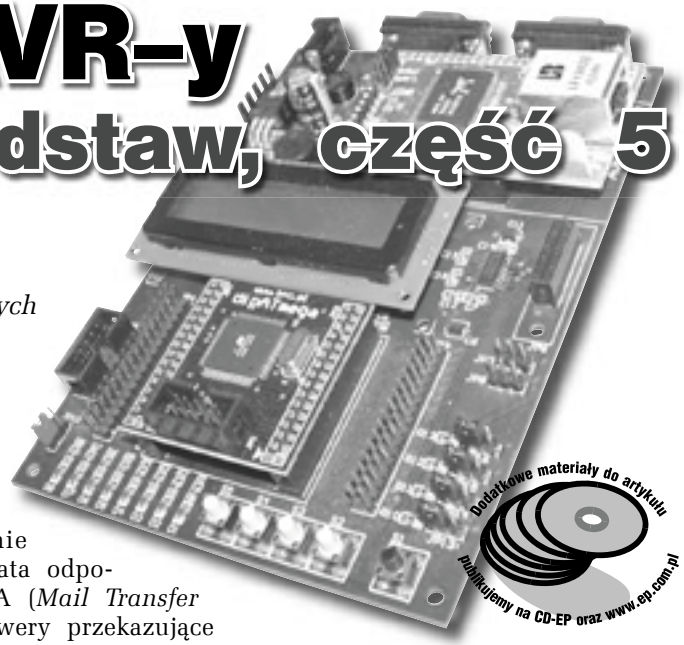
# Ethernet i AVR–y
# Ethernet od podstaw, część 5

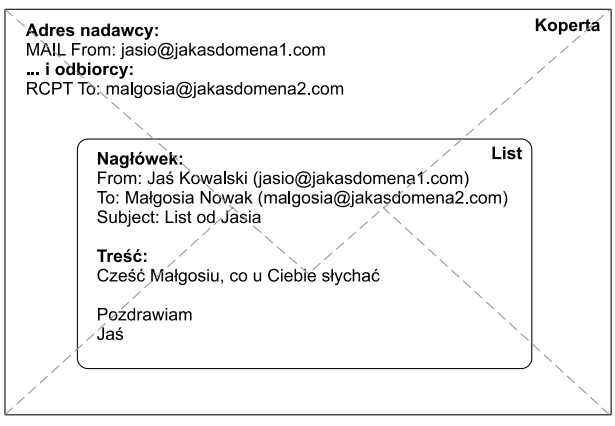



*W poprzednich odcinkach kursu przedstawiłem kilka serwerów protokołu TCP, czyli programów odbierających przychodzące połączenia. Teraz pora na przykład klienta – aplikacji nawiązującej połączenie z określonym serwerem.*

Budowane przez nas układy często muszą pracować niezawodnie przez bardzo długi czas w odległych lub trudno dostępnych miejscach. Przydatna jest wówczas możliwość składania przez urządzenie okresowego raportu zawierającego np. wyniki pomiarów, albo informującego o wystąpieniu awarii. Jeśli ma ono dostęp do Internetu, taki raport można w wygodny sposób dostarczyć odpowiedniej osobie za pomocą poczty elektronicznej. Jak za chwilę się przekonamy, wysłanie e–maila nie jest trudne i posłuży do zademonstrowania budowy i działania prostego klienta protokołu TCP.

Do uruchomienia przykładowej aplikacji opisanej w artykule potrzebne będzie łącze internetowe oraz działające konto pocztowe – do testów można je założyć za darmo np. w jednym z portali internetowych. Będziemy potrzebować kilku parametrów tego konta:
– adresu serwera wysyłania poczty (SMTP).
– nazwy użytkownika i hasła SMTP (jeżeli serwer, z którego usług korzystamy tego wymaga).

Parametry te należy ustawić w odpowiednich makrodefinicjach na początku kodu programu.




**Rys. 6. Struktura wiadomości e–mail**

## Protokół SMTP

Za dostarczenie e–maila do adresata odpowiadają tzw. MTA (*Mail Transfer Agents*), czyli serwery przekazujące pocztę. Wraz z danymi każdego konta pocztowego dostajemy adres MTA (nazywanego też serwerem SMTP), który przekazuje pocztę wysyłaną z tego konta. Protokołem stosowanym w Internecie do przekazywania e–maili jest SMTP (*Simple Mail Transfer Protocol*). Protokół ten jest oparty o proste komunikaty tekstowe, dlatego e–maila można wysłać nawet bez używania progamu pocztowego (Outlook, Thunderbird, itp.), korzystając z klienta telnetu.

Strukturę wiadomości e–mail przedstawiono na **rys. 1.** Podobnie jak tradycyjny list, e–mail składa się z:
– koperty, zawierającej adres nadawcy (`MAIL From`) i adresata (`RCPT To`),
– nagłówka, zawierającego między innymi temat (`Subject`), datę wysłania (`Date`), adres zwrotny (`Return–path`), nazwę nadawcy (`From`) i odbiorcy (`To`), listę MTA które uczestniczyły w doręczaniu e–maila do adresata (`Received`), informacje o formacie wiadomości (`Content-type`) itd.
– treści wiadomości.

## Klient TCP/IP

Zanim wyślemy e–maila, musimy nawiązać połączenie z serwerem SMTP obsługującym nasze konto pocztowe. Szkielet klienta TCP w systemie Nut/OS pokazany jest na **rys. 7.**

Kod funkcji `send_mail()` łączącej się z serwerem SMTP i wysyłającej e–maila przedstawia **list. 9**. Zaczyna ona pracę od określenia adresu IP serwera SMTP na podstawie jego nazwy przez wywołanie funkcji `NutDnsGetHostByName()`. Następnie tworzy nowe gniazdo protokołu TCP za pomocą znanej z poprzednich części kursu funkcji `NutTcpSocketCreate()`. Kolejnym krokiem jest próba nawiązania połączenia TCP z portem 25 serwera (standardowy port usługi SMTP) – funkcja `NutTcpConnect()`. Jeśli serwer zaakceptuje połączenie, za pomocą `_fdopen()` tworzymy strumień połączony z gniazdem sieciowym, dzięki czemu do odbioru i wysyłania danych będziemy mogli używać standardowych funkcji wejścia–wyjścia. Możemy teraz rozpocząć wysyłanie wiadomości.

## Wysyłanie poczty

Spróbujmy wysłać e–mail przedstawiony na **rys. 6.** Bezpośrednio po nawiązaniu połączenia serwer SMTP powinien poinformować klienta o gotowości:

**Tab. 1. Często spotykane kody odpowiedzi SMTP**

| Kod | Opis |
|---|---|
| 220 | Serwer gotowy |
| 221 | Serwer kończy połączenie |
| 235 | Autentykacja pomyślna |
| 250 | Polecenie wykonane |
| 334 | Kontynuuj autentykację |
| 354 | Rozpocznij wysyłanie wiadomości |
| 500 | Nieznane polecenie |
| 501 | Nieprawidłowy parametr |
| 535 | Autentykacja nieudana |

**List. 9. Kod klienta TCP wysyłającego e–mail (bez obsługi błędów i sprawdzania kodów odpowiedzi z serwera SMTP)**

```
/* funkcja odczytuje ze strumienia f odpowiedz serwera SMTP postaci:
XXX jakis tekst odpowiedzi
i zwraca jej kod (XXX) lub –1, gdy odczyt sie nie udal */

int smtp_get_reply_code(FILE *f)
{
  char buf[128];
  int code;

  int len = fgets(buf, 128, f);
  if(len<=0) return –1;
  sscanf(buf, "%d", &code);
  return code;
}

/* funkcja wysyla e-maila z konta o parametrach z bloku „Konfiguracja konta pocztowego SMTP".
rcpt_addr – e-mail adresata
subject – temat wiadomosci
message – tresc wiadomosci */

int smtp_send_mail(char *rcpt_addr, char *subject, char *message)
{
  char buf[64];
// gniazdo TCP reprezentujace polaczenie z serwerem SMTP
  TCPSOCKET *sock;
// deskryptor pliku polaczony z gniazdem
  FILE *f;
  u_long ip_addr;

// Odpytujemy serwer DNS o adres IP serwera wysylania poczty (SMTP) o nazwie podanej
// w SMTP_SERVER_ADDRESS
  ip_addr = NutDnsGetHostByName(SMTP_SERVER_ADDRESS);

// tworzymy nowe gniazdo protokolu TCP
  sock = NutTcpCreateSocket();

// probujemy polaczyc z portem 25 serwera SMTP
  NutTcpConnect(sock, ip_addr, 25);

// tworzymy deskryptor pliku polaczony z gniazdem sock
  f = _fdopen((int) sock, "r+b");

// serwer powinien na poczatku przedstawic sie nam i wyslac kod 220 oznaczajacy gotowosc
  if(smtp_get_reply_code(f) != SMTP_READY) .....

// przedstawiamy sie serwerowi. Wbrew standardowi, podana nazwa nie musi byc nazwa FQDN hosta,
// ktory laczy sie z serwerem.
  fprintf(f, „HELO ethernut.localdomain\n");
// upewniamy sie, ze dane zostaly wyslane przez wywołanie fflush()
  fflush(f);

// sprawdzamy, czy serwer wykonal polecenie
  if(smtp_get_reply_code(f) != SMTP_REQUEST_COMPLETED)....

#ifndef SMTP_DISABLE_AUTH
// probujemy sie zalogowac
  fprintf(f, „AUTH LOGIN\n");fflush(f);
  smtp_get_reply_code(f);

// wysylamy zakodowana w base64 nazwe uzytkownika
  base64_encode(SMTP_USER_NAME, buf, strlen(SMTP_USER_NAME));
  fprintf(f, „%s\n", buf);fflush(f);
  smtp_get_reply_code(f);

// wysylamy zakodowane w base64 haslo
  base64_encode(SMTP_PASSWORD, buf, strlen(SMTP_PASSWORD));
  fprintf(f, „%s\n", buf);fflush(f);

// odczytujemy odpowiedz serwera, jesli podalismy poprawny login i haslo bedzie miala kod 235 (AUTH_SUCCESSFUL)
  if(smtp_get_reply_code(f) != SMTP_AUTH_SUCCESSFUL) RETURN_ERROR(SM_AUTH_ERROR);
#endif

// wysylamy adres nadawcy
  fprintf(f, „MAIL From:<%s>\n", SENDER_EMAIL); fflush(f);
  smtp_get_reply_code(f);

// wysylamy adres adresata :P
  fprintf(f, „RCPT To:<%s>\n", rcpt_addr); fflush(f);
  smtp_get_reply_code(f);

// wysylamy komende DATA oznaczajaca poczatek danych listu
  fprintf(f, „DATA\n", SENDER_EMAIL); fflush(f);
  smtp_get_reply_code(f);

// wysylamy naglowki listu (temat, nadawce, odbiorce):
  fprintf(f, „From: %s\n", SENDER_EMAIL);
  fprintf(f, „To: %s\n", rcpt_addr);
  fprintf(f, „Subject: %s\n", subject);
// miedzy naglowkami a trescia powinna znajdowac sie 1 pusta linia:
  fprintf(f,"\n");
// wysylamy tresc listu
  fprintf(f,"%s\n", message);
//  ... i sygnalizujemy serwerowi koniec wiadomosci przez wyslanie kropki w nowej linii:
  fprintf(f,".\n");
  fflush(f);

  smtp_get_reply_code(f);

// konczymy sesje
  fprintf(f,"QUIT\n");
  fflush(f);
```

**List. 9. c.d.**

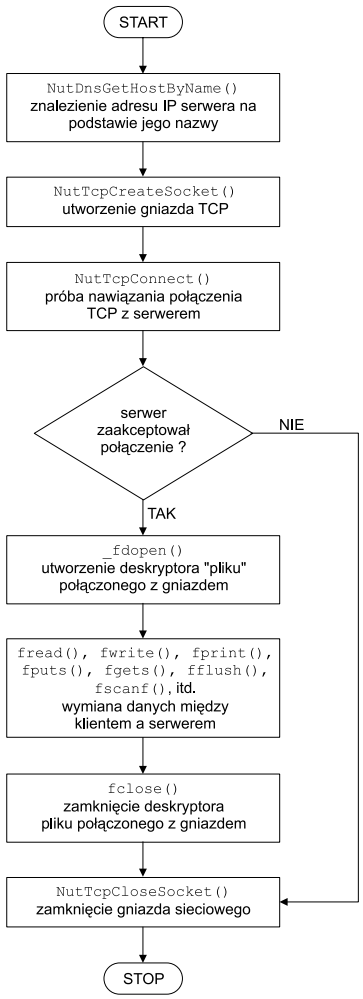
```
  smtp_get_reply_code(f);

// zamykamy gniazdo
  fclose(f);
  NutTcpCloseSocket(sock);
}

int main(void)
{
  initialize();
  init_network();

  smtp_send_mail("jakisemail@serwer.com", "Testowy mail z Ethernuta", "Oto testowy mail wyslany z plytki Ethernut!");

  for(;;);
}
```




**Rys. 7. Struktura prostego klienta TCP w systemie Nut/OS**

**Ważniejsze funkcje spotykane w programie z list. 9**

**int NutTcpConnect(TCPSOCKET *sock, u_long addr, u_short port).**
Funkcja próbuje nawiązać połączenie TCP z portem port serwera o adresie IP addr. Jeżeli połączenie zostanie ustanowione, funkcja zwraca 0, a gniazdo sock reprezentuje nawiązane połączenie. Jeśli próba połączenia nie powiedzie się, NutTcpConnect() zwraca –1, zaś przyczynę niepowodzenia można sprawdzić za pomocą funkcji:

**int NutTcpError(TCPSOCKET *sock).**
Wszystkie kody błędów zwracane przez tę funkcję znajdują się w pliku nagłówkowym net/errno.h, poniżej wymione są najczęściej spotykane:
– ECONNREFUSED: Connection refused – serwer odrzucił połączenie
– EHOSTUNREACH: No route to host – nie można znaleźć serwera
– ENETUNREACH: Network is unreachable – sieć jest niedostępna. Błąd występuje często, gdy mamy niepoprawnie skonfigurowany interfejs sieciowy.

**u_long NutDnsGetHostByName(CONST char *name);**
Funkcja pyta serwer DNS (patrz ramka DNS) o adres IP komputera o nazwie name. Adres IP jest zwracany w postaci liczby 32–bitowej, a jeśli nie udało się go ustalić (np. serwer DNS nie odpowiedział lub host o podanej nazwie nie istnieje), funkcja zwraca wartość 0. Na przykład wywołanie:
NutDnsGetHostByName(„www.ep.com.pl");
zwróci wartość 0x3e81ef92, czyli adres serwera www.ep.com.pl (62.129.239.146) zapisany szesnastkowo. Jeśli nasze urządzenie nie używa DHCP do konfiguracji sieci, przed pierwszym wywołaniem NutDnsGetHostByName() konieczne jest ustawienie adresów serwerów DNS za pomocą funkcji:

**void NutDnsConfig2 ( u_char * hostname, u_char * domain, u_long pdnsip, u_long sdnsip )**
gdzie:
hostname, domain – nazwa i domena DNS naszego urządzenia. Ponieważ system w obecnej wersji nie wykorzystuje tych parametrów, możemy podać tam dowolne wartości.
pdnsip – adres IP pierwotnego serwera DNS
sdnsip – adres IP zapasowego serwera DNS
Przykładowo, jeśli korzystamy z łącza DSL TPSA, DNS–y konfigurujemy w następujący sposób:
NutDnsConfig2(„ethernut","localdomain",
inet_addr(„194.204.159.1"),
inet_addr(„194.204.152.34"));

**serwer:** 220 smtp.jakasdomena.com ESMTP ready\n (\n – znak końca wiersza)

Liczba na początku to kod odpowiedzi serwera SMTP. Kod 220 oznacza że serwer jest gotowy i czeka na dalsze polecenia. Dla naszego programu istotny jest tylko ten kod, tekst następujący po nim jest jedynie objaśnieniem dla użytkownika. Pojawiająca się na **list. 9** funkcja smtp_get_reply_code() odbiera za pomocą fread() odpowiedź z serwera i odczytuje z niej kod, który zwraca w postaci liczby typu int. Najważniejsze kody odpowiedzi SMTP objaśniono w **tab. 1**.

Pierwszą czynnością jest przedstawienie się serwerowi za pomocą komendy HELO:

**klient:** HELO nazwa_klienta\n

Zalecane jest podanie pełnej nazwy domeny (FQDN – *Fully Qualified Domain Name*) klienta. W praktyce wysyłana nazwa może być dowolna – często podaje się adres IP klienta w nawiasach kwadratowych []. Po otrzymaniu komendy HELO serwer powinien zwrócić kod 250 oznaczający poprawne wykonanie polecenia:

**serwer:** 250 smtp.jakasdomena.com: Hello, nazwa_klienta\n

Kolejnym krokiem jest zwykle autentykacja użytkownika, czyli podanie jego nazwy i hasła. Obecnie jest ona wymagana przez większość serwerów SMTP. Istnieje kilka metod autentykacji, jedną z częściej stosowanych jest metoda LOGIN:

**klient:** AUTH LOGIN\n
**serwer:** 334 VXNlcm5hbWU6\n (tekst „Username:" zakodowany w base64)
**klient:** nazwa_uzytkownika_zakodowana_w_base64\n
**serwer:** 334 UGFzc3dvcmQ6\n (tekst „Password:" zakodowany w base64)
**klient:** haslo_uzytkownika_zakodowane_w_base64\n

**Objaśnienia niektórych pojęć występujących w artykule**
**SMTP (*Simple Mail Transfer Protocol*)** – protokół komunikacyjny będący standardem przekazywania poczty elektronicznej w Internecie. Jest stosowany od wczesnych lat 80, do dziś (z wieloma rozszerzeniami). Usługa SMTP zwykle działa na porcie 25 TCP. Szczegółowy opis protokołu SMTP znajduje się w RFC2821 (*http://tools.ietf.org/html/rfc2821*).
**DNS (*Domain Name System*)** – w dużym uproszczeniu jest to system serwerów tłumaczących nazwy komputerów w Sieci na ich adresy IP. Skrót DNS odnosi się też do pojedynczego serwera nazw (Domain Name Server).
**Kodowanie *base64*** – jest to kodowanie, czyli sposób zapisu danych (nie należy mylić z szyfrowaniem!) chroniący je przed przypadkowym uszkodzeniem przy przesyłaniu przez różne, często niekombatybilne ze sobą urządzenia sieciowe lub programy (np. różniące się liczbą bitów przypadającą na znak ASCII – maszyna, która ma 7 bitów na znak nie przekaże poprawnie znaków 8–bitowych). Zakodowanie ciągu bajtów w base64 polega na podzieleniu wejściowego strumienia danych na 3–bajtowe (czyli 24–bitowe) bloki, a następnie na zapisaniu ich w postaci czterech znaków ASCII z 64–elementowego zbioru (kolejno: duże i małe litery alfabetu łacińskiego, cyfry oraz znaki + i /) z których każdy koduje 6 kolejnych bitów wejściowego bloku. Jeśli blok wejściowy ma rozmiar, który nie jest wielokrotnością liczby 3, blok wyjściowy dopełnia się znakami = tak, by jego rozmiar był wielokrotnością liczby 4.
Kodowanie base64 jest wykorzystywane między innymi w poczcie elektronicznej przy przesyłaniu binarnych załączników oraz nazw użytkownika i haseł w metodach AUTH PLAIN i AUTH LOGIN.
Przykład – kodowanie napisu **`TEST`**:

| | T | E | S | T | dopełnienie bloku | |
|---|---|---|---|---|---|---|
| Znaki wejściowe i ich kody ASCII (bin) | 01010100 | 01000101 | 01010011 | 01010100 | | |
| Znaki odpowiadające szóstkom bitów | V | E | V | T | V | A | = | = |

Wynikiem jest ciąg znaków **`VEVTVA==`**

Jeśli wszystko przebiegnie pomyślnie, otrzymamy odpowiedź zbliżoną do poniższej:

```
serwer: 235 Authentication successful\n
```

W przypadku podania niepoprawnej nazwy użytkownika i/lub hasła, odpowiedź serwera będzie następująca:

```
serwer: 535 Authentication failed\n
```

Program z **list. 9** pozwala na pominięcie etapu autentykacji przez odkomentowanie makrodefinicji `SMTP_DISABLE_AUTH` na początku kodu. Po „ceremonii" przedstawiania się i autentykacji przesyłamy serwerowi dane z koperty wiadomości, czyli adres nadawcy i odbiorcy:

```
klient: MAIL from: <jasio@jakasdomena.com>\n
serwer: 250 Sender OK\n
klient: RCPT to: <malgosia@jakasdomena2.com>\n
serwer: 250 Recipient OK\n
```

Następnie wydajemy polecenie `DATA`, po którym wysyłamy nagłówek, jedną pustą linię, a następnie treść wiadomości. Koniec listu sygnalizujemy serwerowi przez wysłanie linii zawierającej pojedynczą kropkę:

```
klient: DATA\n
serwer: 354 Go ahead...\n
klient: From: Jas Kowalski (jasio@jakasdomena.com)\n
klient: To: Malgosia Nowak (malgosia@jakasdomena2.com)\n
klient: Subject: List od Jasia\n
klient: \n (jedna pusta linia)
klient: Czesc Malgosiu\n
kilent: (ciąg dalszy treści)\n
klient: .\n (koniec treści listu)
serwer: 250 Message accepted
```

To wszystko – nasz e–mail został wysłany. Podczas jednej sesji możemy wysłać kilka e–maili. Przy wysyłaniu następnych wiadomości nie ma potrzeby wysyłania komendy `HELO` i powtórnej autentykacji. Po wysłaniu wszystkich widomości sesję należy zakończyć wydając polecenie `QUIT`:

```
klient: QUIT\n
serwer: 221 smtp.jakasdomena.com Out.\n
```

Po poprawnym zakończeniu sesji, zamykamy strumień połączony z *socketem* za pomocą funkcji `fclose()`, a następnie gniazdo sieciowe, wywołując funkcję `NutTcpCloseSocket()`.

**Tomasz Włostowski, EP**
**tomasz.wlostowski@ep.com.pl**

Kody źródłowe przykładowych programów znajdują się na płycie CD–EP oraz na stronach *http://ethernut.ep.com.pl* oraz *http://wlostowski.ep.com.pl*.