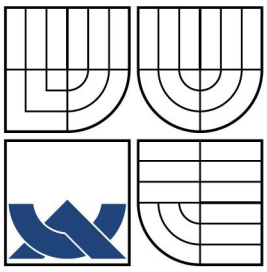
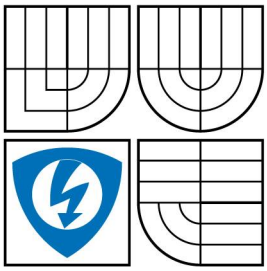


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV AUTOMATIZACE A MERICÍ TECHNIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF CONTROL AND INSTRUMENTATION

KONFIGURAČNÍ APLIKACE PRO EMBEDDED ETHERNET

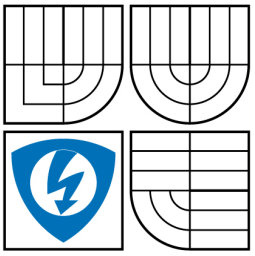
CONFIGURATION APPLICATION FOR EMBEDDED ETHERNET

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE
AUTHOR
MARTIN ZAMRZLA

VEDOUCÍ PRÁCE
SUPERVISOR
Ing. PETR FIEDLER, Ph.D

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav automatizace a měřicí techniky

Bakalářská práce

bakalářský studijní obor
Automatizační a měřicí technika

Student:	Martin Zamrzla	***ID:***	98399
Ročník:	3	***Akademický rok:***	2008/2009

NÁZEV TÉMATU:

Konfigurační aplikace pro embedded Ethernet

POKYNY PRO VYPRACOVÁNÍ:

Proveďte rešerši způsobů a možností zjištění dynamicky přidělené IP adresy u embedded zařízení bez displeje a klávesnice (např. ARP protokolem). Seznamte se s možnostmi operačního systému reálného času na platformě Rabbit (uC/OS-II). Udělejte na platformě Rabbit 2000/3000 projekt, který by obsahoval web server s konfigurační stránkou, kde lze nastavovat: Konfigurace IP - adresa zařízení, maska podsítě, adresa gatewaye; adresa DNS serverů (primární a záložní); adresa NTP serverů (primární a záložní); časová zóna pro zohlednění časového pásma; adresa pro zasílání hlášení od zařízení; konfigurace SMTP pro odeslání hlášení e-mailem.

DOPORUČENÁ LITERATURA:

Dle vlastního literárního průzkumu a doporučení vedoucího práce.

Termín zadání:	9.2.2009	***Termín odevzdání:***	1.6.2009

Vedoucí práce: Ing. Petr Fiedler, Ph.D.

prof. Ing. Pavel Jura, CSc.
Předseda oborové rady

Abstrakt

Tato práce se zabývá možnostmi nastavení síťových parametrů na embedded zařízení. Jsou popisovány dva typy nastavení síťových parametrů a to statické a dynamické. Dále se zabývá zasíláním emailů, ukládání uživatelských konfigurací do Flash paměti a synchronizací času.

Klíčová slova

Síťové parametry, embedded zařízení, statické parametry, dynamické parametry, Flash paměť

Abstract

This work describes possible options when setting up network parameters on the embedded machinery. Further, there are described two ways of setting up the network parameters, static and dynamic ones. Then it is dealing with issue concerning e-mail correspondence like sending as well as saving user´s configurations in the Flash memory and time synchronisation.

Keywords

Network parameters, embedded machinery, static parameters, dynamic parameters, Flash memory

Bibliografická citace:

ZAMRZLA, M. *Konfigurační aplikace pro embedded Ethernet*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2009

Prohlášení

„Prohlašuji, že svou bakalářskou práci na téma Konfigurační aplikace pro embedded Ethernet jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.
Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.“

V Brně dne: **1. června 2009** ……
podpis autora

Poděkování

Děkuji vedoucímu bakalářské práce Ing. Petru Fiedlerovi, Ph.D. za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé bakalářské práce.

V Brně dne: **1. června 2009** ……
podpis autora

OBSAH

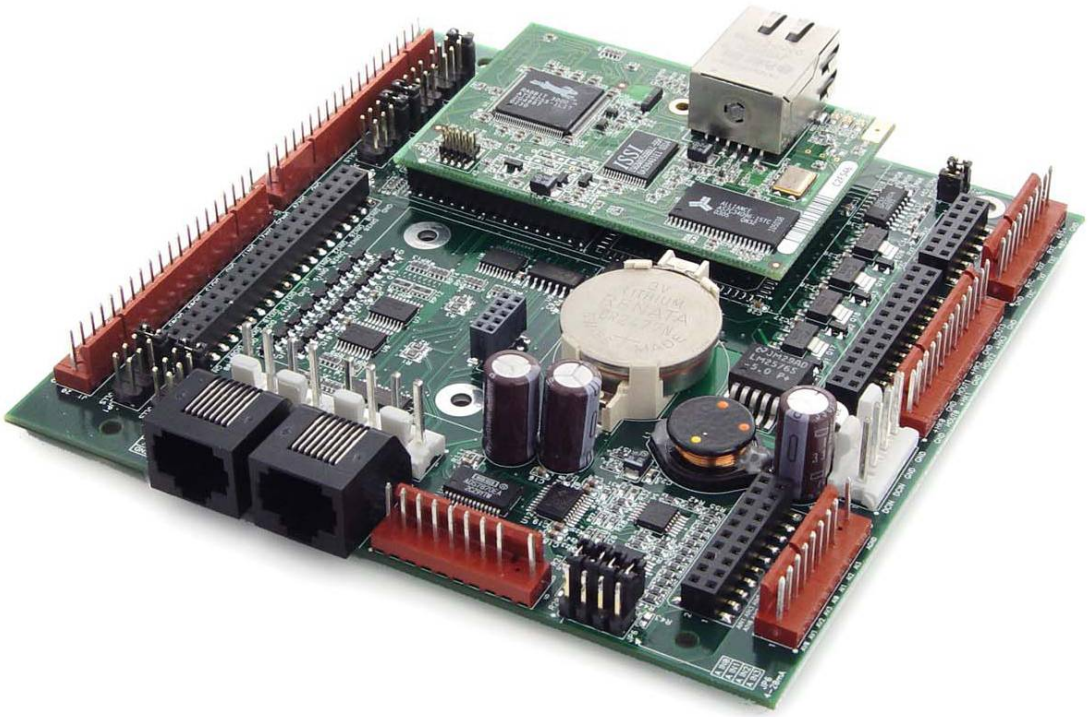
1. ÚVOD

Cílem této bakalářské práce je udělat rešerši o možnostech přidělení IP adresy u embedded zařízení na vývojovém kitu RABBIT 3000. Vytvořit vzorový projekt, který by obsahoval web server, kde je možnost přidělení IP adresy, síťové masky, gatewaye, DNS serverů (primární, sekundární), NTP serverů, konfigurace SMTP. Veškerá konfigurace má být uložena na interní paměť FLASH a odeslána emailem na zadanou adresu.

V první části pojednáme o vývojové desce BL2600 s modulem RCM 3200 a mikroprocesoru RABBIT 3000. Dále se dozvíme něco o embedded systémech, síťových protokolech, TCP, IP protokolu, ARP protokolu, DHCP serveru, DNS, NTP serveru a SMTP serveru. Poté následuje již popis TCP/IP protokolu v prostředí Dynamic C – inicializace, konfigurace síťových parametrů (statická, dynamická). Dále následuje popis TCP socketu jako alokace, otevření a funkce (I/O) TCP socketu. Po této kapitole již následuje popis webového serveru, http serveru. V této části bude popsána architektura webového serveru (aplikační blok, http blok), konfigurace stránek a používání proměnných pro www stránky.

Nakonec je popis a vzhled konfigurační stránky, popis navrženého softwaru pro vývojový kit. V příloze jsou výpisy programů (Bakalářská práce.c vytvořený v Dynamic C, web.html vytvořený v PSPad).

2. VÝVOJOVÝ POČÍTAČ BL2600 S MODULEM RCM3200 A MIKROPROCESOREM RABBIT 3000



2.1 VÝVOJOVÁ DESKA BL2600 A RABBIT 3000

BL2600 je vývojová deska, na které je připojen modul RCM3200 s mikroprocesororem RABBIT 3000, paměť FLASH, RAM, digitální vstupně výstupní obvody, A/D – D/A převodník, sériové porty RS232 / RS485 a Ethernetový port.

Tato deska se programuje pomocí programovacího kabelu, který se připojuje na sériový port počítače, nebo na USB – RS232 převodník, nebo přes Ethernet s RabbitLink.

Software:

Dynamic C je integrovaný vývojový nástroj pro vývoj embedded systémů. Ten běží na PC a je určený pro návrh zařízení založených na Rabbit mikroprocesorech.
Dynamic C poskytuje kompletní vývojové prostředí s editorem, kompilátorem a ladícím programem.

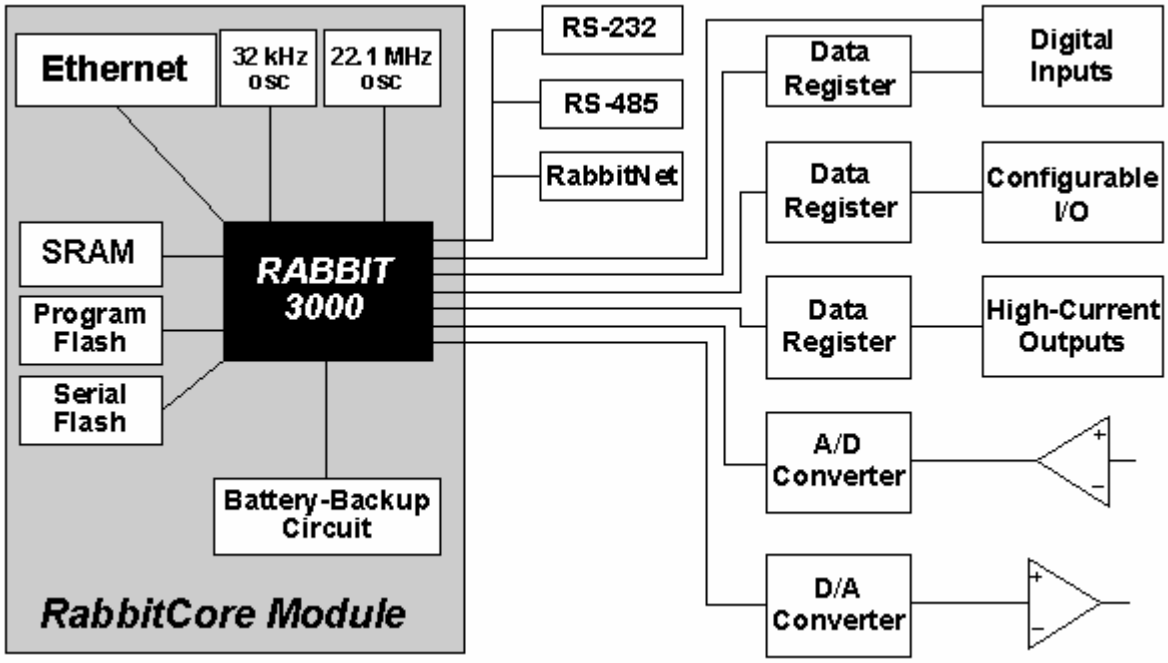
2.1.1 Vlastnosti BL2600

- Modul RCM3200 s mikroprocesorem Rabbit3000 pracující na frekvenci 44.2 MHz
- 512K statické paměti SRAM , 512K paměti FLASH
- 36 digitálních I/O
- 12 analogových kanálů: 8 11-ti bitových A/D převodníků, 4 12-ti bitové D/A převodníky
- RJ-45 Ethernetový port kompatibilní se standardy IEEE 802.3
- 3 sériové porty 2 RS-232 a 1 RS-485
- Hodiny reálného času
- Hlídací časovač Watchdog
- Tři LED diody Ethernetu
- 2 RabbitNet porty

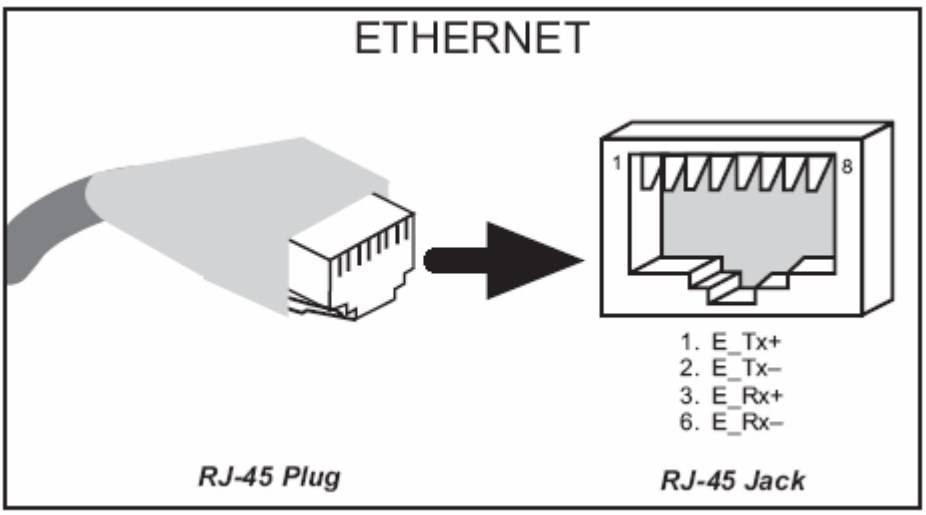
2.1.2 Mikroprocesor Rabbit 3000

Vlastnosti mikroprocesoru:

- Maximální taktovací rychlost 54MHz
- Maximální frekvence krystalu 30MHz
- Maximální operační napětí 3,6V
- 7 8-mi bitových I/O linek
- 6 sériových portů



Obr. 1. *Vývojová deska BL2600*



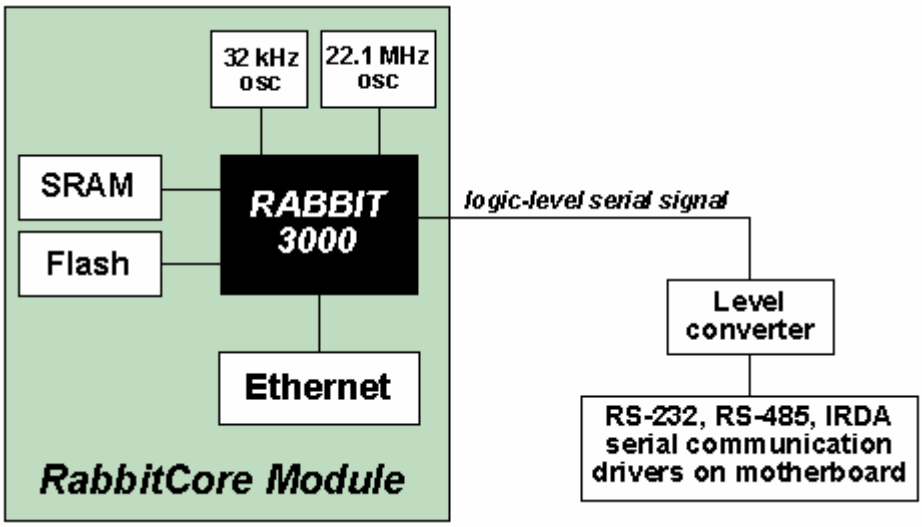
Obr. 2. *Ethernetový Port*

2.2 RABBITCORE MODUL RCM3200

Modul RCM3200 je navrženy pro použití na vývojové desce BL2600. Připojen je pomocí dvou 34 pinových konektorů, které poskytují 52 vstupů/výstupů, 5 sériových rozhraní a kontrolní signály. RCM3200 je dále vybaven Ethernetovým portem, 512K paměti flash, 512K paměti SRAM a 256K SRAM, která má záložní baterii.

Vlastnosti RCM3200:

- Mikroprocesor Rabbit3000 pracující na frekvenci 44.2 MHz
- 52 I/O linek, 44 konfigurovatelných, 4 pevně dané vstupy a 4 pevně dané výstupy
- 512K paměť flash, 512K SRAM, 256K data SRAM
- 10 8-mi bitových časovačů a jeden 10-ti bitový
- Hodiny reálného času
- Watchdog
- RJ-45 Ethernetový port
- 10-ti bitový čítač PWM
- 4 úrovně přerušení
- podporuje IrDA



Obr. 3. *Modul RCM3200*

3. EMBEDDED SYSTÉMY

Embedded systémy patří k nejrozšířenější variantě užívání počítačových systémů. Tyto systémy poskytují větším systémům, jichž jsou součástí, potřebnou inteligenci. Příklady embedded systémů najdeme od přenosného přehrávače hudby po užití v raketoplánu. Jedním z požadavků na tyto systémy je, že musí být tzv. „autonomní“ čili schopny plnit své funkce bez zásahu člověka v průběhu poměrně dlouhého časového intervalu. Proto je při vývoji hlavní důraz kladen na energetickou spotřebu, spolehlivost a robustnost. Autonomní činnost je nezbytná především tam, kde reakce člověka mohou být příliš pomalé, nedostatečně předvídatelné nebo nežádoucí. Dále například systémy, které pracují v reálném čase, musí být velmi rychlé při vykonávání daných funkcí. Další klíčovou charakteristikou většiny embedded systémů je, že by měly být neviditelné (skryté), tj. uživatel by je neměl považovat za počítač.

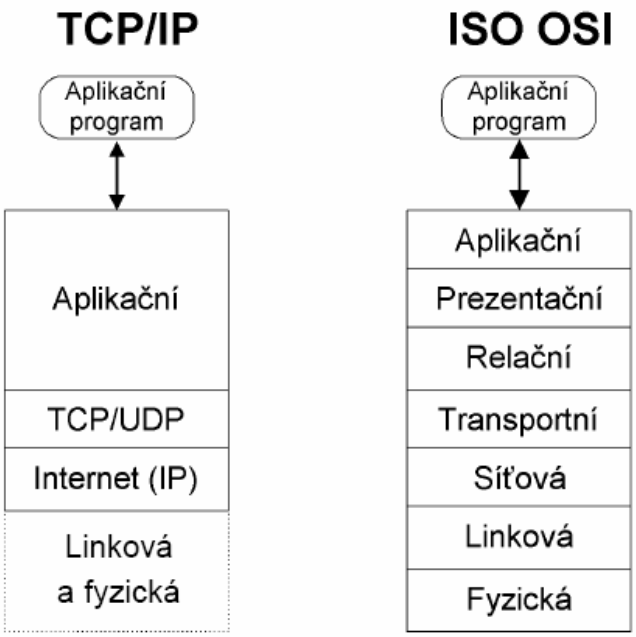
Na rozdíl od všeobecně použitelného počítače (například osobního) embedded systémy jsou navrženy pro konkrétní činnosti. Některé také pracují v reálném čase, tzn. že zpoždění činnosti nebo akce ovládané řídícím procesorem může mít fatální následky nebo poruchu činnosti (přerušení hraní hudby, zaseknutí motoru,…).
Při malých sériích nebo prototypech embedded systémů lze použít hardware osobního počítače a použít pouze konkrétní programy nebo nahradit původní operační systém operačním systémem pracujícím v reálném čase (RTOS).

Software:

Software, psaný pro embedded systémy, obzvláště pro takové, jež nemají diskovou jednotku se nazývá firmware, což je software uložené přímo v hardwarovém zařízení jako například v jednom IC čipu ROM nebo FLASH paměti. Programy na těchto systémech často běží v reálném čase a s omezenými hardwarovými zdroji, často tu nejsou diskové mechaniky, operační systém, klávesnice či obrazovka. Protože se předpokládá, že přístroje, které běží pod embedded systémy budou pracovat dlouhou dobu, je firmware vyvíjen mnohem pečlivěji než software pro osobní počítače. Musí být schopné se samy restartovat v případě selhání systému. Toho je docíleno pomocí speciální elektronické sledovací části zvané “watchdog timer”, která restartuje počítač.

4. SÍŤOVÉ PROTOKOLY

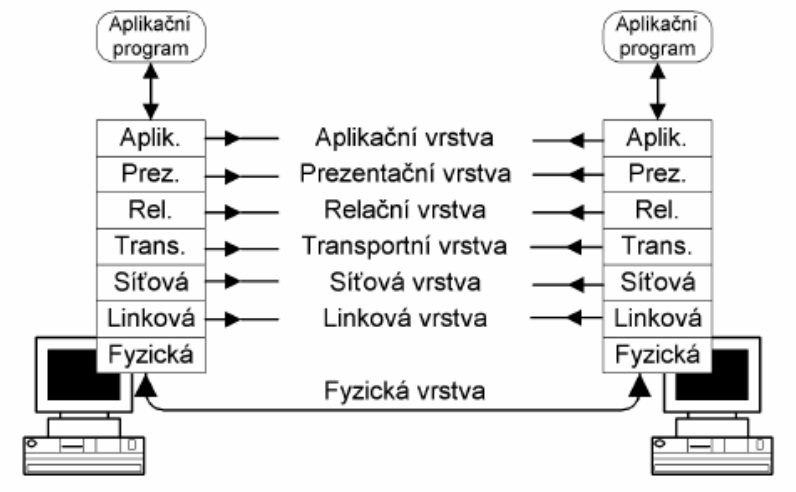
Počítače v počítačových sítích používají pro vzájemnou komunikaci síťové protokoly. Síťových protokolů existuje celá řada. V Internetu se používají síťové protokoly TCP/IP.
Mezinárodní normalizační úřad (ISO) normalizoval soustavu protokolů označovaných jako ISO OSI.
Rodina protokolů TCP/IP využívá čtyři vrstvy a protokoly ISO OSI používají vrstev dokonce sedm, jak je znázorněno na obr. 4.
V počítačových sítích používáme ještě více vrstev. Počet vrstev závisí na tom, jakou soustavu síťových protokolů použijeme. Místo o soustavě síťových protokolů někdy též mluvíme o tzv. síťovém modelu. Nejčastěji se budeme setkávat s modelem, který používá Internet, tento model se též nazývá rodinou protokolů TCP/IP. Kromě protokolů TCP/IP se setkáme ještě s modelem ISO OSI, který standardizoval mezinárodní standardizační úřad (ISO).



Obr. 4. *Síťové modely TCP/IP a ISO OSI*

4.1 MODEL OSI

Komunikace mezi dvěma počítači je schématicky znázorněna na obr. 5.



Obr. 5. *Architektura ISO OSI*

Fyzická vrstva

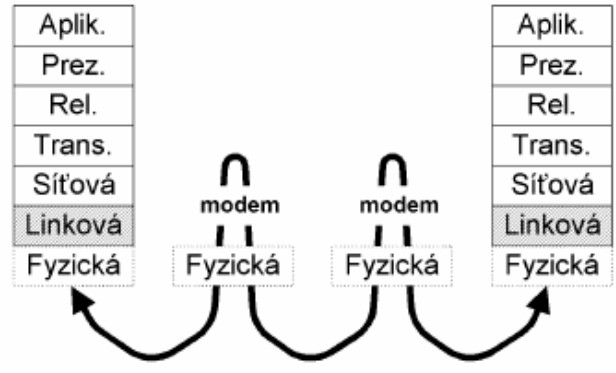
Fyzická vrstva popisuje elektrické či optické signály používané při komunikaci mezi počítači. Na fyzické vrstvě je vytvořen tzv. fyzický okruh.

Linková vrstva

Linková vrstva zajišťuje v případě sériových linek výměnu dat mezi sousedními počítači a v případě lokálních sítí výměnu dat v rámci lokální sítě.

Linkový rámec:

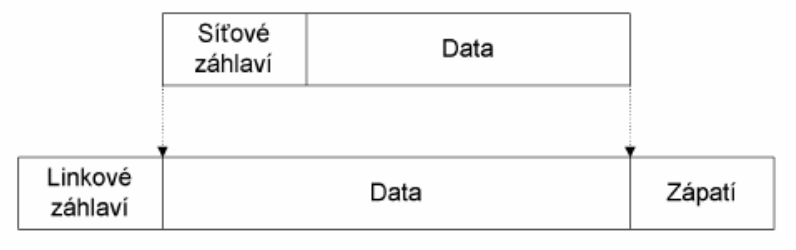
Datový rámec nese v záhlaví linkovou adresu příjemce, linkovou adresu odesílatele a další řídící informace. V zápatí nese obvykle kontrolní součet z přenášených dat. Pomocí něho lze zjistit, zdali nedošlo při přenosu k porušení dat. V přenášených datech je pak zpravidla nesen paket síťové vrstvy.



Obr. 6. *Komunikace na Linkové vrstvě*

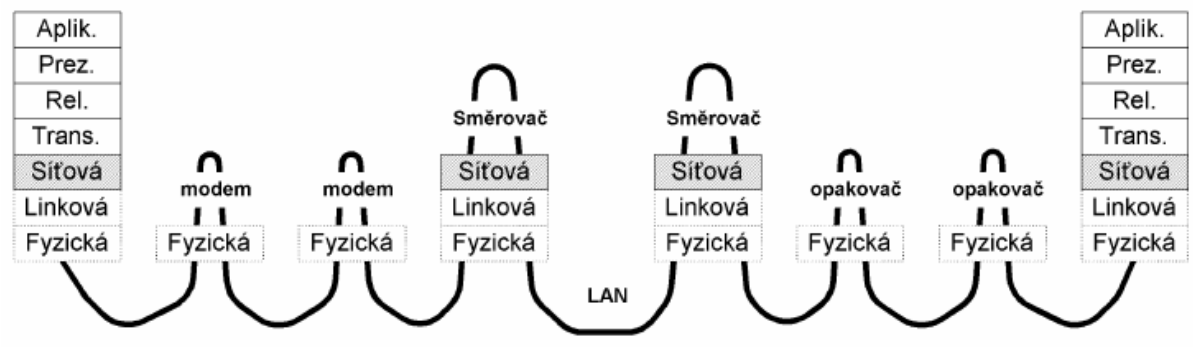
Síťová vrstva

Síťová vrstva zabezpečuje přenos dat mezi vzdálenými počítači WAN. Základní jednotkou přenosu je síťový paket, který se balí do datového rámce. Síťový paket se také skládá ze záhlaví a datového pole. Se zápatím se u síťových protokolů setkáváme jen zřídka.
Z obr. je patrné, že síťové záhlaví společně s daty síťového paketu tvoří data linkového rámce.



Obr. 7. *Síťový paket a jeho vkládání*

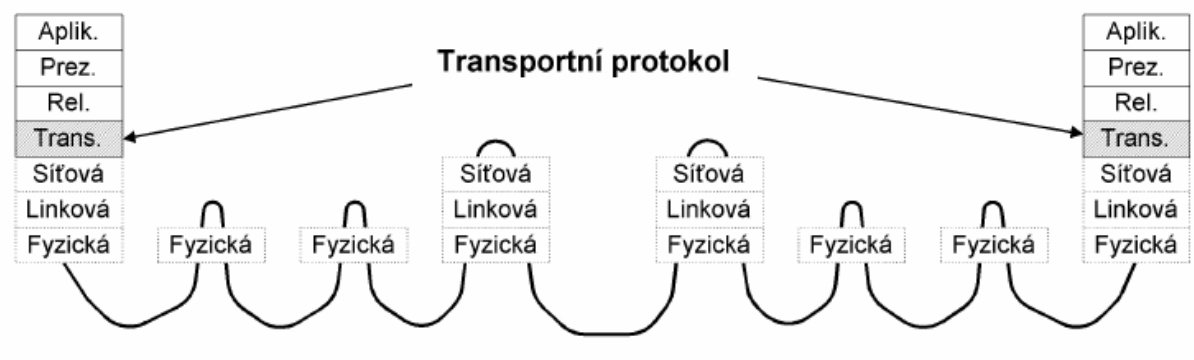
V rozsáhlých sítích mezi počítači leží zpravidla jeden nebo více směrovačů. Mezi sousedními směrovači je na linkové vrstvě vždy přímé spojení. Směrovač vybalí síťový paket z datového rámce (jednoho linkového protokolu) a před odesláním do jiné linky jej opět zabalí do jiného datového rámce (obecně jiného linkového protokolu).



Obr. 8. *Komunikace na síťové vrstvě*

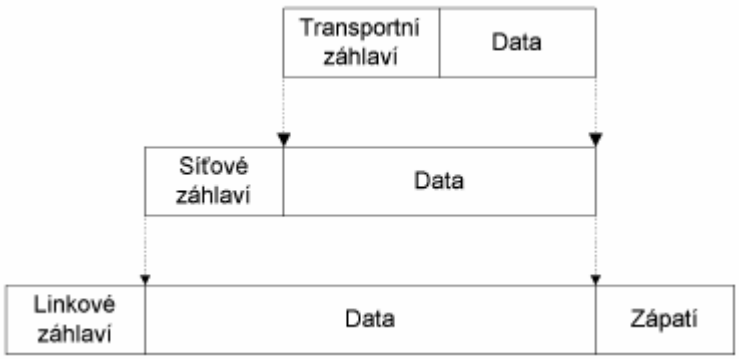
Transportní vrstva

Síťová vrstva zabezpečí spojení mezi vzdálenými počítači, takže transportní vrstvě se jeví jakoby žádné modemy, opakovače, mosty či směrovače na cestě nebyly. Transportní vrstva se zcela spoléhá na služby nižších vrstev. Také předpokládá, že spojení mezi počítači je zajištěno, proto se bez zbytečných starostí může věnovat spojení mezi aplikacemi na vzdálených počítačích.



Obr. 9. *Komunikace na transportní vrstvě*

Mezi dvěma počítači může být několik transportních spojení současně. Z hlediska síťové vrstvy jsou pakety adresovány adresou počítače (resp. jeho síťového rozhraní). Z hlediska transportní vrstvy jsou adresovány jednotlivé aplikace. Aplikace jsou jednoznačně adresovány v rámci jednoho počítače. Jednotkou přenosu je transportní paket, který se opět skládá ze záhlaví a datové části. Transportní paket se přenáší v datové části síťového paketu.



Obr. 10. *Vkládání transportních paketů do síťových paketů*

Relační vrstva

Relační vrstva zabezpečuje výměnu dat mezi aplikacemi, tj. provádí tzv. checkpoint, synchronizaci transakcí (commit), korektní uzavírání souborů atd. Dobře představitelnou relací je např. sdílení síťového disku.
Základní jednotkou je relační paket, který se opět vkládá do transportního paketu. V literatuře se můžeme často sekat s obrázkem, jak se relační paket skládá z relačního záhlaví a relačních dat a celý relační paket se vkládá do transportního paketu. Od transportní vrstvy výše tomu tak být nemusí. Informace relační vrstvy mohou být přenášeny uvnitř dat. Ještě markantnější je tato situace u prezentační vrstvy, která data např. zašifruje, takže změní celý obsah paketu.

Prezentační vrstva

Prezentační vrstva je zodpovědná za reprezentaci a zabezpečení dat. Reprezentace dat může být na různých počítačích různá. Např. se jedná o problém zdali je nejvyšší bit v bajtu zcela vlevo nebo vpravo atp. Zabezpečením se rozumí šifrování, zabezpečení integrity dat, digitální podepisování atd.

Aplikační vrstva

Aplikační vrstva předepisuje v jakém formátu a jak mají být data přebírána/předávána od aplikačních programů. Např. protokol Virtuální terminál popisuje jak mají být data formátována, ale i dialog mezi oběma konci spojení.

5. TCP/IP PROTOKOLY

Rodina protokolů TCP/IP se nezabývá (až na výjimky) fyzickou a linkovou vrstvou. V praxi se i v Internetu používají pro fyzickou a linkovou vrstvu často protokoly vyhovující normám ISO OSI, které standardizoval ITU.
Vztah mezi protokoly ISO OSI a TCP/IP? Každá skupina má vlastní definici svých vrstev i protokolů jednotlivých vrstev. Proto jsou protokoly ISO OSI a TCP/IP obecně nesouměřitelné. V praxi však je třeba využívat komunikační zařízení vyhovující ISO OSI pro přenos IP-paketů nebo např. naopak realizovat služby podle ISO OSI přes Internet.

Internet Protokol

Internet Protokol (dále jen IP-protokol) prakticky odpovídá síťové vrstvě. IP-protokol přenáší tzv. IP-datagramy mezi vzdálenými počítači. Každý IP-datagram ve svém záhlaví nese adresu příjemce, což je úplná směrovací informace pro dopravu IP-datagramu k adresátovi. Takže síť může přenášet každý IP-datagram samostatně. IP-datagramy tak mohou k adresátovi dorazit v jiném pořadí než byly odeslány.
Každé síťové rozhraní v rozsáhlé síti Internet má svou celosvětově jednoznačnou IP-adresu (jedno síťové rozhraní může mít více IP-adres, avšak jednu IP-adresu nesmí používat více síťových rozhraní). Internet je tvořen jednotlivými sítěmi, které jsou propojeny pomocí směrovačů. Směrovač se anglicky nazývá router, ve starších publikacích se však označuje jako gateway.

Protokoly TCP a UDP

Protokoly TCP a UDP odpovídají transportní vrstvě. Protokol TCP dopravuje data pomocí TCP segmentů, které jsou adresovány jednotlivým aplikacím. Protokol UDP dopravuje data pomocí tzv. UDP datagramů. Protokoly TCP a UDP zajišťují spojení mezi aplikacemi běžícími na vzdálených počítačích. Protokoly TCP a UDP mohou zajišťovat i komunikaci mezi procesy běžícími na témže počítači, to je však z našeho pohledu nepříliš zajímavé.
Rozdíl mezi protokoly TCP a UDP spočívá v tom, že protokol TCP je tzv. spojovanou službou, tj. příjemce potvrzuje přijímaná data. V případě ztráty dat (ztráty TCP segmentu) si příjemce vyžádá zopakování přenosu. Protokol UDP přenáší data pomocí datagramů (obdoba telegramu), tj. odesílatel odešle datagram a už se nezajímá o to, zdali byl doručen.
Adresou je tzv. port. Pro pochopení rozdílu mezi IP-adresou a portem se používá srovnání s poštovní adresou. IP-adresa odpovídá adrese domu a port jménu a příjmení osoby, které má být dopis doručen.

Aplikační protokoly

Aplikační protokoly odpovídají několika vrstvám ISO OSI. Relační, prezentační a aplikační vrstva ISO OSI je zredukována do jedné aplikační vrstvy TCP/IP.

Absence prezentační vrstvy se řeší zavedením specializovaných „prezentačních-aplikačních“ protokolů, jako jsou protokoly SSL a S/MIME specializující se na zabezpečení dat. Nebo protokoly Virtuální terminál a ASN.1 určené pro prezentaci dat. Protokol Virtuální terminál (nezaměňovat se stejnojmenným protokolem v ISO OSI) specifikuje prezentaci dat v síti pro protokol Telnet, avšak využívají jej i další protokoly (FTP, SMTP a částečně i HTTP).

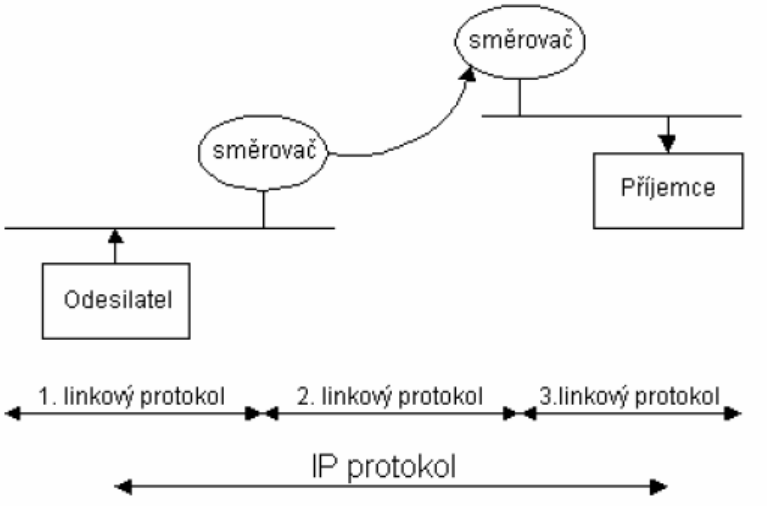
5.1 IP PROTOKOL

IP-protokol je protokol, umožňující spojit jednotlivé lokální sítě do celosvětového Internetu. Od protokolu IP dostal také Internet své jméno. Zkratka IP totiž znamená InterNet Protocol, tj. protokol spojující jednotlivé sítě.

IP-protokol je tvořen několika dílčími protokoly:

- Vlastním protokolem IP.
- Služebním protokolem ICMP sloužícím zejména k signalizaci mimořádných stavů.
- Služebním protokolem IGMP sloužícím pro dopravu adresných oběžníků.
- Služebními protokoly ARP a RARP, které jsou často vyčleňovány jako samostatné, na IP nezávislé protokoly, protože jejich rámce nejsou předcházeny IP-záhlavím.

V linkovém protokolu má každé síťové rozhraní (network interface) svou fyzickou (tj. linkovou) adresu, která je v případě LAN zpravidla šestibajtová, tak v IP-protokolu má každé síťové rozhraní alespoň jednu IP-adresu, která je v případě IP-protokolu verze 4 čtyřbajtová, a v případě IP-protokolu verze 6 šestnáctibajtová.



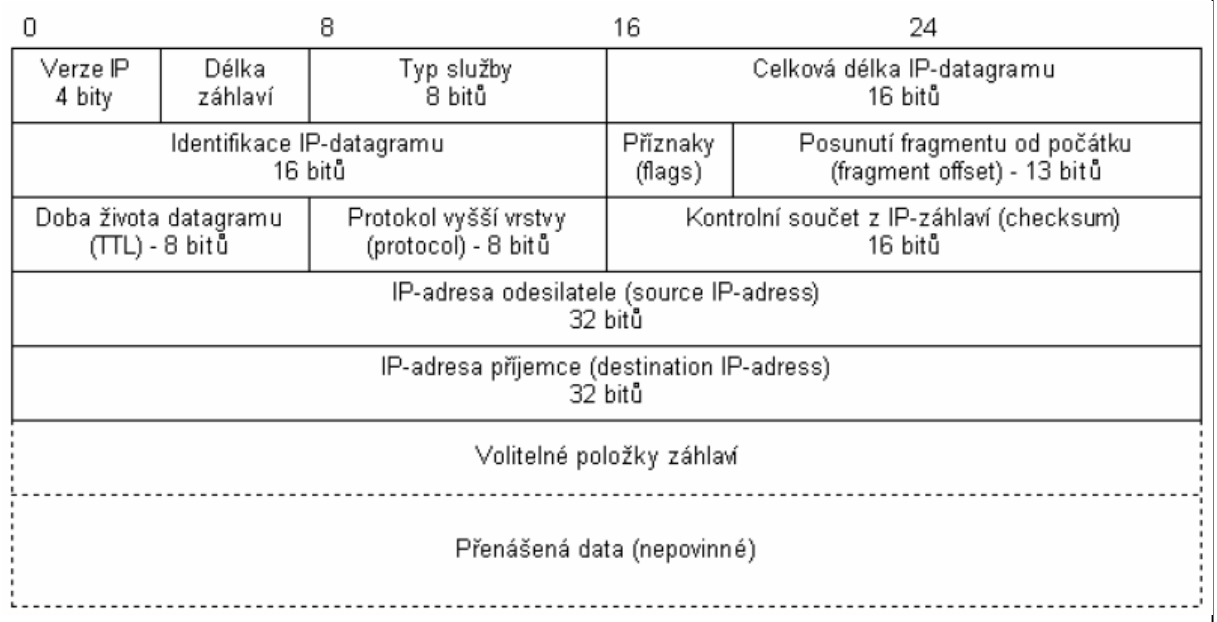
Obr. 11. *Linkové a IP protokol*

Obrázek 11. znázorňuje, že linkový protokol dopravuje datové rámce pouze k následujícímu směrovači, kdežto IP-protokol dopravuje data mezi dvěma vzdálenými počítači rozsáhlé sítě. Zatímco obálka, kterou jsou na linkové vrstvě data obalena je na každém směrovači vždy zahozena a vytvořena nová, tak IP-datagram není směrovačem změněn. Směrovač nesmí změnit obsah IP-datagramu. Výjimkou je pouze položka TTL ze záhlaví IP-datagramu, kterou je každý směrovač povinen zmenšit alespoň o jedničku a v případě změny na nulu se IP-datagram zahazuje. Tímto mechanismem se Internet snaží zabránit nekonečnému toulání paketů Internetem. Existují i další výjimky, ke kterým se také později dostaneme (např. fragmentace). Zatímco u linkových protokolů jsme základní přenášené kvantum dat označovali jako linkový rámec, tak u IP-protokolu je základní jednotkou přenášených dat IP-datagram.

5.2 IP DATAGRAM

IP-datagram se skládá ze záhlaví a přenášených dat. Záhlaví má zpravidla 20 bajtů. Záhlaví však může obsahovat i volitelné položky a v takovém případě je záhlaví o ně delší.

Struktura IP-datagramu je na obrázku 12.



Obr. 12. *IP datagram*

5.3 PROTOKOL ICMP

Protokol ICMP je služební protokol, který je součástí IP-protokolu. Protokol ICMP slouží k signalizaci mimořádných událostí v sítích postavených na IP-protokolu. Protokol ICMP svoje datové pakety balí do IP-protokolu, tj. pokud budeme prohlížet přenášené datagramy, pak v nich najdeme za linkovým záhlavím záhlaví IP-protokolu následované záhlavím ICMP paketu. Protokolem ICMP je možné signalizovat nejrůznější situace, skutečnost je však taková, že konkrétní implementace TCP/IP podporují vždy jen jistou část těchto signalizací a navíc z bezpečnostních důvodů mohou být na směrovačích mnohé ICMP signalizace zahazovány.

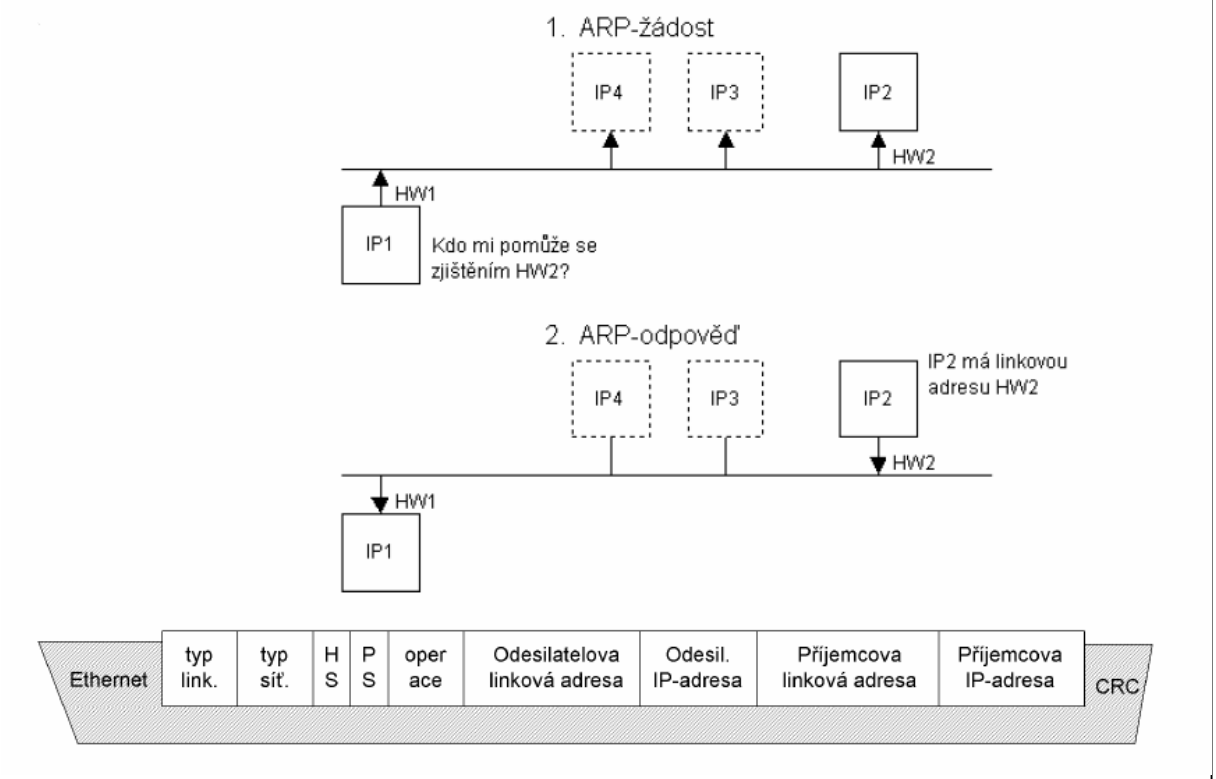
5.4 PROTOKOLY ARP A RARP

Je-li stanice na lokální síti a chce protokolem IP komunikovat s jinou stanicí na téže síti, pak ji v protokolu IP adresuje čtyřbajtovou IP-adresou. Pro komunikaci zná IP-adresu odesílatele (svou IP-adresu) a IP-adresu příjemce. Je tedy schopna sestavit IP-datagram. Jenže potíž je v tom, že tento IP-datagram musí být zabalen do linkového rámce – např. do ethernetového rámce. Aby vytvořil ethernetový rámec, tak potřebuje linkovou (6B) adresu příjemce i odesílatele. Odesílatel zná linkovou adresu, avšak nezná linkovou adresu příjemce. Jak takovou adresu zjistí? To řeší protokol ARP.

Protokol ARP (Address Resolution Protocol) řeší problém zjištění linkové adresy protější stanice ze znalosti její IP-adresy. Řešení je jednoduché, do LAN vyšle linkový oběžník (linková adresa FF:FF:FF:FF:FF:FF) s prosbou: „Já stanice o linkové adrese HW1, IP-adrese IP1, chci komunikovat se stanicí o IP-adrese IP2, kdo mi pomůže s nalezením linkové adresy stanice o IP-adrese IP2? Stanice IP2 takovou žádost uslyší a odpoví. V odpovědi uvede svou linkovou adresu HW2.

ARP-paket je balen přímo do Ethernetu, tj. nepředchází mu žádné IP-záhlaví. Protokol ARP je vlastně samostatný, na IP nezávislý protokol. Proto jej mohou používat i jiné protokoly, které s protokoly TCP/IP nemají nic společného.
Pole typ linkového protokolu specifikuje linkový protokol používaný na LAN. Linkovému protokolu Ethernet II je vyhrazeno číslo 1.

Typ síťového protokolu specifikuje typ síťového protokolu, používají se stejná čísla jako pro pole protocol v protokolu Ethernet II, tj. IP-protokol má přiděleno číslo 80016.
Pole HS určuje délku linkové adresy a pole PS délku síťové adresy. Standardně je tedy HS=6 a PS=4.
Pole operace určuje o jakou operaci jde. Žádost (ARP request) má hodnotu 1 a odpověď (ARP reply) má hodnotu 2. Toto pole je definováno rovněž pro reverzní překlad (protokol RARP), kdy RARP žádost používá hodnotu 3 a RARP odpověď hodnotu 4.
Pak již následuje linková adresa odesílatele, IP-adresa odesílatele, linková adresa příjemce (v dotazu vyplněna nulami) a IP-adresa příjemce.



Obr. 13. *Komunikace ARP protokolu*

Žádost je posílána linkovým oběžníkem a v poli příjemcova linková adresa má vyplněny nuly. Odpověď 1 pak má již vyplněna všechny pole a nemusí být odesílána oběžníkem. Je třeba zdůraznit, že v odpovědi je odesílatelem dotazovaný a příjemce tazatel (došlo k výměně příjemce a odesílatele).

5.4.1 RARP

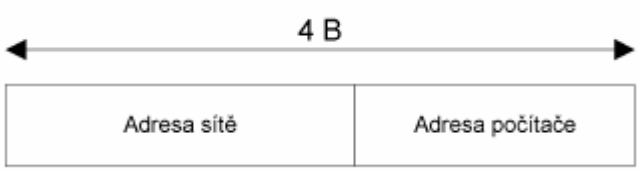
Protokolem ARP je také možné odeslat žádost s vyplněnou IP-adresou odesílatele i příjemce a také s oběma vyplněnými linkovými adresami. Takovou žádost je možné chápat jako: „Neexistuje náhodou“ na LAN ještě jiná stanice, která používá stejnou IP-adresu jako já?”. V případě, že se obdrží odpově1, tak se uživateli signalizuje zpráva „Duplicate IP address sent from Ethernet address xx:xx:xx:xx:xx:xx”. To pochopitelně signalizuje chybu v konfiguraci jedné ze stanic používajících tuto adresu.
Zatímco protokol ARP slouží k překladu IP-adres na linkové adresy reverzní ARP označované jako RARP slouží k překladu linkové adresy na IP-adresu. Avšak proč takový překlad provádět? Smysl protokolu RARP je u bezdiskových stanic.

Bezdisková stanice po svém zapnutí nezná nic jiného než svou linkovou adresu (tu má uloženu výrobcem v paměti ROM). Po svém zapnutí se potřebuje dozvědět svou IP-adresu. Proto do LAN vyšle oběžník s prosbou: „Já mám linkovou adresu HW1, kdo mi řekne jakou mám IP-adresu”. Na LAN pak musí být RARP-server, který jí IP-adresu přidělí a sdělí v odpovědi. Protokol RARP používá stejný formát paketu jako protokol ARP. Pouze hodnota pole operace je zvětšena o jedničku. V RARP žádosti pochopitelně není vyplněna ani IP-adresa žadatele. Protokol RARP se v praxi téměř nepoužívá, nahradil jej protokol DHCP, který je komplexnější.

5.5 IP ADRESA

Protokol IP verze 4 používá IP-adresu o délce čtyři bajty. IP-adresa adresuje jednoznačně síťové rozhraní systému. Anglicky se takováto jednoznačná adresa nazývá unicast. Pokud má systém více síťových karet (více síťových rozhraní) a na všech je provozován protokol IP, pak každé rozhraní má svou IPadresu. Je to podobné jako s adresou domu. Pokud má dům vchod ze dvou ulic, pak má i dům dvě adresy.
IP-adresa je tvořena čtyřmi bajty. IP-adresa se zapisuje notací, kde jednotlivé bajty se mezi sebou oddělují tečkou.
Rozeznáváme:
- Dvojkovou notaci, kde jednotlivé bity každého bajtu se vyjádří jako dvojkové číslo, např.: 10101010.01010101.11111111.11111000
- Desítkovou notaci – čtyři osmiciferná dvojková čísla se převedou do desítkové soustavy, např: 170.85.255.248
- Šestnáctkovou notaci – jednotlivé bajty IP-adresy se vyjádří šestnáctkově (hexadecimálně), např: aa.55.ff.f8

IP-adresa se skládá ze dvou částí:
1. Adresy (lokální) sítě.
2. Adresy počítače v (lokální) síti.



Obr. 14. *Struktura IP adresy*

IP-adresa se dělí na adresu sítě a adresu počítače v rámci této sítě (viz obr.).
Kolik bajtů z IP-adresy tvoří adresu sítě určují počáteční bity prvního bajtu IP-adresy. IP-adresy se dělí do pěti tříd:

Třída A, kde nevyšší bit prvního bajtu má hodnotu 0. Zbylých 7 bitů prvního bajtu tvoří adresu sítě a zbytek je určen pro adresu počítače v rámci sítě. V třídě A máme 126 sítí (0 a 127 mají zvláštní význam. V každé síti je 224-2 adres pro počítače (adresy tvořené samými nulami a samými jedničkami mají zvláštní význam).

Třída B, kde nejvyšší dva bity prvního bajtu mají hodnotu 102. Zbylých 6 bitů a následující druhý bajt je určen pro adresy sítí. Můžeme tedy mít celkem 214 sítí a v každé síti 216-2 počítačů.

Třída C, kde nevyšší tři bity prvního bajtu mají hodnotu 1102. Zbylých 5 bitů a následující dva bajty jsou určeny pro adresu sítě. Můžeme tedy mít 222 sítí a v každé síti 128-2 počítačů.

Třída D, kde nejvyšší čtyři bity prvního bajtu mají hodnotu 11102. Zbytek IP-adresy se pak už nedělí na adresu sítě a adresu počítače. Zbytek IP-adresy tvoří adresný oběžník (multicast).

Třída E tvořící zbytek adres je tč. rezervou.

Třída	Začátek	1.bajt	Standardní maska	Bitů sítě	Bitů stanice	Sítí	Stanic v každé síti
A	0	0-127	255.0.0.0	7	24	126	16777214
B	10	128-191	255.255.0.0	14	16	16384	65534
C	110	192-223	255.255.255.0	21	8	2097152	254
D	1110	224-239	multicast				
E	1111	240-255	Vyhrazeno jako rezerva				

5.5.1 Možnosti přidělení IP adresy

IP adresa může být stanici přidělena několika způsoby:

Ruční nastavení
V tomto případě správce sítě nevyužívá DHCP serveru a konfiguraci jednotlivých stanic zapisuje jednotlivě přímo do konfigurace jednotlivých stanic.
Statická alokace
DHCP server obsahuje seznam MAC adres a k nim příslušným IP adres. Pokud je žádající stanice v seznamu, dostane vždy přidělenu stejnou pevně definovanou IP adresu.
Dynamická alokace
Správce sítě na DHCP serveru vymezí rozsah adres, které budou přidělovány stanicím, které nejsou registrovány. Časové omezení pronájmu IP adresy dovoluje DHCP serveru již nepoužívané adresy přidělovat jiným stanicím. Registrace dříve pronajatých IP adres umožňuje DHCP serveru při příštím pronájmu přidělit stejnou IP adresu.

V IPv6 sítích je automatickému nastavení stanice věnována vyšší pozornost, aby byla konfigurace počítačové sítě ještě jednodušší.

5.6 SÍŤOVÁ MASKA

Síťová maska se používá pro určení adresy sítě. Adresa sítě je částí IP adresy. Síťová maska určuje, které bity v IP-adrese tvoří adresu sítě. Síťová maska je opět čtyřbajtové číslo. Toto číslo vyjádřené v dvojkové soustavě má v bitech určujících adresu sítě jedničky a v ostatních bitech nuly. Princip síťové masky se dobře pochopí, používáme-li dvojkovou notaci.
Jednotlivé třídy sítí používají jako adresu sítě různě dlouhou část IP adresy.
Třída A používá pro adresu sítě první bajt. Čili standardní síťová maska pro adresy třídy A má v prvním bajtu samé jedničky a ve zbylých třech bajtech samé nuly:
11111111.00000000.00000000.00000000
což vyjádřeno v desítkové soustavě je:
255.0.0.0 (šestnáctkově ff.00.00.00)
Obdobně standardní síťová maska pro třídu B je desítkově:
255.255.0.0 (šestnáctkově ff.ff.00.00)
Pro třídu C:
255.255.255.0 (šestnáctkově ff.ff.ff.00).
Síťové masky odpovídající třídám A, B a C se nazývají standardní síťové masky.

5.7 DYNAMICKY PŘIDĚLOVANÉ ADRESY (DHCP)

Má-li síť již interval IP-adres přidělen, pak můžeme začít s přidělováním adres jednotlivým síťovým rozhraním na této síti. Jsou dvě možnosti:

- Staticky (trvale) přidělit IP-adresu.
- Dynamicky (na dobu připojení) přidělit IP-adresu.

Dynamické přidělování přináší výhodu i v tom, že je potřeba jen tolik IP-adres, kolik je současně přihlášených uživatelů. Dynamické přidělování adres řeší aplikační protokol DHCP. Protokol DHCP vychází ze zkušeností a částečně v sobě zahrnuje i podporu starších protokolů z této oblasti, tj. protokolů RARP, DRARP a BOOTP.
V protokolu DHCP žádá klient DHCP-server o přidělení IP-adresy (případně o další služby). DHCP-server může být realizován jako proces na počítači s operačním systémem UNIX, Windows NT atp. Nebo DHCP-server může být realizován i jako součást směrovače.
Zatímco přidělování IP-adres na LAN je v současné době doménou protokolu DHCP, pro přidělování IP-adres počítačům za komutovanou linkou (např. zákazníkům poskytovatele Internetu) se zpravidla přidělují IP-adresy pomocí protokolu PPP.
Protokol PPP je linkovým protokolem používaným na asynchronních sériových linkách. Neumožňuje takové služby jako protokol DHCP, avšak přidělit IP-adresu stanice umí. Více stejně pro připojení uživatele k Internetu nebývá třeba.
Dynamické přidělování IP-adres může být ještě kombinováno s nečíslovanými sítěmi. V případě, že budou sériové linky jako nečíslované sítě, pak směrovač dynamicky přidělující IP-adresy zařídí, že počítače se budou jevit, jakoby byly přímo na lokální síti.

5.8 DNS

DNS (Domain Name System) je hierarchický systém doménových jmen, který je realizován servery DNS a protokolem stejného jména, kterým si vyměňují informace. Jeho hlavním úkolem a příčinou vzniku jsou vzájemné převody doménových jmen a IP adres uzlů sítě. Později ale přibral další funkce (např. pro elektronickou poštu či IP telefonii) a slouží dnes de facto jako distribuovaná databáze síťových informací.
Protokol používá porty TCP/53 i UDP/53, je definován v RFC1035. Servery DNS jsou organizovány hierarchicky, stejně jako jsou hierarchicky tvořeny názvy domén. Jména domén umožňují lepší orientaci lidem, adresy pro stroje jsou však vyjádřeny pomocí adres 32bitových (IPv4) A záznam nebo 128bitových (IPv6) - AAAA záznam. Systém DNS umožňuje efektivně udržovat decentralizované databáze doménových jmen a jejich překlad na IP adresy. Stejně tak zajišťuje zpětný překlad IP adresy na doménové jméno - PTR záznam.

Prostor doménových jmen tvoří strom. Každý uzel tohoto stromu obsahuje informace o části jména (doméně), které je mu přiděleno a odkazy na své podřízené domény. Kořenem stromu je tzv. kořenová doména, která se zapisuje jako samotná tečka. Pod ní se v hierarchii nacházejí tzv. *domény nejvyšší úrovně (Top-Level Domain, TLD)*. Ty jsou buď tematické (*com* pro komerci, *edu* pro vzdělávací instituce atd.) nebo státní (*cz* pro Českou republiku, *sk* pro Slovensko, *jo* pro Jordánsko atd.) Strom lze administrativně rozdělit do zón, které spravují jednotliví správci (organizace nebo i soukromé osoby), přičemž taková zóna obsahuje autoritativní informace o spravovaných doménách. Tyto informace jsou poskytovány autoritativním DNS serverem.
Výhoda tohoto uspořádání spočívá v možnosti zónu rozdělit a správu její části svěřit někomu dalšímu. Nově vzniklá zóna se tak stane autoritativní pro přidělený jmenný prostor. Právě možnost delegování pravomocí a distribuovaná správa tvoří klíčové vlastnosti DNS a jsou velmi podstatné pro jeho úspěch. Ve vyšších patrech doménové hierarchie platí, že zóna typicky obsahuje jednu doménu. Koncové zóny přidělené organizacím připojeným k Internetu pak někdy obsahují několik domén – například doména kdesi.cz a její poddomény vyroba.kdesi.cz, marketing.kdesi.cz a obchod.kdesi.cz mohou být obsaženy v jedné zóně a obhospodařovány stejným serverem.

DNS server může hrát vůči doméně (přesněji zóně, ale ve většině případů jsou tyto pojmy zaměnitelné) jednu ze tří rolí:

- **Primární server** je ten, na němž data vznikají. Pokud je třeba provést v doméně změnu, musí se editovat data na jejím primárním serveru. Každá doména má právě jeden primární server.
- **Sekundární server** je automatickou kopií primárního. Průběžně si aktualizuje data a slouží jednak jako záloha pro případ výpadku primárního serveru, jednak pro rozkládání zátěže u frekventovaných domén. Každá doména musí mít alespoň jeden sekundární server.

5.9 NTP

NTP (*Network Time Protocol*) je protokol pro synchronizaci vnitřních hodin počítačů po paketové síti s proměnným zpožděním. Tento protokol zajišťuje, aby všechny počítače v síti měly stejný a přesný čas. Byl obzvláště navržen tak, aby odolával následku proměnlivého zpoždění v doručování paketů.
NTP klient používá Marzullův algoritmus pro stanovení času z (nepatrně) se lišících odpovědí časových serverů. Používá se čas UTC se speciálními příznaky pro přestupné sekundy. NTP verze 4 obvykle dovede po internetu udržovat čas s chybou pod 10 milisekund (1/100 s), v lokální síti může při ideálních podmínkách dosáhnout přesnosti až 200 mikrosekund (1/5000 s).

Počítač, který chce synchronizovat své hodiny, pošle pár dotazů několika NTP serverům a ty mu v odpovědi pošlou svůj, přesný čas. Klient z odpovědí nejprve vyloučí servery se zřejmě nesmyslným časem (s odchylkou 1000 sekund a více). Poté ponechá skupinu serverů s největším společným průnikem.Běžně se jím dosahuje přesnosti hodin v řádu milisekund.

5.10 SMTP

Simple Mail Transfer Protocol (zkratka SMTP) je internetový protokol určený pro přenos zpráv elektronické pošty (e-mailů) mezi přepravci elektronické pošty (MTA). Protokol zajišťuje doručení pošty pomocí přímého spojení mezi odesílatelem a adresátem; zpráva je doručena do tzv. poštovní schránky adresáta, ke které potom může uživatel kdykoli (off-line) přistupovat (vybírat zprávy) pomocí protokolů POP3 nebo IMAP. Jedná se o jednu z nejstarších aplikací, původní norma RFC 821 byla vydána v roce 1982 (v roce 2001 ji nahradila novější RFC 2821). SMTP funguje nad protokolem TCP, používá port TCP/25.

Doručování elektronické pošty po Internetu se účastní tři druhy programů:

- MUA - *Mail User Agent*, poštovní klient, který zpracovává zprávy u uživatele
- MTA - *Mail Transfer Agent*, server, který se stará o doručování zprávy na cílový systém adresáta
- MDA - *Mail Delivery Agent*, program pro lokální doručování, který umísťuje zprávy do uživatelských schránek, případně je může přímo automaticky zpracovávat (ukládat přílohy, odpovídat, spouštět různé aplikace pro zpracování apod.)

5.10.1 Poštovní klient

Poštovní klient je program, který zajišťuje odesílání zpráv a vybírání schránek. Příkladem je např. Microsoft Outlook, Mozilla Thunderbird, Opera, Mutt, Pine a další. Je to v podstatě specializovaný editor, který umí kromě vytvoření zprávy také manipulovat se schránkami, odeslat zprávu nejbližšímu MTA a převzít zprávu ze serveru prostřednictvím POP3 nebo IMAP. Vlastním doručováním zprávy po síti až k adresátovi se klient nezabývá. Součástí klienta bývá také více či méně složitý adresář, který pomáhá uživateli udržet přehled o adresách.

5.10.2 Poštovní server

Poštovní server (MTA) běží obvykle jako démon a naslouchá na portu TCP/25. K tomuto portu se může připojit (navázat TCP spojení) buď poštovní klient, nebo jiný server, který předá zprávu k doručení. MTA zkontroluje, zda je zpráva určena pro systém, na kterém běží. Pokud ano, předá ji programu MDA (lokální doručení). Pokud je zpráva určena jinému počítači, naváže spojení s příslušným serverem a zprávu mu předá.
Při vyhledávání vzdáleného serveru, kterému má předat zprávu, musí MTA spolupracovat se systémem DNS. Od serveru DNS si vyžádá tzv. *MX záznam* pro cílovou doménu, který obsahuje IP adresu počítače, který se stará o doručení pošty v této doméně. Pokud DNS tento záznam neobsahuje, pokusí se poštovní server doručit správu přímo na počítač uvedený v adrese za zavináčem.
Poštovní server obsahuje v konfiguraci řadu parametrů, pomocí kterých můžeme mimo jiné nastavit, pro které domény MTA přijímá zprávy. Stejně tak je možné určit, od koho bude nebo nebude zprávy přijímat, což je velmi důležité z hlediska bezpečnosti a ochrany proti spamu.
Nejčastějšími programy v roli MTA jsou sendmail, postfix, exim, qmail, Microsoft Exchange, Mercury aj.

5.10.3 Program pro lokální doručování

Server by mohl zprávy do uživatelských schránek ukládat přímo, ale výhodnější je k tomu použít specializovaný program. To umožňuje při doručování ještě dále zprávy zpracovávat nebo filtrovat. Příkladem může být třídění zpráv do různých schránek uživatele podle obsahu (odesilatele, subjektu a pod.), nebo odstraňování nežádoucích zpráv (viry, spam). Tyto volby si může každý uživatel nastavit samostatně nezávisle na ostatních.
Typickými představiteli MDA jsou procmail a maildrop.

6. INICIALIZACE TCP/IP V DYNAMIC C

Tato kapitola popisuje konfigurační makra, datové struktury a funkce k použití pro konfigurování a inicializaci TCP/IP v prostředí Dynamic C. Dynamic C podporuje pouze IP protokol verze 4 ne verzi 6. TCP/IP protokol je naprogramován pomocí několika knihoven *.lib. Hlavní knihovna protokolu je *DCRTCP.lib*.

6.1 INICIALIZACE TCP/IP

Funkce *sock_init()* musí být volána na začátku hlavního bloku programu *main()* k tomu, aby byl inicializován TCP/IP protokol.
Návratová hodnota z funkce *sock_ init()* musí být 0 tedy značit úspěšnou inicializaci před voláním jiných funkcí TCP/IP.

Funkce *sock_init()* vykoná následující operace:

- Volání inicializace podprogramu pro ARP, TCP, UDP a DNS
- Test zda běžel *sock_init()* předtím jestliže ano pak vrací 0 jestliže ne tak následující kroky nejsou vykonané.
- Inicializace ovladače paketu
- Vymaže směrovače a jiné servery
- Když používáme Ethernet, čeká přibližně 1sekundu pro inicializaci hardwaru Ethernetu
- Rozhraní jsou inicializována pomocí nastavení v makru *IFCONGIG_**
- Když používáme *USE_DHCP* tak je konfigurace DHCP dokončená.

Jestliže všechno výše uvedené proběhlo úspěšně je návratová hodnota nastavená na 0. Jinak je návratová hodnota nenulová. Dále ještě můžeme ještě testovat jestli nám funkce nevrátila hodnotu 2 to by znamenalo, že DHCP selhal. Ostatní hodnoty signalizují, že síť není použitelná.

6.2 KONFIGURACE ROZHRANÍ

Konfigurace rozhraní je definováno pomocí několika maker jako například *MY_IP_ADDRESS* stejně jako voláním konfiguračních funkcí např. *sethostid()*.
Pro provoz TCP/IP na vývojovém počítači, host(počítač) potřebuje znát svou IP adresu a také masku. IP adresa a sítová maska jsou nedůležitější konfigurační položky stejně jako další konfigurační položky jsou potřebné pro správný provoz spojení. Pro jiné spojení než na místní síti je důležité taky znát adresu gatewaye(brány) či routeru(směšovače). Router má důležitý úkol zasílat zprávy mezi místním hostem a okolním světem(hosty, kteří nejsou na místní síti). Routery mají přiřazený zvláštní spojení. Každé spojení bude všeobecně požadovat odlišný router, nicméně ve většině případů jen jedno rozhraní bude použito pro spojení s nelokálními hosty tak i jen jeden router se bude starat o spojení hostů na nelokální síti. Některé z konfiguračních položek nejsou specifické pro žádné spojení. Například DNS(Domain Name Server) servery jsou známé jejich IP adresou. DNS servery jsou používané pro překlad jejich IP adres na čitelné adresy např. (www.seznam.cz).

6.2.1 Zdroje konfiguračních spojení

Dynamic C získává zdroje konfiguračních informací z následujících zdrojů:
- Předdefinovanou konfigurační knihovnu *tcp_config.lib*
- Makra definovaná před *#use "dcrtcp.lib"*; statická konfigurace
- Zavaděč síťových protokolů BOOTP a DHCP; dynamická konfigurace
- Volané runtime funkce například *ifconfig()* a *sethostid()*

6.2.2 Předdefinované konfigurace

Používání předdefinovaných maker má výhodu ve snižování počtu definicí maker na začátku každého TCP/IP programu, stejně jako vyloučení potřeby pro kopírování/vkládání hodně nastavení z jednoho programu do druhého. Používání předdefinovaných konfigurací je velmi snadné: stačí *#define* a jednotlivé makra na začátku(vrcholu) každého programu. Makro je definované jako celé číslo (int), které vybírá jedno z předdefinovaných konfigurací v *tcp_config.lib*.
Např:
#define TCPCONFIG 1
#use "dcrtcp.lib"

6.2.3 Statická konfigurace

Pro statickou konfiguraci máme dvě možnosti:

1. Rozhraní konfigurované pomocí sady maker včetně *dcrtcp.lib*. Nejběžnější makra jsou:
   MY_IP_ADDRESS, MY_NETMASK, MY_GATEWAY a MY_NAMESERVER.
   Např:
   #define MY_IP_ADDRESS "192.168.100."
   #define MY_NETMASK "255.255.255.0"
   #define MY_GATEWAY "192.168.1.1"

2. Rozhraní konfigurované použitím makra nazývána *IFCONFIG_* * , kde * je nahrazena jménem rozhraní. Např. *IFCONFIG_ETH0* pro první ethernetový port. *IFCONFIG_ALL* obsahuje konfigurační položky, které nejsou charakteristické pro žádné rozhraní.
   Hodnota makra *IFCONFIG_** je ve skutečnosti seznam položek v syntaktické formě jazyka C.
   Např:
   *#define IFCONFIG_ETH0 *
   *IFS_IPADDR, aton("192.168.1.100"), *
   *IFS_NETMASK, aton("255.255.255.0"), *
   *IFS_ROUTER_SET, aton("192.168.1.1"), *
   IFS_UP

6.2.3.1 IP Adresa nastavená manuálně

Informace, které potřebuje znát každý host na síti:

1. IP adresa hosta (vývojový počítač)
2. Část IP adresy, která rozlišuje počítače na hostitelské síti od počítačů na dalších sítí (Síťová maska)
3. IP adresu směrovače, který spojuje hostitelskou síť se „zbytkem světa“ (router)
4. IP adresu lokálního DNS serveru pro hostitelskou síť.

MY_IP_ADDRESS, MY_NETMASK, MY_GATEWAY a *MY_NAMESERVER* odpovídají těmto čtyřem bodům.

6.2.4 Dynamická konfigurace přes síť

TCP/IP protokol v Dynamic C podporuje DHCP (Dynamic Host Configuration Protocol), nebo BOOTP (Bootstrap Protocol) pro dynamickou konfiguraci. DHCP je modernější než BOOTP, který byl původně navržený k tomu, aby podporoval samozavádějící program u bezdiskových stanic. Použitím těchto protokolů můžeme úplně odstranit potřebu použití statické konfigurace.
Knihovna BOOTP.lib dovoluje aby stanice byla BOOTP nebo DHCP klient. Použitý protokol záleží na typu serveru, který je nainstalovaný na lokální síti. BOOTP a DHCP servery jsou obvykle centrálně instalovány na lokální síti a jsou obsluhovány administrátorem. Inicializace může trvat delší dobu při použití DHCP než při použití lokální konfigurace, ale to vše záleží na použitém serveru na místní sít.

Oba protokoly povolí zaslání konfiguračních parametrů do klienta a to:

- IP adresu klienta (IP_ADDRESS)
- Síťovou masku (NETMASK)
- Adresu brány (GATEWAY)
- DNS (DNS)
- Jména serverů (NAMESERVER)

BOOTP přiřadí stálé IP adresy. DHCP umí přiřadit IP adresu hostovi na omezenou dobu. Jsou tam dvě funkce přiřazení IP adresy hostovi a to: *dhcp_release()* a *dhcp_acquire()*.

dhcp_release() – Tato funkce opustí DHCP server. Toto dovolí serveru aby znovu použil adresu, která byla přidělena. Po volání této funkce je proměnná pro IP adresu nastavená na 0 a není možné volat jinou TCP/IP funkci, která vyžaduje platnou IP adresu. Normálně by *dhcp_release()* byl použitý na síti, kde je malý počet IP adres, ale je tam velké množství hostů, kteří potřebují ojedinělý přístup do místní sítě.

dhcp_acquire() – Tato funkce získává pronájem DHCP serveru, který ještě nebyl získaný, nebo vypršel čas pro získání, nebo se vzdalo používání *dhcp_release()*. Normálně jsou DHCP obnovené automaticky, ale jestli se spojení neobnoví včas, nebude TCP/IP protokol používaný. Když vyprší čas *tcp_tick()* vrátí 0 a proměnná pro IP adresu bude resetována na 0. Později může zkusit tato funkce získat IP adresu znovu.

BOOTP a DHCP můžou být použité jen na standardním rozhraní, které je specifikované hodnotou makra *IF_DEFAULT* . Jestliže používáme víc než jedno rozhraní, pak by jsme měli zajistit jeho správné nastavení.

Úspěšné použití DHCP konfigurace zajistí splnění následujících podmínek:
- *#define USE_DHCP* před *dcrtcp.lib*
- *IF_DEFAULT* – ukazuje na požadované rozhraní
- Definice *IFCONFIG_** makra, aby obsahoval *IFS_DHCP* parametry ID

Příklad použití DHCP serveru:
#define USE_DHCP
#define IF_DEFAULT 0
#define IFCONFIG_ETH0 IFS_DHCP, 1, IFS_UP
#use "dcrtcp.lib"

Můžeme také použít předdefinovanou konfiguraci číslo 3 v *tcp_config.lib*, která je DHCP.
#define TCPCONFIG 3
#use "dcrtcp.lib"

Tyto konfigurace používají veškerá makra potřebné pro DHCP nebo BOOTP. Samozřejmě musí být DHCP server dostupný na místní síti a musí být nastavený tak, aby obsahoval použité konfigurační volby.

6.2.5 Runtime konfigurace *ifconfig()*

ifconfig() udělá většinu práce za všechny konfigurační techniky. Například statická konfigurace přes (makro *IFCONFIG_**) v základu volá *ifconfig()* s nahrazenými specifickými parametry. *ifconfig()* vezme proměnný počet parametrů a tento seznam parametrů je ukončený zvláštním symbolem *IFS_END*.

Příklad použití *ifconfig()* pro statickou konfiguraci:
```
ifconfig(IF_ETH0, IFS_IPADDR, aton("192.168.1.100"),
        IFS_NETMASK, aton("255.255.255.0"),
        IFS_ROUTER_SET, aton("192.168.1.1"),
        IFS_UP,
       IFS_END);
```

ifconfig() je také možno použít k získání aktuální IP adresy:
longword ipaddr;
ifconfig(IF_ETH0, IFG_IPADDR, &ipaddr, IFS_END);
v proměnné *ipaddr* je aktuální IP adresa prvního ethernetového rozhraní.

První parametr *ifconfig()* je číslo rozhraní. Pro jisté nastavení, může být tento parametr *IF_ANY*, který použije nastavení pro všechny rozhraní. Následující parametry můžou být v libovolném množství. Tyto parametry musí být ukončeny zvláštním identifikátorem *IFS_END*.

6.2.6 MAC adresa

MAC adresa je prezentovaná jako sekvence šesti čísel oddělené dvojtečkami např: 00:01:02:03:04:05. MAC adresa nesouvisí s IP adresou. IP adresa identifikuje každého hosta na místní síti. MAC adresa identifikuje hardwarovou adresu každého zařízení. Mac adresy jsou jedinečné a jsou používané pro specifikaci Ethernetových adaptérů. Obvykle se MAC adresa používá když zařízení žádá o přidělení IP adresy DHCP serverem. DHCP server užívá MAC adresu k tomu, aby přiřadil „starou“ IP adresu stejnému zařízení.

6.3 DYNAMICKÉ STARTOVÁNÍ A ZASTAVENÍ SPOJENÍ

Dynamic C dovoluje individuálně přenést rozhraní nahoru nebo dolů pomocí užití funkcí *ifup(), ifdown()* nebo *ifconfig()*. Počáteční požadovaný stav rozhraní je nastavený v makru *IFCONFIG_**. Standardně rozhraní nejsou nastavená když je volána funkce *sock_init()* v bootovací době. Jen jestli makro *IFCONFIG_** obsahuje IFS_UP příkaz bude provedený v bootovací době.
Většina aplikací by neměla potřebovat dynamicky měnit status rozhraní. Výjimka může být u použití PPP přes sériovou linku, kde je modem užívaný při vytáčení.

6.3.1 Stav rozhraní

Jsou použité dvě funkce pro test aktuálního stavu rozhraní:
ifstatus() – jenom vrací booleovskou hodnotu zda je rozhraní „nahoru“. Jestliže je návratová hodnota nenulová, tak je rozhraní připravené na normální TCP/IP komunikaci. Jinak není rozhraní dostupný. To může být buď „dole“,nebo v procesu stoupání.
ifpending() – dává více informací: jeho návratová hodnota nejen signalizuje jeho stav, ale také jestli je v procesu výměny.
Pro registr volání funkce voláme *ifconfig()* s *IFS_IF_CALLBACK* jako identifikační parametr a adresu našeho volání jako hodnotu parametru.

6.3.2 Přenos rozhraní nahoru

Můžeme volat *ifup()*, nebo *ifconfig()* s identifikujícím parametrem *IFS_UP*. Výhoda používání *ifconfig()* je, že můžeme specifikovat počet rozhraní *IF_ANY*, který převádí všechny rozhraní nahoru.
Návrat z volání *ifup()* musí aplikace testovat dokončení používání funkcí.

6.3.3 Přenos rozhraní dolů

Můžeme volat *ifdown()*, nebo *ifconfig()* s identifikujícím parametrem *IFS_DOWN*. Výhoda používání *ifconfig()* je, že můžeme specifikovat počet rozhraní *IF_ANY*, který převádí všechny

7. ROZHRANÍ TCP A UDP SOCKET

TCP (Transmission Control Protocol) a UDP (User Datagram Protocol) jsou protokoly transportní vrstvy. TCP je používaný pro spolehlivý tok dat mezi dvěma hostiteli na síti. O protokolu UDP říkáme, že nedává záruky na datagramy, které přenáší mezi počítači v síti. Někdy je označován jako „nespolehlivý“, ale to je velmi zavádějící označení. Na rozdíl od protokolu TCP totiž nezaručuje, zda se přenášený datagram neztratí, zda se nezmění pořadí doručených datagramů nebo zda se některý datagram nedoručí vícekrát.
V Dynamic C se TCP/IP knihovny realizují přes IP (Internet protokol). TCP zahrnují další různé protokoly: HTTP protokol, SMTP protokol a FTP. Naproti tomu UDP zahrnují: DNS a SNMP.
TCP mají hodně složitých důležitých detailů, které jsou nutné k tomu, aby zajistili spolehlivý tok dat mezi hostiteli na síti navzdory možným chybám v síti, jako ztracené, nebo znovu poslané pakety. Například TCP bude automaticky znovu posílat data, dokud nebudou potvrzena příjemcem v dané platné době. TCP ale neposílá tolik dat, aby nepřetekl příjmový buffer u příjemce a nepřetěžuje routery v místní síti.
UDP naproti tomu tyto všechny detaily nepoužívá, nicméně má jiný výhody, které zas nemůže poskytnout TCP. Jedna výhoda je to, že může UDP posílat data více něž jednomu účastníkovi a další je, že chrání „záznamové hranice“, které mohou být užitečné pro některé aplikace.
TCP je spojově orientovaný protokol. Dva účastníci založí TCP spojení, které vytrvá do té doby než budou vzájemně uzavřené. Naproti tomu u UDP není žádná inicializace pro spojení. UDP může zaslat packet kdykoliv jakémukoliv cíli. Samozřejmě nemusí být cíl připraven přijmout UDP packety, proto s touto možností musí aplikace počítat.

7.1 SOKET

TCP socket představuje stav spojení mezi hostem a vzdáleným hostitelem. Když hovoříme o spojení, která jdou přes internet tak je socket popisovaný do 4 čísel: místní a vzdálené IP adresy (každá 32 bitů) a místní a vzdálené číslo portu (každé 16 bitů). Spojení, která nejdou přes internet (např. po místní LAN) jsou stále ojedinělý uvnitř sítě.
Z praktického hlediska – socket je struktura v RAM paměti, která obsahuje celou nezbytnou informaci o stavu. TCP sockety jsou značně rozsáhlejší oproti UDP socketům, protože je tam více informací, aby se udrželo spojení. TCP sockety taký vyžadují oba příjmový i vysílací buffer zatímco UDP sockety vyžadují jen příjmový buffer.
Každý socket má v Dynamic C přidružený *tcp_Socket* – struktura z 145 bytů, nebo *udp_Socket* – struktura z 62 bytů. Vstupně/Výstupní buffery jsou v přídavné paměti.

7.2 ČÍSLO PORTU

Oba TCP i UDP používají číslo portu. Číslo portu dovoluje, aby existovalo několik spojení mezi dvěma hosty na síti. Čísla portu jsou také používané známými protokoly – např. TCP port je 23 a použitý pro běžná telnet spojení. Obecně čísla portu pod 1024 jsou použité pro standardní služby. Porty mezi 1024 a 65536 jsou použité pro dočasné spojení. Často, jeden z tvůrců spojení vybere jedno z dočasných čísel portu pro konec spojení.
Když otevřeme socket použitím TCP/IP knihoven, můžeme specifikovat číslo portu, nebo můžeme použít knihovnu pro výběr dočasného čísla portu pro tzv. „krátká spojení“.

Některé ze známých portů jsou uvedeny v tabulce:

Číslo portu	Aplikace
20/21	FTP (File Transport Protokol)
23	Telnet
25	SMTP (Simple Mail Transfer Protocol)
53	DNS (Domain Name Server)
80	HTTP server

7.3 ALOKACE TCP A UDP SOCKETU

TCP a UDP socket struktury musí být alokovány ve statické paměti dat. To je jednoduše uděláno deklarací statické proměnné typu *tcp_Socket* nebo *udp_Socket:*
Např:
static tcp_Socket s;
static udp_Socket my_udp_s[20];

7.3.1 Alokace bufferu socketu

V Dynamic C jsou dvě makra definující počet dostupných socketů. Tyto makra neurčují kolik socketů můžeme přidělit, ale omezují kolik socketů můžeme úspěšně použít.
Významná makra jsou:

MAX_TCP_SOCKET_BUFFERS – určuje maximální množství TCP socketů s předalokovaným bufferem. Defaultně je 4. Buffer je přiřazený k socketu s prvním voláním *tcp_open()*, nebo *tcp_listen()*.

MAX_UDP_SOCKET_BUFFERS - určuje maximální množství UDP socketů s předalokovaným bufferem. Defaultně je 0. Buffer je přiřazený k socketu s prvním voláním *udp_open()*.

7.3.2 Velikost bufferu socketu

V Dynamic C jsou jsou I/O buffery pro TCP a UDP odděleně používány:

TCP_BUF_SIZE – Určuje velikost TCP bufferu. Defaultně 4096 bytů.
UDP_BUF_SIZE – Určuje velikost UDP bufferu. Defaultně 4096 bytů.

Jestliže *SOCK_BUF_SIZE* je deklarovaná, budou mít *TCP_BUF_SIZE* a *UDP_BUF_SIZE* její velikost.
Jestliže *SOCK_BUF_SIZE* není deklarovaná, ale je *tcp_MaxBufSize* pak budou mít *TCP_BUF_SIZE* a *UDP_BUF_SIZE* velikost *tcp_MaxBufSize* * 2.

7.4 OTEVŘENÍ TCP SOCKETU

Máme dva způsoby jak otevřít TCP socket a to: aktivní a pasivní.
Pasivní otevření znamená, že je socket dostupný pro dalšího hosta. Tento typ otevírání je obvyklý pro internetové servery, které „poslouchají“ na známých portech jako např. 80 pro HTTP server.
Aktivní otevření je použito, když založí počítač spojení s dalším hostem, který „poslouchá“ na specifickém portu. Toto je typicky použito, když je počítač jako klient pro jiný server.
Rozdíl mezi aktivním a pasivním otevíráním je ztraceno jakmile je spojení plně navázáno. Když je spojení navázáno oba hosté „operují“ na základě peer-to-peer. Rozdíl mezi tím kdo je klient a kdo je server je až v aplikaci. TCP sám o sobě nedělá rozdíly.

7.4.1 Pasivní otevírání

Pasivní otevírání socketu volá funkci *tcp_listen()* nebo *tcp_extlisten()*. Zásobujeme *tcp_listen()* ukazatelem na strukturu *tcp_Socket* , jiné číslo portu bude kontaktovat na našem zařízení IP adresu a číslo portu. Jestliže chceme být schopný přijímat z nějaké IP adresy, nebo nějakého čísla portu, nastavíme jedno nebo obojí na 0. Pro ovládání vícenásobných spojení, každé nové spojení bude požadovat vlastní *tcp_Socket* a oddělené volání k *tcp_listen()*, ale použití stejného čísla portu(*lport*).
tcp_listen() se ihned vrací a musíme zvolit přicházející spojení. Můžeme použít makro *sock_wait_established* , které bude volat *tcp_tick()* a blokuje dokud spojení není definitivní, nebo můžeme manuálně zvolit socket použitím *sock_established.*

7.4.2 Aktivní otevírání

Když internetový prohlížeč obnoví stránku, to je aktivně otevřené jedno nebo více spojení s web serverovými pasivně otevřenými sockety. Pro aktivní otevření socketu volá funkce *tcp_open()* nebo *tcp_extopen()* s použitými parametry, které jsou podobné při volání *tcp_listen()*. Je důležité dát přesné parametry pro *remip* a *port*, ale parametr *lport* muže být 0, který říká knihovně *DCRTCP.lib* aby vybrala nepoužitý port mezi 1024 a 65536.
Když voláme *tcp_open()*, Dynamic C zkouší kontaktovat i další zařízení k tomu aby založilo spojení. *tcp_open()* selže a vrátí 0 jestliže spojení nemohlo být otevřené kvůli potížím například se směrovačem, nebo problémem s hostitelskou hardwarovou adresou.

7.4.3 Čekání na navázání spojení

Když se otevře TCP socket ať už pasivně nebo aktivně, musí čekat na kompletní TCP spojení. Toto je metoda známá jako „3-way handshake“. Jak název naznačuje tak nejméně 3 pakety musí být vyměněny mezi komunikujícími. Po dokončení tohoto procesu, který zabírá jeden cyklus, je spojení plně navázáno takové, že může začít přesun dat v aplikaci.
Bohužel metoda „3-way handshake“ nemusí být vždy úspěšná: síť může být odpojená; jeden z komunikujících může spojení zrušit, nebo se mohl dokonce rozbít. Tyto možnosti musí být správně ošetřeny v naší aplikaci.

7.4.4 Zpoždění spojení

Občas je vhodné přijmout žádost spojení když zdroje dělají že nejsou dostupné. Toto se může stát s webovými servery které mají několik grafických obrazů a každý vyžaduje vlastní socket.
Makro *USE_RESERVEPORTS* je definovaný defaultně. To dovoluje použít funkci *tcp_reserveport()*. Když se nepodaří spojit s portem v *tcp_reserveport()*, metoda 3-way handshake je navrhnutá dokonce tak i když tam není socket dostupný. Toto je uděláno nastavením parametru v TCP hlavičce na 0. Význam tohoto je: „Já můžu vzít nyní 0 bytů dat“. Druhá strana spojení bude čekat, až hodnota parametru bude signalizovat, že data mohou být poslána.

7.5 FUNKCE TCP SOCKETU

7.5.1 Kontrolní funkce TCP socketu

Tyto funkce mění status socketu, nebo mění jeho I/O buffer:

- *sock_abort*
- *sock_close*
- *sock_flush*
- *sock_flushnext*
- *tcp_extlisten*
- *tcp_extopen*
- *tcp_listen*
- *tcp_open*

sock_close – toto volání ukončí spojení. Nemusí ihned ukončit spojení, protože může mít nějaký čas na to, poslat žádost k ukončení spojení a čekat na potvrzení. Jestliže si chceme být jistý, že je spojení kompletně zavřené zavoláme *tcp_tick()* se strukturovanou adresou socketu. Když *tcp_tick()* vrátí 0 je spojení kompletně zavřené.
sock_abort – toto volání zavře otevřené spojení. Tato funkce způsobí reset TCP a veškeré přijaté budoucí packety budou ignorovány.
Data nemusí být hned zaslána ze socketu do jeho cíle. Jestliže aplikace vyžaduje zaslání dat ihned, můžeme volat funkci *sock_flush()*. Tato funkce se pokusí zaslat data ihned.
Jestliže víme dopředu, že data musí být zaslána ihned, zavoláme funkci *sock_flushnext()*. Tato funkce způsobí zaslání souboru dat ihned a je účinnější než *sock_flush()*.

7.5.2 Stavové funkce TCP socketu

Tyto funkce vrací informace o stavu socketu, nebo jeho I/O bufferu.

- *sock_alive*
- *sock_bytesready*
- *sock_dataready*
- *sock_iface*
- *sock_rbleft*
- *sock_rbsize*
- *sock_tbleft*
- *sock_tbsize*
- *sock_tbused*
- *tcp_tick*

tcp_tick – řídí TCP/IP protokol, ale také vrací informaci o stavu. Když zásobíme *tcp_tick*() ukazatelem na *tcp_Socket*, tak za prvé bude zpracovávat packety a potom kontrolovat socket. *tcp_tick*() vrací 0 když je socket úplně zavřený. Můžeme použít tuto návratovou hodnotu po volání *sock_close* k určení jestli je socket úplně zavřený.

7.5.3 I/O funkce socketu

Tyto funkce ovládání všechny vstupy a výstupy pro socket

- *sock_aread*
- *sock_awrite*
- *sock_axread*
- *sock_fastread*
- *sock_fastwrite*
- *sock_getc*
- *sock_gets*
- *sock preread*
- *sock_putc*
- *sock_read*
- *sock_write*
- *sock_xfastread*
- *sock_xfastwrite*

Máme dva režimy psaní a zápis TCP socketů: ASCII a binárně. Standardně je socket otevřený v binárním módu, ale můžeme ho změnit voláním funkce *sock_mode()*.
sock_bytesready vrátí >0 jen když je kompletní řetězec v bufferu, nebo je buffer plný.
sock_puts() – funkce automaticky umístí znak na konec řetězce.

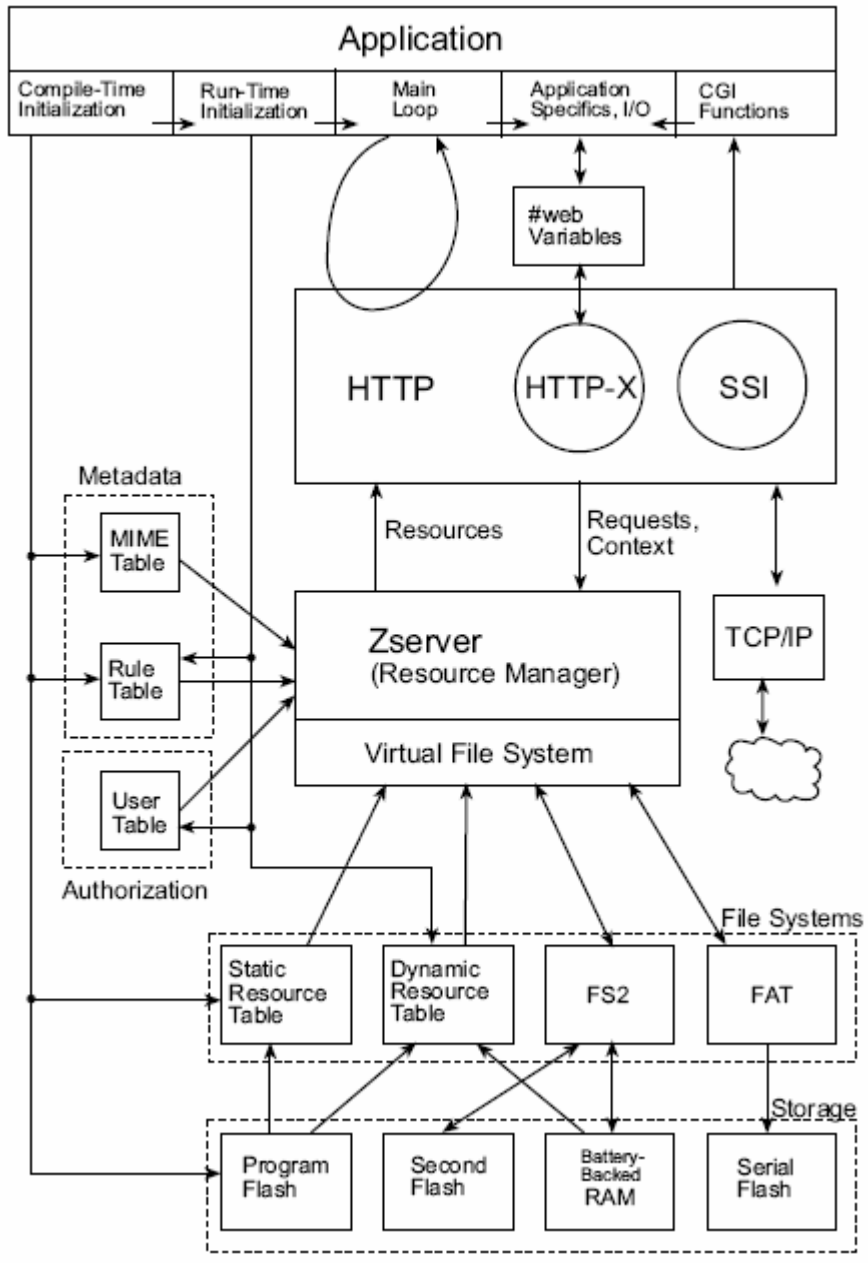
7.6 TCP/IP FUNKCE: *TCP_TICK()*

tcp_tick() je základní funkce pro TCP/IP knihovny. Má dvě užití: řídí udržování nejčerstvějších informací na pozadí; může být také užívaný k testování TCP socketu. Všimněme si, že *tcp_tick()* dělá více než TCP proces: je také nutný pro UDP a další vnitřní protokoly jako ARP nebo ICMP. Také ovládá status rozhraní.
Strojový čas, který potřebuje *tcp_tick()* se při každém volání mění. Hrubá čísla jsou: méně než milisekunda jestli tam není nic na zpracování, desítky milisekund pro typický zpracování packetu, a stovky milisekund pro výjimečnou situaci. Obecně čím více aktivních socketů, které se užívají součastně tím delší čas pro dokončení *tcp_tick()*, ale nebude se tolik zvětšovat pro rozumné použití množství socketů.
Je doporučené volat *tcp_tick()* v hlavní programové smyčce. Jestliže máme v aplikaci více čekajících smyček, měli by jsme zařídit, aby byl volán v každé z nich. TCP/IP funkční knihovny, které jsou vedené jako „blokovací“ jsou vždy zahrnuty v *tcp_tick()*.
Funkční knihovny, které jsou vedené jako „neblokovací“ (např. *sock_fastread()*) nejsou volány v tcp_tick() a musíme si na to dát pozor v naší aplikaci.
Některé z poskytujících aplikačních protokolů (jako např. HTTP a FTP) mají vlastní „tick“ funkce (jako *http_handler()* a *ftp_tick()*). Když voláme takovou funkci, není třeba volat *tcp_tick()*.

8. WEBOVÝ SERVER, HTTP SERVER

8.1 ARCHITEKTURA WEBOVÉHO SERVERU

Následující obrázek ukazuje všechny důležité části webové aplikace. Zdá se, že je tam velké množství součástí, ale všechny jsou potřeba pro naší webovou aplikaci.



Obr. 15. *Architektura webového serveru*

8.1.1 Aplikační blok

Na vrcholu tohoto schématu je blok nazývaný „Application“, který se skládá z 5 dílčích bloků. Aplikační blok představuje kód, kterou musíme v naší aplikaci vytvořit. Všechno pod tímto blokem je poskytnuto knihovnami, ačkoliv budeme potřebovat specifikovat některé části rozhraní k těmto blokům.

Aplikační blok se skládá:

1. Compile-time initialization – Tento blok zahrnuje věci jako výběr vhodných knihoven; inicializace statických(konstanty) datových struktur a tabulek; výběr konfigurace sítě a zahrnutí statických zdrojů(externích souborů) přes *#ximport* nebo *#zimport*.
2. Run-Time initialization – Hlavní *main()* funkce potřebuje zavolat specifické funkční knihovny než se začne vykonávat:

- *sock_init()* toto je důležitá funkce, která inicializuje TCP/IP
- *sspec_automount()* toto není povinné, inicializuje dostupný filesystem (FS2 nebo FAT) pro použití zdrojovým manažerem Zserver.
- *http_init()* toto je důležité, inicializuje HTTP server.

3. Main Loop - hlavní cyklus. Finální program v *main()* nepřetržitě volá funkci *http_handler()* a jiné možné funkce. Toto je důležité pro HTTP server, aby zpracoval žádosti ze sítě.
4. Application specifics, I/O – Je zde určitý počet možností, kterými může naše aplikace komunikovat s HTTP serverem. Aplikace může přímo volat funkce v HTTP serveru, ve zdrojovém manažeru (Zserver), v TCP/IP, nebo někde jinde. Jeden velmi dobrý způsob je přes rozhraní poskytnuté pomocí proměnných *#web*.
5. CGI Functions – CGI stojí za „běžným Gateway rozhraním“, nicméně tato zkratka má více specifických použití v Dynamic C – odkazuje na C funkci, která je volaná HTTP serverem k tomu, aby generovala dynamický obsah pro prohlížeč.

8.1.2 HTTP blok

Ve schématu je tento blok tvořen dvěma kruhy. Server je odpovědný za chytání žádostí z venku. Každá žádost je analyzovaná k tomu, aby se určil zdroj, uživatel který žádost zaslal a zad je uživatel oprávněný tento zdroj získat. Jestli je zdroj k dispozici, uživatel známý a má řádná oprávnění, pak je zdroj přenesen zpět do prohlížeče.
Malý blok s názvem *#web Varibles* mezi aplikací a HTTP-X ukazuje, že *#web* proměnné jsou dopravními prostředky mezi aplikací a serverem. *#web* proměnné jsou obyčejné C proměnné (pole nebo struktury).
Server také komunikuje s nižšími vrstvami ve schématu. Na pravé straně je TCP/IP blok – toto je cesta k vnějšímu světu. Obvykle server komunikuje s TCP/IP přes TCP socket.

8.1.3 Zserver blok

Přímo pod HTTP serverem je blok Zserver (Resource Manager). Toto je centrální jednotka celé aplikace. Ovládá přístup k mnoha dalším blokům ve schématu. I když je to centrální jednotka tak nepotřebujeme znát do podrobna jeho práci. Zserver může být použit také na další typy serverů jako například FTP, protože poskytuje stejné rozhraní k různým druhům zdrojů. Jak je uvedeno ve schématu, Zserver je navržený jako zdrojový manager a virtuální filesystém.

8.2 HTTP SERVER

HTTP (Hypertext Transfer Protocol) server dává k dispozici HTML (Hypertext Markup Language) klientům pomocí webového prohlížeče. HTTP server je implementovaný v *HTTP.lib* , proto musíme napsat na začátku našeho programu *#use „http.lib“*. HTTP závisí na připojení k síti, které je konfigurováno v knihovně *dcrtcp.lib*. Nastavení sítě je nezbytné pro použití HTTP. HTTP používá knihovny Zserveru k tomu, aby řídil zdroje a řídil přístup.

8.2.1 Datové struktury HTTP

Použití struktury *HttpState* je důležité pro CGI funkce. Ukazatel na *HttpState* je jen jeden parametr všem CGI funkcím. Po většinu času by měl být tento ukazatel předaný dalším HTTP funkčním knihovnám. Struktura *HttpState* platí jen uvnitř CGI funkce, která byla volána z HTTP serveru.

8.3 SERVEROVÉ KNIHOVNY

Zdrojový manager, *Zserver.lib*, obsahují struktury, funkce a konstanty k tomu, aby dovolily HTTP a FTP serverům sdílet data a uživatelské informace.
V základním příslušenství, které poskytuje Zserver je schopnost k tomu, aby přeložil zdrojová jména (URL případně HTTP) do odkazů na file systém a paměťové objekty. Zdroj odkazuje na objekty (soubory, funkce a proměnné) se kterými manipuluje Zserver za server. Podpora pro HTML je také zahrnuta v Zserver.lib.

8.3.1 Použité konstanty v Zserver.lib

Konstanty jsou přiřazené do polí struktur *ServerSpec* a *ServerAuth*.

8.3.1.1 Pole typu ServerSpec

Možné hodnoty jsou například:

SSPEC_XMEMFILE – Data sídlící v xmem
SSPEC_ROOTVAR – Proměnné data v root paměti (pro HTTP)
SSPEC_FUNCTION – Funkce (pro HTTP)
SSPEC_VARIABLE – Proměnná data (pro HTTP)
SSPEC_XMEMVAR – Proměnná data v root paměti (pro HTTP)

8.3.1.2 Makra pro řízení a inicializaci

Static MIME Type Table
Tato tabulka mapuje rozšířený název souboru a MIME typy. Potřebujeme takovou tabulku jestliže používáme server, který vyžaduje MIME typy. Současně to potřebuje jen HTTP server.

SSPEC_MIMETABLE_START
SSPEC_MIME(extension, type)
SSPEC_MIME_FUNC(extension, type, function)
SSPEC_MIMETABLE_END

Tato sekvence nastavuje mapovací tabulku MIME. Aktuálně je podporována jen statická MIME tabulka.

Statické zdrojová tabulka

Statická zdrojová tabulka přidruží jména webových zdrojů (souborů, funkcí a proměnných) k odkazům na paměťové objekty.

SSPEC_RESOURCETABLE_START
SSPEC_RESOURCE_XMEMFILE(name, addr)
SSPEC_RESOURCE_ROOTVAR(name, addr, type, format)
SSPEC_RESOURCE_FUNCTION(name, addr)
SSPEC_RESOURCETABLE_END

Např:
SSPEC_RESOURCETABLE_START
SSPEC_RESOURCE_XMEMFILE("/", index_html),
SSPEC_RESOURCE_XMEMFILE("/index.shtml", index_html),
SSPEC_RESOURCE_XMEMFILE("/rabbit1.gif", rabbit1_gif),
SSPEC_RESOURCE_ROOTVAR("dhcp", &dhcp, PTR16, "%s"),
SSPEC_RESOURCE_ROOTVAR("ipaddresa", ipaddresa, PTR16, "%s"),
SSPEC_RESOURCE_ROOTVAR("mac", &mac, PTR16, "%s"),
SSPEC_RESOURCE_FUNCTION("/OK.cgi", OK),
SSPEC_RESOURCE_FUNCTION("/refresh.cgi", refresh)
SSPEC_RESOURCETABLE_END

8.4 KONFIGURACE WEBOVÝCH STRÁNEK

8.4.1 Přidání souborů

Přidání souborů do kontroléru, aby sloužili jako webové stránky: *#ximport* a za ním cestu, kde jsou soubory umístěny.

Např:
#ximport "semestralka.html" index_html
#ximport "rabbit1.gif" rabbit1_gif

Přidání souboru s různými koncovkami:

SSPEC_MIMETABLE_START
SSPEC_MIME_FUNC(".shtml", "text/html", shtml_handler),
SSPEC_MIME(".html", "text/html"),
SSPEC_MIME(".gif", "image/gif"),
SSPEC_MIME(".cgi", "")
SSPEC_MIMETABLE_END

8.4.2 Dynamické proměnné na webové stránce

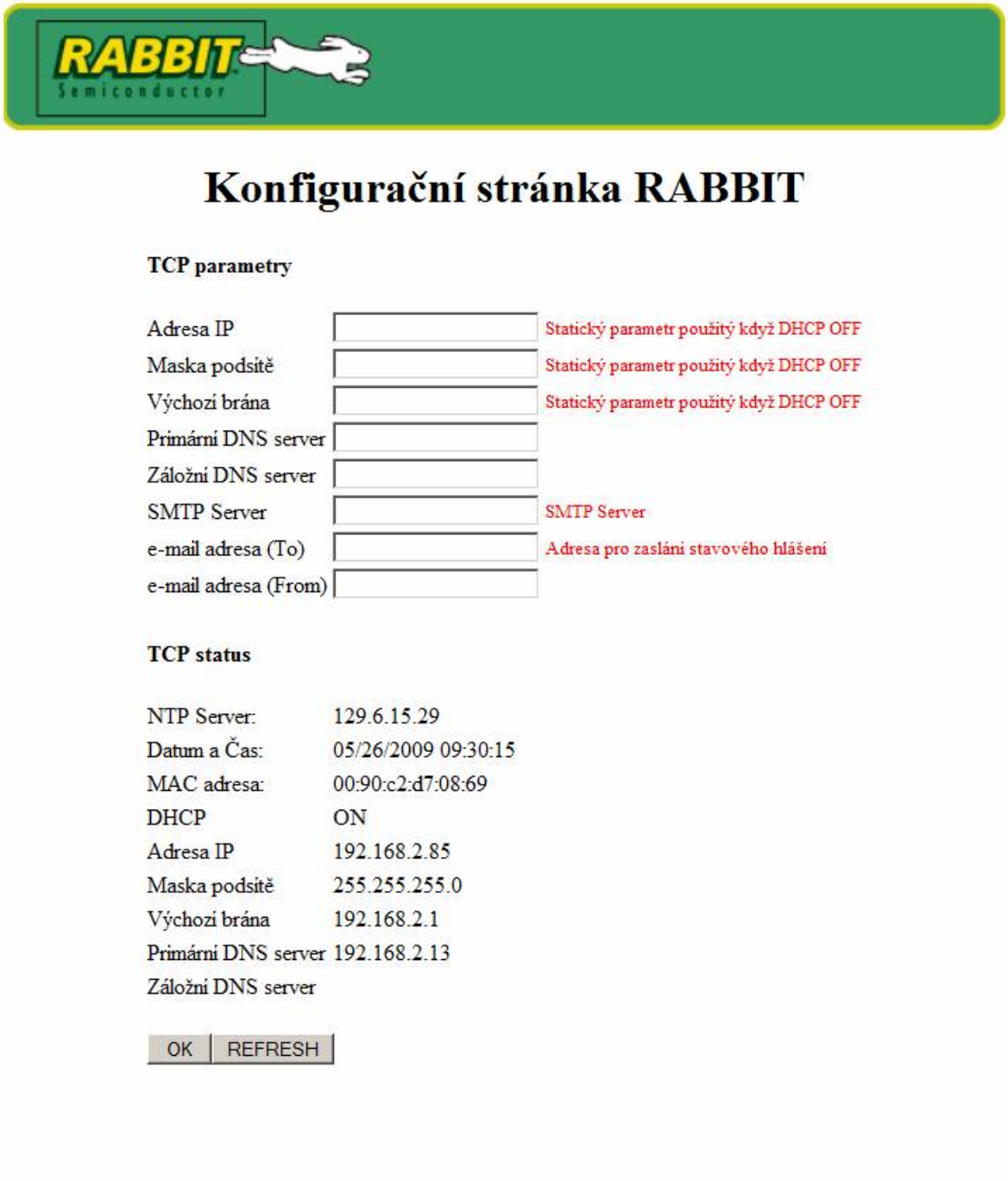
Pro dynamickou změnu proměnné na webové stránce se používá *#echo var*.

Např.
<td align="left">Adresa IP </td>
<td align="left"><!--#echo var="ip"--></td>
V shtml souboru je *<!--#echo var="ip"-->*nahrazen hodnotou proměnné s názvem *ip* ze statické zdrojové tabulky.

SSPEC_RESOURCETABLE_START
...
SSPEC_RESOURCE_ROOTVAR("ip", &ip, PTR16, "%s")
...
SSPEC_RESOURCETABLE_END

9. POPIS KONFIGURAČNÍ STRÁNKY

9.1 VZHLED KONFIGURAČNÍ STRÁNKY



Obr. 16. *Vzhled konfigurační stránky*

9.2 POPIS STRÁNKY

9.2.1 TCP parametry:

Adresa IP – Statický parametr, defaultní nastavení na 192.168.1.100
Maska podsítě – Statický parametr, defaultní nastavení na 255.255.255.0
Výchozí brána - Statický parametr, defaultní nastavení na 192.168.1.254
Primární DNS – Statický parametr defaultní nastavení na 192.168.1.101
NTP server – trvale nastaven na time-b.nist.gov (IP 129.6.15.29)
Časová zóna – trvale nastavena na +1 hodinu
SMTP Server – Zadání příslušného SMTP Serveru – trvale nastaven na smtp.seznam.cz
adresa adresáta, adresa odesílatele

9.2.2 TCP status:

Načtení nastavených parametrů včetně MAC adresy zařízení, která nelze na této konfigurační stránce měnit.

Server – Nastavený server pro synchronizaci času
Datum a Čas – Zobrazení datumu a času synchronizovaného NTP serverem
DHCP – signalizuje zda je DHCP zapnutý či vypnutý (ON/OFF)
Adresa IP – Zobrazí nastavenou či přidělenou IP adresu
Maska podsítě – Zobrazí nastavenou či přidělenou masku
Výchozí brána – Zobrazí nastavenou či přidělenou bránu (gateway)
Primární,Sekundární DNS server – Zobrazí nastavený či přidělený DNS server

9.2.3 Tlačítka OK, REFRESH

OK – Toto tlačítko slouží pro potvrzení nastavených statických parametrů, které se poté uložení do UserBlock a odešlou emailem.

REFRESH – Znovunačtení nastavených parametrů.

10. POPIS PROGRAMU

10.1 POPIS POUŽITÝCH MAKER

#define TCPCONFIG 1; #define TCPCONFIG 5 – Tyto makra definují zda se bude požívat statická či dynamická konfigurace použitím DHCP serveru. To ukazuje následující tabulka:
Možné hodnoty nastavení *TCPCONFIG*:

TCPCONFIG	Ethernet	PPP	DHCP	Runtime
1	ANO	NE	NE	NE
2	NE	ANO	NE	NE
3	ANO	NE	ANO	NE
4	ANO	ANO	NE	NE
5	ANO	NE	ANO	NE
6	ANO	NE	NE	ANO
7	ANO	NE	ANO	NE
8	NE	ANO	ANO	NE
9	ANO - wifi	NE	NE	NE
10	ANO - wifi	NE	ANO	NE

Jestliže budeme používat statickou konfiguraci zadáme *TCPCONFIG 1*, při použití DHCP serveru bude *TCPCONFIG 5*.

TCP_BUF_SIZE – popsáno v kap. 7.3.2
HTTP_MAXSERVERS 1 – maximální počet HTTP serverů na portu 80
MAX_TCP_SOCKET_BUFFERS 1 – určuje maximální množství TCP socketů v předalokovaném bufferu.
NIST_SERVER_IP - definuje server pro synchronizaci času
TIMEZONE – nastavení časového pásma

#define DEFAULTIP "192.168.1.100"
#define DEFAULTMASKA "255.255.255.0"
#define DEFAULTPNAMESERVER "192.168.1.100"
#define DEFAULTSNAMESERVER "192.168.1.102"
#define DEFAULTDOMAINNAME "192.168.1.101"
#define DEFAULTROUTER "192.168.1.1"
#define SMTP_SERVER "smtp.seznam.cz"

- defaultní nastavení počátečních adres jako ip, maska, dns, gateway

10.2 POPIS FUNKCÍ

10.2.1 Funkce *main()*

V hlavní funkci ***main()*** jsou postupně volány tyto funkce:

sock_init() – popsáno v kapitole 6.1
http_init() – inicializace HTTP serveru. Musí být volána po *sock_init()*, a před voláním hlavní smyčky *http_handler()*
tcp_reserveport(80) – tato funkce dovoluje, aby bylo spojení navázáno I když tam zatím není dostupný socket. Toto je vytvořeno nastavením parametru v hlavičce TCP během nastavovací fáze spojení, které signalizuje, že může být přijato 0 bytů.
init() – inicializace proměnných používaných na webové stránce
readflash() – načtení parametrů uložených v paměti.
usedhcp() – funkce, která řídí použití DHCP serveru
savepara() – funkce, která uloží parametry do paměti.
gettime() – funkce, která se připojí k serveru a získá aktuální datum a čas.
status() – tato funkce načte proměnné a aktualizuje je pro webovou stránku.

Hlavní cyklus:

http_handler() – Toto je základní řídící funkce pro HTTP server. Tato funkce musí být volána cyklicky, aby doména pracovala.
tcp_tick() – popsáno v kapitole 7.6

10.2.2 Funkce tlačítek OK a REFRESH

Tlačítko OK – Při zmáčknutí tohoto tlačítka se vyvolá funkce deklarovaná jako *int OK(HttpState* state)*. Při vykonávání této funkce se nejprve skenují proměnné na webové stránce, které se poté ukládají do pole struktur *FORMSpec*. Dále se testuje zda je použitý DHCP server. Jestliže není tak se začnou nastavovat síťové parametry (ip, maska, gateway…) použitím funkce *ifconfig()* Toto nastavení probíhá v cyklu, který obslouží všechny existující sítě. Po nastavení parametrů se tyto parametry uloží do paměti a zašlou se emailem na uvedenou adresu. Nakonec se vyvolá funkce *cgi_redirectto()*.

Tlačítko REFRESH – Při zmáčknutí tohoto tlačítka se vyvolá funkce deklarovaná jako *int refresh(HttpState* state)*, která dále vyvolá funkci status(). Nakonec se vyvolá funkce *cgi_redirectto()*. Funkce *status()* získá pomocí funkce *ifconfig()* parametry sítě spolu s mac adresou zařízení.

Popis použitých funkcí ve funkcích int OK(HttpState* state)* a *int refresh(HttpState* state)

ifconfig() – Tato funkce nastaví parametry sítě v runtimu. Navíc je možné touto funkcí získat parametry sítě a podporuje vícenásobná rozhranní. Libovolné množství parametrů může být nastaveno, nebo získáno v jednou volání.

Popis použitých parametrů při statické konfiguraci:
- IFS_IPADDR – nastavení IP adresy
- IFS_NETMASK – nastavení masky sítě
- IFS_ROUTER_SET – nastavení routeru
- IFG_HWA – získání mac adresy
- IFG_IPADDR – získání IP adresy
- IFG_NETMASK – získání síťové masky
- IFG_ROUTER_DEFAULT – získání routeru
- IFS_END – označuje konec parametrů

Popis použitých parametrů při aktivním DHCP:
- IFG_DHCP_INFO – získání DHCP informací
- IFS_DHCP – použití DHCP konfigurace
- IFS_DHCP_TIMEOUT – nastavení DHCP time-out v sekundách.
- IFG_DHCP_OK – nastavení DHCP v pořádku
- IFG_DHCP_FELLBACK – získání zda DHCP dovoluje stat. konfiguraci
- IFG_IPADDR – získání IP adresy
- IFG_NETMASK – získání síťové masky
- IFS_UP – vyšší rozhraní
- IFS_DOWN – nižší rozhraní

Dále je tato funkce popsána v kapitolách 6.2.5 a taká 6.3

cgi_redirectto(HttpState *state, char *url) – Tato funkce volá CGI funkce k tomu, aby přesměrovala uživatele na jinou stránku. Pošle uživatele na URL uloženém v parametru url. CGI funkce je považována za vyvolanou když jí zavoláme a bude v nedefinovaném stavu.
Parametry funkce:
- *State* – aktuální struktura serveru přijatá CGI funkcí
- *url* – adresa k přesměrování

10.3 POPIS FUNKCÍ ZASÍLÁNÍ EMAILU

V první části zasílání emailu se provede inicializace pomocí funkce *init_email()*. Tato funkce je volána poté co se zmáčkne tlačítko OK. Za inicializační funkcí dále následuje funkce zaslání emailu *SendMail()*.
Popis funkcí:
init_email() – V této funkci se nastaví do deklarované struktury – komu se má email zaslat, od koho je, předmět a požadovaná zpráva, která je sestavená před inicializací.

SendMail() – V této funkci začne posílání emailu pomocí funkce: ***smtp_sendmail(char *to, char *from, char *subject, char *message)*** – Tato funkce je použitá pro krátké zprávy, které jsou sestaveny před tím než jsou poslány. Za touto funkcí následuje cyklicky volaná funkce *smtp_mailtick()*.
smtp_mailtick(void) – Tato funkce je cyklicky volána dokud není email odeslán

Návratové hodnoty:
SMTP_SUCCESS – e-mail úspěšně odeslán
SMTP_PENDING - e-mail ještě není poslaný, opakování volání smtp_mailtick
SMTP_TIME - e-mail nebyl poslán během SMTP_TIMEOUT sekund.
SMTP_UNEXPECTED – dostal špatnou odezvu od SMTP server.
SMTP_ABORTED – transakce nepodařená

10.4 POPIS UKLÁDÁNÍ A ČTENÍ VNITŘNÍ FLASH PAMĚTI

10.4.1 Zápis do paměti Flash

Pro ukládání do vnitřní paměti Flash je použitá funkce:
int writeUserBlockArray(unsigned addr, void sources[], unsigned numbytes[] , int numsources)* – Uživatelský blok je chráněný před normálním zápisem na Flafh paměť. To je zpřístupněno pouze přes tuto funkci, nebo přes funkci w*riteUserBlock()*. Tato funkce zapíše sadu dat z paměti do uživatelského bloku. Jestliže jsou data zapsána v sousedících bytech používá se funkce w*riteUserBlock()*. Funkce *writeUserBlockArray()* se používá když jsou data zapsána v ne-sousedících bytech. To může být případ např. struktury dat sítě.

Parametry:
addr – offset adresa v paměti kam se má zapsat.
sources – pole ukazatelů kopírujících dat.
numbytes – pole bytů pro kopírování každého zdroje. Součet délek v tomto poli nesmí být delší jak 32767 bytů.
numsources – Počet datových zdrojů.

10.4.2 Čtení z paměti Flash

Pro čtení z paměti Flash je použita funkce:
*int readUserBlockArray(void *dests[], unsigned numbytes[], int numdests, unsigned addr)* – Čte byty z User block na primárním flash paměti do bufferu v root paměti. Tato funkce je užívaná jako opak *writeUserBlockArray()*.

Parametry:
dests – ukazatel na pole cíle, aby se odtud kopírovali data.
numbytes – pole bytů zapsaný každému cíli.
numdests – počet cílů.
addr – offset adresa v User Block ze které má číst.

10.5 POPIS SYNCHRONIZACE ČASU

Tato funkce je v programu nazvaná *gettime()*. Po příslušné deklaraci použitých proměnných následuje testování zda je aktivní nastavení přes DHCP server. Za tímto následují již funkce pro připojení na patřičný server zadaný v makru *NIST_SERVER_IP*.
Nyní je úprava IP adresy funkcí *resolve()*, která převede ip adresu zadanou jako znak na longword IP adresu.
Po této úpravě se volá funkce:
*int tcp_open(tcp_Socket *s, word lport, longword remip, word port, dataHandler_t datahandler)*
Tato funkce vytváří aktivní spojení s dalším zařízením. Po volání *tcp_open()* musí být volána funkce *sock_established()*(nebo makro *sock_wait_established()* dokud není spojení plně navázáno.

Parametry:
s – ukazatel na strukturu socketu
lport – lokální port, používá se 0 pro rozsah 1025 - 65536
remip – adresa na kterou se má připojit
port – číslo portu
datahandler – funkce volaná když jsou data přijmuta. NULL pro umístění dat v socketovém přijímacím bufferu.

*void sock_wait_established(void *s, int seconds, int (*fptr)(), int *status)* – Toto čeká dokud není spojení navázáno pro specifikovaný TCP socket, nebo se přerušuje jestli nastává time-out. Jestliže nastane chyba skáče na návěští *sock_err*.

Parametry:
s – ukazatel na socket
seconds – počet sekund kolik má čekat před time-outem. Obvykle je *sock_delay* 20s.
fptr – funkce opakovaně volána při čekání.
status – ukazatel na stavové slovo.

Další použitá funkce *void sock_wait_input(void *s, int seconds, int (*fptr)(), int *status)* a *int sock_gets(tcp_Socket *s, char *text, int len);*

*void sock_wait_input(void *s, int seconds, int (*fptr)(),int *status)*- Tato funkce čeká dokud existuje vstup pro funkce jako *sock_read()* nebo *sock_gets()*. Tato funkce se v ASCII módu vrátí jen pouze je-li plný řetězec, nebo je plný buffer.

Parametry:
s – ukazatel na socket
seconds – počet sekund kolik má čekat před time-outem. Obvykle je *sock_delay* 20s
fptr – funkce opakovaně volána při čekání
status – ukazatel na stavové slovo

*int sock_gets(tcp_Socket *s, char *text, int len)* – Tato funkce čte řetězec ze socketu, jestliže je řetězec delší než *len* tak je řetězec ukončený a zbývající znaky jsou vyřazeny.

Parametry:
s – ukazatel na socket
text – buffer na řetězec
len – maximální délka bufferu

10.6 SEZNAM POUŽITÝCH KNIHOVEN

```
#use "tcp_config.lib"
#use "dcrtcp.lib"
#use "http.lib"
#use smtp.lib
```

11. ZÁVĚR:

Cílem této bakalářské práce bylo vytvořit konfigurační stránku pro embedded zařízení s mikroprocesorem RABBIT3000, kde měla být možnost nastavovat IP adresa jak staticky tak dynamicky užitím DHCP serveru, maska podsítě, výchozí brána, DNS server, SMTP server (adresy odesílatele a příjemce), NTP server pro synchronizaci času a časové pásmo. Konfigurační stránka byla vyvíjena v PSPadu a software pro mikroprocesor RABBIT v prostředí Dymamic C. Program je napsán v jazyce ANCI C.

Nastavování parametrů staticky lze měnit bez problémů. Veškeré parametry se ukládají do UserBlock a poté jsou sestaveny do emailové zprávy, která je zaslaná na požadovanou adresu příjemce. Když chceme používat (nastavovat) statické parametry je nutné aktivovat makra *TCPCONFIG 1* a *DHCP 0* – jsou deaktivované a použité jako komentář. Při nastavování parametrů dynamicky (použitím DHCP serveru) se používají makra TCPCONFIG 5 a DHCP 1. Požadované parametry se získají bez problémů z DHCP serveru, na který jsme momentálně připojeni.
U této bakalářské práce ale také nastaly určité problémy:
1. problém je, když chceme synchronizovat čas při použití statických parametrů - nedaří se připojit na požadovaný server.
2. problém je u odeslání emailu (nelze měnit SMTP server – nastaven trvale na smtp.seznam.cz) – tento parametr se musí nastavit podle SMTP serveru na který jsme aktuálně připojeni.

LITERATURA

Dokumentace dodaná k vývojové desce:

[1] Z-WORLD RABBIT. *Wolf (BL2600) User's Manual* [CD]. DCRABBIT_9.21/Docs/manuals/BL2600/BL2600UM.pdf
[2] Z-WORLD RABBIT. *RabbitCore RCM 3200 User's Manual* [CD]. DCRABBIT_9.21/Docs/manuals/RCM3200/UsersManual/RC3200UM.pdf
[3] Z-WORLD RABBIT. *Dynamic C User's Manual* [CD]. DCRABBIT_9.21/Docs/manuals/DC/DCUserManual/DCPUM.pdf
[4] Z-WORLD RABBIT. *Dynamic C Function Reference Manual* [CD]. DCRABBIT_9.21/Docs/manuals/DynCFunctionReference/DynCFunRef.pdf
[5] Z-WORLD RABBIT. *An Introduction to TCP/IP* [CD]. DCRABBIT_9.21/Docs/manuals/TCPIP/Introduction/tcpintro.pdf
[6] Z-WORLD RABBIT. *Dynamic C TCP/IP User's Manual Volume 1* [CD]. DCRABBIT_9.21/Docs/manuals/TCPIP/UsersManualV1/tcpV1.pdf
Z-WORLD RABBIT. *Dynamic C TCP/IP User's Manual Volume 2* [CD].
[7] DCRABBIT_9.21/Docs/manuals/TCPIP/UsersManualV2/tcpV2.pdf

Internet:

[1] WIKIPEDIE SERVER. *Dynamic Host Configuration Protokol* [online]. http://cs.wikipedia.org/wiki/Dynamic_Host_Configuration_Protocol
[2] WIKIPEDIE SERVER. *Network Time Protokol* [online]. http://cs.wikipedia.org/wiki/Network_Time_Protocol
[3] WIKIPEDIE SERVER. *Simple Mail Transfer Protokol* [online]. http://cs.wikipedia.org/wiki/Simple_Mail_Transfer_Protocol
[4] WIKIPEDIE SERVER. *Embedded systém* [online]. http://cs.wikipedia.org/wiki/Embedded_syst%C3%A9m

Príloha A - Bakalarska prace.c

```
#class auto

//#define TCPCONFIG     1     //Statická konfigurace
//#define DHCP 0              // DHCP OFF
#define TCPCONFIG    5        //Dynamická konfigurace použití DHCP
#define DHCP 1                // DHCP ON

#define USE_IF_CALLBACK
#define TCP_BUF_SIZE 2048
#define HTTP_MAXSERVERS 1
#define MAX_TCP_SOCKET_BUFFERS 1

#define NIST_SERVER_IP "129.6.15.29"    //NTP server
#define NIST_PORT       13
#define TIMEZONE        1               //Časová zóna

#define DEFAULTIP "192.168.2.85"            // -
#define DEFAULTMASKA "255.255.255.0"        //  \
#define DEFAULTPNAMESERVER "192.168.2.85"   //   \
#define DEFAULTSNAMESERVER "192.168.1.102"  //     -  defaultní nastavení
#define DEFAULTDOMAINNAME "192.168.1.101"   //   /
#define DEFAULTROUTER "192.168.2.1"         //  /
#define SMTP_SERVER "smtp.seznam.cz"        // -

#define DELKA 20
#define MAX_LENGTH_EMAIL 100
#define LENGTH_URL 100

#use "tcp_config.lib"
#ifndef DHCP_CLASS_ID
   #define DHCP_CLASS_ID "Rabbit-TCPIP:Z-World:DHCP-Test:1.2.0"
#endif
#ifndef DHCP_CLIENT_ID_MAC
   #define DHCP_CLIENT_ID_MAC
#endif

#memmap xmem
#use "dcrtcp.lib"
#use "http.lib"
#use smtp.lib

#ximport "web.html"  index_html
#ximport "rabbit1.gif"       rabbit1_gif

SSPEC_MIMETABLE_START
   SSPEC_MIME_FUNC(".shtml", "text/html", shtml_handler),
   SSPEC_MIME(".html", "text/html"),
   SSPEC_MIME(".gif", "image/gif"),
   SSPEC_MIME(".cgi", "")
SSPEC_MIMETABLE_END

//---------- Deklarace proměnných ----------
char gateway[DELKA],dns[DELKA],secdns[DELKA],hwmac[DELKA],ip[DELKA];
char hostname[DELKA],dns2[DELKA],email[DELKA],smtpserver[DELKA],dhcp[DELKA];
char ipaddresa[DELKA],maska[DELKA];
char dhcpgateway[DELKA],dhcpdns[DELKA],dhcpip[DELKA],dhcpmaska[DELKA];
char ntpserver[DELKA],timezone[DELKA],server[30],datetime[30];
char text[MAX_LENGTH_EMAIL];
char url[LENGTH_URL];
int i,j;
longword a,b,c,d,e,aa,bb,cc,dd,ee;          // testovací proměnné
char m[20],n[20],o[20],p[20],q[20],r[20];   // testovací proměnné
```

```
//---------------------------------------------

tcp_Socket s;

unsigned int save_lens[5];
void* save_data[5];

#define MAX_FORMSIZE 64

typedef struct {                //
   char *name;                  //
   char value[MAX_FORMSIZE];    // - - dekarace proměnných stránky
} FORMType;                     //
FORMType FORMSpec[8];           //

typedef struct {                //
   char *from;                  //
   char *to;                    //
   char *subject;               // - - deklarace proměnných emailu
   char *body;                  //
}Email;                         //
Email emailArray[1];            //

void init()                     // inicializace proměnných
{
word iface;
 strcpy(dhcp,"ON");
 strcpy(ip,"");
 strcpy(maska,"");
 strcpy(gateway,"");
 strcpy(dns,"");
 strcpy(dns2,"");

for (iface = 0; iface < IF_MAX; ++iface)
        {
         ifconfig(iface,
                 IFS_IPADDR, aton(DEFAULTIP),
                 IFS_NETMASK, aton(DEFAULTMASKA),
                 IFS_ROUTER_SET, aton(DEFAULTROUTER),
                 IFS_END);
        }

 setdomainname(DEFAULTDOMAINNAME);

 FORMSpec[0].name = "ipaddresa";
 FORMSpec[1].name = "subnetmask";
 FORMSpec[2].name = "gateway";
 FORMSpec[3].name = "primdns";
 FORMSpec[4].name = "secdns";
 FORMSpec[5].name = "smtpserver";
 FORMSpec[6].name = "emailto";
 FORMSpec[7].name = "emailfrom";
 FORMSpec[0].value[0] = '\0';
 FORMSpec[1].value[0] = '\0';
 FORMSpec[2].value[0] = '\0';
 FORMSpec[3].value[0] = '\0';
 FORMSpec[4].value[0] = '\0';
 FORMSpec[5].value[0] = '\0';
 FORMSpec[6].value[0] = '\0';
 FORMSpec[7].value[0] = '\0';
}
```

```
void init_email(void)                                  //inicializace emailu
{
   emailArray[0].from = FORMSpec[7].value;
   emailArray[0].to = FORMSpec[6].value;
   emailArray[0].subject = "Server RABBIT";
   emailArray[0].body = text;
}


void readflash()                        //načtení proměnných z UserBlock
{
word iface;
readUserBlockArray(save_data, save_lens, 5, 0);

for (iface = 0; iface < IF_MAX; ++iface)
        {
         if(strcmp(ip,"\0"))
          {
           ifconfig(iface,
                  IFS_IPADDR, aton(ip),
                  IFS_END);
          }
         if(strcmp(maska,"\0"))
          {
           ifconfig(iface,
                  IFS_NETMASK, aton(maska),
                  IFS_END);
          }
         if(strcmp(gateway,"\0"))
          {
           ifconfig(iface,
                  IFS_ROUTER_SET, aton(gateway),
                  IFS_END);
          }
        }
if(strcmp(dns,"\0"))
 {
  setdomainname(dns);
 }
return;
}

void SendMail()                       //Odeslání emailu
{
smtp_sendmail(emailArray[0].to,       emailArray[0].from,
              emailArray[0].subject, emailArray[0].body);

   // čekání až bude email odeslán
   while(smtp_mailtick()==SMTP_PENDING)
      continue;
   // Uspěšné odeslání emailu
   if(smtp_status()==SMTP_SUCCESS)
   {
      printf("Email odeslán");
   }
   else
   {
      printf("Chyba odeslání emailu");
   }
}
```

```
int parse_post(HttpState* state)          //skenování proměnných stránky
{
   auto int retval;
   auto int i;
   retval = sock_aread(&state->s, state->p,
                        (state->content_length < HTTP_MAXBUFFER-1)?
                        (int)state->content_length:HTTP_MAXBUFFER-1);
   if (retval < 0)
    {
     return 1;
    }
   state->subsubstate += retval;
   if (state->subsubstate >= state->content_length)
    {
     state->buffer[(int)state->content_length] = '\0';
     for(i=0; i<(sizeof(FORMSpec)/sizeof(FORMType)); i++)
      {
 http_scanpost(FORMSpec[i].name, state->buffer, FORMSpec[i].value,MAX_FORMSIZE);
      }
      return 1;
    }
   return 0;
}

void status()                        //tisk statusu na stránku
{
 word iface;
 for (iface = 0; iface < IF_MAX; ++iface)
          {
           ifconfig(iface,
                    IFG_HWA, hwmac,
                    IFG_IPADDR, &a,
                    IFG_NETMASK, &b,
                    IFG_ROUTER_DEFAULT, &c,
                    IFS_END);
          }


sprintf(hwmac,"%02x:%02x:%02x:%02x:%02x:%02x", hwmac[0],hwmac[1],
        hwmac[2],hwmac[3],hwmac[4],hwmac[5]);

if(!DHCP)
{
 strcpy(dhcp,"OFF");
 inet_ntoa(ip,a);
 inet_ntoa(maska,b);
 inet_ntoa(gateway,c);
}
}



void print_results(int iface, int up)        //tisk parametrů přiřazených DHCP
{                                            //serverem
   auto word dhcp_ok, dhcp_fb;
   auto DHCPInfo * di;
   auto long myip;
   auto long mynetmask;
   auto int i;

   if (ifconfig(iface,
```

```
            IFG_DHCP_INFO, &di,
            IFG_DHCP_OK, &dhcp_ok,
            IFG_DHCP_FELLBACK, &dhcp_fb,
            IFG_IPADDR, &myip,
            IFG_NETMASK, &mynetmask,
            IFS_END)
         || !di) {
         printf("No DHCP info obtained!\n");

   }
   printf("Network Parameters:\n");
   inet_ntoa(dhcpip,myip);
   strcpy(ip,dhcpip);
   inet_ntoa(dhcpmaska,mynetmask);
   strcpy(maska,dhcpmaska);
   printf("  My IP Address = %s\n", dhcpip);
   printf("  Netmask =       %s\n", dhcpmaska);

   #if DHCP_NUM_ROUTERS
      printf("  Routers:        ");
      for (i=0; i < DHCP_NUM_ROUTERS; ++i)
       {
        inet_ntoa(dhcpgateway,di->router[i]);
        printf("%s",dhcpgateway);
        strcpy(gateway,dhcpgateway);
       }
      printf("\n");
   #endif
   #if DHCP_NUM_DNS
      printf("  DNS servers:    ");
      for (i=0; i < DHCP_NUM_DNS; ++i)
       {
        inet_ntoa(dhcpdns,di->dns[i]);
        printf("%s",dhcpdns);
        strcpy(dns,dhcpdns);
       }
      printf("\n");
   #endif
      ip_print_ifs();
      router_printall();
}

//---------- připojení na NTP server a získání času ---------

void gettime()
{
int i,dst,health,stat;
char buff[2048];
struct tm  t;
unsigned long  longsec;
longword ipadr;
strcpy(datetime,"Aktivni jen pri DHCP");
strcpy(server,"Aktivni jen pri DHCP");

if(DHCP)            // Test zapnutí DHCP
{
sock_init();

while (ifpending(IF_DEFAULT) == IF_COMING_UP)
 {
  tcp_tick(NULL);
 }
```

```
ipadr=resolve(NIST_SERVER_IP);                          //
tcp_open(&s, 0, ipadr, NIST_PORT, NULL);                // Připojení k serveru
sock_wait_established(&s, 0, NULL, &stat);              //
sock_mode(&s, TCP_MODE_ASCII);                          //

while (tcp_tick(&s))
 {
  sock_wait_input(&s, 0, NULL, &stat);                 //
  sock_gets(&s, buff, 48);                             // Získání dat
 }

sock_err:
if (status == -1)
 {
      printf("\nConnection timed out, exiting.\n");
 }
   if (status != 1)
   {
      printf("\nUnknown sock_err (%d), exiting.\n", status);
   }

sock_close(&s);

// převedení času do formy pro tisk

   t.tm_year = 100 + 10*(buff[6]-'0') + (buff[7]-'0');
   t.tm_mon  = 10*(buff[9]-'0') + (buff[10]-'0');
   t.tm_mday = 10*(buff[12]-'0') + (buff[13]-'0');
   t.tm_hour = 10*(buff[15]-'0') + (buff[16]-'0');
   t.tm_min  = 10*(buff[18]-'0') + (buff[19]-'0');
   t.tm_sec  = 10*(buff[21]-'0') + (buff[22]-'0');
   dst       = 10*(buff[24]-'0') + (buff[25]-'0');
   health    = buff[27]-'0';

   longsec = mktime(&t);
   longsec += 3600ul * TIMEZONE;     // nastavení časové zóny
   if (dst != 0)
      longsec += 3600ul;

   mktm(&t, longsec);
   printf("Current time:  %02d:%02d:%02d  %02d/%02d/%04d\n",
            t.tm_hour, t.tm_min, t.tm_sec,
            t.tm_mon, t.tm_mday, 1900 + t.tm_year);

   strcpy(server,NIST_SERVER_IP);

   sprintf(datetime,"%02d/%02d/%04d    %02d:%02d:%02d",t.tm_mon, t.tm_mday,
   1900 + t.tm_year,t.tm_hour, t.tm_min, t.tm_sec);

 }
}
//-----------------------------------------------------

void savepara()
{
if(DHCP)
 {
  save_data[0] = &dhcp;                                 //
  save_lens[0] = sizeof(dhcp);                          //
  save_data[1] = &ip;                                   //
  save_lens[1] = sizeof(ip);                            //
```

```
  save_data[2] = &maska;                              //
  save_lens[2] = sizeof(maska);                       // - - uložení proměnných do
  save_data[3] = &gateway;                            //     UserBlock
  save_lens[3] = sizeof(gateway);                     //
  save_data[4] = &dns;                                //
  save_lens[4] = sizeof(dns);                         //
  writeUserBlockArray(0, save_data, save_lens, 5);//
 }
}


void usedhcp()
{
word iface;
DHCPInfo *di;
#define DTIMEOUT 20
if(DHCP)                // Test zapnutí DHCP
 {
       for (iface = 0; iface < IF_MAX; iface++)
         {
          if (!is_valid_iface(iface))
            continue;
          ifconfig(iface,
               IFS_DOWN,
               IFG_DHCP_INFO, &di,
               IFS_UP,
               IFS_END);
          ifconfig(iface,
               IFS_IF_CALLBACK, print_results,
               IFS_DHCP, di != NULL,
               IFS_DHCP_TIMEOUT, DTIMEOUT,
               IFS_END);
         }
 }

}

int refresh(HttpState* state)               //znovu načtení proměnných
{                                           //a tisk na stránku
status();
sprintf(url,"http://" "%s" "/index.shtml",ip);
cgi_redirectto(state,url);
return 0;
}


int OK(HttpState* state)                //nastavení parametrů na stránce
{

word iface;
DHCPInfo *di;

 if(parse_post(state))
  {
   if(!DHCP)                          // Test zapnutí DHCP
    {
      {
       for (iface = 0; iface < IF_MAX; iface++)
        {
          if(strcmp(FORMSpec[0].value,"\0"))
           {
            ifconfig(iface,
```

```
                IFS_IPADDR, aton(FORMSpec[0].value),
                IFS_END);
          }
         if(strcmp(FORMSpec[1].value,"\0"))
          {
           ifconfig(iface,
                IFS_NETMASK, aton(FORMSpec[1].value),
                IFS_END);
          }
         if(strcmp(FORMSpec[2].value,"\0"))
          {
           ifconfig(iface,
                IFS_ROUTER_SET, aton(FORMSpec[2].value),
                IFS_END);
          }
       }

      if(strcmp(FORMSpec[3].value,"\0"))
       {
        setdomainname(FORMSpec[3].value);
       }
     }
   }

  else
   {
    if(DHCP)                    // Test zapnutí DHCP
     {
    #define DTIMEOUT 20

      for (iface = 0; iface < IF_MAX; iface++)
        {
         if (!is_valid_iface(iface))
           continue;
         ifconfig(iface,
              IFS_DOWN,
              IFG_DHCP_INFO, &di,
              IFS_UP,
              IFS_END);
         ifconfig(iface,
              IFS_IF_CALLBACK, print_results,
              IFS_DHCP, di != NULL,
              IFS_DHCP_TIMEOUT, DTIMEOUT,
              IFS_END);
        }
      }
   }
}
status();

save_data[0] = &dhcp;                           //
save_lens[0] = sizeof(dhcp);                    //
save_data[1] = &ip;                             //
save_lens[1] = sizeof(ip);                      //
save_data[2] = &maska;                          //
save_lens[2] = sizeof(maska);                   // -  uložení proměnných
save_data[3] = &gateway;                        //    do UserBlock
save_lens[3] = sizeof(gateway);                 //
save_data[4] = &dns;                            //
save_lens[4] = sizeof(dns);                     //
writeUserBlockArray(0, save_data, save_lens, 5);//
```

```
// ----- Sestavení textu emailu -----

sprintf(text,"DHCP: %s; IP adresa: %s; Maska podsítě: %s; Brána: %s; DNS: %s; ",
        dhcp, ip, maska, gateway, dns);
init_email();                       // Inicializace e-mailu
SendMail();                         // Odeslání e-mailu

// ---------------------------------

sprintf(url,"http://" "%s" "/index.shtml" ,ip);
cgi_redirectto(state,url);
return 0;
}

SSPEC_RESOURCETABLE_START
   SSPEC_RESOURCE_XMEMFILE("/", index_html),
   SSPEC_RESOURCE_XMEMFILE("/index.shtml", index_html),
   SSPEC_RESOURCE_XMEMFILE("/rabbit1.gif", rabbit1_gif),
   SSPEC_RESOURCE_ROOTVAR("server", &server, PTR16, "%s"),     //
   SSPEC_RESOURCE_ROOTVAR("datetime", &datetime, PTR16, "%s"),
   SSPEC_RESOURCE_ROOTVAR("hwmac", &hwmac, PTR16, "%s"),      //
   SSPEC_RESOURCE_ROOTVAR("dhcp", &dhcp, PTR16, "%s"),        //
   SSPEC_RESOURCE_ROOTVAR("ip", &ip, PTR16, "%s"),            //  proměnné pro
   SSPEC_RESOURCE_ROOTVAR("maska", &maska, PTR16, "%s"),      //  Status web.
   SSPEC_RESOURCE_ROOTVAR("gateway", &gateway, PTR16, "%s"),  //  web. stránky
   SSPEC_RESOURCE_ROOTVAR("dns", &dns, PTR16, "%s"),          // /
  SSPEC_RESOURCE_ROOTVAR("dns2", &dns2, PTR16, "%s"),        //
   SSPEC_RESOURCE_FUNCTION("/OK.cgi", OK),                   // Tlačítko OK
   SSPEC_RESOURCE_FUNCTION("/refresh.cgi", refresh)          // Tlačítko REFRESH
SSPEC_RESOURCETABLE_END

void main(void)
{
  gettime();
  sock_init();
  http_init();
  tcp_reserveport(80);
  init();
  readflash();
  usedhcp();
  savepara();
  status();

   while (1)
   {
    http_handler();
    tcp_tick(NULL);
   }
}
```

Priloha B - web.html

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
<title>Konfigurační stránka RABBIT</title>
</head>
<body bgcolor="White">
<center>
  <img SRC="rabbit1.gif" width="50%">
  <br>
</center>
    <H1 align="center" >Konfigurační stránka RABBIT</H1>
    <FORM ACTION="OK.cgi" METHOD="POST">
        <table width="100%" height="30%" bordercolor="white" >
          <td align="center" bgcolor="white">
           <table bordercolor="white">
              <tr>
                <td align="left"><b>TCP parametry</b><br><br></td>
              </tr>
              <tr>
                <td align="left">Adresa IP </td>
                <td align="left"><INPUT TYPE="TEXT" NAME="ipaddresa" SIZE="20"></td>
                <td align="left" style="color: #ff0000;"><small>Statický parametr
použitý když DHCP OFF</small></td>
              </tr>
              <tr>
                <td align="left">Maska podsítě</td>
                <td align="left"><INPUT TYPE="TEXT" NAME="subnetmask" SIZE="20"
MAXLENGTH="15" VALUE=""></td>
                <td align="left" style="color: #ff0000;"><small>Statický parametr
použitý když DHCP OFF</small></td>
              </tr>
              <tr>
                <td align="left">Výchozí brána</td>
                <td align="left"><INPUT TYPE="TEXT" NAME="gateway" SIZE="20"
MAXLENGTH="15" VALUE=""></td>
                <td align="left" style="color: #ff0000;"><small>Statický parametr
použitý když DHCP OFF</small></td>
              </tr>
              <tr>
                <td align="left">Primární DNS server</td>
                <td align="left"><INPUT TYPE="TEXT" NAME="primdns" SIZE="20"
MAXLENGTH="15" VALUE=""></td>
              </tr>
              <tr>
                <td align="left">Záložní DNS server</td>
                <td align="left"><INPUT TYPE="TEXT" NAME="secdns" SIZE="20"
MAXLENGTH="15" VALUE=""></td>
              </tr>
              <tr>
                <td align="left">SMTP Server</td>
                <td align="left"><INPUT TYPE="TEXT" NAME="smtpserver" SIZE="20"
MAXLENGTH="30" VALUE=""></td>
                <td align="left" style="color: #ff0000;"><small>SMTP
Server</small></td>
              </tr>
              <tr>
                <td align="left">e-mail adresa (To)</td>
                <td align="left"><INPUT TYPE="TEXT" NAME="emailto" SIZE="20"
MAXLENGTH="30" VALUE=""></td>
                <td align="left" style="color: #ff0000;"><small>Adresa pro zaslání
stavového hlášení</small></td>
```

```
              </tr>
              <tr>
                <td align="left">e-mail adresa (From)</td>
                <td align="left"><INPUT TYPE="TEXT" NAME="emailfrom" SIZE="20"
MAXLENGTH="30" VALUE=""></td>
              </tr>
              <tr>
                <td><br></td>
              </tr>
              <tr>
                <td align="left"><b>TCP status</b><br><br></td>
              </tr>
              <tr>
                 <td align="left">NTP Server:</td>
                 <td align="left"><!--#echo var="server"--><br></td>
              </tr>
               <tr>
                 <td align="left">Datum a Čas:</td>
                 <td align="left"><!--#echo var="datetime"--><br></td>
              </tr>
              <tr>
                 <td align="left">MAC adresa:</td>
                 <td align="left"><!--#echo var="hwmac"--><br></td>
              </tr>
              <tr>
                 <td align="left">DHCP </td>
                 <td align="left"><!--#echo var="dhcp"--></td>
              </tr>
              <tr>
               <td align="left">Adresa IP </td>
               <td align="left"><!--#echo var="ip"--></td>
              </tr>
              <tr>
               <td align="left">Maska podsítě</td>
               <td align="left"><!--#echo var="maska"--></td>
              </tr>
              <tr>
               <td align="left">Výchozí brána</td>
               <td align="left"><!--#echo var="gateway"--></td>
              </tr>
              <tr>
               <td align="left">Primární DNS server</td>
               <td align="left"><!--#echo var="dns"--></td>
              </tr>
              <tr>
               <td align="left">Záložní DNS server</td>
               <td align="left"><!--#echo var="dns2"--></td>
              </tr>
              <tr>
                <td align="left" colspan="2">
                  <table width="100%">
                    <tr><br>
                      <INPUT TYPE="SUBMIT" NAME="ADOPT" VALUE="  OK  "></FORM>
                      <FORM ACTION="refresh.cgi" METHOD="POST"><INPUT TYPE="SUBMIT"
NAME="RESET" VALUE=" REFRESH "></FORM>
                    </tr>
                  </table>
                </td>
              </tr>
            </table>
```

```
          </td>
        </table>
</body>
</html>
```