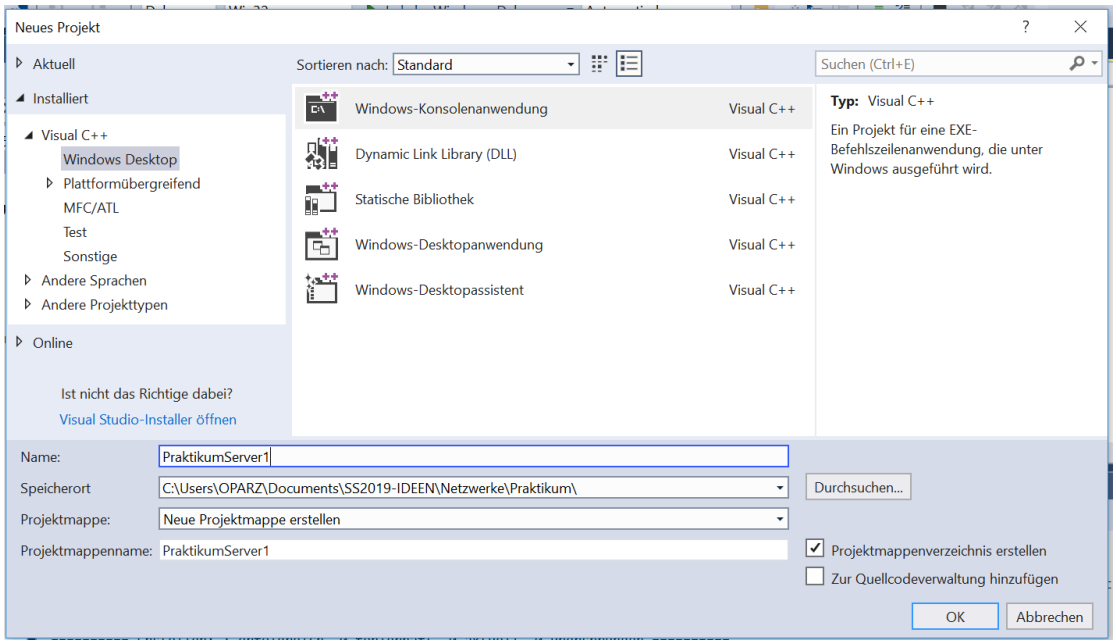
Praktikum Netzwerke

Socketprogrammierung mit Visual Studio

Programmieren eines Servers (1)

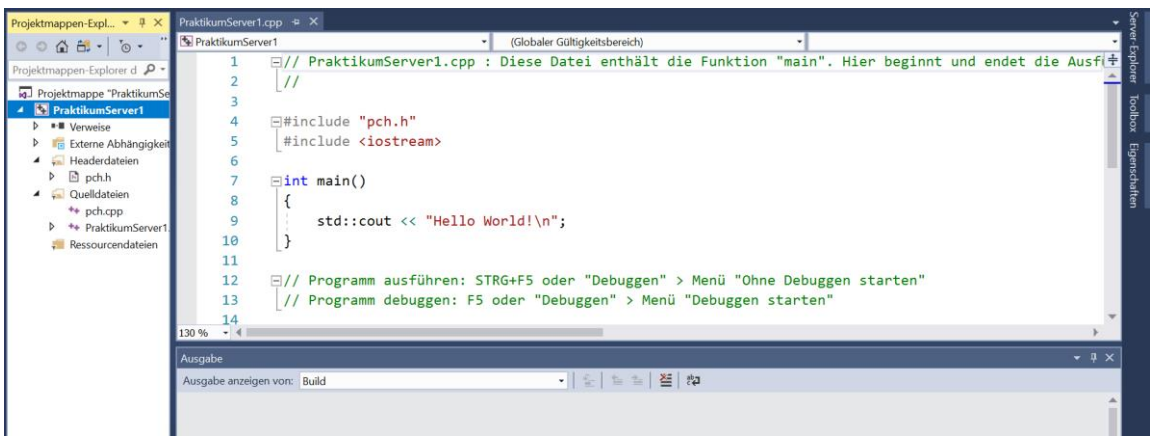
Erstellen Sie ein neues Projekt mit Visual Studio:



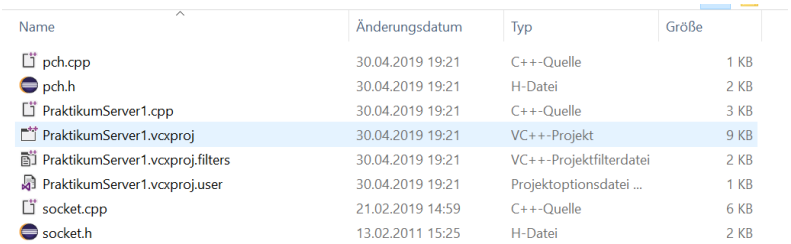
Für den Speicherort tragen Sie Ihr Netzlaufwerk und entsprechende Unterverzeichnisse ein, z.B.:

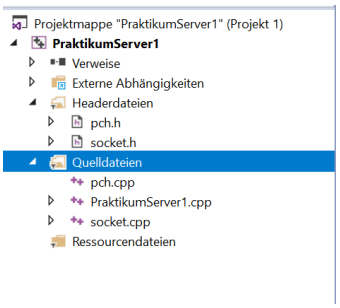
\\192.168.10.113\homes\twpn_pa\

Nach dem erfolgreichen Erstellen des Projekts sieht Ihr Starbildschirm so aus:

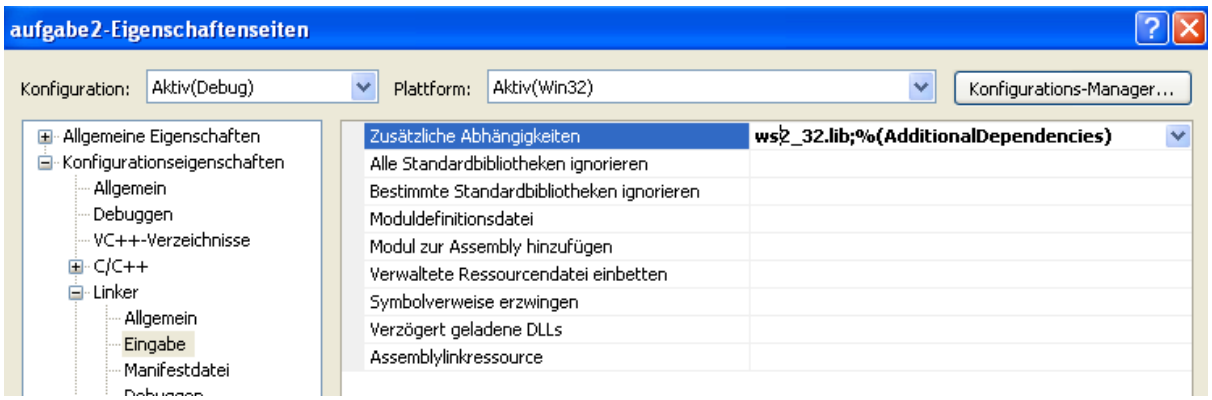


Die bereitgestellten Dateien socket.cpp und socket.h müssen Sie in das Verzeichnis der anderen Quelldateien kopieren und dann in Ihr Projekt hinzufügen:





Am Ende müssen Sie nur noch die Library ws2_32.lib in das Projekt eintragen:



1. Programmieren Sie einen Server, der eine Verbindung annimmt und anschließend beendet wird.
   - Der Server soll nur Verbindungen an Port 60 annehmen
   - Die Ausgabe an der Konsole soll jede der Socketfunktionen dokumentieren:
     - *Server: socket() erfolgreich*
     - *Server: bind()...*
     - *Server: listen()...*
     - *Server: accept...*
     - *Server: close()...*

Bei den Socketfunktionen (bind(), ... close()) soll zwischen erfolgreich und nicht erfolgreich unterschieden werden. Bei Erfolg soll z.B.

- *Server: bind an Port xxx erfolgreich.*

bzw. bei nicht erfolgreichen „Binden“ des Sockets

- *Server: bind an Port xxx nicht erfolgreich.*

ausgegeben werden.

Bei der Socketfunktion accept(..) soll die folgende Ausgabe erscheinen:

- *Server: accept()...Verbindung mit < irgendeine IP> an Port 60 hergestellt*

bzw:

- *Server: accept()...Verbindung mit < irgendeine IP> an Port 60 nicht hergestellt*

Das Testen des Servers ist sehr einfach mit einem Browser möglich. Geben Sie in die Eingabezeile des Browsers den folgenden Text ein:

http://127.0.0.1:60

alternativ können Sie auch den symbolischen Namen `localhost` benutzen

http://localhost:60

Der Browser wird damit veranlasst, eine http Verbindung mit dem „loopback“ Interface herzustellen.
Sie können den Server natürlich auch von einem anderen beliebigen Rechner im Netzwerk ansprechen, vorausgesetzt Sie kennen die IP-Adresse des Servers.

2. Erweitern Sie nun den Server so dass er nicht nach jeder Anfrage abbricht. Bei jeder Verbindungsanfrage soll ein Zähler inkrementiert und auf der Konsole ausgegeben werden. Die Ausgabe soll in der folgenden Form erfolgen:

   - *Server: accept()...Verbindung Nr. xx an Port 60 hergestellt*

Anstelle eines Browsers können Sie zum Testen des Servers auch den in der letzten Aufgabe programmierten Client benutzen

3. Wireshark:

Untersuchen Sie die Datenpakete auf dem Server- und dem Client-Rechner.
Beachten Sie, dass es mit Wireshark nicht möglich ist auf das loopback Interface einen Capture durchzuführen. Sie müssen also jeweils einen anderen Rechner im Netzwerk als Client benutzen.

- Welcher der beiden Rechner baut die Verbindung auf und welcher baut sie wieder ab?
- Welche Portnummern werden verwendet?
- Welches Protokoll wird verwendet?

Programmieren eines Servers (2):

Codeschnippsel:

```
char web_seite[] = "HTTP/1.1 200 OK\r\n\r\n\
<html><body><h1>Hallo Du schönes weites Web</h1></body></html>";
```

oder:

```
string web_seite ("HTTP/1.1 200 OK\r\n\r\n\
<html><body><h1>Hallo Du schönes weites Web</h1></body></html>");
```

1. Schreiben Sie ein Server-Programm, das HTML-Dateien an einen Web-Browser sendet. Als Beispiel kann der oben gezeigte String verwendet werden. Im Browser (oder einem selbst geschriebenen Client) sollte bei Erfolg der oben gedruckte String angezeigt werden. Das Serverprogramm soll beliebig oft den Zugriff auf die „Webseite“ ermöglichen. Es muss also für jeden Zugriff ein neuer Client- Socket erzeugt, und nachher auch wieder geschlossen werden. Der Server soll über den Port 60 erreichbar sein.
2. Es soll nun der folgende String bei jedem Aufruf des Clients angezeigt werden:

   ```
   Vielen Dank für den Zugriff auf den Server. Sie sind der xxx.
   Besucher dieser Webseite
   ```

   Tip: Benutzen Sie die Funktion sprintf(...)

Die Anzeige an der Server-Konsole soll wie folgt aussehen:

```
Server: socket(). . .
Server: bind(). . .
Server: listen(). . .
Server: Server an Port xxx wartet . . .

Server: accept()...Verbindung angenommen
Server: empfangen:

→Geben Sie an dieser Stelle die http Anfrage des Clients auf der
Konsole aus
```

Musterdatei:

```
/* Server, der 10 Zugriffe eines Clients zulässt und sich dann beendet.     */
#include "pch.h"
#include "socket.h"
#include <iostream>

int main(int argc, char* argv[])
{
	bool ret;
	string str("hallo");
	string buffer;
	Socket sock_server;
	Socket sock_client;
	unsigned short portnummer = 51;

	sock_server.create();
	ret = sock_server.bind(portnummer);
	if (ret == false){
		std::cout << "Server: bind failed" << endl;
		exit(1);
	}
	else
		std::cout << "Server: bind successful on port: " << portnummer << endl;

	ret = sock_server.listen();
	if (ret == false){
		std::cout << "Server: listen failed" << endl;
		exit(1);
	}
	else
		std::cout << "Server: listen" << endl;
	int i=0;
	while(i++ <10){
		ret = sock_server.accept(sock_client);
		if (ret == false){
			std::cout << "Server: accept failed" << endl;
			exit(1);
		}
		else{
			std::cout << "Server: accept with portnumber: "<< portnummer << endl;

			sock_client.recv(buffer);
			cout << buffer << endl;
			sock_client.send("Hallo Welt");
			sock_client.close();
		}
	}
	ret = sock_server.close();
	if (ret == false){
		std::cout << "Server: close failed" << endl;
		exit(1);
	}
	else
		std::cout << "Server: close erfolgreich" << endl;

	sock_server.cleanup();		//needed for Winsock.dll
	std::cout << "Das wars dann" << endl;
	return 0;
}
```

```
// aufgabe5.cpp : Serverprojekt, das den die Zugriffe auf den Server zählt.

#include "pch.h"
#include "socket.h"
#include <iostream>
#include <sstream>

int main(int argc, char* argv[])
{
	bool ret;
	string str;
	Socket sock_server;
	Socket sock_client;

	string buffer;
	string webseite;
	string webseite1
	(" <html><head>\
	   <meta http-equiv=Content-Type content='\"'text/html; charset=windows-1252'\"'> \
	       <meta name=Generator content='\"'Microsoft Word 10 (filtered)'\"'> \
	       <title>Hallo, jetzt hats endlich geklappt</title>\
	       <style>\
	       <!--\
	        /* Style Definitions */\
	        p.MsoNormal, li.MsoNormal, div.MsoNormal\
	              {margin:0cm;\
	              margin-bottom:.0001pt;\
	              font-size:12.0pt;\
	              font-family:'\"'Times New Roman'\"';}\
	          @page Section1\
	              {size:595.3pt 841.9pt;\
	              margin:70.85pt 70.85pt 2.0cm 70.85pt;}\
	   div.Section1      {page:Section1;}\
	   -->\
	   </style></head><body lang=DE><div class=Section1>\
	  <p class=MsoNormal><b><span style='font-size:20.0pt'>Hallo, jetzt hats endlich\
	       geklappt. "
	);

	string webseite2
	("     </span></b></p>\
	       </div></body></html>\
	       \r\n\r\n"
	);


	string serverResponse
	(
	       "HTTP/1.1 200 OK\r\n \
	       Date: Wed, 04 May 2017 01:21:22 GMT\r\n \
	       Server: oparz \r\n \
	       Last Modified: Tue, 29 Oct 2002 22:21:19 GMT\r\n \
	       Content-Length: 2232 \r\n \
	       Content-Type: text/html \r\n \
	       \r\n"
	);
```

```
	const unsigned short int port = 80;

	sock_server.create();
	ret = sock_server.bind(port);
	if (ret == false){
		cout << "Server: bind failed" << endl;
		exit(1);
	}
	else
		cout << "Server: bind successful on port: " << port << endl;

	ret = sock_server.listen();
	if (ret == false){
		cout << "Server: listen failed" << endl;
		exit(1);
	}
	else{
		cout << "Server: listen" << endl;

		int count_website = 1;
		while(1){

			ostringstream sstream;
			sstream << "Herzlichen Glückwunsch, Du bist der  " <<  count_website++
				<< " te Besucher der Webseite." ;
			string s2 = sstream.str();

			webseite = webseite1 + s2 + webseite2 ;

			ret = sock_server.accept(sock_client);
			if (ret == false){
				cout << "Server: accept failed" << endl;
				exit(1);
			}
			else
				cout << "Server: accept with portnumber: "<< port << endl;
			sock_client.recv(buffer);
			cout << endl << buffer << endl;
			/*
			if (ret = sock_client.send(serverResponse) == false)
				cout << "Server: send response failed" << endl;
			cout << endl << "Antwort des Servers:   " << endl  << serverResponse
				<< endl;
			*/
			sock_client.send(webseite);		//
			cout << endl << "Webseite an den Client   :"  << endl << webseite
				<< endl;
			Sleep(500);

			ret = sock_client.close();
			if (ret == false){
				cout << "Client: close  failed" << endl;
				exit(1);
			}
			else
				cout << "Client: close  success" << endl;
		}
	}
	return 0;
}
```