############# SERVER #######################

1. Create the project “Win32 Console Application” with “itec\_srv01” name for example-> OK -> Next -> Finish

2. Copy the code ask below (“yellow” highlight)

|  |
| --- |
| // itec\_srv01.cpp : Defines the entry point for the console application.  //  #include "stdafx.h"  #define \_WINSOCK\_DEPRECATED\_NO\_WARNINGS  #include <winsock2.h>  #include <stdio.h>  #include <tchar.h>  #include <WS2tcpip.h>  // Need to link with Ws2\_32.lib  #pragma comment (lib, "Ws2\_32.lib")  void errexit(const char\*, ...);  void pause(void);  int \_tmain(int argc, TCHAR\* argv[])  {  WORD wVersionRequested;  WSADATA wsaData;  int retcode;  SOCKET socket\_descriptor;  struct sockaddr\_in sin;  SOCKET new\_socket;  struct sockaddr\_in new\_sin;  int addrlen;  char ClientAddr[4];  HOSTENT\* pClientHostEnt;  char Buffer[4096];  int length;  char Message[] = "Message from the server.";  /\*  [1] WSAStartup() must be called before any other socket  routines. The following code prints all returned  information except for the vendor-specific data structure.  WSAStartup returns the following information in the  WSAData structure.  struct WSAData {  WORD wVersion;  WORD wHighVersion;  char szDescription[WSADESCRIPTION\_LEN+1];  char szSystemStatus[WSASYSSTATUS\_LEN+1];  unsigned short iMaxSockets; (ignore in WinSock 2)  unsigned short iMaxUdpDg; (ignore in WinSock 2)  char FAR \* lpVendorInfo;(ignore in Winsock 2)  };  Note that WSAGetLastError() should not be used since  the error code may not be saved properly if winsock.dll  did not load.  \*/  printf("WSAStartup()\n");  wVersionRequested = MAKEWORD(2, 2); // Use MAKEWORD(1,1) if you're at WinSock 1.1  retcode = WSAStartup(wVersionRequested, &wsaData);  if (retcode != 0)  errexit("Startup failed: %d\n", retcode);  printf("Return Code: %i\n", retcode);  printf("Version Used: %i.%i\n", LOBYTE(wsaData.wVersion), HIBYTE(wsaData.wVersion));  printf("Version Supported: %i.%i\n", LOBYTE(wsaData.wHighVersion), HIBYTE(wsaData.wHighVersion));  printf("Implementation: %s\n", wsaData.szDescription);  printf("System Status: %s\n", wsaData.szSystemStatus);  printf("\n");  if (LOBYTE(wsaData.wVersion) != LOBYTE(wVersionRequested) ||  HIBYTE(wsaData.wVersion) != HIBYTE(wVersionRequested))  {  printf("Supported version is too low\n");  WSACleanup();  return 0;  }  pause();  /\*  [2] Once WSAStartup has been called, the socket can be  created using the socket() call. The following creates  an Internet protocol family (PF\_INET) socket providing  stream service (SOCK\_STREAM).  \*/  printf("socket()\n");  socket\_descriptor = socket(PF\_INET, SOCK\_STREAM, 0);  if (socket\_descriptor == INVALID\_SOCKET)  errexit("Socket creation failed: %d\n", WSAGetLastError());  printf("Socket Descriptor: %i\n", socket\_descriptor);  printf("\n");  pause();  /\*  [3] Before accepting a connection, the server must bind  a host address and port number to the socket using bind().  bind() requires that we first initialize an Internet  address family structure.  struct sockaddr\_in {  short sin\_family;  u\_short sin\_port;  struct in\_addr sin\_addr;  char sin\_zero[8];  };  The structure is first set to all zeros. The family is  AF\_INET. We'll use 4984 as the port number for this  example. It is mapped to network byte order using htons().  We use the special identifier INADDR\_ANY (00.00.00.00) to  indicate that connections can be accepted on any interface  for this host.  \*/  memset(&sin, 0, sizeof(sin));  sin.sin\_family = AF\_INET;  sin.sin\_port = htons(4984);  sin.sin\_addr.s\_addr = INADDR\_ANY;  /\*  [4] bind() is used to bind the address to the socket.  \*/  printf("bind()\n");  retcode = bind(socket\_descriptor, (struct sockaddr\*) & sin, sizeof(sin));  if (retcode == SOCKET\_ERROR)  errexit("Bind failed: %d\n", WSAGetLastError());  printf("Return Code: %d\n", retcode);  printf("\n");  pause();  /\*  [5] listen() is used to make the socket passive so that it  is ready to accept incoming connection requests. For this  example, we'll not allow any enqueued requests, so the  queuelen parameter is set to 0.  \*/  printf("listen()\n");  retcode = listen(socket\_descriptor, 0);  if (retcode == SOCKET\_ERROR)  errexit("Listen failed: %d\n", WSAGetLastError());  printf("Return Code: %d\n", retcode);  printf("\n");  pause();  /\*  [6] accept() is used to accept a connection. accept()  returns a descriptor for a new socket for the new  connection and provides the address of the remote host.  Note that accept() blocks until a connection is accepted.  \*/  printf("accept()\n");  addrlen = sizeof(new\_sin);  new\_socket = accept(socket\_descriptor, (struct sockaddr\*) & new\_sin, &addrlen);  if (new\_socket == INVALID\_SOCKET)  errexit("Accept failed: %d\n", WSAGetLastError());  printf("New Socket Descriptor: %d\n", new\_socket);  printf("Remote Port: %d\n", ntohs(new\_sin.sin\_port));  printf("Host Address: %s\n", inet\_ntoa(new\_sin.sin\_addr));  printf("\n");  pause();  /\*  [7] Although not required for most servers, a server  can determine information about the remote end point,  i.e. the client. gethostbyaddr() is used to resolve the  IP address from accept() to determine client host name.  \*/  printf("gethostbyaddr(\"%s\")\n", inet\_ntoa(new\_sin.sin\_addr));  memcpy(ClientAddr, &new\_sin.sin\_addr, sizeof(ClientAddr));  pClientHostEnt = gethostbyaddr(ClientAddr, sizeof(ClientAddr), PF\_INET);  if (pClientHostEnt == NULL)  errexit("Get host by address failed: %d\n", GetLastError());  printf("Remote Host: %s\n", pClientHostEnt->h\_name);  printf("\n");  pause();  /\*  [8] The server is now ready to receive data using recv().  No flags are set. We assume here that all data will be  read with one recv() call. This is, in general, not a  good assumption because a stream transport protocol is  used.  \*/  printf("recv()\n");  length = recv(new\_socket, Buffer, sizeof Buffer, 0);  if (length == SOCKET\_ERROR)  errexit("Receive failed: %d\n", WSAGetLastError());  printf("Bytes received: %d\n", length);  printf("Message: %s\n", Buffer);  printf("\n");  pause();  /\*  [9] send() is used to send a message to the client.  No flags are set.  \*/  printf("send()\n");  retcode = send(new\_socket, Message, sizeof Message, 0);  if (retcode == SOCKET\_ERROR)  errexit("Send failed: %d\n", WSAGetLastError());  printf("Bytes Sent: %d\n", retcode);  printf("\n");  pause();  /\*  [10] The server closes both sockets using closesocket();  In a normal server, the server would probably just close  new\_socket, but keep socket\_descriptor open to accept  new connections.  \*/  printf("closesocket()\n");  retcode = closesocket(socket\_descriptor);  if (retcode == SOCKET\_ERROR)  errexit("Close socket failed: %d\n", WSAGetLastError());  printf("Return Code: %d\n", retcode);  printf("\n");  printf("closesocket()\n");  retcode = closesocket(new\_socket);  if (retcode == SOCKET\_ERROR)  errexit("Close socket failed: %d\n", WSAGetLastError());  printf("Return Code: %d\n", retcode);  printf("\n");  pause();  /\*  [11] WSACleanup() is used to terminate use of socket  services. A normal server would probably continue to  execute "forever."  \*/  printf("WSACleanup()\n");  retcode = WSACleanup();  if (retcode == SOCKET\_ERROR)  errexit("Cleanup failed: %d\n", WSAGetLastError());  printf("Return Code: %d\n", retcode);  printf("\n");  pause();  return 0;  }  void errexit(const char\* format, ...)  {  va\_list args;  va\_start(args, format);  vfprintf(stderr, format, args);  va\_end(args);  WSACleanup();  pause();  exit(1);  }  void pause(void)  {  char c;  printf("Press Enter to continue\n");  scanf\_s("%c", &c);  } |

3. Build and run the code

############# CLIENT #######################

1. Create the project “Win32 Console Application” with “itec\_clnt01” name for example-> OK -> Next -> Finish

2. Copy the code ask below (“yellow” highlight)

|  |
| --- |
| // itec\_clnt01.cpp : Defines the entry point for the console application.  //  #include "stdafx.h"  #define \_WINSOCK\_DEPRECATED\_NO\_WARNINGS  #include <winsock2.h>  #include <stdio.h>  #include <tchar.h>  #include <WS2tcpip.h>  #include <string.h>  // Need to link with Ws2\_32.lib  #pragma comment (lib, "Ws2\_32.lib")  void errexit(const char\*, ...);  void pause(void);  int \_tmain(int argc, \_TCHAR\* argv[])  {  WORD wVersionRequested;  WSADATA wsaData;  int retcode;  SOCKET socket\_descriptor;  char ServerName[64];  HOSTENT\* pHostEnt;  struct sockaddr\_in sin;  char Message[] = "Greetings from the client";  char Buffer[4096];  int length;  /\*  [1] WSAStartup() must be called before any other socket  routines. The following code prints all returned  information valid in WinSock 2.  WSAStartup returns the following information in the  WSAData structure.  struct WSAData {  WORD wVersion;  WORD wHighVersion;  char szDescription[WSADESCRIPTION\_LEN+1];  char szSystemStatus[WSASYSSTATUS\_LEN+1];  unsigned short iMaxSockets; (ignore in WinSock 2)  unsigned short iMaxUdpDg; (ignore in WinSock 2)  char FAR \* lpVendorInfo;(ignore in WinSock 2)  };  Note that WSAGetLastError() should not be used since the  error code may not be saved properly if winsock.dll did  not load.  \*/  printf("WSAStartup()\n");  wVersionRequested = MAKEWORD(2, 2); // Use MAKEWORD(1,1) if you're at WinSock 1.1  retcode = WSAStartup(wVersionRequested, &wsaData);  if (retcode != 0)  errexit("Startup failed: %d\n", retcode);  printf("Return Code: %i\n", retcode);  printf("Version Used: %i.%i\n", LOBYTE(wsaData.wVersion), HIBYTE(wsaData.wVersion));  printf("Version Supported: %i.%i\n", LOBYTE(wsaData.wHighVersion), HIBYTE(wsaData.wHighVersion));  printf("Implementation: %s\n", wsaData.szDescription);  printf("System Status: %s\n", wsaData.szSystemStatus);  printf("\n");  pause();  if (LOBYTE(wsaData.wVersion) != LOBYTE(wVersionRequested) ||  HIBYTE(wsaData.wVersion) != HIBYTE(wVersionRequested))  {  printf("Supported version is too low\n");  WSACleanup();  return 0;  }  /\*  [2] Once WSAStartup has been called, the socket can be  created using the socket() call. The following creates an  Internet protocol family (PF\_INET) socket providing stream  service (SOCK\_STREAM).  \*/  printf("socket()\n");  socket\_descriptor = socket(PF\_INET, SOCK\_STREAM, 0);  if (socket\_descriptor == INVALID\_SOCKET)  errexit("Socket creation failed: %d\n", WSAGetLastError());  printf("Socket Descriptor: %i\n", socket\_descriptor);  printf("\n");  pause();  /\*  [3] Before making a connection, an Internet address  family structure must be initialized.  struct sockaddr\_in {  short sin\_family;  u\_short sin\_port;  struct in\_addr sin\_addr;  char sin\_zero[8];  };  It is first set to all zeros. The family is AF\_INET.  We'll use 4984 as the port number for this example.  htons() converts to network byte ordering.  gethostbyname() is used to determine the address of the  remote host. For this example, we connect to the host  specified in the first command line argument or the local  host ("localhost") if no command line arguments are  provided, using the first address for this host in the  h\_addr\_list.  gethostbyname() returns a pointer to the following  structure. Some of the contents are printed below.  struct hostent {  char FAR \* h\_name;  char FAR \* FAR \* h\_aliases;  short h\_addrtype;  short h\_length;  char FAR \* FAR \* h\_addr\_list;  };  \*/  memset(&sin, 0, sizeof(sin));  sin.sin\_family = AF\_INET;  sin.sin\_port = htons(4984);  strcpy\_s(ServerName, "localhost");  printf("gethostbyname(\"%s\")\n", ServerName);  if (pHostEnt = gethostbyname(ServerName)) {  memcpy(&sin.sin\_addr, pHostEnt->h\_addr\_list[0], pHostEnt->h\_length);  printf("Address Length: %d\n", pHostEnt->h\_length);  printf("Host Address: %s\n", inet\_ntoa(sin.sin\_addr));  printf("Host Name: %s\n", pHostEnt->h\_name);  printf("\n");  pause();  }  else errexit("Can't get %s\" host entry: %d\n", ServerName, WSAGetLastError());  /\*  [4] connect() is used to establish a connection to a remote  endpoint.  \*/  printf("connect()\n");  retcode = connect(socket\_descriptor, (struct sockaddr\*) & sin, sizeof(sin));  if (retcode == SOCKET\_ERROR)  errexit("Connect failed: %d\n", WSAGetLastError());  printf("Return Code: %d\n", retcode);  printf("\n");  pause();  /\*  [5] send() is used to send a message to the server. No  flags are set.  \*/  printf("send()\n");  retcode = send(socket\_descriptor, Message, sizeof Message, 0);  if (retcode == SOCKET\_ERROR)  errexit("Send failed: %d\n", WSAGetLastError());  printf("Bytes Sent: %d\n", retcode);  printf("\n");  pause();  /\*  [7] recv() is used to receive a message from the server.  No flags are set. This code assumes that all data is  read with the first and only recv() call. In general,  this is not a good assumption since a stream transport  service is used.  \*/  printf("recv()\n");  length = recv(socket\_descriptor, Buffer, sizeof Buffer, 0);  if (length == SOCKET\_ERROR)  errexit("Receive failed: %d\n", WSAGetLastError());  printf("Bytes received: %d\n", length);  printf("Message: %s\n", Buffer);  printf("\n");  pause();  /\*  [8] The client closes its socket using closesocket();  \*/  printf("closesocket()\n");  retcode = closesocket(socket\_descriptor);  if (retcode == SOCKET\_ERROR)  errexit("Close socket failed: %d\n", WSAGetLastError());  printf("Return Code: %d\n", retcode);  printf("\n");  pause();  /\*  [9] WSACleanup() is used to terminate use of socket services.  \*/  printf("WSACleanup()\n");  retcode = WSACleanup();  if (retcode == SOCKET\_ERROR)  errexit("Cleanup failed: %d\n", WSAGetLastError());  printf("Return Code: %d\n", retcode);  printf("\n");  pause();  return 0;  }  void errexit(const char\* format, ...)  {  va\_list args;  va\_start(args, format);  vfprintf(stderr, format, args);  va\_end(args);  WSACleanup();  pause();  exit(1);  }  void pause(void)  {  char c;  printf("Press Enter to continue\n");  scanf\_s("%c", &c);  } |

3. .Build and run the code