Politechnika Częstochowska
Wydziału Inżynierii Mechanicznej i Informatyki
Informatyka

UDP Chat

Dariusz Synowiec
Dariusz Wilk
Pawel Orlowski

Częstochowa, 2010

Spis treści

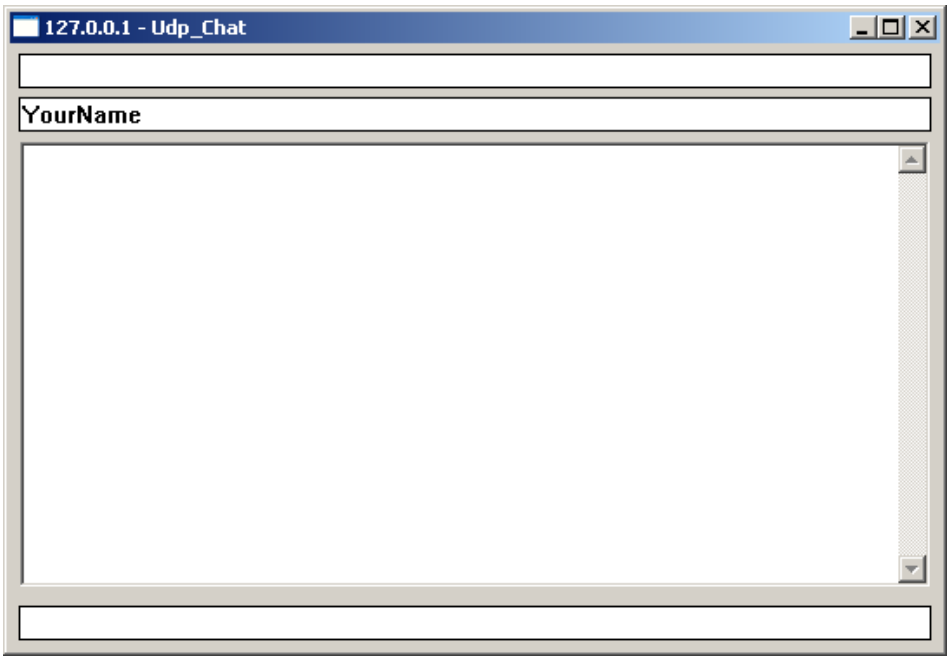
Część I

Dokumentacja

1. Wstęp

Program, zgodnie z nazwą, jest prostą aplikacją realizującą podstawowe założenia chatu. Po uruchomieniu programu, wyświetla się okno z czterema polami tekstowymi (patrz Rys. 1.1). W Tytule programu możemy zobaczyć do jakiego hosta będziemy wysyłać wiadomości. Pierwsze pole tekstowe służy do zmiany hosta docelowego (wystarczy wpisać żądany adres i nacisnąć Enter). Druga kontrolka stanowi listę wiadomości wysłanych i odebranych w programie. W trzecim polu użytkownik powinien wpisać nazwę którą się identyfikuje. Aby wysłać komunikat należy umieścić kursor w ostatnim polu, wpisać go i nacisnąć enter.

Program wysyła wiadomości poprzez UDP w funkcji obsługującej komunikaty Windows. Odbieranie komunikatów realizowane jest w osobnym wątku. Program nasłuchuje na porcie 13 z wszystkich adresów wszelkich komunikatów i wyświetla je w polu drugim.



Rysunek 1.1. Główne okno programu.

2. Wstęp teoretyczny

2.1. Protokół UDP

UDP (ang. User Datagram Protocol) jest jednym z elementów TCP/IP. Przy pomocy UDP, programy komputerowe mogą wysyłać komunikaty (zwane datagramami), do innych hostów. UDP jest protokołem pakietowym, czyli nie wymaga zestawiania specjalnego kanału komunikacyjnego.

UDP używa prostej transmisji bez tzw hand-shake. UDP jest zawodny, datagramy mogą dochodzić w złej kolejności, mogą być duplikowane lub utracone bez żadnej informacji. Protokół zakłada, że korekcja albo nie jest potrzebna albo robiona jest w aplikacji, unikając dodatkowego nakładu na obsługę interfejsu sieciowego. Aplikacje czasu rzeczywistego często używają UDP, ponieważ dla nich lepiej jest opuścić jeden pakiet niż czekać na nadejście spóźnionego. Gdy korekcja błędów jest niezbędna na poziomie interfejsu sieciowego, aplikacje powinny używać TCP lub SCTP, które są właśnie do tego celu zaprojektowane.

UDP jest też użyteczny dla serwerów, które muszą odpowiadać małą ilością danych do ogromnej ilości klientów. Inaczej niż w TCP, UDP umożliwia transmisją typu broadcast (do wszystkich w sieci) i multicast (do wielu hostów).

UDP jest używane przez DNS (Domain Name System), audio/video stream, VoIP (Voice over IP) oraz w wielu grach.

2.2. Porty

Aplikacje wykorzystujące UDP używają socketów do zestawiania komunikacji pomiędzy hostami. Sockety dowiązują aplikację z portami serwisowymi, które służą jako ostatnie punkty transmisji danych. Port to struktura języka C identyfikowana 16-sto bitowym numerem (czyli dozwolone wartości pochodzą z zakresu 0..65535) Port 0 jest zarezerwowany ale jest dozwolone jego użycia przez aplikację w przypadku gdy nie spodziewa się ona odpowiedzi.

Porty 1 do 1023 są portami dedykowanymi. Na systemach Unixowych, łączenie z tymi portami wymaga praw roota.

Porty 1024 do 49151 to porty zarejestrowane.

Porty 49152 do 65535 używane są przez aplikacje klienckie do tymczasowych połączeń z serwerem.

2.3. Protokół IP

Protokół Internetowy IP (ang. Internet Protocol) jest używany do komunikacji poprzez sieć pakietową.

IP to głównie protokół warstwy Internet Pakietu IP (Internet Protocol Suite) i ma za zadanie dostarczać unikalne datagramy (pakiety) z IP hosta-źródła do komputera docelowego tylko bazując na ich adresach. Do tego celu IP definiuje metody adresowania oraz struktury pakowania datagramów (pakietów). Pierwsza główna wersja struktury adresowania, nazywana teraz IPv4, wciąż dominuje w internecie, pomimo że jego następca IPv6 jest aktywnie wdrażany na całym świecie.

Dane z warstw wyższych protokołu są pakowane w pakiety/datagramy. Zestawianie fizycznego połączenia nie jest potrzebne tuż przed wysłaniem pakietów przez jednego hosta do drugiego. IP jest, w przeciwieństwie do tradycyjnej telefonii gdzie połączenie musi być zestawiane przed każdą rozmową, protokołem pakietowym (przez co bezpołączeniowym).

Dzięki abstrakcji jaką oferuje protokół, IP może być używany w sieciach heterogenicznych tzn sieć może się składać z Ethernetu, ATM, FDDI, Wi-Fi i innych. Każda warstwa połączenia może mieć swoją własną metodę adresowania (lub też nie mieć jej wcale) oraz analogicznie potrzebę znalezienia adresu docelowego do zrealizowania połączenia. Znajdowanie adresu jest realizowane poprzez Protokół Znajdowania Adresu (ang. Adress Resolution Protocol, ARP) dla IPv4 oraz Protokół Odkrywania Sąsiadów (ang. Neighbor Discovery Protocol, NDP) dla IPv6.

3. Kod źródłowy

Cały program oparty jest na podstawowych funkcjach dostępnych w WinApi oraz bibliotece Winsock. Cały program znajduje się w pliku `main.cpp`, w którym znajdują się funkcje:

WinMain jest główną funkcją programu przyjmującą standardowe parametry jak dla każdego programu okienkowego opartego o WinApi.

TextWindowProcedure to funkcja przechwytująca naciśnięcia przycisków. Jest ona użyta do obsługi pól tekstowych z adresem hosta i polem wpisywania wiadomości. Po wykryciu że w polu tekstowym naciśnięty został jakiś przycisk, wywoływana jest funkcja MainWindowProcedure, w pozostałych przypadkach domyślna procedura obsługi okna.

MainWindowProcedure zawiera całą funkcjonalność obsługi komunikatów. W niej interpretowane są przyciśnięcia klawisza `Enter` oraz wszystkich pozostałych obsługiwanych komunikatów.

ResizeComponents dba o to by wszystkie komponenty były zawsze prawidłowo rozmieszczone w oknie nadają komponentom rozmiary proporcjonalne do nowej wielkości okna.

WinSockInit inicjuje biblioteką winsock. Szerzej funkcja opisana jest w sekcji 3.3.

UDPListenerThreadFunction| to główna funkcja wątku odpowiadającego za nasłuchiwanie. Patrz sekcja 3.3.2.

addressOf prosta funkcja pobierająca adres IP hosta. Pełny opis funkcji w sekcji 3.3.1.

3.1. Zmienne globalne

W programie występuje kilka zmiennych globalnych takich jak bufory, numery handle okien, nazwa programu. Spośród nich dwie zmienne globalne dotyczą tematu UDP (patrz Listing 3.1):

sa reprezentuje adres IP do którego będziemy wysyłać komunikaty. Zmienna jest typu `sockaddr_in` i przechowuje:
- — rodzinę adresowania (w naszym przypadku jest to AF_INET),
- — numer portu (13)
- — adres IP (w postaci unii typu `in_addr`).

soc przechowuje numer o socketa, którego używamy do wysyłania komunikatów. Jest typu `SOCKET`, co sprowadza się do `unsigned int`.

Obie zmienne są globalne ponieważ są używane w funkcjach `WinMain` oraz `MainWindowProcedure`. W pierwszej funkcji obie są inicjowane. W drugiej obie są używane podczas wywołania funkcji `sendto`. Dodatkowo zmienna `sa` jest aktualizowana podczas zmiany hosta.

Listing 3.1. globalVariables

```
1 struct sockaddr_in sa;
2 SOCKET soc;
```

3.2. Funkcje

Spośród wielu instrukcji zawartych w funkcjach, opisane zostaną tylko te dotyczące komunikacji z użyciem protokołu UDP. Na początku każdej sekcji znajduje się listing opisywanej części programu.

3.2.1. WinMain

Listing 3.2. WinMain

```
1 soc = socket(PF_INET, SOCK_DGRAM, 0);
2
3 char  host[200];
4 sprintf(host, "127.0.0.1");           //loopback
5
6 sa.sin_port = htons(13);
7 sa.sin_family = AF_INET;
8 sa.sin_addr.s_addr = addressOf(host);
```

Linia 1 inicjuje socket w systemie. Używamy rodziny protokołu PF_INET. Jako typ transmisji wybieramy SOCK_DGRAM. Trzeci parametr pozwala sprecyzować, który z wszystkich dostępnych protokołów danego typu w danej rodzinie chcemy wybrać, podając 0 żądamy domyślnego protokołu.

W liniach 3-8 inicjujemy strukturę adresu docelowego. Do zmiennej `host` wpisujemy adres "127.0.0.1co oznacza interfejs zwrotny. Wszystkie wysłane pakiety odbierzemy z powrotem poprzez wątek nasłuchujący. Wybieramy port 13, rodzinę adresowania AF_INET. Następnie używamy funkcji `addressOf` by pobrać wartość numeryczną adresu IP.

3.2.2. MainWindowProcedure

Listing 3.3. MainWindowProcedure

```
1  DWORD e;
2  /* send the message */
3  e = sendto(soc, BuforT, nameLen + dlugoscT + 1, 0, (struct ↩
       sockaddr *) & sa, sizeof (sa));
4
5  if (e == SOCKET_ERROR) {
6      MessageBox(hMainWindow, "Sendto Failed", "Error",MB_OK | ↩
           MB_ICONERROR );
7  }
8
9  ...
10
```

```
11 temp = addressOf(BuforT);
12 if (INADDR_NONE != temp)
13 {
14    /* change the destination adress */
15    sa.sin_addr.s_addr = temp;
16    /* update the window title */
17    strcat(BuforT, " - ");
18    strcat(BuforT, WINDOW_NAME);
19    SetWindowText(hMainWindow, BuforT);
20 }
```

Pierwsza część listingu 3.3 pokazuje jak realizowane jest wysyłanie komunikatów to hosta. Po pobraniu tekstu z pola tekstowego wywoływana jest funkcja `sendto` wysyłająca komunikat (BuforT o długości nameLen + dlugoscT + 1), do hosta (sa), przy użyciu socketu (soc). Listing 3.4 pokazuje deklarację funkcji sendo. I tak:
SOCKET s oznacza numer gniazdka, który ma zostać użyty,
const char *buf to wskaźnik na bufor z danymi do wysłania,
int len mówi o ilości bajtów, które mają być z wysłane,
int flags przez ten parametr można przekazać flagi modyfikujące działanie funkcji,
const struct sockaddr *to jest wskaźnikiem na strukturę zawierającą adres, port, oraz typ adresowania,
int tolen jest długością (w bajtach) struktury wskazywanej prze `to`.

Listing 3.4. Deklaracja funkcji sendto

```
1 int sendto(
2   __in  SOCKET s,
3   __in  const char *buf,
4   __in  int len,
5   __in  int flags,
6   __in  const struct sockaddr *to,
7   __in  int tolen
8 );
```

Druga część listingu 3.3 zawiera część kodu zmieniającą adres wysyłkowy. Ten kawałek programu jest wywoływany po naciśnięciu klawisz enter w polu nazwy hosta. W linii 11 znajduje się wywołanie funkcji `addressOf` zwracającej adres IP hosta docelowego w postaci liczbowej. Jeżeli adres został poprawnie znaleziony (linia 12) następuje zmiana adresu (linia 15) oraz zmiana tytułu okna (linie 7-9).

3.3. Funkcja WinSockInit

Listing 3.5. WinSockInit

```
1 int WinSockInit() {
2    int retVal = 0;
3    WORD version = MAKEWORD(1, 1);
```

```
4     WSADATA wsaData;
5
6     retVal = WSAStartup(version, &wsaData);
7
8     if (0 != retVal) {
9        MessageBox(hMainWindow, "Init Failed", "Error",MB_OK | ↩
             MB_ICONERROR );
10    }
11
12    return retVal;
13 }
```

Funkcja ta ma za zadanie „podnieść" bibliotekę winsock (patrz Listing 3.5). Najpierw wybieramy wersję 1.1 biblioteki (linia 3). Następnie wywołujemy funkcję `WSAStartup` (linia 6). Gdy nie udało się podnieść biblioteki, wyświetlamy stosowny komunikat (linie 8-10).

3.3.1. addressOf

Listing 3.6. addressOf

```
1  u_long addressOf(const char * addrStr)
2  {
3     u_long retVal;
4     struct hostent* phe;
5
6     char* p1;
7     char* p2;
8
9     retVal = inet_addr(addrStr);
10    if (INADDR_NONE == retVal)
11    {
12       phe = gethostbyname(addrStr);
13
14       if (NULL == phe)
15       {
16          MessageBox(hMainWindow, "Nie znalalem hosta", "Error",↩
                MB_OK | MB_ICONERROR );
17       }
18       else
19       {
20
21          p1 = (char *) & retVal;
22          p2 = &phe->h_addr[0];
23
24          for (int i=0; i<sizeof(retVal); i++)
25          {
26             p1[i]=p2[i];
27          }
28       }
29    }
30
31    return retVal;
32 }
```

Funkcja przyjmuje jako parametr wskaźnik na ciąg znaków zakończony '0', który stara się zamienić na numeryczną wartość z adresem IP. W linii 9 zakłada się, że w buforze wejściowym znajduje się adres IP w gotowej postaci (np „127.0.0.1"). Jeżeli nie (tzn funkcja `inet_addr` zwróci `INADDR_NONE`) wywoływana jest funkcja `gethostbyname`. Gdy host zostanie poprawnie znaleziony, adres IP jest kopiowany ze struktury hostent do zmiennej retVal i zwracany.

3.3.2. UDPListenerThreadFunction

Listing 3.7. UDP server

```
1  struct sockaddr_in A;
2  int s, d;
3
4  A.sin_family = AF_INET;
5  A.sin_port = htons(13);
6  A.sin_addr.s_addr = INADDR_ANY;
7
8  s = socket(AF_INET, SOCK_DGRAM, 0);
9
10 d = bind(s, (struct sockaddr *) & A, sizeof (A));
11
12 ...
13
14 while(1)
15 {
16   memset(inBuf,0,100);
17   d = recvfrom(s, &inBuf[2], 100, 0, (sockaddr *) & A, &dw);
18
19   ...
20
21 }
```

Pokazana część programu (listing 3.7 w uproszczeniu pokazuje zasadę działania tej funkcji. Linie 1-6 inicjują strukturę `A` rodziną adresową AF_INET, portem 13. Adresem w tym przypadku jest INADDR_ANY (0) co oznacza, że będą przyjmowany komunikaty ze wszystkich adresów IP (w tym loopback).

Utworzenie nowego socketu odbiorczego (o takich samych parametrach jak socket odbiorczy zadeklarowany wcześniej) znajduje się w linii 8.

W linii 10 jest wywołanie funkcji `bind` wiążącej socket `s` z adresem `A`. Po wywołaniu tej funkcji, nasz program nasłuchuje na sockecie `s` wiadomości przychodzących z adresu `A`.

Linia 14 i dalsze pokazują (obrazowo) jak realizowane jest nasłuchiwanie na porcie. Pętla nieskończona najpierw czyści bufor odbiorczy, następnie wywołuję funkcję `recvfrom`. Wątek tak długo „wisi" w tej funkcji aż nie otrzyma porcji danych. Z listingu 3.8 widać, że parametry funkcji są praktycznie tego samego typu co w `sentdo` z tą różnicą, że parametry `buf` oraz `from` są wyjściowe.

Listing 3.8. Deklaracja funkcji recvfrom

```
int recvfrom(
  __in          SOCKET s,
  __out         char *buf,
  __in          int len,
  __in          int flags,
  __out         struct sockaddr *from,
  __inout_opt   int *fromlen
);
```

Część II

Dodatki

A. Pełny listing programu

```
1  /*
2   * =====================================================================
3   *
4   *       Filename:  main.cpp
5   *
6   *    Description:  UDP Chat based on win api and winsck library and threads.
7   *
8   *        Version:  1.0
9   *        Created:  2009-11-24 23:02:40
10  *       Revision:  none
11  *       Compiler:  gcc
12  *
13  *         Author:  Dariusz Synowiec
14  *
15  * =====================================================================
16  */
17 // TODO: Strona tytulowa (UDP chat)
18 //    Dariusz Synowiec
19 //    Dariusz Wilk
20 //    Pawel Orlowski
21 // uzyty protokol (teoria) (udp, )
22 // uzyte rozkazy z winsocka
23 // uzyte elementy (edytor kompilator itd).
24 // dokumentacja z kodu
25
26 /* #####   HEADER FILE INCLUDES   ########################## */
27
28 #include <windows.h>
29 #include <windowsx.h>
30
31 #include <stdio.h>
32 #include <time.h>
33
34
35 /* #####   MACROS  -  LOCAL TO THIS SOURCE FILE   ######## */
36
37
38 #define ON 1
39 #define OFF 0
40
41 #define ECHO ON
42 #define MAX_BUF 20000
43
44 #define WINDOW_NAME "Udp_Chat"
45
46
```

```
47 /* #####   PROTOTYPES  -  LOCAL TO THIS SOURCE FILE   ### */
48
49 LRESULT CALLBACK MainWindowProcedure (HWND, UINT, WPARAM, ↩
       LPARAM);
50 LRESULT CALLBACK TextWindowProcedure (HWND hwnd, UINT mesg, ↩
       WPARAM wParam, LPARAM lParam);
51 void ResizeComponents(HWND hwnd);
52 int WinSockInit(void);
53 DWORD WINAPI UDPListenerThreadFunction(LPVOID lpParam);
54 u_long addressOf(const char * addrStr);
55
56
57 /* #####   VARIABLES  -  LOCAL TO THIS SOURCE FILE   #### */
58
59 char szClassName[ ] = "UDPChat";
60 HWND hText;
61 HWND hMessages;
62 HWND hMainWindow;     /* This is the handle for our window */
63 HWND hName;
64 HWND hHostName;
65 WNDPROC g_OldWndProc;
66
67 //char BuforT[MAX_BUF];
68 char* BuforT;
69 char* inBuf;
70 char* name;
71 struct sockaddr_in sa;
72 SOCKET soc;
73 unsigned int nameLen;
74
75
76 /* #####   FUNCTION DEFINITIONS  -  LOCAL TO THIS SOURCE FILE ↩
       ######################## */
77
78
79 /*
80  * ===  FUNCTION  ======================================================
81  *         Name:  WinMain
82  *  Description:  Main function
83  * =====================================================================
84  */
85 int WINAPI WinMain (HINSTANCE hThisInstance, HINSTANCE ↩
       hPrevInstance, LPSTR lpszArgument, int nFunsterStil)
86 {
87     char  host[200];
88
89     MSG messages;            /* Here messages to the application↩
           are saved */
90     WNDCLASSEX wincl;        /* Data structure for the ↩
           windowclass */
91
92     /* Bring up winsock */
93     (void) WinSockInit();
94
95     /* Open socket */
96     soc = socket(PF_INET, SOCK_DGRAM, 0);
97
```

```
98
99      sprintf(host, "127.0.0.1");           //loopback
100     //    sprintf(host, "149.223.36.86");    //krzysiek
101     //    sprintf(host, "R01184");           //krzysiek
102     //    sprintf(host, "140.171.179.171"); //cytrix
103     //    sprintf(host, "R01772");           //ja
104
105     /* create default sending address */
106     sa.sin_port = htons(13);
107     sa.sin_family = AF_INET;
108     sa.sin_addr.s_addr = addressOf(host);
109
110     /* Allocate and clear buffers */
111     BuforT = (char*) calloc(MAX_BUF, sizeof(char));
112     inBuf  = (char*) calloc(MAX_BUF, sizeof(char));
113     name   = (char*) calloc(MAX_BUF, sizeof(char));
114
115
116     /*-------------------------------------------------------
117      *  Auto created by DevCpp BEGIN
118      *-------------------------------------------------------*/
119     /* The Window structure */
120     wincl.hInstance     = hThisInstance;
121     wincl.lpszClassName = szClassName;
122     wincl.lpfnWndProc   = MainWindowProcedure;      /* This ↩
           function is called by windows */
123     wincl.style         = CS_DBLCLKS;               /* Catch ↩
           double-clicks */
124     wincl.cbSize        = sizeof (WNDCLASSEX);
125
126     /* Use default icon and mouse-pointer */
127     wincl.hIcon         = LoadIcon (NULL, IDI_APPLICATION);
128     wincl.hIconSm       = LoadIcon (NULL, IDI_APPLICATION);
129     wincl.hCursor       = LoadCursor (NULL, IDC_ARROW);
130     wincl.lpszMenuName = NULL;                      /* No menu */
131     wincl.cbClsExtra    = 0;                        /* No extra ↩
           bytes after the window class */
132     wincl.cbWndExtra    = 0;                        /* structure or↩
            the window instance */
133     /* Use Windows's default color as the background of the ↩
           window */
134     wincl.hbrBackground = (HBRUSH) COLOR_BACKGROUND;
135
136     /* Register the window class, and if it fails quit the ↩
           program */
137     if (!RegisterClassEx (&wincl))
138     {
139        return 0;
140     }
141
142     /* The class is registered, let's create the program*/
143     hMainWindow = CreateWindowEx (
144           0,                   /* Extended possibilites for ↩
                variation */
145           szClassName,         /* Classname */
146           WINDOW_NAME,          /* Title Text */
147           WS_OVERLAPPEDWINDOW, /* default window */
```

```
148         CW_USEDEFAULT,       /* Windows decides the position */
149         CW_USEDEFAULT,       /* where the window ends up on the screen */
150         544,                 /* The programs width */
151         375,                 /* and height in pixels */
152         HWND_DESKTOP,        /* The window is a child-window to desktop */
153         NULL,                /* No menu */
154         hThisInstance,       /* Program Instance handler */
155         NULL                 /* No Window Creation data */
156         );
157     /*-------------------------------------------------------------
158      *  Auto created by DevCpp END
159      *-------------------------------------------------------------*/
160
161     /* Adjust window name to: "%host% - %WINDOW_NAME%" */
162     strcpy(BuforT, host);
163     strcat(BuforT, " - ");
164     strcat(BuforT, WINDOW_NAME);
165     SetWindowText(hMainWindow, BuforT);
166
167     /* Create host name text box */
168     hHostName = CreateWindowEx (0, "EDIT", NULL, WS_CHILD | WS_VISIBLE | WS_BORDER, 5, 5, 150, 20,
169         hMainWindow, NULL, hThisInstance, NULL);
170
171     /* Create user name text box */
172     hName = CreateWindowEx (0, "EDIT", NULL, WS_CHILD | WS_VISIBLE | WS_BORDER, 5, 25, 150, 20,
173         hMainWindow, NULL, hThisInstance, NULL);
174
175     /* Create messages list text box */
176     hMessages = CreateWindowEx (WS_EX_CLIENTEDGE, "EDIT", NULL, WS_CHILD | WS_VISIBLE | WS_BORDER |
177         WS_VSCROLL | ES_MULTILINE | ES_AUTOVSCROLL, 5, 45, 150, 130, hMainWindow, NULL, hThisInstance, NULL);
178
179     /* Create new message text box */
180     hText = CreateWindowEx (0, "EDIT", NULL, WS_CHILD | WS_VISIBLE | WS_BORDER, 5, 160, 150, 20,
181         hMainWindow, NULL, hThisInstance, NULL);
182
183     /* Adjust User name string to be "YourName" */
184     sprintf(name, "YourName");
185     SetWindowText (hName, name);
186     nameLen = strlen(name);
187
188     /*
189      * Override normal TEXT window class Procedure so we can easily catch enter key.
190      * Note that it wll override all windows of class TEXT (so hHostName as well)
191      */
192     g_OldWndProc = (WNDPROC) SetWindowLong (hText, GWL_WNDPROC, (LONG)TextWindowProcedure);
```

```
193     SetWindowLong (hHostName, GWL_WNDPROC, (LONG)↩
            TextWindowProcedure);
194 
195     /*
196      * Register Win+C lobal hotkey, it will bring our window on ↩
            top and set the focus on new message
197      * text box
198      */
199     RegisterHotKey(hMainWindow, 1, MOD_WIN, 0x43);
200 
201     /* Adjust components sizes */
202     ResizeComponents(hMainWindow);
203 
204     /* Make the window visible on the screen */
205     ShowWindow (hMainWindow, nFunsterStil);
206 
207     /* Create listener thread.  */
208     HANDLE hThread;
209     DWORD dwThreadId;
210     hThread = CreateThread(
211           NULL,                        // default security ↩
                attributes
212           0,                           // use default stack size
213           UDPListenerThreadFunction,   // thread function name
214           NULL,                        // argument to thread ↩
                function
215           0,                           // use default creation ↩
                flags
216           &dwThreadId);                // returns the thread ↩
                identifier
217 
218 
219     SetFocus(hText);
220 
221     /* Run the message loop. It will run until GetMessage() ↩
            returns 0 */
222     while (GetMessage (&messages, NULL, 0, 0))
223     {
224         /* Translate virtual-key messages into character messages↩
                */
225         TranslateMessage(&messages);
226         /* Send message to MainWindowProcedure */
227         DispatchMessage(&messages);
228     }
229 
230     /* The program return-value is 0 - The value that ↩
            PostQuitMessage() gave */
231     return messages.wParam;
232 }
233 
234 
235 /*
236  * ===  FUNCTION  ======================================================
237  *         Name:  TextWindowProcedure
238  *  Description:  Function is responsible for intercepting ↩
            enter key in text fields.
239  * ======================================================================
```

```
240  */
241  LRESULT CALLBACK TextWindowProcedure (HWND hwnd, UINT mesg, ↩
         WPARAM wParam, LPARAM lParam)
242  {
243     switch (mesg)
244     {
245        case WM_KEYDOWN:
246           {
247              /* Let MainWindowProcedure handle the keys */
248              CallWindowProc (MainWindowProcedure, hwnd, mesg, ↩
                    wParam, lParam);
249           }
250           break;
251     }
252
253     return CallWindowProc (g_OldWndProc, hwnd, mesg, wParam, ↩
           lParam);
254  }
255
256
257  /*
258   * ===  FUNCTION  ======================================================
259   *         Name:  MainWindowProcedure
260   *  Description:  This function is called by the Windows ↩
          function DispatchMessage()
261   * =====================================================================
262   */
263  LRESULT CALLBACK MainWindowProcedure (HWND hwnd, UINT message, ↩
         WPARAM wParam, LPARAM lParam)
264  {
265     DWORD dlugoscM = 0;
266     DWORD dlugoscT = 0;
267
268     switch (message)                  /* handle the messages */
269     {
270        case WM_DESTROY:
271           PostQuitMessage (0);       /* send a WM_QUIT to the ↩
                 message queue */
272           break;
273        case WM_KEYDOWN:
274           /* When a key was pressed */
275           {
276              switch ( (int) wParam )
277              {
278                 case VK_RETURN: /* if it was <RETURN> */
279                    if (hwnd == hText) /* if focus was set to new↩
                          message window */
280                    {
281                       nameLen = GetWindowTextLength (hName);
282                       GetWindowText (hName, name, nameLen + 1);
283
284                       dlugoscM = GetWindowTextLength(hMessages);
285                       dlugoscT = GetWindowTextLength(hText);
286
287                       if (dlugoscT > MAX_BUF)
288                       {
```

```
289                         MessageBox(hMainWindow, "To long ↩
                                Message", "Error",MB_OK | ↩
                                MB_ICONERROR );
290                         SetFocus(hText);
291                         break;
292                     }
293
294                     /* Paste username to the buffer */
295                     strncpy(BuforT, name, nameLen);
296
297                     /* add message */
298                     GetWindowText (hText, &BuforT[nameLen + ↩
                            1], dlugoscT + 1);
299
300                     /* add ":" and "\0" */
301                     BuforT[nameLen] = ':';
302                     BuforT[nameLen + dlugoscT + 1] = '\0';
303
304                     DWORD e;
305                     /* send the message */
306                     e = sendto(soc, BuforT, nameLen + dlugoscT↩
                            + 1, 0, (struct sockaddr *) & sa, ↩
                            sizeof (sa));
307
308                     if (e == SOCKET_ERROR) {
309                         MessageBox(hMainWindow, "Sendto Failed"↩
                                , "Error",MB_OK | MB_ICONERROR );
310                     }
311
312 #if ECHO == ON
313                     if (dlugoscM > 0)
314                     {
315                         BuforT[nameLen - 1] = '\r';
316                         BuforT[nameLen + 0] = '\n';
317                         Edit_SetSel(hMessages, dlugoscM, ↩
                                dlugoscM);
318                         Edit_ReplaceSel(hMessages, &BuforT[↩
                                nameLen - 1]);
319                     }
320                     else
321                     {
322                         SetWindowText (hMessages, &BuforT[↩
                                nameLen + 1]);
323                     }
324 #endif
325
326                     SetWindowText(hText, "");
327
328                     SendMessage(hMessages, WM_VSCROLL, ↩
                            SB_BOTTOM, 0);
329                 }
330                 else if (hwnd == hHostName) /* if focus was ↩
                        set to host address window */
331                 {
332                     dlugoscT = GetWindowTextLength(hHostName);
333
334                     if (0 < dlugoscT)
```

```
335                         {
336                             u_long temp;
337                             GetWindowText(hHostName, BuforT, ↩
                                    dlugoscT + 1);
338 
339                             temp = addressOf(BuforT);
340                             if (INADDR_NONE != temp)
341                             {
342                                 /* change the destination adress */
343                                 sa.sin_addr.s_addr = temp;
344                                 /* update the window title */
345                                 strcat(BuforT, " - ");
346                                 strcat(BuforT, WINDOW_NAME);
347                                 SetWindowText(hMainWindow, BuforT);
348                             }
349                         }
350                         else
351                         {
352                             MessageBox(hMainWindow, "Wpisz adres ↩
                                    hosta.", "Error",MB_OK | ↩
                                    MB_ICONERROR );
353                         }
354                         SetFocus(hText);
355                     }
356                     break;
357                 case VK_ESCAPE:
358                     /* If <Esc> was pressed - minimize the window↩
                            */
359                     ShowWindow (hMainWindow, SW_MINIMIZE);
360                     break;
361             }
362         }
363         break;
364     case WM_HOTKEY:
365         /*
366          * Since we only have one hotkey registered, we don't ↩
                have to check which hotkey was
367          * pressed. Bring the window to the top.
368          */
369         ShowWindow (hMainWindow, SW_MINIMIZE);
370         ShowWindow (hMainWindow, SW_SHOWNORMAL);
371         SetFocus(hText);
372         break;
373     case WM_SIZE:
374         /* We get this message each time size of the window ↩
               has changed. Update controls sizes. */
375         ResizeComponents(hMainWindow);
376         break;
377     default:                        /* for messages that we don↩
            't deal with */
378         return DefWindowProc (hwnd, message, wParam, lParam);
379   }
380 
381   return 0;
382 }
383 
384 /*
```

```
385  * ===  FUNCTION  ======================================================
386  *         Name:  ResizeComponents
387  *  Description:  Function resizes all components to their proper size using main window dimensions.
388  * =====================================================================
389  */
390 void ResizeComponents(HWND hwnd)
391 {
392     RECT rect;
393     GetClientRect (hwnd, &rect);
394 
395     SetWindowPos(hHostName , HWND_TOP , 5 , rect.top + 5      , rect.right - 10 , 20                , 0);
396     SetWindowPos(hName     , HWND_TOP , 5 , rect.top + 30     , rect.right - 10 , 20                , 0);
397     SetWindowPos(hMessages , HWND_TOP , 5 , rect.top + 55     , rect.right - 10 , rect.bottom - 90 , 0);
398     SetWindowPos(hText     , HWND_TOP , 5 , rect.bottom - 25 , rect.right - 10 , 20                , 0);
399 }
400 
401 /*
402  * ===  FUNCTION  ======================================================
403  *         Name:  WinSockInit
404  *  Description:  Loads winsock library.
405  * =====================================================================
406  */
407 int WinSockInit() {
408     int retVal = 0;
409     WORD version = MAKEWORD(1, 1);
410     WSADATA wsaData;
411 
412     retVal = WSAStartup(version, &wsaData);
413 
414     if (0 != retVal) {
415         MessageBox(hMainWindow, "Init Failed", "Error",MB_OK | MB_ICONERROR );
416     }
417 
418     return retVal;
419 }
420 
421 /*
422  * ===  FUNCTION  ======================================================
423  *         Name:  UDPListenerThreadFunction
424  *  Description:  THread listens on port 13 and accepts all udp transmissions. It displays every
425  *                message in hMessages window.
426  * =====================================================================
427  */
428 DWORD WINAPI UDPListenerThreadFunction(LPVOID lpParam) {
429     DWORD dlugoscM = 0;
430     DWORD dlugoscT = 0;
431     struct sockaddr_in A;
432     int s, d;
433 
434     s = socket(AF_INET, SOCK_DGRAM, 0);
```

```
435
436     A.sin_family = AF_INET;
437     A.sin_port = htons(13);
438     A.sin_addr.s_addr = INADDR_ANY;
439
440     d = bind(s, (struct sockaddr *) & A, sizeof (A));
441
442     if (d >= 0) {
443        int dw = sizeof (A);
444        while(1)
445        {
446           memset(inBuf,0,100);
447           d = recvfrom(s, &inBuf[2], 100, 0, (sockaddr *) & A, &↩
                dw);
448
449 #if ECHO == ON
450           if (127 != A.sin_addr.s_net)
451 #endif
452           {
453              dlugoscM = GetWindowTextLength(hMessages);
454
455              if (dlugoscM > 0)
456              {
457                 inBuf[0] = '\r';
458                 inBuf[1] = '\n';
459                 Edit_SetSel(hMessages, dlugoscM, dlugoscM);
460                 Edit_ReplaceSel(hMessages, inBuf);
461              }
462              else
463              {
464                 SetWindowText (hMessages, &inBuf[2]);
465              }
466
467              SendMessage(hMessages, WM_VSCROLL, SB_BOTTOM, 0);
468           }
469        }
470     }
471 }
472
473
474 /*
475  * ===  FUNCTION  ======================================================
476  *         Name:  addressOf
477  *  Description:  Tries to translate a char array containing ↩
     host address to a u_long address.
478  *         Note:  User still have to perform htonl.
479  * =====================================================================
480  */
481 u_long addressOf(const char * addrStr)
482 {
483     u_long retVal;
484     struct hostent* phe;
485
486     char* p1;
487     char* p2;
488
489     retVal = inet_addr(addrStr);
```

```
490     if (INADDR_NONE == retVal)
491     {
492         phe = gethostbyname(addrStr);
493 
494         if (NULL == phe)
495         {
496             MessageBox(hMainWindow, "Nie znalalem hosta", "Error",↩
                    MB_OK | MB_ICONERROR );
497         }
498         else
499         {
500 
501             p1 = (char *) & retVal;
502             p2 = &phe->h_addr[0];
503 
504             for (int i=0; i<sizeof(retVal); i++)
505             {
506                 p1[i]=p2[i];
507             }
508         }
509     }
510 
511     return retVal;
512 }
```