Inhaltsverzeichnis

1. Ein Stream Socket API für C++....1
1.1. Die Basisklasse: Socket.h....2
1.2. Die Basisklasse: Socket.cpp....2
1.3. Die Unterklasse: ServerSocket....5
1.3.1. Ein Beispielprogramm: simple_server_main.cpp....5
1.3.2. Die Unterklasse: ServerSocket.h....6
1.3.3. Die Unterklasse: ServerSocket.cpp....7
1.4. Die Unterklasse: ClientSocket....8
1.4.1. Ein Beispielprogramm: simple_client_socket.cpp....8
1.4.2. Die Unterklasse: ClientSocket.h....9
1.4.3. Die Unterklasse: ClientSocket.cpp....9
1.5. Das Makefile....10
1.6. +Aufgaben....11
Aufgabe: rate-server.c, rate-client.c....11
Aufgabe: tictactoe-server.c, tictactoe-client.c....11
1.7. Zusammenfassung....12

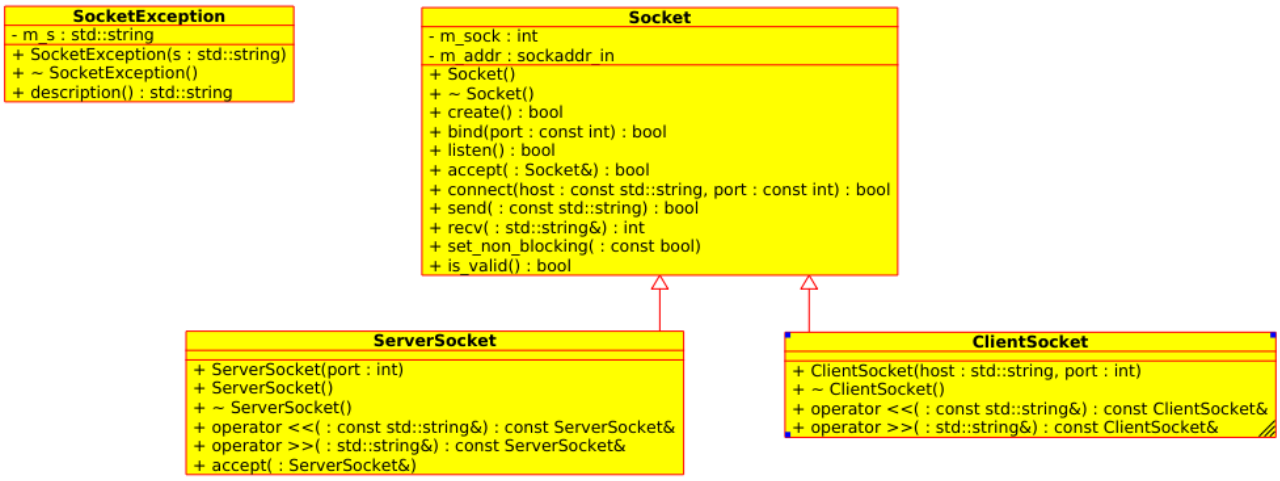
☑ Socket-Klassen erstellen:
http://www.pcs.cnu.edu/~dgame/sockets/socketsC++/sockets.html

☑ C++ Socket-Library Tutorial:
http://www.alhem.net/Sockets/tutorial/index.html

1. Ein Stream Socket API für C++

Dies hier ist eine Implementierung der klassischen Socket-C-Funktionen in einer UNIX-Umgebung. Das Lesen/Schreiben von Daten erfolgt über Stream-Klassen. Auch das Lesen blockiert, wenn keine Daten im Socket-Eingang vorhanden sind.

Sehen Sie zunächst das Klassendiagramm:



1.1. Die Basisklasse: Socket.h

Socket ist die **Basis Wrapper Klasse** der Standard Socket API-Funktionen in C:

Hier die Socket.h (Ein Auszug)

```
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;
```

Die Socket Klasse wird benutzt, um

1. einen ServerSocket
2. einen ClientSocket

zu erzeugen. Diese beiden bieten Exception Handling und ein Stream Interface an.

1.2. Die Basisklasse: Socket.cpp

Hier nun der Programmcode für die Socket-Schnittstelle: Socket.cpp (Ein Auszug)

Betrachten Sie vor allem die fett markierten Teile.

```
// Socket.cpp: Implementation of the Socket class.

#include "Socket.h"
#include "string.h"
#include <string.h>
#include <errno.h>
```

```
#include <fcntl.h>

Socket::Socket() :  m_sock ( -1 ){
  memset ( &m_addr,0, sizeof ( m_addr ) );
}

Socket::~Socket(){
  if ( is_valid() )
    ::close ( m_sock );
}

bool Socket::create(){
  m_sock = socket ( AF_INET,SOCK_STREAM,0 );

  if ( ! is_valid() ) return false;

  // …

  return true;
}

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;
}


bool Socket::listen() const
{
  if ( ! is_valid() ) return false;

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


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

  return true;
}
```

```
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;
}


bool Socket::send ( const std::string s ) const {

  int status = ::send ( m_sock, s.c_str(), s.size(), MSG_NOSIGNAL );
  if ( status == -1 )
      return false;
  else
      return true;
}


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 ){
      cout << "status == -1 errno == " << errno << ;
      cout << "  in Socket::recv\n";
      return 0;
    }
  else if ( status == 0 ){
      return 0;
    }
  else{
      s = buf;
      return status;
    }
}

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;
}

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 );

}
```

1.3. Die Unterklasse: ServerSocket

Die Unterklasse ServerSocket kann zB. auf folgende Weise verwendet werden:

1.3.1. Ein Beispielprogramm: simple_server_main.cpp

```
#include "ServerSocket.h"
#include "SocketException.h"
#include <string>

//ht
#include <iostream>
using namespace std;

// ht
int main ( int argc, char* argv[] ){

  try
```

```
    {
      // Create the socket
      ServerSocket server ( 30000 );
      std::cout << "running.... port " << 30000<<endl;

      while ( true )
     {
      ServerSocket new_sock;
      server.accept ( new_sock );

      try
        {
          while ( true )
         {
           // READ
           std::string data;
           new_sock >> data;

            cout << "SERVER: got <" << data << ">" << endl;

           //WRITE
           new_sock << data;
         }
        }
      catch ( SocketException& ) {}

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

  return 0;
}
```

1.3.2. Die Unterklasse: ServerSocket.h

// Definition of the ServerSocket class

```
#ifndef ServerSocket_class
#define ServerSocket_class

#include "Socket.h"


class ServerSocket : private Socket
{
```

```
 public:

  ServerSocket ( int port );
  ServerSocket (){};
  virtual ~ServerSocket();

  const ServerSocket& operator << ( const std::string& ) const;
  const ServerSocket& operator >> ( std::string& ) const;

  void accept ( ServerSocket& );

};
#endif
```

1.3.3. Die Unterklasse: ServerSocket.cpp

// Implementation of the ServerSocket class

```
#include "ServerSocket.h"
#include "SocketException.h"


ServerSocket::ServerSocket ( int port )
{
  if ( ! Socket::create() )
    {
      throw SocketException ( "Could not create server socket." );
    }

  if ( ! Socket::bind ( port ) )
    {
      throw SocketException ( "Could not bind to port." );
    }

  if ( ! Socket::listen() )
    {
      throw SocketException ( "Could not listen to socket." );
    }

}

ServerSocket::~ServerSocket()
{
}


const ServerSocket& ServerSocket::operator << ( const std::string& s )
const
{
  if ( ! Socket::send ( s ) )
    {
      throw SocketException ( "Could not write to socket." );
```

```
    }

  return *this;

}


const ServerSocket& ServerSocket::operator >> ( std::string& s ) const
{
  if ( ! Socket::recv ( s ) )
    {
      throw SocketException ( "Could not read from socket." );
    }

  return *this;
}

void ServerSocket::accept ( ServerSocket& sock )
{
  if ( ! Socket::accept ( sock ) )
    {
      throw SocketException ( "Could not accept socket." );
    }
}
```

1.4. Die Unterklasse: ClientSocket

Auch hier wieder ein kleines Demo-Programm:

1.4.1. Ein Beispielprogramm: simple_client_socket.cpp

```
#include "ClientSocket.h"
#include "SocketException.h"
#include <iostream>
using namespace std;
#include <string>

// ht
int main ( int argc, char* argv[] )
{
  try
    {

      ClientSocket client_socket ( "localhost", 30000 );

      std::string reply;

      try
     {
       client_socket << "Test message.";
       client_socket >> reply;
     }
```

```
      catch ( SocketException& ) {}

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

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

  return 0;
}
```

Der Client sieht folgendermaßen aus:

1.4.2. Die Unterklasse: ClientSocket.h

```
// Definition of the ClientSocket class

#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
```

1.4.3. Die Unterklasse: ClientSocket.cpp

```
// Implementation of the ClientSocket class

#include "ClientSocket.h"
#include "SocketException.h"

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." );
    }

}


const ClientSocket& ClientSocket::operator << ( const std::string& s )
const
{
  if ( ! Socket::send ( s ) )
    {
      throw SocketException ( "Could not write to socket." );
    }

  return *this;

}


const ClientSocket& ClientSocket::operator >> ( std::string& s ) const
{
  if ( ! Socket::recv ( s ) )
    {
      throw SocketException ( "Could not read from socket." );
    }

  return *this;
}
```

1.5. Das Makefile

```
# Makefile for the socket programming example
#
socklibs = -lnsl -lsocket
simple_server_objects = ServerSocket.o Socket.o simple_server_main.o
simple_client_objects = ClientSocket.o Socket.o simple_client_main.o


all : simple_server simple_client

simple_server: $(simple_server_objects)
        g++ -o simple_server $(socklibs) $(simple_server_objects)
```

```
simple_client: $(simple_client_objects)
        g++ -o simple_client $(socklibs) $(simple_client_objects)


Socket: Socket.cpp
ServerSocket: ServerSocket.cpp
ClientSocket: ClientSocket.cpp
simple_server_main: simple_server_main.cpp
simple_client_main: simple_client_main.cpp


clean:
        rm -f *.o simple_server simple_client
```

Aufruf:
Terminal1: ./simple_server_main
Terminal2: ./simple_client_main

1.6. +Aufgaben

Aufgabe: rate-server.c, rate-client.c

Schreiben Sie ein Programm zum Zahlenraten

Der Server:
nach dem Verbindungsaufbau durch den Client denkt sich der Server
eine Zahl zwischen 0 und 99 aus.

Dann wiederholt, bis zum Treffer...
1.Client
* liest vom User eine Zahl (0-99) ein und
* schickt diese an den Server
2.Server
* liest die geratene Zahl vom Client
* bewertet die geratene Zahl mit
  "zu tief", "getroffen" oder "zu hoch"
* sendet die Bewertung an den Client
3.Client
* liest die Bewertung und
* leitet daraus einen neuen Rateversuch ab, oder (weil getroffen) endet

Aufgabe: tictactoe-server.c, tictactoe-client.c

Schreiben Sie ein Client- und ein Serverprogramm zum Spiel TicTacToe. Verwendet werden soll der Port 71234.

Der Client baut eine Verbindung auf und spielt gegen den Server.

1.7. Zusammenfassung

Gib je ein Beispiel zu:

SocketClient
SocketServer