Ethernet i AVR–y
Ethernut od podstaw, część 3

Kontynuujemy rozpoczęty miesiąc temu opis serwera WWW wbudowanego w system Nut/OS. Jego najprostsza aplikacja pokazana w poprzednim odcinku pozwalała na wyświetlenie statycznej strony internetowej. W tej części pokażę, jak budować strony interaktywne – z formularzami i dynamicznie generowaną zawartością.

W tej części cyklu pokażę, jak zbudować prostą aplikację Ethernuta sterowaną za pomocą przeglądarki internetowej (**rys. 4**). Jej zadaniem jest zapalanie i gaszenie diody LED po kliknięciu „przycisku" oraz pokazywanie stanu przełączników znajdujących się na płytce z uruchomionym w Ethernutem w oknie przeglądarki (**rys. 5**). Aby taka aplikacja mogła działać, ethernutowy serwer WWW wyposażono w 2 mechanizmy:

– dynamiczne generowanie zawartości stron, które wykorzystamy do wypisywania stanu diod i przycisków,
– skrypty CGI, których będziemy używać do zapalania i gaszenia LED–ów.

Strony WWW z dynamiczną zawartością

W „dużych" serwerach WWW do tego generacji stron z dynamiczną zawartością wykorzystuje się zwykle rozbudowane języki skryptowe, jak np. PHP, serwlety w Javie czy ASP. Ponieważ Nut/OS działa na bardzo skromnym sprzęcie, mechanizm generacji stron z dynamiczną zawartością jest bardzo uproszczony i ogranicza się do wywołania wybranej funkcji programu w języku C przy napotkaniu odpowiedniego znacznika w kodzie strony w HTML–u:

```
<html>
<head>
<title>Strona testowa</title>
</head>
<body>Napis poniżej będzie
wygenerowany dynamicznie:<br>
<%napis%>
</body>
</html>
```

Po napotkaniu znacznika `<%napis%>`, serwer HTTP wbudowany w Nut/OS wywoła wybraną funkcję *callback*, w polskiej terminologii określaną jako *funkcja zwrotna*. Obsługę mechanizmu generacji dynamicznej zawartości stron, nazwanego w Ethernucie ASP (*Active Server Pages* – nie należy mylić z technologią ASP Microsoftu) należy zainicjalizować w następujący sposób:

```
NutRegisterAsp(); // uruchomienie
mechanizmu ASP
NutRegisterAspCallback(ASP
_callback);       // wybór funkcji
wywoływanej
                  // przy napotkaniu
znacznika <%znacznik%>
```

Funkcja *callback* w najprostszym przypadku wygląda następująco:

```
static int ASP_callback (char *tag_
name, FILE *f)
{
        if(!strcmp(tag_
name,"napis"))
   {
   fprintf(f, "Ten napis jest
wygenerowany dynamicznie!");
   return 0;
   }

   return -1;
}
```

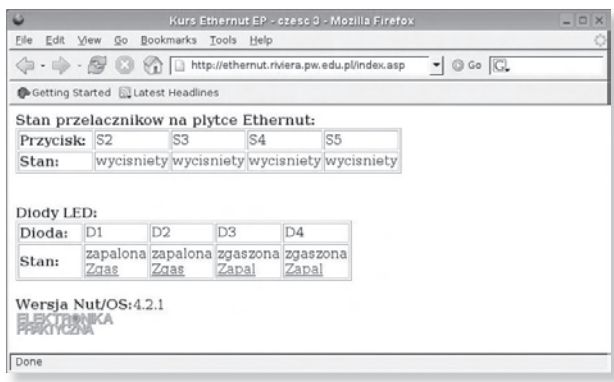
Parametr `tag_name` to nazwa napotkanego w kodzie strony znacznika ASP, natomiast `f` to strumień (`FILE`), do którego możemy zapisać dane, które zostaną przesłane do klienta zamiast znacznika ASP. A więc powyższy kod, w miejsce znacznika `<%napis%>` umieści tekst „*Ten napis jest wygenerowany dynamicznie!*".

Wspomnę tu o jeszcze jednej bardzo ważnej rzeczy: aby mechanizm dynamicznej generacji zawartości stron WWW mógł działać, plik z kodem strony musi mieć rozszerzenie `.asp` (np. `index.asp`)

ASP – bardziej rozbudowany przykład

Opisany poniżej przykład będzie generował dwie tabelki, zawierające stan diod oraz przełączników na płytce ZL9AVR oraz zestaw odnośników, których kliknięcie będzie powodowało zapalanie/gaszenie LED–ów. Szkielet strony w języku HTML jest następujący (plik `index.asp`):

```
<html><head>
<title>Kurs Ethernut EP - część 3
</title>
</head>
<body>
<b>Stan przełączników na płytce
Ethernut: </b>
<br>
<%przelaczniki%>
<br><br>
<b>Diody LED: </b><br>
<%diody%>
<br>
<br>
<b>Wersja Nut/OS:</b>
<%nut_version%><br>
</body>
</html>
```



Rys. 4. Okno przeglądarki z przykładową aplikacją uruchomioną na płytce ZL9AVR z modułem ZL1ETH firmy Kamami

List. 4.
```
static int CGI_callback(FILE *f, REQUEST *r)
{
// kod HTML przekierowujacy do pliku index.asp
    static prog_char webpage_code[] = "<html><head><meta http-equiv=\
"refresh\" content=\"0;url=../index.asp\"></head><body></body></html>";

// wysylamy naglowek protokolu HTTP 200 OK - wszystko w porzadku
    NutHttpSendHeaderTop(f, r, 200, "Ok");

// wysylamy "Content-Type" - rodzaj danych, czyli plik tekstowy w formacie
HTML (text/html)
    NutHttpSendHeaderBot(f, "text/html", -1);

// wysylamy kod strony w HTMLu
    fputs_P(webpage_code, f);

// upewniamy sie, ze wszystko zostalo juz wyslane do klienta
    fflush(f);

// sprawdzamy, czy skrypt zostal wywolany z parametrami
    if (r->req_query) {
        char *name;
        char *value;
        int i;
        int count;

// pobieramy liczbe parametrow
        count = NutHttpGetParameterCount(r);

        for (i = 0; i < count; i++) {

// pobieramy nazwe i wartosc kolejnego parametru
            name = NutHttpGetParameterName(r, i);
            value = NutHttpGetParameterValue(r, i);

// analizujemy go i podejmujemy odpowiednie czynnosci
            if(!strcmp(name,"zapal"))
                led_state |= (1<<atoi(value));
            else if(!strcmp(name,"zgas"))
                led_state &= ~(1<<atoi(value));
        }
    }


// aktualizujemy stan diod LED przez zapis do Portu F
    PORTF&=0xf0;
    PORTF|=(led_state)&0xf;

    return 0;
}
```

Kod funkcji `ASP_callback` oraz funkcji pomocniczych dla tej strony przedstawiono na **list. 4.** Obsługuje on dwa znaczniki: `<%przelaczniki%>` i `<%diody%>`, umieszczając w ich miejscu tabelki zawierające odpowiednio stan przełączników (odczytany bezpośrednio z portu D mikrokontrolera) oraz diod LED (przechowywany w zmiennej `led_state`). Dodatkowo w tabelce „*Diody LED*" znajdą się odnośniki umożliwiające ich zapalanie lub gaszenie. W pliku `index.asp` występuje także znacznik: `<%nut_version%>` – jest on wbudowany w system Nut/OS i w jego miejsce serwer WWW wstawia wersję używanego Ethernuta.

Aby sprawdzić, czy program poprawnie działa należy przytrzymać przycisk na płytce, a następnie (cały czas trzymając przycisk wciśnięty) kliknąć *Odśwież* w przeglądarce internetowej.

Skrypty CGI

Mamy już opanowane wysyłanie dynamicznie generowanych stron do *klienta*. Teraz umożliwimy przeglądarce internetowej wydawanie rozkazów dla naszej aplikacji. Do tego celu posłuży my się mechanizmem CGI (*Common Gateway Interface*). W kodzie HTML generowanym przez funkcję `ASP_callback()` pojawiają się następujące odnośniki do pliku `cgi-bin/diody.cgi`:

```
<a href=\"cgi-bin/diody.cgi?zgas=1">Zgas</a>
<a href=\"cgi-bin/diody.cgi?zapal=1">Zapal</a>
```

W przypadku „dużych" serwerów WWW, taki plik fizycznie istnieje i jest zwykle skryptem w Perlu. Skrypty CGI są przechowywane w oddzielnym katalogu o nazwie `cgi-bin`. W systemie Nut/OS plik skryptu jest zastąpiony funkcją w języku C.

Cechą mechanizmu CGI pozwalającą na przekazywanie poleceń dla serwera przez przeglądarkę internetową jest możliwość wywoływania skryptów z parametrami podanymi w ich adresie po znaku „`?`". Na przykład wpisanie w pasku adresu przeglądarki:

```
http://jakis_serwer/cgi-bin/diody.cgi?zapal=1
```

spowoduje (w systemie Ethernut) wywołanie funkcji odpowiadającej skryptowi `diody.cgi` z parametrem `zapal` o wartości `1`.

Stworzymy teraz skrypt `diody.cgi`, który umożliwi sterowanie diodami LED na płytce ZL9AVR. Będzie on przyjmował parmetry o postaci `zapal=numer_diody` i `zgas=numer_diody`, gdzie `numer_diody` to numer diody LED (0…3), która ma być zapalona lub zgaszona. Skrypt ten będzie musiał także wygenerować poprawną stronę WWW. W naszym przypadku po ustawieniu odpowiedniego stanu LED–ów, skrypt będzie przekierowywał przeglądarkę internetową do pliku `index.asp`.

Kod funkcji w języku C przypisanej do skryptu `diody.cgi` przedstawiono na **list. 5**. Funkcja `CGI_callback` przyjmuje 2 parametry: `FILE *f` – strumień, do którego zapis powoduje wysłanie zapisanych danych do przeglądarki klienta i `REQUEST *req` – strukturę opisującą żądanie odebrane przez serwer, zawierającą m.in. adres do którego odwołuje się klient oraz listę parametrów (tekst po znaku „`?`" w adresie strony)

W odróżnieniu od opisanego wcześniej mechanizmu ASP zastępującego określone miejsca w szablonie strony, funkcja obsługująca skrypt CGI musi oprócz kodu strony w HTML–u wysyłać klientowi nagłówki protokołu HTTP. Odpowiadają za to funkcje:

```
NutHttpSendHeaderTop(FILE *f, REQUEST *req, int status, char *title);
```

oraz

```
NutHttpSendHeaderBot(FILE *f, char *mime_type, long bytes);
```

Pierwsza z nich wysyła kod statusu (`status`) serwera HTML i odpowiadający mu komunikat (`title`) oraz informację o rodzaju i wersji oprogramowania serwera WWW. W naszym przypadku wysyłany kod to 200 („OK"). Oznacza on, że otrzymane polecenie jest poprawne i żądane dane zostaną wysłane. Często spotykanymi kodami statusu są np. 404 (*Not Found* – strona nie istnieje) albo 403 (*Forbidden* – dostęp zabroniony).

Druga z wymienionych funkcji wysyła klientowi informację o typie serwowanych danych (parametr `mime_type`) – w naszym przypadku "*text/html*", czyli dane tekstowe w formacie HTML oraz (jeżeli parametr `bytes` jest liczbą nieujemną) – długość przesyłanych danych, przydatną zwłaszcza przy przesyłaniu dużych plików (przeglądarka może wówczas podać

List. 5.
```
static int ASP_callback (char *tag_name, FILE *f)
{
    if(!strcmp(tag_name,"przelaczniki"))
    {
        tabelka_przelaczniki(f);
        return 0;
    } else if(!strcmp(tag_name,"diody"))
    {
        tabelka_diody(f);
        return 0;
    }

    return -1;
}

void tabelka_przelaczniki(FILE *f)
{
    int i;

// wysylamy do klienta kod zwyczajnej tablelki w HTMLu
    fprintf(f,"<table border=1 rows=5 cols=2><tr><td><b>Przycisk:</b>");

    for(i=0;i<4;i++)
        fprintf(f,"<td>S%d", i+2);              // nazwa przycisku na plytce

    fprintf(f,"</tr><tr><td><b>Stan:</b>");

    for(i=0;i<4;i++)
    {
        char stan = PIND & (1<<(i+4));      // odczytujemy stan przycisku
z portu D

// ... i wypisujemy w tabelce czy wlaczony, czy nie
        if(!stan)
            fprintf(f,"<td>wcisniety");
        else
            fprintf(f,"<td>wycisniety");
    }

    fprintf(f,"</tr></table>");             // koniec tabelki
}

static unsigned char stan_led = 0;

void tabelka_diody(FILE *f)
{
    int i;

    fprintf(f,"<table border=1 rows=5 cols=2><tr><td><b>Dioda:</b>");

    for(i=0;i<4;i++)
        fprintf(f,"<td>D%d", i+1);              // nazwa diody LED na płytce

    fprintf(f,"</tr><tr><td><b>Stan:</b>");

    for(i=0;i<4;i++)
    {
// sprawdzamy, czy dioda powinna byc zapalona:
        char stan = stan_led & (1<<i);

// ... i wypisujemy w tabelce jej stan
  if(stan)
        {
            fprintf(f,"<td>zapalona<br>");
// oraz odnosnik pozwalajacy na zmiane stanu diody:
            fprintf(f,"<a href=\"cgi-bin/diody.cgi?zgas=%d\">Zgas</a>", i);
        }
        else
        {
            fprintf(f,"<td>zgaszona<br>");
            fprintf(f,"<a href=\"cgi-bin/diody.cgi?zapal=%d\">Zapal</a>",
i);
        }
    }

    fprintf(f,"</tr></table>");                 // koniec tabelki
}
```

procentową wartość objętości ściąganego pliku).

Następnie wysyłamy kod strony (`webpage_code`) za pomocą funkcji `fputs_P()`. Jak wiemy mikrokontrolery AVR mają oddzielne przestrzenie adresowe pamięci Flash i RAM, konieczne więc było wprowadzenie dwóch wariantów funkcji w bibliotece standardowej. Funkcje z sufiksem `_P` w nazwie przyjmują dane z pamięci Flash, bez `_P` – z pamięci RAM. Kod HTML wysyłany w funkcji *CGI_callback* jest przechowywany w pamięci Flash (typ danych *prog_char*), aby zaoszczędzić pamięć RAM (kompilator AVR–GCC domyślnie umieszcza wszystkie ciągi znaków w pamięci RAM).

W wyniku działania powyższych funkcji, przeglądarka klienta odbierze od serwera następujące dane:

```
HTTP/1.0 200 Ok
nagłówki protokołu HTTP
Server: Ethernut 4.2.1
Content-type: text/html
1 pusta linia
<html><head> (.....) kod HTML zawarty w stałej webpage_code
```

Następnie sprawdzamy, czy skrypt CGI został wywołany z parametrami (element `req_query` struktury `REQUEST` jest wskaźnikiem do listy argumentów, ich brak powoduje przyjęcie wartości `NULL`). Jeśli mamy jakieś parametry, sprawdzamy ich liczbę (funkcja `NutHttpGetParameterCount()`), a następnie w pętli pobieramy ich nazwy i wartości za pomocą funkcji:

```
char *NutHttpGetParameterName(REQUEST *req, int index);
char *NutHttpGetParameterValue(REQUEST *req, int index);
```

gdzie `index` jest numerem interesującego nas parametru.

Mając nazwy i wartości argumentów, możemy je przeanalizować i podjąć stosowne czynności. W opisywanym przykładzie, napotkanie parametru o nazwie *zapal* (lub *zgas*) powoduje ustawienie (lub wyzerowanie) bitu w zmiennej `led_state` o numerze podanym w wartości parametru.

Na końcu funkcji `CGI_callback` przepisujemy 4 najmłodsze bity zmiennej `led_state` do rejestru PORTF, aby zaktualizować stan diod LED.

Pozostało nam jeszcze zarejestrować skrypt CGI w systemie i przypisać mu wybraną funkcję w języku C. Należy to wykonać przed uruchomieniem głównej pętli serwera WWW za pomocą funkcji:

```
int NutRegisterCgi (char *name, int(*func)(FILE *, REQUEST *));
```

podając jako `name` nazwę skryptu (w naszym przypadku *diody.cgi*) i adres obsługującej go funkcji jaki `func`.

Co dalej?

Opisany przykład serwera może w danym momencie obsługiwać tylko jedno połączenie. Przeglądarki internetowe potrafią wysyłać kilka żądań jednocześnie, których nasz serwer nie będzie w stanie obsłużyć. Taka sytuacja zakończy się błędem *connection refused* – połączenie odrzucone. Na szczęście dzięki wbudowanej w Nut/OS obsłudze wątków, można ten problem rozwiązać uruchamiając kilka kopii serwera działających jednocześnie. Ale o tym – za miesiąc!

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

Przykłady przedstawione w artykule zostały uruchomione na zestawie składającym się z płytki ewaluacyjnej ZL9AVR, interfejsu Ethernet z RTL8019 – ZL1ETH oraz modułu dipAVR – ZL7AVR, które udostępniła redakcji firma Kamami (*www.kamami.pl*).