operator << ( const std::string& ) const;
  const ClientSocket& operator >> ( std::string& ) const;
};

#endif
//
// File:     ClientSocket.h
// Author:   Adam.Lewis@athens.edu
// Purpose:
// Header file for the client portion of our C++ Sockets interface.   This code
// is based on code from an 2002 article in Linux Gazette
#include "ClientSocket.h"
#include "SocketException.h"
//
// ClientSocket::ClientSocket(std::string, int)
// Construct a new instance of the client side of our C++ socket interface.
// Pre-condition:
// Host and port passed from calling application.
```

```
//
// Post-condition:
// A new instance of our class is created on the heap.
//
ClientSocket::ClientSocket ( std::string host, int port )
{
  if ( ! Socket::create() )
    {
      throw SocketException ( "Could not create client socket." );
    }

  if ( ! Socket::connect ( host, port ) )
    {
      throw SocketException ( "Could not bind to port." );
    }
}
//
// ClientSocket::operator <<(const std::string &)
// Put a piece of data to the socket
// Pre-condition:
// String to be output is passed into method.
//
// Post-condition:
// The information is sent to the socket or an exception is thrown if the
// send fails.
//
const ClientSocket& ClientSocket::operator << ( const std::string& s ) const
{
  if ( ! Socket::send ( s ) )
    {
      throw SocketException ( "Could not write to socket." );
    }

  return *this;

}
//
// ClientSocket::operator >>(const std::string &)
// Get a piece of data from the socket.
// Pre-condition:
// String into which the data is to be dumped is passed into method.
//
// Post-condition:
// The information is read from the socket or an exception is thrown if the
// read failed.
//
const ClientSocket& ClientSocket::operator >> ( std::string& s ) const
{
  if ( ! Socket::recv ( s ) )
    {
      throw SocketException ( "Could not read from socket." );
    }

  return *this;
}
//
// File:     Socket.h
// Author:   Adam.Lewis@athens.edu
```

```
// Purpose:
// Header file for the main portion of our C++ Sockets interface.   This code
// is based on code from an 2002 article in Linux Gazette
//
#ifndef Socket_class
#define Socket_class
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <unistd.h>
#include <string>
#include <arpa/inet.h>
const int MAXHOSTNAME = 200;
const int MAXCONNECTIONS = 5;
const int MAXRECV = 500;
class Socket
{
 public:
  Socket();
  virtual ~Socket();

  // Server initialization
  bool create();
  bool bind ( const int port );
  bool listen() const;
  bool accept ( Socket& ) const;

  // Client initialization
  bool connect ( const std::string host, const int port );

  // Data Transimission
  bool send ( const std::string ) const;
  int recv ( std::string& ) const;


  void set_non_blocking ( const bool );

  bool is_valid() const { return m_sock != -1; }

 private:
  int m_sock;
  sockaddr_in m_addr;
};
#endif
//
// File:     Socket.cpp
// Author:   Adam.Lewis@athens.edu
// Purpose:
// Implementation file for the main portion of our C++ Sockets interface.  This
// code is based on code from an 2002 article in Linux Gazette.
//
#include "Socket.h"
#include "string.h"
#include <string.h>
#include <iostream>
#include <errno.h>
#include <fcntl.h>
```

```
//
// Socket::Socket()
// Construct a new instance of our class.
// Pre-condition:
// NONE.
//
// Post-condition:
// A cleared space for the address instance variable has been created.
//
Socket::Socket() :
  m_sock ( -1 )
{

  memset ( &m_addr,
         0,
         sizeof ( m_addr ) );

}
//
// Socket::~Socket()
// Destory an existing instance of our class.
// Pre-condition:
// A valide socket is available
//
// Post-condition:
// The socket is closed as needed.
//
Socket::~Socket()
{
  if ( is_valid() )
    ::close ( m_sock );
}
//
// Socket::create()
// Create a new socket and associate it with our instance.
// Pre-condition:
// Instance has been created.
//
// Post-condition:
// A reference to a valid socket can be found in our private variable.
//
bool Socket::create()
{
  m_sock = socket ( AF_INET,
                SOCK_STREAM,
                0 );

  if ( ! is_valid() )
    return false;


  // TIME_WAIT - argh
  int on = 1;
  if ( setsockopt ( m_sock, SOL_SOCKET, SO_REUSEADDR, ( const char* ) &on,
sizeof ( on ) ) == -1 )
    return false;
  return true;
```

```
}
//
// Socket::bind()
// Create a new socket and associate it with our instance.
// Pre-condition:
// We have a valid socket
//
// Post-condition:
// That socket is bound to the IP and port.
//
bool Socket::bind ( const int port )
{

  if ( ! is_valid() )
    {
      return false;
    }
  m_addr.sin_family = AF_INET;
  m_addr.sin_addr.s_addr = INADDR_ANY;
  m_addr.sin_port = htons ( port );
  int bind_return = ::bind ( m_sock,
                        ( struct sockaddr * ) &m_addr,
                        sizeof ( m_addr ) );
  if ( bind_return == -1 )
    {
      return false;
    }

  return true;
}
//
// Socket::listen()
// Create a new socket and associate it with our instance.
// Pre-condition:
// We have a valid socket
//
// Post-condition:
// We are listening on a valid socket.
//
bool Socket::listen() const
{
  if ( ! is_valid() )
    {
      return false;
    }

  int listen_return = ::listen ( m_sock, MAXCONNECTIONS );


  if ( listen_return == -1 )
    {
      return false;
    }

  return true;
}
//
// Socket::accept()
```

```
// Accpet a connection on this socket.
// Pre-condition:
// A valid and correct socket is passed to us for connection purposes
//
// Post-condition:
// We are accepting connections on that socket
//
bool Socket::accept ( Socket& new_socket ) const
{
  int addr_length = sizeof ( m_addr );
  new_socket.m_sock = ::accept ( m_sock, ( sockaddr * ) &m_addr, ( socklen_t * )
&addr_length );

  if ( new_socket.m_sock <= 0 )
    return false;
  else
    return true;
}
//
// Socket::send(const std::string)
// Send data out over our socket.
// Pre-condition:
// A valid and correct socket is passed to us for connection purposes
// Data is passed to us in the method parameter.
//
// Post-condition:
// The data gets sent over the socket.
//
bool Socket::send ( const std::string s ) const
{
  int status = ::send( m_sock, s.c_str(), s.size(), 0);
  if ( status == -1 )
    {
      return false;
    }
  else
    {
      return true;
    }
}
//
// Socket::recv(std::string&)
// Get data from our socket.
// Pre-condition:
// A valid and correct socket is passed to us for connection purposes
// We have a valid string to put data into
//
// Post-condition:
// The data gets read from the socket.
//
int Socket::recv ( std::string& s ) const
{
  char buf [ MAXRECV + 1 ];

  s = "";

  memset ( buf, 0, MAXRECV + 1 );
```

```
  int status = ::recv ( m_sock, buf, MAXRECV, 0 );

  if ( status == -1 )
    {
      std::cout << "status == -1   errno == " << errno << "  in Socket::recv\n";
      return 0;
    }
  else if ( status == 0 )
    {
      return 0;
    }
  else
    {
      s = buf;
      return status;
    }
}
//
// Socket::connect(const std::string, const int)
// Connect a socket to a host and port
// Pre-condition:
// A valid and correct socket is passed to us for connection purposes
// We have a valid host and port
//
// Post-condition:
// Our socket is connected to the host and port.
//
bool Socket::connect ( const std::string host, const int port )
{
  if ( ! is_valid() ) return false;

  m_addr.sin_family = AF_INET;
  m_addr.sin_port = htons ( port );

  int status = inet_pton ( AF_INET, host.c_str(), &m_addr.sin_addr );

  if ( errno == EAFNOSUPPORT ) return false;

  status = ::connect ( m_sock, ( sockaddr * ) &m_addr, sizeof ( m_addr ) );

  if ( status == 0 )
    return true;
  else
    return false;
}
//
// Socket::set_non_blocking(bool)
// The socket state is set to non-blocking.
// Pre-condition:
// A valid and correct socket is passed to us for connection purposes
//
// Post-condition:
// The state of the socket has been changed.
//
void Socket::set_non_blocking ( const bool b )
{
  int opts;
  opts = fcntl ( m_sock,
```

```
            F_GETFL );
  if ( opts < 0 )
    {
      return;
    }
  if ( b )
    opts = ( opts | O_NONBLOCK );
  else
    opts = ( opts & ~O_NONBLOCK );
  fcntl ( m_sock,
        F_SETFL,opts );
}
#include "ClientSocket.h"
#include "SocketException.h"
#include <iostream>
#include <string>

int main ( int argc, char *argv[] )
{
  std::string ary[10] = {
    "0)A beautiful, smart, and loving person will be coming into your life.",
    "1)A dubious friend may be an enemy in camouflage.",
    "2)A fresh start will put you on your way.",
    "3)A friend asks only for your time not your money.",
    "4)A friend is a present you give yourself.",
    "5)A gambler not only will lose what he has, but also will lose what he
doesn't have.",
    "6)A golden egg of opportunity falls into your lap this month",
    "7)A good friendship is often more important than a passionate romance.",
    "8)A hunch is creativity trying to tell you something.",
    "9)A lifetime friend shall soon be made."
  };
  for (int i = 0; i < 10; ++i){
  try
  {

    ClientSocket client_socket("localhost", 30000);

    std::string reply;

    try
    {
      int x = rand() % 10;
      client_socket << ary[x];
      client_socket >> reply;
    }
    catch (SocketException &)
    {
    }

    std::cout << "We received this response from the server:\n\"" << reply <<
"\"\n";
    ;

    }
  catch ( SocketException& e )
    {
      std::cout << "Exception was caught:" << e.description() << "\n";
```

```
    }
  }
  return 0;
}
#include "ServerSocket.h"
#include "SocketException.h"
#include <string>
#include <iostream>

int main ( int argc, char * argv[] )
{
  std::cout << "running....\n";

  try
    {
      // Create the socket
      ServerSocket server ( 30000 );

      while ( true )
      {

        ServerSocket new_sock;
        server.accept ( new_sock );

        try
          {
            while ( true )
            {
              std::string data;
              new_sock >> data;
              new_sock << data;
            }
          }
        catch ( SocketException& ) {}

      }
    }
  catch ( SocketException& e )
    {
      std::cout << "Exception was caught:" << e.description() << "\nExiting.\n";
    }

  return 0;
}
```