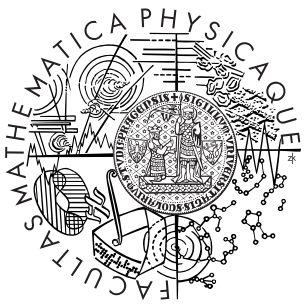
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Jiří Tlach

TCP/IP síťový kontrolér

Katedra softwarového inženýrství

Vedoucí bakalářské práce: RNDr. Jakub Yaghob, Ph.D.
Studijní program: Informatika, obor Programování

2006

Na tomto místě bych rád poděkoval vedoucímu mé bakalářské práce RNDr. Jakubu Yaghobovi, Ph.D. za jeho cenné rady a připomínky, firmě Uinfo s.r.o. za odborné konzultace a pomoc při oživování hardware. Poděkování patří také mým rodičům, kteří mi studium umožnili.

Prohlašuji, že jsem svou bakalářskou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce a jejím zveřejňováním.

V Praze dne Jiří Tlach

Obsah

Název práce: TCP/IP síťový kontrolér
Autor: Jiří Tlach
Katedra: Katedra softwarového inženýrství
Vedoucí bakalářské práce: RNDr. Jakub Yaghob, Ph.D.
e-mail vedoucího: Jakub.Yaghob@mff.cuni.cz

Abstrakt: Cílem této bakalářské práce je návrh a implementace software pro vestavěný systém, jehož jádrem je 8bitovým RISC procesor AVR připojený k ethernetovému řadiči umožňující síťovou komunikaci. Software bude zahrnovat nejen ovládání samotného procesoru a k němu připojených periferií, ale hlavně TCP/IP knihovnu pro internetovou komunikaci. Hlavní důraz přitom bude kladen na prostorovou nenáročnost vzniklého software, protože kapacita kódové paměti dvou AVR procesorů pro které je software určen, je pouze 8 resp. 16 kilobyte.

Klíčová slova: TCP/IP, vestavěná zařízení, AVR

Title: TCP/IP Network Controller
Author: Jiří Tlach
Department: Department of Software Engineering
Supervisor: RNDr. Jakub Yaghob, Ph.D.
Supervisor's e-mail address: Jakub.Yaghob@mff.cuni.cz

Abstract: The aim of this work is design and implementation of software for an embedded system. Its is based on 8-bit RISC AVR processor connected to Ethernet controller that allows network communication. The software will include processor control as well as control of its peripherals and TCP/IP library for Internet communication. The main goal is to minize code size of developed software because both target AVR processors have very limited size of code memory. The first one has capacity of 8 kilobytes, the second one capacity of 16 kilobytes.

Keywords: TCP/IP, embedded devices, AVR

Kapitola 1

Úvod

Komunikace v rámci sítě Internet dnes již není pouze doménou klasických PC vybavených velkou výpočetní a paměťovou kapacitou, nýbrž ke slovu se dostávají i malá zařízení disponující schopností internetové komunikace. Jedná se obvykle o jednoúčelová zařízení – embedded(vestavěné) systémy, která nabízejí pouze minimální paměťové zdroje a malou výpočetní kapacitu. I přesto však mohou fungovat např. jako webový server nabízející informace o naměřených hodnotách z připojených čidel nebo zasílat e-mailové zprávy. Uplatnění takových zařízení lze najít hlavně v průmyslové automatizaci, kde internetové připojení umožňuje řídit či monitorovat celý systém na dálku z jakéhokoli místa, kde existuje připojení k Internetu.

1.1 Cíl práce

Pro tuto bakalářskou práci bylo k dispozici vestavěné zařízení ve dvou variantách nazvané S8 Server. Jeho jádrem je AVR procesor s malou paměťovou kapacitou, ke kterému jsou připojeny další periferie jako sériový port, digitální vstup a výstup a hlavně ethernetový řadič umožňující zasílání a příjem ethernetových rámců. Obě varianty se od sebe liší pouze použitým AVR procesorem – v prvním případě se jedná o AT90S8515, ve druhém pak o ATmega161. Cílem práce bylo navrhnout a implementovat softwarové řešení pro toto zařízení skládající se ze dvou hlavních části: funkcí pro ovládání hardware procesoru včetně připojených periferií a TCP/IP knihovny pro síťovou komunikaci. Jelikož nebyl dopředu znám konkrétní repertoár aplikací, které by měl S8 Server hostit, bylo také jedním z požadavků na tento software jeho snadná konfigurovatelnost a přizpůsobení se individuálním požadavkům každé aplikace.

1.2 Motivace

I přesto, že podobných projektů jako je S8 Server existuje celá řada, většina z nich využívá buď přímo hardwarových obvodů s nativní podporou TCP/IP protokolů nebo jsou jejich základem mikroprocesory s poměrně velkou paměťovou kapacitou (řádově desítky kilobyte). Proto se ukázala jako velmi zajímavá možnost pokusit se vytvořit

TCP/IP implementaci na systému s tak omezenými paměťovými podmínkami jaké nabízí S8 Server.

1.3 Struktura práce

Celá práce je tématicky rozdělena do sedmi kapitol.

V první z nich je podán přehled o rodině protokolů TCP/IP, což je soustava protokolů pro komunikaci v rámci počítačových sítí a hlavně v Internetu. Podrobněji jsou rozebrány klíčové protokoly z této soustavy (IP, ICMP, UDP, TCP) a další protokoly, které byly implementovány v S8 Serveru.

Druhá kapitola se zabývá hardwarem S8 Serveru, na kterém byl prováděn vývoj a testování software. Popsán je hlavně AVR procesor, který je jádrem celého systému, a stručně také další moduly a periferie, které jsou k němu připojeny.

Třetí kapitola se věnuje implementaci software S8 Serveru zahrnující jednoduchý operační systém a samotnou TCP/IP knihovnu pro síťovou komunikaci.

Na třetí kapitolu navazuje kapitola čtvrtá, která vysvětluje konfiguraci a překlad aplikací využívajících TCP/IP knihovnu a operační systém.

Předposlední kapitola přibližuje některé projekty podobného charakteru a podává jejich srovnání s S8 Serverem.

Poslední (závěrečná) kapitola obsahuje zhodnocení vyvinutého software a provádí diskuzi jeho nedostatků s návrhy pro vylepšení a další práci.

Kapitola 2

Rodina protokolů TCP/IP

Tato kapitola se zabývá popisem protokolů z rodiny TCP/IP, jenž byly implementovány v S8 Serveru. U každého protokolu je uveden stručný popis s výjimkou TCP, na který byl kladen poměrně velký důraz, jelikož se jedná o jeden z klíčových protokolů pro spolehlivou komunikaci v rámci Internetu.

2.1 Základní popis

Rodina protokolů TCP/IP představuje sadu protokolů pro komunikaci v počítačových sítích. Její původ sahá do 70. a 80. let 20. století, kdy byla navržena pro komunikaci v americké vládní síti ARPANET, ze které se později vyvinula dnešní celosvětová síť Internet. Základní myšlenkou těchto protokolů je *packet switching* (přepínání paketů), což znamená že přenášená data nejsou posílána jako souvislý proud, nýbrž po samostatných blocích – paketech. O tom, kudy tyto pakety putují pak rozhodují jednotlivé uzly sítě, a s tím je také spojeno několik problémů, které tyto protokoly musí řešit. Mezi ty nejzávažnější patří možnost ztráty paketu v síti zapříčiněná hlavně z důvodu zahlcení sítě. Dalším problémem je například pořadí příchozích paketů, které se může lišit od pořadí v jakém byly pakety odesílány. To je způsobeno tím, že pakety mohly putovat po různých cestách odlišné délky.

TCP/IP protokoly rozdělujeme celkem do čtyř vrstev. Každé vrstvě odpovídá ve struktuře paketu hlavička příslušného protokolu této vrstvy. Samotná data paketu jsou uložena v nejvyšší (aplikační) vrstvě, před kterou jsou při sestavování paketu postupně přidávány hlavičky protokolů nižších vrstev. Na straně příjemce paketu pak dochází k postupnému odebírání hlaviček v opačném pořadí. Příklad TCP/IP paketu je uveden na obrázku 2.1.

Obrázek 2.1: Příklad TCP/IP paketu s linkovou vrstvou Ethernet.

Linková vrstva

Nejnižší vrstvou je linková vrstva, která slouží pro přenos dat po fyzickém médiu. Jejím hlavním úkolem je přenos paketu mezi přímo připojenými uzly v rámci tohoto fyzického média. Referenční model TCP/IP nijak nespecifikuje protokol linkové vrstvy. Jediným požadavkem na něj je zajištění spolehlivého přenosu, který předpokládá síťová vrstva. Příkladem linkového protokolu je Ethernet, FDDI nebo Token Ring.

Síťová vrstva

Globální adresaci a směrování paketů zajišťuje síťová vrstva realizovaná převážně IP protokolem. Každému uzlu sítě je jednoznačně přidělena IP adresa, díky které lze uzel v síti identifikovat. Tyto adresy jsou přidělovány hierarchicky tak, že rozdělení jednotlivých rozsahů odpovídá topologii sítě, což umožňuje pro každý paket s IP adresou určit, kudy má být dále poslán na cestě ke svému cíli.

Transportní vrstva

Síťová vrstva sice zajišťuje přenos mezi uzly sítě, ale již neřeší rozlišování jednotlivých entit (běžících procesů) v rámci těchto uzlů. Tuto funkci plní protokoly transportní vrstvy. Díky nim je tedy možné určit, zda příchozí paket patří např. procesu webového serveru či poštovního klienta. Jedná se o tzv. end-to-end komunikaci. Mezi hlavní zástupce protokolů transportní vrstvy patří UDP a TCP, které identifikují entity v rámci uzlů číslem portu.

Aplikační vrstva

Nejvýše stojí aplikační vrstva reprezentována velkým množstvím protokolů, které jsou navrženy pro účely konkrétních aplikací. Mezi nejznámější patří pravděpodobně HTTP protokol pro webové služby, SMTP pro zasílání pošty či FTP pro přenos souborů.

2.2 IP

IP (Internet Protocol) verze 4 [28] je protokol síťové vrstvy představující základní mechanismus pro přenos paketů mezi jednotlivými uzly v Internetu. Jeho hlavními činnostmi je adresování, směrování a případné fragmentování paketů.

Adresování a směrování

IP protokol verze 4 používá pro adresaci jednotlivých uzlů sítě 4bytovou IP adresu, která je pro každý tento uzel unikátní. Díky hierachickému rozdělování těchto adres je pak možné provádět směrování paketů IP paketů putujících k adresátovi právě podle jeho IP adresy.

Fragmentace

K fragmentaci paketu dochází v situaci, kdy jeho velikost přesáhne hodnotu MTU (Maximum Transfer Unit), což je velikost maximálního linkového rámce, který může být přenesen. V takovém případě je paket rozdělen do více menších paketů (fragmentů) a tyto již přenést lze. Údaj o provedené fragmentaci je zaznamenán do IP hlavičky každého fragmentu tohoto paketu a příjemce si na základě těchto informací sestaví po přijetí všech fragmentů původní paket.

2.3 ICMP

ICMP (Internet Control Message Protocol) [27] je služební protokol, který je součástí IP protokolu. ICMP zprávy jsou typicky generovány při chybách v IP paketech nebo pro diagnostické a směrovací účely. Přehled několika vybraných ICMP zpráv je uveden v tabulce 2.1.

Zpráva	**Význam**
Echo Request	Žádost o zaslání odezvy
Echo Reply	Odezva na Echo Request
Destination Unreachable	Cílový hostitel je nedosažitelný
Source Quench	Zahlcení sítě
Redirect	Informace o přesměrování

Tabulka 2.1: Příklad některých ICMP zpráv

Jeden z velmi používaných jednoduchých diagnostických nástrojů je program ping, který skrze ICMP zprávu Echo Request zjišťuje, zda hostitel se zadanou IP adresou je připojen do sítě. Pokud ano, odpoví zprávou Echo Reply.

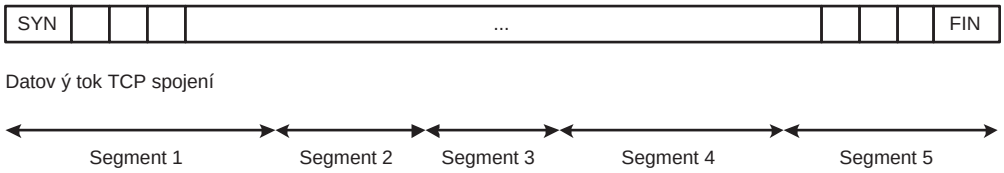
2.4 TCP

TCP (Transmission Control Protocol) [29] je protokol transportní vrstvy zajišťující spojovaný a spolehlivý přenos dat nad nespolehlivými přenosovými cestami, kde hrozí nebezpečí ztráty, poškození, duplikace či přeuspořádání přenášených paketů. Jedná se o jeden z hlavních protokolů z rodiny TCP/IP, který hrál klíčovou roli při rozvoji komunikace v počítačových sítích. Hlavním důvodem jeho úspěchu je schopnost adaptace na vlastnosti přenosových cest a dosažení pokud možno co nejefektivnějšího datového přenosu.

2.4.1 Spolehlivý přenos

Každý byte přenášených dat je označen pořadovým číslem začínajícím na libovolné hodnotě. Datový proud je rozdělen do tzv. TCP segmentů (viz obrázek 2.2, význam

SYN a FIN je vysvětlen dále) vkládaných do IP paketů obsahujících kromě pořadového čísla prvního bytu dat obsažených v segmentu také řídicí informace o spojení.



Obrázek 2.2: Datový tok rozdělený do TCP segmentů

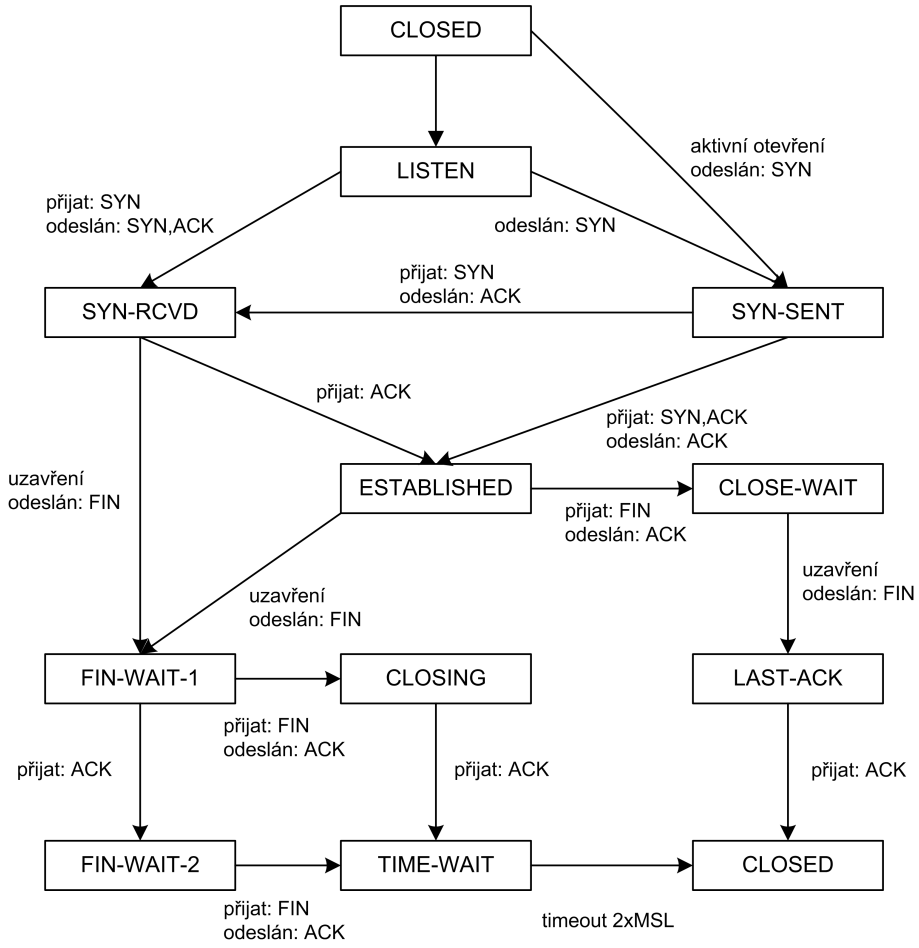
Spolehlivost datového přenosu je zaručena tím, že obdrží-li příjemce TCP segment s daty, zašle odesílateli TCP segment s ACK příznakem a s pořadovým číslem následujícího bytu, který je schopen dále přijmout. Tímto potvrzením dává najevo, že obdržel všechny byty s menšími pořadovými čísly. ACK tedy není potvrzením o přijetí určitého TCP segmentu, nýbrž potvrzením o přijetí dat do určitého pořadového čísla. Pokud odesílatel do jisté doby neobdrží toto potvrzení, zašle svá data znovu. Pokud ani po několikanásobném přeposlání dat druhá strana neodpovídá, je spojení na straně odesílatele zrušeno.

2.4.2 Spojovaná komunikace

TCP protokol vytváří virtuální okruh mezi dvěma komunikujícími uzly nazývaný TCP spojení. To je jednoznačně identifikováno IP adresami a TCP porty obou stran spojení. Aby bylo možné určit v jakém stavu se spojení nachází, musí se každý z komunikujících uzlů řídit stavovým diagramem TCP protokolu (viz obrázek 2.3). Komplikovanost tohoto diagramu je dána především třífázovým zahajováním resp. ukončováním TCP spojení a také tím, že TCP protokol dovoluje polouzavřená resp. polootevřená spojení. K těm může dojít v situaci, kdy jedna strana spojení již nemá žádná data k odeslání a zašle oznámení o ukončení spojení, avšak druhá strana stále posílá svá data. V takovém případě strana iniciující uzavření spojení musí akceptovat příchozí data až do okamžiku, kdy odesílatel také zašle žádost o ukončení spojení.

2.4.3 Otevření spojení

Při otevírání spojení musí jedna ze stran vystupovat jako klient a druhá jako server. Server se musí nacházet ve stavu LISTEN, kde čeká na požadavky na otevření spojení. Klient nacházející se ve stavu CLOSED zahájí celý proces otevření spojení zasláním segmentu s příznakem SYN, v němž také udává pořadové číslo prvního bytu, které bude po ustanovení spojení odesílat. Poté vchází do stavu SYN-SENT. Pokud server akceptuje klientův požadavek, odešle segment s příznaky SYN, ACK a vstupuje do stavu SYN-RECVD. Jakmile klient odpoví ACK segmentem, vstoupí obě strany do stavu ESTABLISHED a mohou začít se zasíláním svých dat. Celý tento proces označovaný jako třífázové podání ruky (three way handshake) je schematicky popsán na obrázku 2.4. Příznak SYN spolu s příznakem FIN, který se používá při



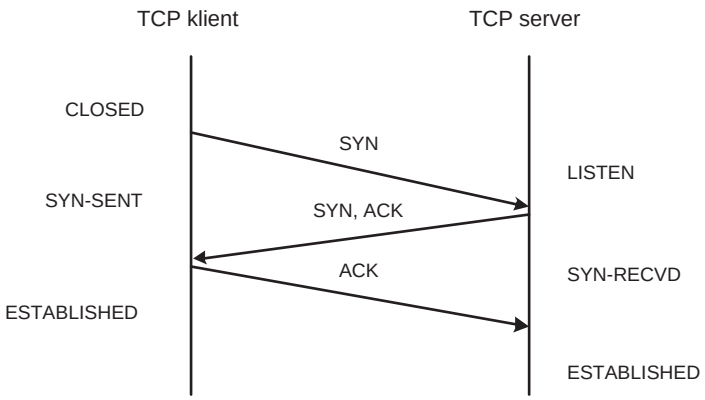
Obrázek 2.3: Stavový diagram TCP spojení

uzavírání spojení, jsou zařazeny do bytového proudu přenášených dat, což umožňuje jejich spolehlivý přenos k druhé straně spojení.

2.4.4 Uzavření spojení

Proces uzavření spojení je mnohem komplikovanější než jeho otevření, protože pro úplné uzavření spojení musí dát souhlas obě komunikující strany.

Strana požadující ukončení spojení nejprve zašle FIN segment a přesune se postupně do stavů FIN-WAIT-1 a FIN-WAIT-2, případně také do CLOSING, po nichž následuje TIME-WAIT. V tomto stavu musí komunikující uzel setrvat po dobu dvojnásobku hodnoty MSL (Maximum Segment Lifetime – doby života segmentu v síti) než vstoupí do stavu CLOSED, čímž je spojení ukončeno. Druhá strana spojení přechází při příjmu FIN segmentu ze stavu ESTABLISHED do CLOSE-WAIT, ve kterém ještě může zasílat svá data. Jakmile také zašle FIN segment signalizující ukončení spojení, vstoupí do stavu LAST-ACK a následně do CLOSED.



Obrázek 2.4: Třífázové navazování TCP spojení

2.4.5 Střední doba obrátky

Podle střední doby obrátky (Round-Trip Time – RTT) přenášených segmentů se určuje časový limit (Round-trip Timeout – RTO), po jehož uplynutí jsou přeposílány nepotvrzené segmenty. Hraje důležitou roli hlavně z hlediska časové efektivity přenosu, protože je-li vypočtená hodnota mnohem menší než skutečná doba obrátky, TCP segmenty budou znovu odeslány ještě dříve, než mohlo přijít ACK potvrzení o jejich přijetí. Tím dochází ke zbytečnému zahlcení sítě. Na druhou stranu je-li vypočtená hodnota příliš velká, zvolené časové limity budou delší než je nutné, což způsobí rapidní zpomalení přenosu dat.

Algoritmus výpočtu střední doby obrátky [29] je následující:

$$R \leftarrow \alpha * R + (1 - \alpha) * M,$$

kde R je střední doba obrátky, M je poslední naměřená doba obrátky a α je konstanta v intervalu $< 0, 1 >$ stanovující míru adaptace RTT na změny. Obvyklá hodnota konstanty α je blízká číslu 1. Časový limit RTO přeposlání segmentu je pak určen vzorcem

$$RTO \leftarrow \beta * R,$$

kde β je konstanta větší než 1 zvolená tak, aby pravděpodobnost, že doba obrátky dalšího odesílaného segmentu překročí hodnotu RTO, byla malá. Vylepšení tohoto algoritmu navrhnul V. Jacobson v [17], kde hodnota β není pevná, nýbrž počítána na základě předchozích hodnot RTT. Další modifikaci přinesla práce P. Karna [20] doporučující provádět výpočet RTT pouze z těch odesílaných segmentů, které nebyly přeposílány. Došlo-li totiž k přeposílání segmentu a vzápětí byl přijat ACK, pak nelze určit, ke kterému z těchto segmentů se ACK vztahuje. V takovém případě by mohla být do výpočtu RTT zanesena chyba.

2.4.6 Řízení toku dat

Řízení toku dat je mechanismus zabraňující situacím, kdy jedna strana spojení zahlcuje příjemce svými daty aniž by je příjemce zvládal zpracovávat. Proto každý TCP segment obsahuje údaj o velikosti tzv. okna (window), což je velikost volné paměti, kterou může uzel využít pro uložení příchozích dat ke zpracování. Odesílatel pak nesmí zaslat data přesahující velikost okna inzerovaného příjemcem.

Tento způsob řízení toku dat označovaný jako technika okna také umožňuje odesílateli zaslat větší množství datových TCP segmentů za sebou aniž přijal potvrzení o přijetí některého z nich druhou stranou spojení.

2.4.7 Mechanismy pro zabránění zahlcení

Zatímco řízení toku dat technikou okna se snaží zabránit přetečení příjmových vyrovnávacích pamětí komunikujících uzlů, mechanismy pro zabránění zahlcení (congestion avoidance) [29] mají za úkol předejít přetečení paměti směrovačů, přes které TCP segmenty putují. Pokud by totiž TCP protokol nerespektoval dostupnou přenosovou kapacitu sítě a neustále zvyšoval objem přenášených segmentů, došlo by k zahlcení sítě, což by mělo za následek zahazování TCP segmentů a zbytečnou degradaci TCP spojení. Z tohoto důvodu byly vyvinuty dva základní algoritmy pro předcházení těchto situací:

- slow start algoritmus
- congestion avoidance algoritmus

Oba přidávají k řídicím informacím každého TCP spojení novou proměnnou nazvanou *congestion window*. Při zasílání dat pak odesílatel pošle minimum z hodnoty *congestion window* a velikosti okna příjemce, čímž se reguluje velikost nepotvrzených dat putujících v TCP segmentech sítí od odesílatele k příjemci. Právě vhodné výpočty hodnoty *congestion window* během přenosu dat zabraňují zahlcení sítě. Na začátku přenosu je tato hodnota vždy nastavena na velikost maximálního TCP segmentu.

Slow Start

Algoritmus slow start slouží k postupné adaptaci na přenosovou kapacitu sítě při zahájení odesílání dat. Po přijetí každého ACK potvrzení je *congestion window* zvětšeno o jeden maximální segment. Po přesáhnutí určité hodnoty označovaného jako *slow start threshold*, je řízení předáno algoritmu congestion avoidance.

Congestion Avoidance

Zatímco ve fázi slow start docházelo k exponenciálnímu vzrůstu hodnoty *congestion window*, ve fázi congestion avoidance algoritmu je tato hodnota zvyšována pouze lineárně dokud nedojde k prvnímu zahození či ztrátě segmentu, což je signalizace zahlcení sítě. Pak následuje snížení *congestion window* na jeden segment, nastavení *slow start threshold* na polovinu velikosti příjemcova okna a řízení přebírá opět slow

start algoritmus. Tím sice dochází k dočasnému zpomalení celé TCP komunikace, ale na druhou se TCP přenos adaptuje na vlastnosti přenosových cest.

Fast Restransmit a Fast Recovery

Je-li některý ze segmentů ve fázi congestion avoidance při přenosu ztracen a přesto se nejedná o zahlcení sítě, výše popsaný algoritmus sníží *congestion window* na jeden segment, což má za následek drastické zpomalení celého TCP přenosu. Právě pro tyto účely slouží algoritmus Fast Retransmit (rychlé přeposlání), který detekuje ztrátu jediného segmentu a zašle jej znovu ještě před tím, než vyprší časový limit pro jeho přeposlání. Algoritmus Fast Recovery (rychlé zotavení) pak zajistí setrvání ve fázi congestion avoidance a tím zabrání degradaci spojení, ke které by došlo snížením *congestion window* na jeden segment a vstupem do fáze slow start.

2.5 UDP

UDP (User Datagram Protocol) [30] je jednoduchou alternativou k protokolu TCP. Zajišťuje nespojovanou a nespolehlivou službu, což znamená, že odesílatel svá data pouze zabalí do UDP datagramu[1], pošle příjemci, ale již se nezabývá tím, zda datagram skutečně dorazí ke svému cíli. Tuto funkci musí zastat protokol nadřazené aplikační vrstvy.

To co nabízí UDP protokol oproti TCP navíc je skutečnost, že adresátem UDP datagramu nemusí být pouze jediná IP adresa, nýbrž určitá skupina stanic (multicast – adresný oběžník) či všechny stanice v síti (broadcast – všeobecný oběžník). Právě toho lze úspěšně využít např. při rádiovém či televizním vysílání přes Internet.

2.6 HTTP

HTTP (Hypertext Transfer Protocol) je protokol aplikační vrstvy nad TCP protokolem na portu 80 sloužící hlavně pro přenos dokumentů (HTML stránek, obrázků, ...) v síti Internet. První verze tohoto protokolu označená HTTP/0.9 umožňovala pouze přenos holých dat. Další verze HTTP/1.0 [3] již s sebou přinesla hlavičky, což jsou krátké textové zprávy s podrobnějším popisem přenášeného dokumentu jako je např. délka dokumentu nebo datum jeho poslední změny. Poslední verzí je HTTP/1.1 [4], která dále rozšiřuje funkce protokolu o možnost přenosu více dokumentů v rámci jednoho transportního spojení, o automatický výběr dokumentu, zaslání pouze vybrané části dokumentu a další.

HTTP je protokol typu klient – server. Každá zpráva zasílaná klientem či serverem se skládá ze dvou částí – hlavičky v textové podobě a datové části (obvykle obsah přenášeného dokumentu). Klient zasílá na server požadavek obsahující tzv. metodu, URI (Uniform Resource Identifier) – identifikace požadovaného zdroje a nakonec verzi protokolu. Server odpovídá verzí protokolu a stavovým kódem. Za hlavičkou pak volitelně následují přenášená data, která mohou být i v binární podobě.

[1] datagram je označení pro datový paket

2.6.1 Stavové kódy

Stavovým kódem informuje server klienta o úspěšnosti vyřízení jeho požadavku. Je tvořen trojciferným číslem doplněným o krátký text popisující význam kódu. Nejvýznamnější je první cifra podávající nejdůležitější informaci – např. zda požadavek byl či nebyl vyřízen. Další dvě cifry poskytují zpřesňující informace. Význam hodnot první cifry je následující:

- 1xx – odpověď je pouze informačního charakteru
- 2xx – požadavek úspěšně vyřízen
- 3xx – aby byl požadavek vyřízen, je nutné zaslat na server další doplňující údaje
- 4xx – jedná se o chybný požadavek
- 5xx – chyba serveru

Stavové kódy nejsou záležitostí pouze HTTP protokolu, nýbrž jsou využívány i v jiných aplikačních protokolech založených na textové komunikaci (SMTP, FTP, ...).

2.6.2 Metody

Metoda je požadavek klienta na server. Základní metodou je GET a HEAD, které slouží k získání zadaného zdroje (souboru) ze serveru. Odpovědí serveru na tyto metody je hlavička s informacemi o zdroji (pokud existuje) a v případě metody GET následuje za hlavičkou také samotný obsah dokumentu.

Postupem času byl HTTP protokol doplňován další metody z nichž některé jsou povinné, ostatní volitelné. Zde je uveden přehled několika nejpoužívanějších metod:

- POST – umožňuje zaslat na server určité údaje ke zpracování. Ta pak může zpracovat např. nějaký proces běžící na straně serveru
- OPTIONS – slouží k získání informací o možnostech komunikace s uvedeným zdrojem určeného pomocí URI
- PUT – slouží k uložení zasílaných dat na server
- DELETE – požadavek na zrušení dokumentu na serveru

2.7 SMTP

SMTP (Simple Mail Transfer Protocol) [21] je textový protokol pro přenos e-mailových zpráv nad TCP protokolem na portu 25. Komunikace tímto protokolem je typu klient – server. Klient nejprve zadá jednoho nebo i více příjemců své zprávy a poté

```
S: 220 smtp1.kolej.mff.cuni.cz ESMTP
K: HELO jiri
S: 250 Helo jiri, pleased to meet you
K: MAIL FROM: <myaddress@mydomain.com>
S: 250 Sender Ok
K: RCPT TO: <recipient@abcdomain.com>
S: 250 Recipient Ok
K: DATA
S: 354 Enter mail, end with "." on a line by itself
K: Subject: Zprava
K: From: <myaddress@mydomain.com>
K: To:<recipient@abcdomain.com>
K:
K: Toto je testovaci e-mail.
K: S pozdravem
K: Jiri Tlach
K: .
S: 250 Message accepted for delivery
K: QUIT
S: 221 closing connection
```

Obrázek 2.5: Příklad SMTP komunikace

následuje přenos textu zprávy. Demonstrativní příklad SMTP komunikace je uveden na obrázku 2.5, kde text klienta je vždy uvozen „K:“ a text serveru „S:“.

Na každý požadavek klienta (zadání příkazu či zaslání textu zprávy) zašle server odpověď ve formě trojciferného stavového kódu podobně jako u HTTP protokolu. Jejich význam je následující:

- 2xx – požadavek úspěšně vyřízen
- 3xx – pro úspěšné vyřízení požadavku je nutné zaslat doplňující informace
- 4xx – chyba přechodného charakteru na serveru
- 5xx – požadavek je neplatný a nebyl vyřízen

2.8 TFTP

TFTP (Trivial File Transfer Protocol) [32] je jednoduchý protokol pro přenos souborů nad UDP protokolem. Jedná se o velmi zjednodušenou podobu FTP protokolu určenou hlavně pro případy, kdy není použití FTP protokolu pro jeho komplikovanost vhodné. Typickým příkladem je nabíhání bezdiskových počítačů ze sítě (s využitím protokolu BOOTP [9]), kdy se přenosový protokol musí vejít do omezeného množství paměti, např. do ROM paměti, která je na stroji k dispozici.

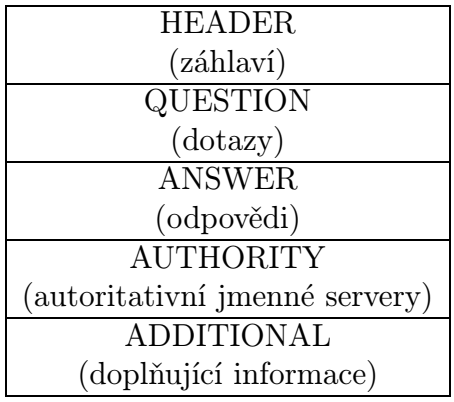
Jelikož TFTP funguje nad nespolehlivou a nespojovanou službou, kterou nabízí UDP, musí obsahovat vlastní mechanismy pro řízení spojení. Koncepce je taková, že při jednom spojení je možné přenést pouze jediný soubor. Během komunikace se v síti pohybuje pouze jediný paket, protože odesílatel vždy čeká na potvrzení než odešle další paket. To je důvodem malé přenosové rychlosti TFTP protokolu na linkách s velkou latencí.

Oproti FTP protokolu má TFTP další významná omezení. Tím nejvýznamnějším je skutečnost, že TFTP server neumožňuje přihlašování připojeného klienta uživatelským jménem ani heslem a také nedovoluje procházení adresářů. Klient proto musí přesně znát přístupovou cestu k požadovanému souboru.

2.9 DNS

DNS (Domain Name System) je hierarchický systém doménových jmen uložený v celosvětově distribuované databázi. Jeho účelem je překlad doménových jmen na IP adresy a nazpět. Celá databáze je realizována pomocí DNS serverů, které vyřizují požadavky na tento překlad. Komunikace mezi těmito servery nebo libovolným klientem a DNS serverem je náplní práce DNS protokolu [18].

DNS je protokol aplikační vrstvy využívající služeb UDP i TCP protokolu na portu 53. Pracuje způsobem dotaz/odpověď (klient/server). Pro dotaz i odpověď se používá datagram stejného formátu (viz obrázek 2.6). Ten se může skládat až z pěti sekcí, z nichž povinná je pouze sekce záhlaví.



Obrázek 2.6: Formát DNS datagramu

V záhlaví (HEADER) je uveden identifikátor datagramu, který slouží k párování dotazu a odpovědi, a parametry dotazu zahrnující např. typ dotazu (překlad z doménového jména na IP adresu či obráceně), celkový počet dotazů obsažených v datagramu a další. Do položek záhlaví také zapíše DNS server informace týkající se odpovědi.

V sekci dotazy (QUESTION) je specifikován samotný dotaz. Ten se skládá ze tří položek:

- doménové jméno pro překlad (QNAME)
- typ dotazu (QTYPE)
- třída dotazu (QCLASS)

Typ dotazu udává, jaký Resource Record je v odpovědi požadován, kde Resource Record (v překladu zdrojová věta) je základní položka v celém DNS systému. Některé z těchto položek jsou uvedeny v tabulce 2.2. Třídou dotazu je obvykle Internet.

Resource Record	**Význam**
A	IP adresa verze 4
NS	Jméno autoritativního serveru pro zónu
CNAME	Alias pro doménové jméno
PTR	Doménové jméno pro reverzní překlad
MX	Priorita a doménové jméno poštovního serveru
TXT	Textový řetězec s popisem

Tabulka 2.2: Příklad Resource Record položek

Za sekcí s dotazy následují tři sekce, každá ve formátu Resource Record položek. Tyto sekce jsou vyplněny pouze v datagramu odpovědi. Sekce odpovědi (ANSWER) obsahuje klientem požadované údaje, např. požadovanou IP adresu k doménovému jménu. Sekce autoritativní jmenné servery (AUTHORITY) je seznam serverů, které

jsou pro doménová jména uvedená v dotazu autoritativními. Poslední sekce (ADDITIONAL) obsahuje různé doplňkové informace.

2.10 DHCP

DHCP (Dynamic Host Configuration Protocol) [10] je aplikační protokol využívající služeb UDP protokolu na portech 68 (DHCP klient) a 69 (DHCP server), který se používá pro automatické přidělování IP adres koncovým stanicím v síti. Současně s IP adresou posílá DHCP server také další doplňkové informace jako adresa nejbližšího směrovače, masku sítě a adresy DNS serverů. V rozsáhlejších sítích se obvykle zasílají ještě další informace.

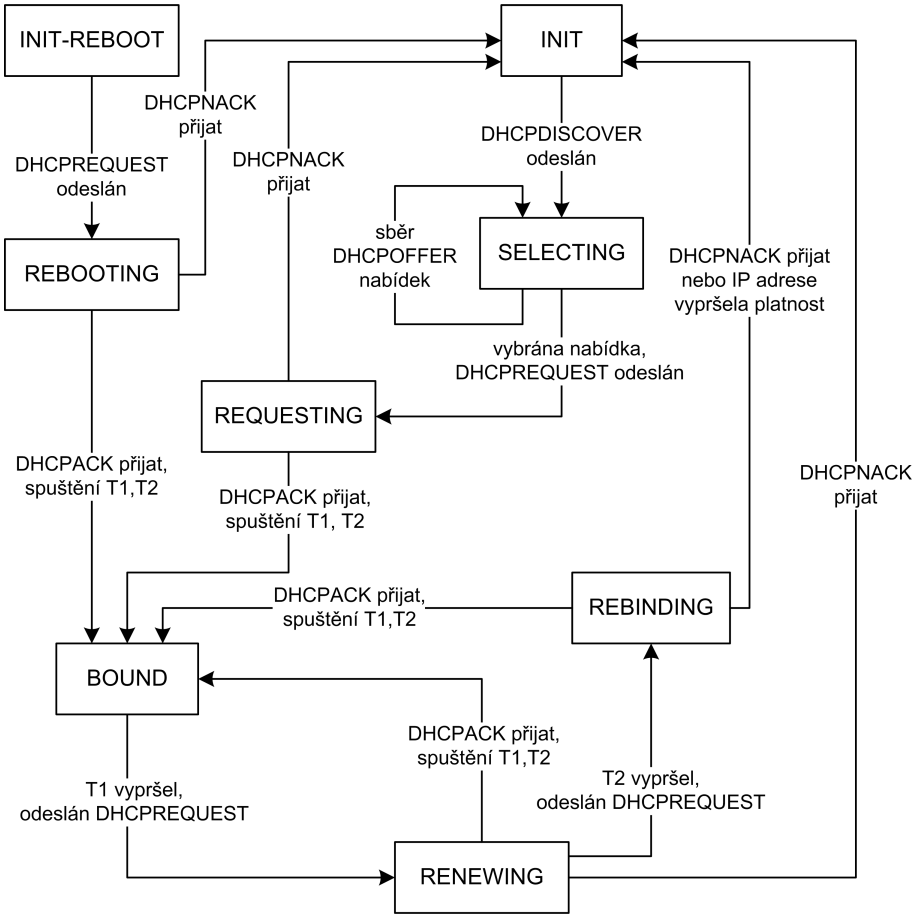
DHCP je rozšířením staršího protokolu BOOTP [9], který přiděloval IP adresy na neomezenou dobu, a je s tímto protokolem zpětně kompatibilní. DHCP server přidělující IP adresy si u každé stanice eviduje půjčenou IP adresu a čas, do kdy ji klient smí používat – doba zapůjčení (lease time). Poté co vyprší, smí server adresu přidělit jinému klientovi.

Komunikace mezi DHCP serverem a klientem probíhá prostřednictvím DHCP zpráv sestávajících z hlavičky a volitelných položek. V hlavičce jsou v případě žádosti uvedeny všechny potřebné údaje klienta a v případě odpovědi přidělená IP adresa a informace o DHCP serveru. Volitelné položky obsahují další doplňující informace.

2.10.1 Činnost klienta

Po připojení do sítě klient vyšle všeobecným oběžníkem zprávu DHCPDISCOVER, na niž odpoví DHCP server zprávou DHCPOFFER s nabídkou IP adresy. Klient si pak z několika těchto příchozích nabídek vybere jednu IP adresu a o tu požádá zprávou DHCPREQUEST. Pokud přidělení proběhne v pořádku, server vrátí DHCPACK. Poté klient používá IP adresu. Po vypršení její platnosti může zažádat server o prodloužení této platnosti. Zde uvedený postup je poměrně zjednodušen, kompletní stavový diagram klienta popisující jeho chování je uveden na obrázku 2.7.

Klient začíná ve stavu INIT resp. INIT-REBOOT pokud si žádá o explicitní IP adresu. Jakmile se dostane do stavu BOUND, znamená to, že mu byla úspěšně přidělena IP adresa. V tento okamžik si nastaví časovače T1 (tzv. renewing timer) a T2 (tzv. rebinding timer), které určují, za jak dlouhou dobu vyprší její platnost. Po uplynutí prvního z nich klient ještě může IP adresu používat, avšak již se musí postarat o prodloužení její platnosti. Tato činnost odpovídá stavu RENEWING. Pokud se to z nějakých důvodů nezdaří a vyprší časovač T2, pak ztrácí klient právo na užívání této adresy a ve stavu REBINDING musí zažádat DHCP server o přidělení nové IP adresy. Nepodaří-li se to, dostává se klient opět na začátek do stavu INIT.



Obrázek 2.7: Stavový diagram DHCP klienta

Kapitola 3

Hardware

V této kapitole je uveden stručný popis desky plošných spojů S8 Serveru, na které byl prováděn vývoj a testování software. Popis je omezen pouze na vysvětlení principů fungování celého systému, a proto nezabíhá do podrobností. Podrobnější informace lze najít na přiloženém CD-ROM disku.

3.1 Základní popis

S8 Server je tvořen dvouvrstvou deskou plošných spojů, kde jádrem je programovatelný 8bitový RISC procesor z řady AVR od firmy Atmel. Všechny ostatní mikročipy na desce jsou k tomuto procesoru připojeny a jím ovládány. Blokové schéma desky plošných spojů je zobrazeno na obrázku 3.1 [1].

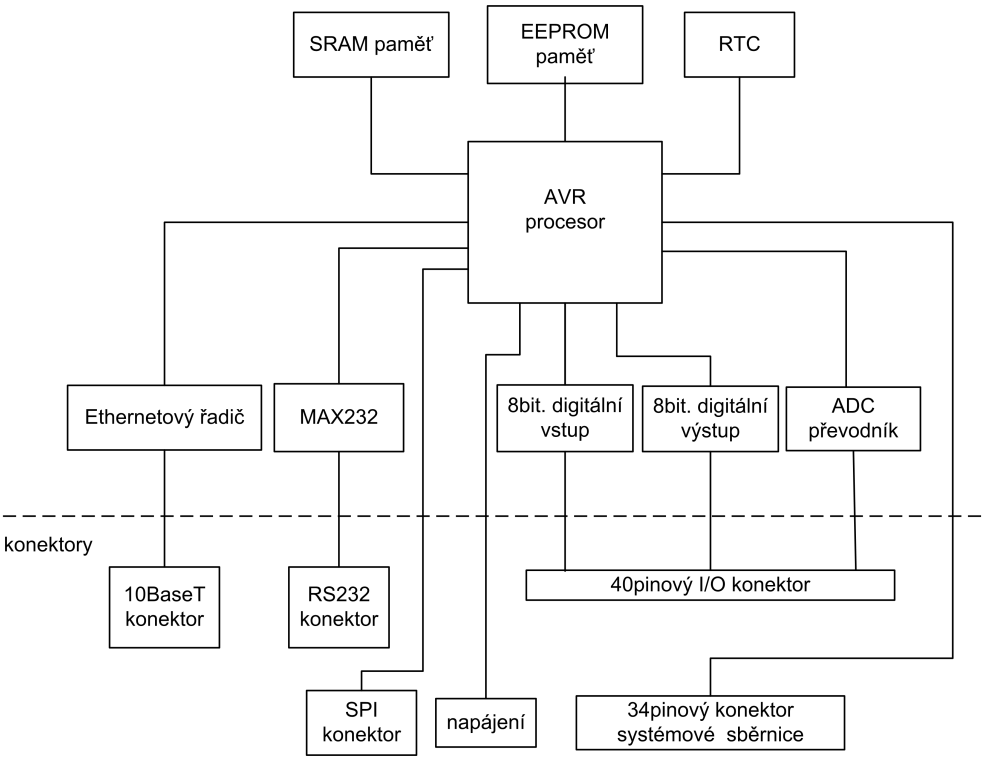
3.2 AVR mikroprocesor

AVR je rodina 8bitových RISC mikroprocesorů s harvardskou architekturou navržená pro efektivní běh programů psaných v jazyce C. Zahrnuje celou škálu mikroprocesorů rozdělených do tří hlavních skupin:

- tinyAVR
- klasická AVR
- megaAVR

Rozdíl mezi zařízeními z těchto skupin spočívá hlavně v dostupných vlastnostech. Zatímco tinyAVR mikroprocesory nabízejí paměť pouze v jednotkách kilobytů a omezené množství dalších prostředků pro běh aplikací, řada megaAVR může být využita pro náročnější aplikace. Jejich společnou vlastností je stejná instrukční sada a organizace paměti, což dovoluje snadný přechod programátora na jiné AVR zařízení. Rychlost těchto procesorů nepřesahuje 20 MIPS.

[1] ADC převodník uvedený na obrázku je instalován pouze na desce plošných spojů s procesorem AT90S8515.



Obrázek 3.1: Blokové schéma desky plošných spojů S8 Serveru

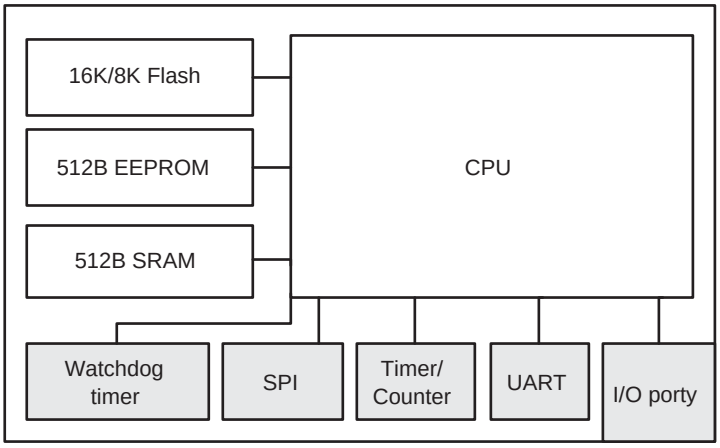
S8 Server je k dispozici ve dvou variantách lišících se pouze použitým AVR. V první variantě se jedná o mikroprocesor AT90S8515 ze skupiny klasických AVR, zatímco ve druhé o ATmega161 ze skupiny megaAVR. Oba mikroprocesory jsou taktovány připojeným krystalem na frekvenci 11,0592 MHz. Společné schéma těchto procesorů je na obrázku 3.2.

3.2.1 Vestavěné paměti

Všechna AVR zařízení v sobě zahrnují persistentní programovatelnou Flash paměť, která slouží pro uložení kódu a statických dat aplikace. Při spuštění aplikace je kód načítán po jednotlivých instrukcích přímo z této paměti. Velikost Flash pro AVR AT908515 je 8 kilobytů a pro ATmega161 16 kilobytů. Pro dynamická data aplikace je k dispozici SRAM paměť, ve které se na nejnižších adresách nacházejí registry procesoru, dále zásobník aplikace a nakonec samotná data.

3.2.2 Vestavěné moduly

Kromě základní paměti nabízejí AVR mikroprocesory také řadu vestavěných modulů, která se může mírně lišit v závislosti na zvoleném modelu. Vestavěné moduly AVR procesorů S8 Serveru jsou znázorněny na obrázku 2 šedou barvou. Následuje jejich stručný popis.



Obrázek 3.2: Blokové schéma AVR procesoru

UART

UART (Universal Asynchronous Receiver and Transmitter) je modul pro sériový přenos dat. Na desce plošných spojů jsou piny tohoto modulu připojeny k čipu MAX232, který provádí konverzi napěťových úrovní a rozhraní je zakončeno RS-232 konektorem, který umožňuje připojit přes sériový kabel jiné zařízení podporující toto sériové rozhraní. Maximální přenosová rychlost je omezena na 115 200 bitů za sekundu. Jedná se v podstatě o klasický sériový port, který je standardním rozhraním také každého klasického PC.

Timer/Counter

Timer/Counter (čítač/časovač) slouží ke generování nebo měření časových intervalů či periodických signálů a dokáže také vyvolávat pravidelná přerušení. Jeho klasickým využitím může být např. odpočet tiků od startu systému.

Watchdog timer

Jedná se modul, který slouží jako pojistka proti chybné funkci software. Jeho základním prvkem je čítač, který je pravidelně snižován o jedna až k nule. Úkolem aplikace je provádět pravidelné restartování tohoto čítače, čímž je nastaven zpět na hodnotu vyšší než nula. Dosáhne-li čítač hodnoty nula, znamená to pravděpodobně chybnou funkci sotftware, jehož úkolem bylo provést restart čítače. V takovém případě je proveden restart AVR procesoru a software je spuštěn znovu.

I/O porty

Oba AVR procesory nabízejí 32 vstupně-výstupních pinů, ke kterým jsou připojeny externí moduly či konektory na desce plošných spojů. Přes tyto piny probíhá veškerá komunikace AVR procesoru s okolním světem.

SPI

SPI (Serial Peripheral Interface) je jednoduché rozhraní pro sériovou komunikaci, které je u S8 Serveru připojeno k ISP In-System Programming) konektoru a umožňuje tak komunikaci s jiným externím zařízením podporujícím SPI protokol. Tento protokol představuje také standardní způsob přenosu zkompilované aplikace z počítače do interní Flash paměti.

3.2.3 Externí paměti

Protože S8 Sever byl navrhnut jako univerzální zařízení pro sběr a snímání dat z různých zdrojů a internetovou komunikaci, bylo nutné jej také vybavit dostatkem pracovní paměti. Interní paměť poskytovaná AVR procesorem v tomto ohledu nebyla dostačující, a proto jsou na desce plošných spojů nainstalovány dvě rozšiřující paměti.

První pamětí je 32 kilobytová SRAM, která je připojena k adresové a datové sběrnici mikroprocesoru a namapována do adresového prostoru aplikace spolu s interní SRAM. Dovoluje tak aplikaci využívat celých 32 kilobyte této paměti.

Druhou pamětí je persistentní 16 kilobytová EEPROM určená pro uchovávání dat trvalejšího charakteru, jako jsou HTML soubory, pokud aplikace funguje například jako HTTP server. Externí EEPROM je připojená k dvoudrátové sběrnici desky plošných spojů a komunikace s AVR mikroprocesorem zde probíhá přes I^2C protokol.

3.2.4 Hodiny reálného času

Externí modul RTC (Real Time Clock) rozšiřuje funkce S8 Serveru o možnost nastavení a získání reálného času. Díky zálohování RTC modulu lithiovým článkem, který je při provozu celého kontroléru regenerován nabíjecím obvodem, je reálný čas udržován v tomto modulu i při vypojení napájení.

RTC je připojen k AVR mikroprocesoru přes dvoudrátovou sběrnici podobně jako externí EEPROM paměť a komunikace proto probíhá I^2C protokolem.

3.2.5 Ethernetový řadič

Jedná se o hlavní periferii S8 Serveru, která přijímá a vysílá ethernetové rámce podle standardu 802.3. Čip řadiče je připojen ke konektoru 10BaseT, který je kompatibilní s klasickým síťovým kabelem a umožňuje připojení zařízení k jinému počítači, lokální síti či globální síti (Internetu).

3.2.6 Digitální vstup a výstup

8bitový digitální vstup a výstup vyvedený na 40pinový konektor na desce plošných spojů může být využit pro připojení čidel či jiných zařízení s dvoustavovými vstupy či výstupy.

3.2.7 Analogový převodník

ADC (Analog to Digital Convertor) je převodník, který umožňuje převod analogové hodnoty na digitální. Například přivedení hodnoty 1V na vstup tohoto převodníku při maximálním rozsahu 2,5V bude výsledná hodnota rovna 40 % maximální digitální hodnoty. ADC převodník MAX1240 instalovaný pouze na desce plošných spojů s procesorem AT90S8515 reprezentuje naměřené hodnoty 12bitovou hodnotou.

3.2.8 Konektor systémové sběrnice

Pro možnost připojení dalších externích modulů k S8 Serveru je na desce plošných spojů instalován 34pinový konektor obsahující piny adresové a datové sběrnice, dvoudrátové sběrnice pro komunikaci I^2C protokolem, piny SPI rozhraní a některé další piny AVR procesoru.

Kapitola 4

Software

Tato kapitola se zabývá popisem softwarového řešení S8 Serveru. V její první části je nastíněna celková koncepce, za níž následuje popis jednoduchého operačního systému a TCP/IP knihovny pro síťovou komunikaci.

4.1 Koncepce

Softwarové řešení S8 Serveru se skládá ze tří hlavních částí. Jedná se o

- Jednoduchý operační systém
- TCP/IP knihovna pro síťovou komunikaci
- Ukázkové aplikace

Cílovou platformou jsou dva AVR procesory AT90S8515 a ATmega161, jejichž popis byl uveden v kapitole 3. Vzhledem k tomu, že velikost dostupné kódové paměti těchto procesorů je 8 resp. 16 kilobytů, byla hlavním kritériem celého software právě co nejmenší prostorová náročnost, avšak nikoli na úkor jeho funkčnosti. Dalším důležitým kritériem byla snadná konfigurovatelnost software ještě před jeho překladem, jelikož každá aplikace může mít své vlastní specifické nároky.

Veškeré zdrojové kódy jsou napsány v jazyce C, což činí celý projekt snadno přenositelný na jiný mikrokontrolér s podobnými vlastnostmi. Výjimkou je pouze jeden modul operačního systému pro pozdržení běhu aplikace o přesně zadaný počet milisekund, který je kompletně napsán v AVR assembleru.

4.2 Operační systém

Při návrhu operačního systému bylo nutné zvolit, zda se má jednat o operační systém podporující vícevláknové zpracování, zda má obsahovat podporu pro real-time operace či naopak podporovat pouze běh jediné aplikace. V současné době je k dispozici celá řada volně dostupných převážně real-time operačních systémů pro vestavěná zařízení, které by mohly být pro S8 Server použity. Patří mezi ně např. FreeRTOS [16],

Liquorice [22] nebo Ethernut Nut/OS [15] – všechny s podporou AVR platformy. Při prvních experimentech s těmito operačními systémy bylo však zjištěno, že jejich prostorové nároky jsou větší, než jaké může ve stávající konfiguraci S8 Server nabídnout. Z tohoto důvodu bylo nutné přikročit k napsání vlastního malého operačního systému.

Výsledkem je jednoduchý operační systém podporující pouze běh jediné aplikace, jelikož podpora vícevláknového zpracování by byla prostorově poměrné náročná a u mnohých aplikací S8 Serveru, které provádějí např. pouze monitorování hodnot získaných z periferií, také nevyužita. S8 operační systém (S8 OS) se skládá z těchto základních částí:

- Správa paměti
- Souborový systém pouze v režimu pro čtení
- Přístup k hardware

Jedná se v podstatě o soubor funkcí, které aplikace volá podle své potřeby. S8 OS nemá žádné jádro a ani nijak nezasahuje do běhu aplikace. Jedinou jeho samostatnou činností je inicializace AVR procesoru po startu mikrokontroléru a předání řízení aplikaci.

4.2.1 Správa paměti

Správa paměti je volitelná vlastnost S8 OS umožňující aplikaci dynamicky alokovat a uvolňovat bloky paměti požadované velikosti. Jako alokační strategie byl zvolen algoritmus first-fit. Hlavním důvodem pro tuto volbu je jednoduchost a prostorová nenáročnost jeho implementace. Kladem je také rychlost tohoto algoritmu a míra fragmentace paměti, která může do značné míry ovlivnit běh aplikací, je ve srovnání s jinými alokačními strategiemi poměrně malá, což dokládá studie v [33].

Dostupná paměť je tímto algoritmem rozdělena na jednotlivé bloky propojené v obousměrný spojový seznam. Každý blok je opatřen hlavičkou, ve které se nachází příznak určující, zda se jedná o volný či alokovaný blok, dále velikost bloku a ukazatele na předchozí resp. následující blok. Struktura hlavičky je zobrazena na obrázku 4.1. Použití obousměrného spojového seznamu je výhodné hlavně pro slučování volných sousedních bloků.

Při žádosti aplikace na alokaci paměti se provede průchod tímto seznamem od začátku a je vrácen první vyhovující blok (odtud název first-fit). Pokud je velikost bloku větší, než jakou aplikace vyžaduje, vezme se z bloku požadovaná část, která se označí jako alokovaná a zbytek utvoří nový, nealokovaný blok.

4.2.2 Souborový systém

Hlavně z důvodu podpory HTTP serveru jako jedné z aplikací S8 Serveru byl implementován jednoduchý souborový systém umožňující pouze čtení zadaných souborů. Ty nejsou uloženy na žádném disku či médiu podobného typu jako u klasického PC,

```
typedef struct mcb_t {
  uint8_t used;       /* označuje volný nebo alokovaný blok */
  uint16_t size;      /* velikost bloku */
  struct mcb_t *prev; /* ukazatel na předchozí blok */
  struct mcb_t *next; /* ukazatel na následující blok */
};
```

Obrázek 4.1: Hlavička paměťového bloku

nýbrž přímo v některé z dostupných pamětí S8 Serveru, což je EEPROM nebo Flash paměť. Samotná implementace souborového systému je přitom nezávislá na použité paměti, jelikož při inicializaci je jí předán ukazatel na aplikační funkci, která bude zajišťovat načítání bloků zadané velikosti z paměti, ve které jsou soubory uloženy.

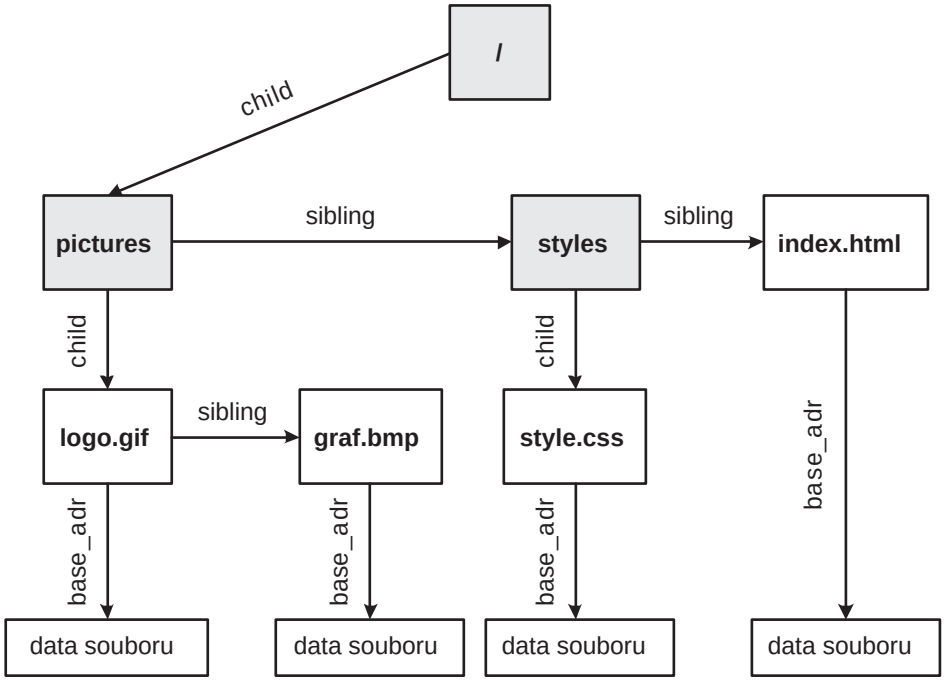
```
typedef struct fnode_t {
  char name[FS_FILENAME_LENGTH];/* jméno souboru/adresáře */
  uint16_t length;              /* délka souboru (0 pro adresář) */
  uint16_t base_adr;            /* bázová adresa souboru */
  uint16_t sibling;             /* ukazatel na pravého sourozence */
  uint16_t child;               /* ukazatel na potomka */
};
```

Obrázek 4.2: Struktura fnode_t

Pro vnitřní organizaci hierarchie souborů byla zvolena stromová struktura resp. její kanonická reprezentace, což umožňuje použití adresářů. Jeden uzel stromu je popsán strukturou `fnode_t`, která je popsána na obrázku 4.2. Zda se jedná o uzel popisující soubor nebo adresář lze poznat z položky base_adr udávající bázovou adresu souboru, která je pro adresář nulová. Každá hladina stromu reprezentuje obsah adresářů stejné úrovně a postup směrem od kořene k listům stromu odpovídá sestupu v adresářové struktuře. Možný příklad adresářové struktury sestavěné ze struktur `fnode_t` je zobrazen na obrázku 4.3. Šedě jsou vyznačeny uzly odpovídající adresářům, bílou barvou pak soubory. Pokud bychom chtěli přistoupit např. k souboru `/style/styles.css`, museli bychom projít přes uzel kořenového adresáře (označován `/`), dále přes uzel adresáře `pictures` a nakonec přes uzel `styles`, jehož potomkem je požadovaný soubor `styles.css`.

4.2.3 Přístup k hardware

Většina klasických operačních systémů nabízí svým aplikacím abstrakci nad přístupem k hardware. Tj. poskytují určité standardní rozhraní nezávislé na konkrétním



Obrázek 4.3: Příklad stromové struktury souborového systému s uzly fnode_t

hardware pro jednotlivé typy periferií jako je např. klávesnice, myš, síťová karta, atd. Aplikace pak přistupuje k periferiím pouze přes toto předepsané rozhraní až na několik výjimek.

V případě malého operačního systému jakým je S8 OS, by jen implementace samotného rozhraní přístupu k hardware zabírala příliš mnoho prostoru, a proto byla zcela vypuštěna. Aplikace tedy volá přímo funkce ovladačů jednotlivých komponent S8 Serveru a má obsluhu periferií pouze ve svých rukou. Operační systém v tomto směru nijak nezasahuje do jejího jednání.

Implementovány byly ovladače těchto zařízení [1]:

- UART
- 8bitový čítač/časovač
- Ethernetový řadič
- I^2C
- 8bitový digitální vstup/výstup
- MAX1240 – analogový konvertor
- Real Time Clock – hodiny reálného času

[1] Zdrojové kódy ovladačů I^2C, 8bitového digitálního vstupu/výstupu, MAX1240 dodala firma Uinfo s.r.o.

- Externí 16KB EEPROM

Stručný popis těchto komponent byl uveden v kapitole 3.

4.3 TCP/IP knihovna

Stěžejní částí celého projektu je knihovna vybraných protokolů z rodiny TCP/IP, která umožňuje S8 Serveru síťovou komunikaci. I přesto, že existuje velké množství volně dostupných a přenositelných implementací TCP/IP protokolů jako součást operačních systémů či jako samostatný software, žádná ze zkoumaných implementací nakonec nebyla vybrána pro S8 Server a získané zdrojové kódy těchto knihoven posloužily jako inspirace pro vytvoření vlastní TCP/IP implementace. Hlavním důvodem pro tento krok je skutečnost, že většina knihoven se vyznačovala buď robustností nebo se naopak jednalo o příliš „skromné“ implementace s velmi omezenou funkčností.

Robustní implementace sice poskytovaly plnou funkčnost a spolehlivost všech protokolů, avšak jejich velikost se pohybovala v řádech desítek kilobyte, což je mimo možnosti S8 Serveru. Mezi tyto implementace patří například lwIP [11], Ethernut Nut/Net [15], nebo Kadak KwikNET [19].

Na druhou stranu „skromné“ implementace jako je uIP [13] a TinyTCP [8] si obvykle vystačí pouze s několika stovkami byte RAM paměti a s kódovou pamětí v řádech jednotek kilobyte, ale mají řadu nevýhod, které velmi omezují jejich použitelnost. Příkladem je omezení počtu současně otevřených TCP spojení pouze na jedno, absence sestavování fragmentovaných IP datagramů či minimální podpora řízení toku dat v TCP protokolu.

4.3.1 Koncepce knihovny

Organizace zdrojových souborů

Protokoly TCP/IP jsou charakteristické tím, že pracují téměř nezávisle na sobě a vzájemná komunikace protokolů sousedních vrstev probíhá přes předem dané rozhraní. To umožnilo rozdělení celého zdrojového kódu knihovny do jednotlivých modulů, kde každý reprezentuje jeden protokol. Toto rozdělení také umožňuje, aby se programátor při překladu své aplikace zvolil, které protokoly má jeho aplikace podporovat a přilinkoval k výslednému kódu pouze moduly těchto protokolů.

API

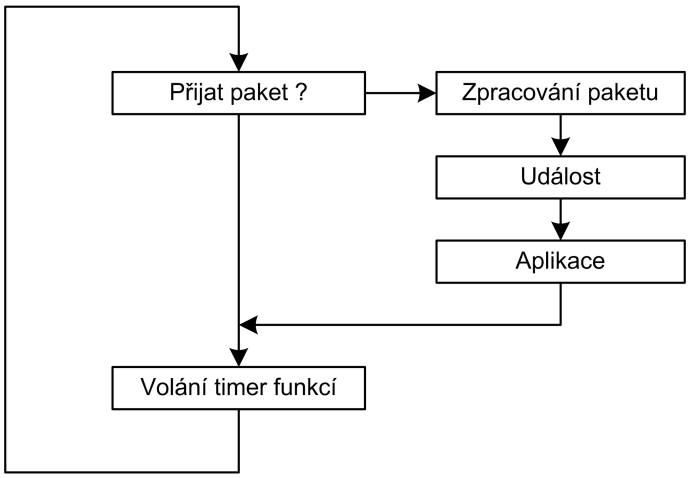
Aplikační programové rozhraní (API) předepisuje způsob jakým aplikace komunikuje s TCP/IP knihovnou. Běžným rozhraním jsou tzv. BSD sockety používané na unixových systémech, které se prosadily také v prostředí operačního systému Windows jako Windows Sockets. Jejich principem jsou blokované operace pro čtení ze socketu a zápis do socketu. To znamená, že vydá-li aplikace příkaz pro čtení ze socketu, ale socket žádná příchozí data neobsahuje, je aplikace v obvyklém případě zablokována

do doby, než jsou data z druhého konce spojení přijata. Takový mechanismus však nutně potřebuje podporu multitaskingového operačního systému, který na S8 Serveru není k dispozici. Proto byl zvolen zcela jiný přístup založený na událostech, kde při výskytu určité události je aplikace vyvolána, aby provedla její obsluhu. Příkladem takových událostí je požadavek na otevření nového TCP spojení, příjem TCP dat nebo obdržení DNS odpovědi.

Kontrolní smyčka

Hlavním jádrem knihovny je kontrolní smyčka, kterou musí implementovat sama aplikace. Provádí se v ní dvě základní činnosti:

1. Kontrola na přijetí paketu ze sítě.
2. Volání tzv. timer funkcí jednotlivých protokolů TCP/IP knihovny.



Obrázek 4.4: Kontrolní smyčka jádra TCP/IP knihovny

Je-li přijat paket ze sítě, je předán TCP/IP knihovně ke zpracování. Zpracování paketu obvykle způsobí vznik určité události a následně je vyvolána aplikace pro obsluhu této události. Součástí obsluhy může být také zasílání vyprodukovaných síťových paketů.

Timer funkce je rutina, ve které si každý TCP/IP protokol obvykle provádí kontrolu na vypršení svých časovačů, což může mít za následek přeposlání posledně odesílaných dat, ukončení spojení s druhou stranou, protože do vypršení časového limitu neprovedla požadovanou reakci apod.

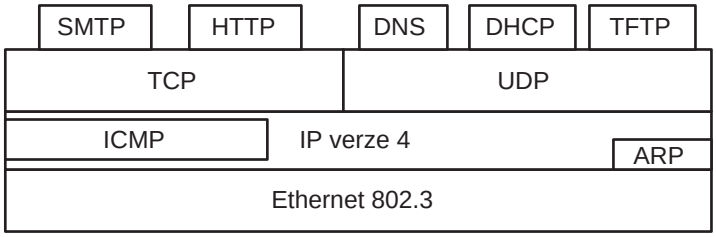
Příjem a vysílání paketů

Pro příjem paketů je v TCP/IP knihovně staticky alokována paměťová oblast jménem `packet`. Ethernetový řadič do něj uloží celý paket a při jeho zpracování si každá TCP/IP vrstva čte potřebné údaje.

Pro odesílání paketu je využita stejná paměťová oblast, aby byly co nejvíce minimalizovány nároky na paměť. Zatímco zpracování přijatého paketu probíhá od nejnižší vrstvy (linkové) směrem k vyšším vrstvám TCP/IP, při vytváření paketu je postup opačný. Aby tedy mohla nejvyšší vrstva zapsat svá data na správné místo do paměťové oblasti `packet`, potřebuje znát objem dat, které vyprodukují vrstvy pod ní. Tento problém je vyřešen tím způsobem, že každá vyšší vrstva si nejprve zjistí délku hlaviček všech protokolů, které budou použity v nižších vrstvách, což jí dovoluje zapsat svá data na absolutní pozici v paměti `packet`. Nižší vrstvy již budou pouze předřazovat hlavičky svých protokolů před tato data. Nejnižší vrstva se tímto postupem dostane až na začátek celé paměťové oblasti `packet`.

Podporované protokoly

Rodina protokolů TCP/IP čítá na desítky protokolů a v implementaci pro S8 Server z ní proto byly vybrány pouze ty klíčové. Přehled protokolů podporovaných S8 Serverem je uveden na obrázku 4.5.



Obrázek 4.5: Podporované TCP/IP protokoly

Jejich implementace byla prováděna podle příslušných RFC dokumentů, které obsahují jejich detailní popis. Důležitým zdrojem byl také dokument RFC 1122 [5] stanovující nároky kladené na implementaci protokolů jednotlivých vrstev pro každý počítač nebo zařízení připojené k Internetu. Uváděné nároky jsou v tomto dokumentu rozděleny do tří kategorií:

1. MUST – požadavky, které musí být splněny v každé implementaci.
2. SHOULD – požadavky, jejichž splnění se doporučuje.
3. MAY – volitelné požadavky, jejichž splnění může být chápáno jako nadstandardní.

V případě S8 Serveru byl brán zřetel pouze na první kategorii stanovující v podstatě minimální množinu požadavků, které musí být splněny, aby zařízení vůbec bylo schopno síťové komunikace.

Další text je věnován popisu implementace jednotlivých protokolů.

4.3.2 Ethernet

Z protokolů linkové vrstvy podporuje S8 Server pouze Ethernet standardu 802.3 [14], jelikož má k dispozici ethernetový řadič, který umožňuje zasílat a přijímat rámce [2] tohoto formátu. Při přijetí rámce je ověření kontrolního součtu a kontrola adresátovy linkové adresy prováděna přímo ethernetovým řadičem a samotná implementace protokolu Ethernet provádí pouze předání datové části rámce protokolu vyšší vrstvy, kterým je IP. Při vysílání je sestavena hlavička rámce a do datové části je vložen IP paket. Rámec je předán ethernetovému řadiči, který připojí za jeho konec kontrolní součet a rámec odešle.

4.3.3 ARP

Protokol ARP (Address Resolution Protocol) [26] stojí mezi linkovou a síťovou vrstvou, jelikož zprostředkovává překlad linkové adresy na IP adresu v rámci lokální sítě. Implementace ARP protokolu zahrnuje funkce pro zaslání odpovědi na ARP dotaz a také zaslání požadavku na překlad IP adresy.

Důležitou součástí implementace je také tzv. ARP tabulka, která slouží pro dočasné uchování IP adres a k nim příslušejících linkových adres. Pokud přijde z IP vrstvy požadavek na překlad určité IP adresy, nejprve je prohledána tato tabulka a je-li nalezen odpovídající záznam, je vrácena příslušná linková adresa. Tím se výrazně přispívá ke zmenšení provozu na lokální síti a šetří se tím dostupná přenosová kapacita. Není-li záznam v tabulce nalezen, je vyslán ARP dotaz a po přijetí odpovědi je získaná linková adresa uložena spolu s IP adresou do tabulky jako nová položka.

Věk	**IP adresa**	**Ethernetová adresa**
10	172.17.93.161	00:50:53:f6:12:40
12	192.168.29.43	00:60:47:d5:27:54
2	172.29.65.216	00:50:80:37:ed:81

Tabulka 4.1: Příklad ARP tabulky

V tabulce 4.1 je uveden příklad několika ARP záznamů. Hodnota ve sloupci věk udává délku života položky v minutách. Při klesnutí na nulu je položka vymazána a přijde-li dotaz na jí odpovídající IP adresu, musí být vyslán ARP požadavek. Tím se zajistí, že při určitých změnách na lokální síti (např. výměna síťové karty u směrovače, čímž se změní jeho linková adresa) budou mít všechny uzly sítě implementující mechanismus ARP tabulky po krátkém čase aktuální položky této tabulky. Obvyklá počáteční doba života položky je několik minut.

[2] rámec je označení pro paket linkové vrstvy

4.3.4 IP

S8 Server implementuje pouze základní funkce IP protokolu – příjem a vysílání paketů. Neumožňuje směrování a nerozeznává volitelné položky IP záhlaví. K dispozici je mechanismus pro sestavování fragmentovaných paketů.

Příjem paketů

Po přijetí paketu je nejprve zkontrolována verze IP protokolu a ověřen kontrolní součet. Nesouhlasí-li některý z těchto údajů, paket je zahozen. Dalším krokem je kontrola adresy příjemce, která musí odpovídat buď IP adrese S8 Serveru nebo oběžníku (tzv. broadcast). Pokud se jedná o fragmentovaný paket, je uschován do paměti, kde bude kompletní paket sestaven až po přijetí všech jeho částí, v jiném případě je předán protokolu vyšší vrstvy.

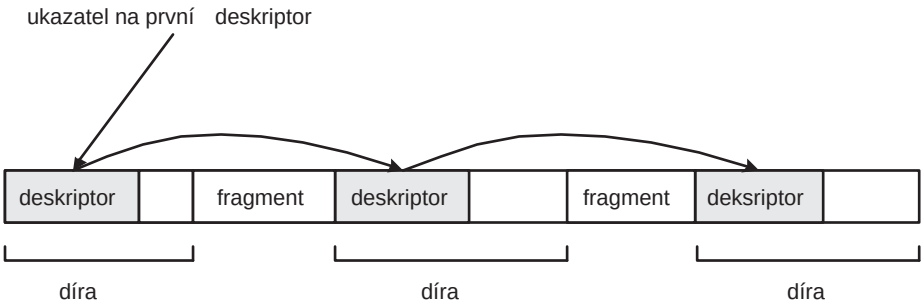
Odesílání paketů

Data k odeslání jsou vložena do IP paketu, u kterého je dopočítán kontrolní součet. Pokud není známa linková adresa příjemce, paket je uložen do fronty čekajících paketů a je odeslán ARP požadavek na zjištění této linkové adresy. Je-li známa nebo je zjištěna z příchozí ARP odpovědi u paketu čekajícího ve frontě, paket je předán linkové vrstvě k odeslání.

Sestavování fragmentovaných paketů

Přestože k fragmentaci IP paketů dochází zřídka, jsou situace, ve kterých se může vyskytnout [6]. Z tohoto důvodu S8 Server volitelně podporuje sestavení paketu z IP fragmentů.

Pro implementaci byl zvolen jednoduchý algoritmus popsaný v RFC 815 [7]. Pro každý nový sestavovaný paket je v paměti vyhrazen tzv. reassembly buffer, do kterého se vkládají příchozí fragmenty tohoto paketu na pozice odpovídající v sestaveném paketu. Zbývající část reassembly bufferu je tvořena tzv. dírami, což jsou místa, kde dosud nějaký fragment chybí. Každá tato díra je jednoznačně určena pozicí svého prvního a posledního bytu – dvojicí hodnot označovaných jako popisovač (deskriptor) díry. Všechny popisovače děr jsou seřazeny do spojového seznamu, který se nachází přímo v reassembly bufferu (viz obrázek 4.6). Pouze ukazatel na první popisovač se nachází mimo něj. Při příchodu nového fragmentu je nejprve určena jeho pozice v reassembly bufferu, aktualizován spojový seznam popisovačů a nakonec je fragment vložen na patřičnou pozici. Pokud po provedení tohoto postupu obsahuje ukazatel na první popisovač neplatnou hodnotu, znamená to ukončení sestavovacího procesu, protože v reassembly bufferu se již nenachází žádná díra. Paket je v takovém případě předán IP vrstvě ke zpracování.



Obrázek 4.6: Příklad reassembly bufferu

4.3.5 ICMP

Podpora ICMP protokolu v S8 Serveru je volitelná a zahrnuje pouze dvě funkce. První z nich je reakce na požadavek Echo Request, který je zasílán diagnostickým nástrojem ping pro zjištění, zda je zadaný uzel připojen do sítě. Druhou funkcí je zasílání zprávy Destination Unreachable pokud je přijat požadavek na protokol transportní nebo aplikační vrstvy, který není implementován.

4.3.6 TCP

TCP protokol je základním transportním protokolem pro spolehlivý přenos dat, čehož využívá velké množství nadřazených aplikačních protokolů, a proto byl implementován i v S8 Serveru. I přesto, že se jedná o velmi komplexní protokol a většina dnešních TCP implementací zabírá řádově desítky až stovky kilobytů, bylo cílem této implementace dosažení velikosti v řádku jednotek kilobyte. Z tohoto důvodu nejsou podporovány všechny vlastnosti TCP protokolu, nýbrž pouze ty z nich, které jsou podstatné pro dodržení pravidel správné komunikace.

Vzhledem k různým potřebám aplikací S8 Serveru byla vytvořena konfigurovatelná TCP implementace s těmito vlastnostmi:

- řízení toku dat – technika okna
- volitelná podpora výpočtu RTT
- volitelná podpora algoritmů pro zamezení zahlcení
- volitelná podpora úschovy odchozích dat do vyrovnávacích pamětí, což usnadňuje přeposílání nepotvrzených TCP segmentů.

Výpočet RTT (Round-trip Time – střední doba obrátky) a podporu algoritmů pro zamezení zahlcení využijí spíše aplikace komunikující v rámci rozsáhlejších sítí (WAN a Internet), zatímco v rámci lokální sítě, kde je předpoklad velké přenosové rychlosti, nejsou tyto mechanismy zcela nutné. Podpora úschovy odchozích dat do

vyrovnávacích pamětí usnadňuje práci aplikacím, protože je-li odeslaný TCP segment v síti na cestě k adresátovi ztracen, je tento TCP segment vyhledán ve frontě uvnitř vyrovnávací paměti segmentů a přeposlán. Značnou nevýhodou tohoto mechanismu je velká paměťová náročnost, obzvlášť je-li současně otevřen větší počet TCP spojení. Není-li tento mechanismus implementován, znamená to pro aplikaci režii navíc, protože data, která předala TCP modulu k odeslání a která se ztratila, musí sama aplikace vygenerovat znovu a předat TCP modulu k přeposlání.

Oproti většině robustních TCP implementací zde chybí podpora pro uchovávání TCP segmentů příchozích mimo pořadí. To je situace, která se může běžně vyskytnout a je důsledkem přeuspořádání TCP segmentů na cestě od odesílatele k příjemci. Zaslal-li odesílatel např. 5 TCP segmentů za sebou a dojde-li k výměně pořadí prvních dvou segmentů, pak implementace TCP na S8 Serveru přijme pouze segment s pořadovým číslem 2 a ostatní zahodí, jelikož jsou mimo pořadí. Při podpoře úschovy segmentů mimo pořadí by bylo přijato všech 5 segmentů a sama TCP implementace by zajistila jejich přeuspořádání. Tento mechanismus by však vyžadoval velké nároky na RAM paměť, a proto nebyl implementován.

API

Aplikační programové rozhraní odpovídá celkové koncepci TCP/IP knihovny a je založeno na tzv. TCP socketech, což je datová struktura uchovávající řídicí informace každého spojení. Aplikace nejprve vytvoří TCP socket zavoláním funkce `tcp_listen()` pokud chce přijímat spojení od klientů či `tcp_connect()` pokud chce provést aktivní navázání spojení se zadaným hostitelem. Další komunikace TCP modulu s aplikací probíhá pouze přes funkci zpětného volání (callback funkce) specifikovanou aplikací při vytvoření socketu, která je volána při výskytu určité události na socketu. Událostí je např. úspěšné navázání spojení s druhou stranou, příjem dat nebo ukončení spojení. Uvnitř obsluhy této události aplikace používá funkce `tcp_write()` a `tcp_read()` pro zaslání či přečtení dat nebo `tcp_close()` a `tcp_abort()` pro řádné ukončení spojení či okamžité přerušení spojení.

Příklad velmi jednoduché aplikace TCP echo serveru je uveden na obrázu 4.7. Podstatou echo serveru je vracení všech přijatých dat zpět druhému konci spojení. Uvedená aplikace však předpokládá u TCP implementace podporu úschovy odchozích dat ve vyrovnávací paměti a také neprovádí kontrolu na úspěšný zápis všech dat do TCP socketu při volání `tcp_write()`.

Kromě základních funkcí `tcp_read()`a `tcp_write()` poskytuje TCP implementace aplikaci další funkce, kterými může aplikace provádět přímé řízení toku dat či získat informace o velikosti okna druhého konce spojení a podle ní přizpůsobit své chování.

4.3.7 UDP

Implementace UDP (User Datagram Protocol) je na rozdíl od TCP protokolu velmi jednoduchá. Jelikož UDP zajišťuje nespojovanou a nespolehlivou komunikaci, není potřeba uchovávat žádné informace spojené s příjmem či odesíláním dat, a proto

```
void callback(TCP_SOCKET *s, uint8_t event)
{
  void *buf; uint16_t length;
  switch(event) {
  case TCPEV_RECV: /* přijata data */
         tcp_read(s, &buf, &length);
         tcp_write(s, buf, length);    /* odešleme přijatá data */
         break;
  case TCPEV_CLOSE_RQ: /* požadavek na ukončení spojení */
         tcp_close(s);
  }
}
int main()
{
  /* .. inicializace TCP/IP knihovny a celého systému .. */

  tcp_listen(1025, TCP_WND, callback);
  /* TCP_WND je velikost našeho okna*/

  /* hlavní smyčka pro TCP/IP knihovnu */
  for(;;) {
    net_poll();
  }
}
```

Obrázek 4.7: Příklad aplikace TCP echo serveru

nejsou použity sockety ani jiné datové struktury. Chce-li aplikace přijímat UDP datagramy, zaregistruje si u UDP modulu obslužnou funkci, která je automaticky volána při příjmu datagramu. Pro odeslání datagramu stačí zavolat pouze příslušnou funkci, která připraví UDP hlavičku, doplní zadaná data i kontrolní součet a zhotovený datagram odešle.

4.3.8 TFTP

TFTP (Trivial File Transfer Protocol) je jednoduchý protokol pro přenos souborů nad UDP. Jeho imlementace v S8 Serveru zahrnuje funkce pro klienta i server. Jelikož UDP nezajišťuje spojovaný ani spolehlivý přenos, bylo nutné tyto mechanismy realizovat přímo v TFTP. To se uskutečnilo zavedením tzv. TFTP socketu (analogie TCP socketu), který je popsán na obrázku 4.8.

Narozdíl od TCP nejsou přenášená data přímo číslována, nýbrž jsou číslovány jednotlivé datagramy s těmito daty. Také není zavedeno okno příjemce ani odesílatele,

```
typedef struct tftp_socket_t {
  char *filename;       /* jméno přenášeného souboru */
  uint8_t flags;
  uint16_t block;       /* číslo aktuálního bloku */
  void *dgram;          /* poslední odeslaný datagram */
  uint16_t dgram_len;   /* délka datagramu */
  uint8_t rexmits;      /* počet přeposlání */
  uint32_t lastsent;    /* časový údaj o posledním odeslání */
  IP_ADDR remote_ip;    /* IP adresa druhého konce spojení */
  uint16_t remote_port; /* UDP port druhého konce spojení */
  uint16_t local_port;  /* lokální UDP port */
  struct tftp_socket_t *next;
} TFTP_SOCKET;
```

Obrázek 4.8: Struktura TFTP socketu

tudíž odeslaný datagram musí být nejprve potvrzen druhou stranou spojení než může být odeslán další datagram – technika jednotlivého potvrzování. Jakmile jsou data přenesena, socket je uvolněn z paměti.

4.3.9 HTTP

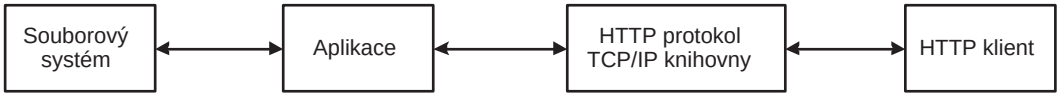
HTTP (Hypertext Transfer Protocol) je protokol aplikační vrstvy využívající služby TCP pro přenos dokumentů, nejčastěji HTML stránek nebo obrázků. V S8 Serveru byla implementována podpora verze 1.0 tohoto protokolu zahrnující funkce nezbytné pro chod jednoduchého HTTP serveru. Tyto funkce zahrnují pouze vyřizování požadavků GET a POST, kterými připojení klienti mohou požádat o získání dokumentů uložených na serveru (metoda GET) či zaslat určité údaje na server ke zpracování (metoda POST).

Po spuštění HTTP serveru je vytvořen naslouchací TCP socket na portu 80. Při příchozím spojení je nejprve vytvořen nový naslouchací socket, aby bylo možné obsloužit i další potenciální klienty a původní socket je pevně svázán s připojeným klientem. Následně se provede analýza požadavku klienta, je vyvolána aplikace stojící nad HTTP vrstvou, která rozhodne o akceptování či zamítnutí požadavku a klientovi je vrácena příslušná odpověď. Pokud se jednalo o požadavek GET, který aplikace akceptovala, následuje přenos dokumentu, který si klient metodou GET vyžádal. Nakonec je TCP spojení uzavřeno.

Souborový systém a CGI skripty

Pro organizaci webových stránek HTTP serveru je k dispozici jednoduchý souborový systém. Soubory jsou uloženy v některé z připojených pamětí S8 Serveru, tj. buď v EEPROM nebo přímo ve Flash paměti společně s kódem aplikace. Aplikace

zde funguje jako prostředník mezi HTTP protokolem a souborovým systémem, jelikož při příchodu klientova požadavku GET pouze otevře zadaný soubor (pokud existuje) a při přenosu kopíruje data ze souboru do HTTP vrstvy, která zajišťuje jejich odesílání klientovi. Toto schéma se nachází na obrázku 4.9.



Obrázek 4.9: Mechanismus zasílání souboru HTTP klientovi

Z hlediska většiny aplikací, které přes HTTP protokol nabízejí svá data, je mnohem zajímavější mít statické soubory jen jako šablonu, do níž je možné doplňovat aktuální údaje (např. naměřené hodnoty z periferií). Z tohoto důvodu byl implementován jednoduchý mechanismus známých CGI skriptů, který je přímo napojený na souborový systém. Aplikace nyní nečte data ze souborového systému, nýbrž přes CGI modul. Ten vždy načte do paměti požadovaný blok souboru, vyhledá v něm všechny CGI příkazy a pro každý z nich vyvolá aplikaci, která se postará o doplnění správných dat za tyto příkazy.

Obrázek 4.10: Mechanismus zasílání souboru HTTP klientovi

Ukázka jednoduché HTML stránky s CGI příkazy je na obrázku 4.11 v levé části. Příkaz vždy začíná znakem „$“, za kterým následuje jeho identifikátor, což může být číslo nebo řetězec. V pravé části se nachází již výsledný soubor po doplnění aktuálního údaje času a data.

4.3.10 SMTP

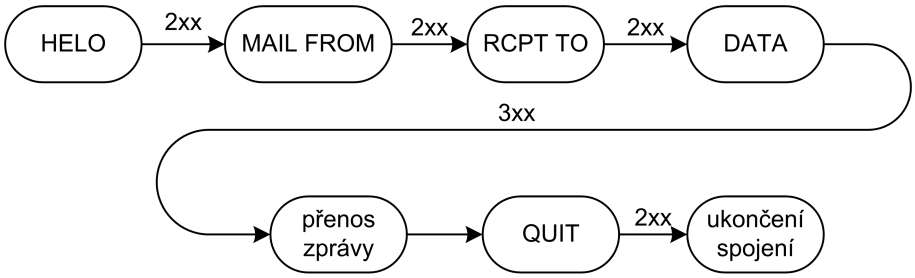
SMTP (Simple Mail Transfer Protocol) je protokol aplikační vrstvy nad TCP protokolem na portu 25 umožňující zasílání e-mailových zpráv. Byla implementována jen klientská část SMTP, která dovoluje v jeden okamžik zaslat pouze jednu e-mailovou zprávu specifikovanou adresou odesílatele, adresou příjemce, předmětem a obsahem. Žádné další funkce, jako je např. zasílání příloh, nejsou podporovány.

Samotná komunikace se SMTP serverem je při zasílání e-mailové zprávy řízena jednoduchým konečným automatem, který se nachází na obrázku 4.12.

Jeden stav automatu odpovídá zaslání SMTP příkazu (na obrázku jsou vepsány přímo do stavu) na SMTP server, který musí odpovědět kódem a případným textovým hlášením (kód je na obrázku zapsán nad šipkami, znak „x“ nahrazuje libovolnou cifru). Odpoví-li server jiným kódem, než se očekává, spojení je ihned přerušeno

```
<html>                                  <html>
<head>                                  <head>
<title>S8 Server</title>                <title>S8 Server</title>
</head>                                 </head>
<body>                                  <body>
<p>Welcome to S8 Server</p>             <p>Welcome to S8 Server</p>
<p>Current time is $time</p>            <p>Current time is 15:11</p>
<p>Current date is $date     </p>       <p>Current date is 24.04.2006</p>
</body>                                 </body>
</html>                                 </html>
```

Obrázek 4.11: Ukázka jednoduché HTML stránky před a po zpracování CGI příkazů



Obrázek 4.12: Konečný automat pro odeslání e-mailové zprávy SMTP protokolem

a aplikaci, která zadala příkaz k odeslání e-mailu, je ohlášena chyba. Naopak dojde-li konečný automat až do posledního stavu, ve kterém je spojení řádně uzavřeno, aplikace je informována o úspěšném odeslání e-mailu.

4.3.11 DNS

DNS (Domain Name System) protokol aplikační vrstvy slouží pro přenos Resource Records (zdrojové věty), které obsahují informace o doménových jménech a jim příslušejících IP adresách a ostatní informace distribuované službou DNS. Jedná se o bezestavový protokol využívající k přenosu dat jak UDP tak i TCP protokol na portu 53.

V S8 Serveru byl implementován pouze DNS klient nad UDP vrstvou zajišťující překlad doménového jména na IP adresu. Aplikace předá DNS klientovi doménové jméno pro překlad a ten vyšle rekurzivní dotaz k zadanému jménnému serveru. Odpověď od jmenného serveru, který ji získal buď přímo ze své vyrovnávací paměti nebo přeposláním rekurzivního dotazu jiným serverům, by pak měla obsahovat IP adresu příslušející doménovému jménu. Pokud odpověď nepřijde do uplynutí určitého časového limitu, je aplikace informována o neúspěchu překladu.

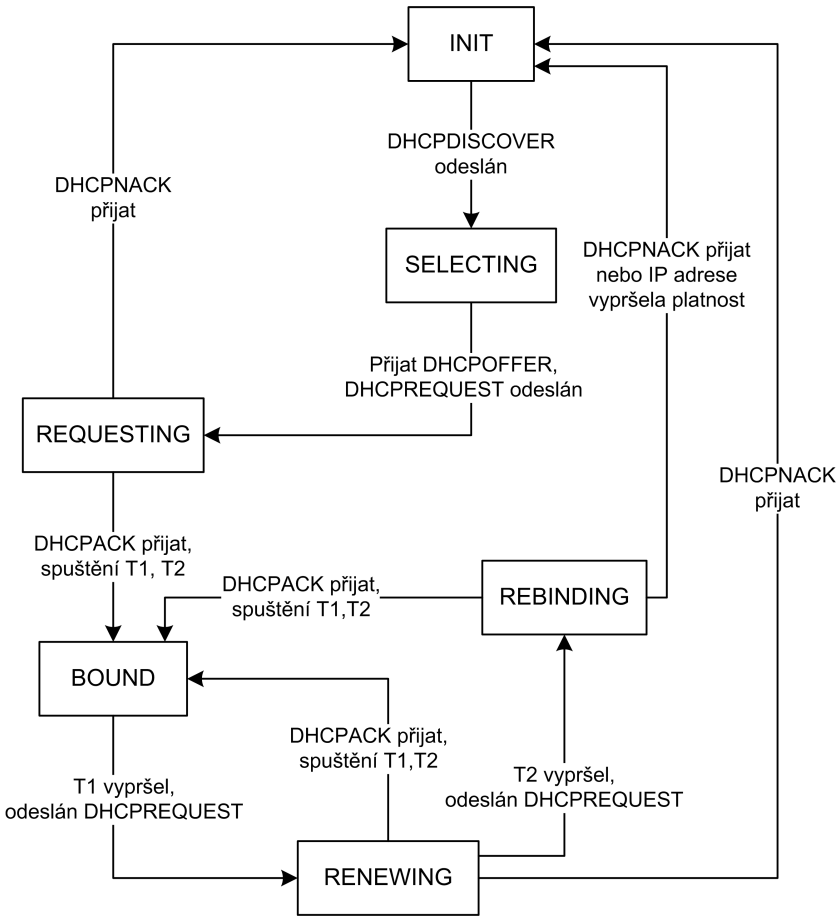
4.3.12 DHCP

DHCP (Dynamic Host Configuration Protocol) je aplikační protokol využívající UDP na portech 67 (server) a 68 (klient) pro dynamickou konfiguraci klienta. Ta zahrnuje hlavně přidělení IP adresy a s ní souvisejících informací jako např. adresa nejbližšího směrovače či DNS jmenného serveru.

Právě díky podpoře DHCP protokolu se stává S8 Server univerzálním síťovým zařízením, které je možné zapojit téměř do libovolné sítě aniž by bylo nutné měnit nastavení IP adres, což znamená jeho přeprogramování.

Implementace DHCP klienta v S8 Serveru je oproti jiným značně zjednodušena hlavně z důvodu úspory prostoru tak jako u všech ostatních protokolů popisovaných v této kapitole. Pro dosažení co nejmenší implementace bylo nutné udělat ve stavovém diagramu DHCP klienta, který je popsaný v RFC 2131 dokumentu (viz obrázek 2.7 v kapitole Rodina protokolů TCP/IP), několik malých změn. Výsledný stavový diagram, podle kterého byla implementace prováděna, je na obrázku 4.13.

Časovače T1 a T2 mají stejný význam, jak bylo popsáno v sekci 2.10 v kapitole Rodina protokolů TCP/IP. Provedené změny se týkají odebrání stavů INIT-REBOOT a REBOOTING, které jsou využity pouze v případě, že DHCP klient na začátku vyžaduje přidělení explicitně zadané IP adresy. Také činnost ve stavu SELECTING byla zjednodušena – neshromažďují se DHCPOFFER nabídky, nýbrž se odpoví na první z těchto příchozích nabídek.



Obrázek 4.13: Statový diagram DHCP klienta implementovaného v S8 Serveru

4.4 Velikost kódu TCP/IP knihovny

Tabulka 4.2 udává přehled velikosti každého modulu TCP/IP knihovny pro procesory ATmega161 a AT90S8515 v bytech. K překladu byl použit kompilátor avr-gcc [2] s volbou pro maximální optimalizaci velikosti výsledného kódu a k získání velikosti kódu nástroj avr-size.

Modul	**Velikost kódu (v bytech)**	
	AT90S8515	**ATmega161**
Ethernet	174	160
ARP	818	782
IP	1 220	1 126
IP/TCP/UDP checksum	224	204
ICMP	188	156
UDP	540	482
TCP minimální	3 330	3 134
TCP maximální	5 278	4 902
DNS	804	776
DHCP	1 817	1 762
HTTP	778	716
SMTP	1135	1 095
TFTP server	1 442	1 260
TFTP klient	1 364	1 326

Tabulka 4.2: Velikost kódu TCP/IP protokolů

Je patrné, že největší podíl zaujímá TCP protokol, který i při své minimální konfiguraci bez výpočtu doby obrátky segmentů, bez podpory úschovy odchozích dat a algoritmů pro zamezení zahlcení zabírá přes 3 KB. Je to způsobeno hlavně celkovou komplexností tohoto protokolu a také častým používáním 32bitové aritmetiky, která je na 8bitových AVR procesorech poměrně drahá co se týká počtu strojových instrukcí na jednu 32bitovou operaci.

Do velikosti modulů v tabulce však nejsou započítány moduly S8 operačního systému, na jejichž implementaci je TCP/IP knihovna závislá. Jedná se o modul dynamické alokace paměti a ovladač ethernetového řadiče. Reálnou velikost kódu funkční aplikace jednoduchého HTTP serveru, který pro svou činnost využívá přímo TCP API a nikoli modul HTTP, podává tabulka 4.3. Jedinou činností tohoto serveru je poskytování statické HTML stránky, která je přímo součástí kódu aplikace.

Modul	**Velikost kódu (v bytech)**	
	AT90S8515	**ATmega161**
S8 OS	1 140	1 076
moduly stndardní knihovny jazyka C	237	361
Ethernet, ARP, IP, minimální TCP	5 964	5 562
HTTP server	821	843
Celkem	**8 162**	**7 842**

Tabulka 4.3: Velikost kódu aplikace HTTP serveru

Kapitola 5

Konfigurace a překlad software

Tato kapitola se zabývá konfigurací a překladem software pro S8 Server včetně stručného popisu nástrojů, které byly pro tyto účely vyvinuty.

5.1 Princip překladu

Většinu zdrojových modulů TCP/IP knihovny a také některé moduly S8 OS je možné konfigurovat při jejich překladu definováním či naopak nedefinováním symbolů pro C/C++ preprocesor. Např. při definování symbolu _TCP_CALC_RTT bude v TCP implementaci zahrnut výpočet střední doby obrátky TCP segmentů a z něj vycházející výpočet časového limitu pro přeposílání segmentů. Není-li tento symbol definován, bude při překladu kód pro tyto výpočty vypuštěn, což je zajištěno direktivou pro podmíněný překlad `#ifdef`.

Díky tomuto principu konfigurace zdrojových modulů není možné uložit přeložený kód modulů TCP/IP a S8 OS do statické knihovny, která by se při překladu aplikací pouze přilinkovala k výslednému kódu, nýbrž je nutné provádět jejich překlad vždy spolu s aplikací. K tomuto překladu však musí programátor vybrat pouze ty zdrojové moduly TCP/IP nebo S8 OS, které jeho aplikace skutečně využije.

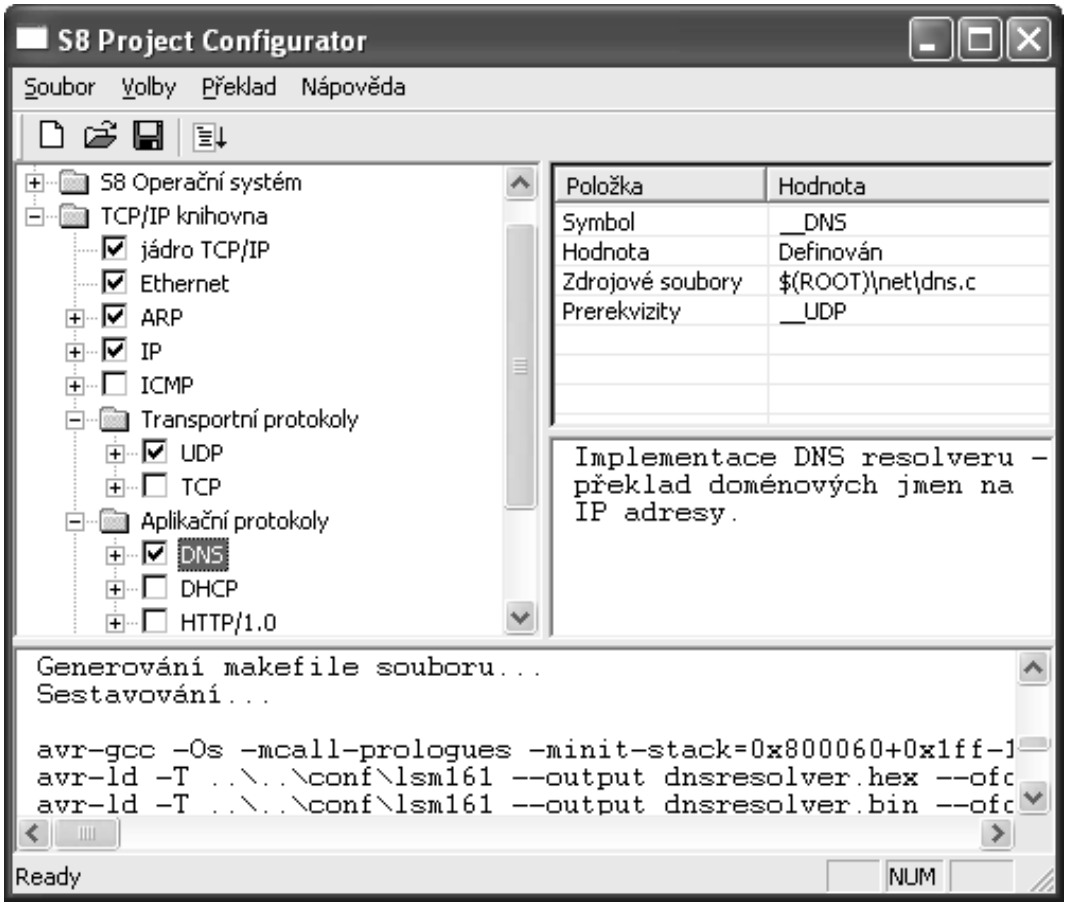
5.2 Použitý C/C++ překladač

Díky velkému rozšíření AVR mikroprocesorů je v dnešní době k dispozici celá řada komerčních kompilátorů i některé volně dostupné (v rámci GPL licence) kompilátory pro tato zařízení. Jedním z nejpoužívanějších je nekomerční překladač avr-gcc [2], který byl použit také v této bakalářské práci. Kromě překladu kódu psaného v jazyce C a C++ umožňuje také překlad zdrojových kódů v assembleru. Pro překlad aplikací na S8 Server byl použit avr-gcc verze 3.4.5.

5.3 S8 Project Configurator

Za účelem snadné a pohodlné konfigurace aplikací byl vytvořen jednoduchý program pro Windows nazvaný S8 Project Configurator. Ten umožňuje pro každou aplikaci vytvořit projekt zahrnující seznam zdrojových souborů aplikace a veškeré konfigurační údaje pro překlad (symboly pro C/C++ preprocesor a jejich hodnoty) získané z tzv. konfigurační šablony popisující parametry TCP/IP knihovny a S8 OS.

Na základě tohoto projektového souboru je vygenerován soubor makefile sloužící jako vstup pro sestavovací program make, který je součástí nástrojů avr-gcc. Ten zajistí kompletní překlad vybraných zdrojových kódů TCP/IP knihovny, S8 OS a zdrojových kódů projektu na výsledný spustitelný obraz aplikace.



Obrázek 5.1: Aplikace S8 Project Configurator

Aplikace S8 Project Configuratoru je znázorněna na obrázku 5.1. V levé části obrazovky se nachází hierarchická struktura modulů TCP/IP knihovny a S8 operačního systému sestavená z konfigurační šablony, v pravé části informace týkající se zvoleného uzlu této struktury a ve spodní části se nachází okno s výstupem sestavovacího programu.

5.3.1 Konfigurace

Jednotlivé uzly hierarchické struktury TCP/IP a S8 OS modulů popisují buď jeden zdrojový modul (např. pro UDP protokol se jedná o modul udp.c) nebo nastavení či parametr modulu. Má-li být modul součástí aplikace, pak stačí příslušný uzel označit kliknutím myši. Stejný postup lze aplikovat na uzel týkající se nastavení modulu (např. povolení výpisu ladících informací na sériový port). Uzlům, které odpovídají určitým parametrům modulu (např. velikost vyrovnávací paměti pro příjem dat z UART), lze opět kliknutím myši nastavit požadovanou číselnou hodnotu.

Při výběru modulů TCP/IP knihovny a S8 OS výše popsaným způsobem jsou také automaticky zohledňovány závislosti mezi nimi. To znamená, že při označení modulu IP bude automaticky označen také modul protokolu Ethernet a ethernetového řadiče, protože pro odeslání IP paketu volá IP protokol funkce Ethernetu a ten po vložení paketu do rámce využívá funkce ethernetového řadiče pro jeho odeslání. Všechny tyto závislosti jsou popsány v konfigurační šabloně a uživatel si je může případně upravit podle svých potřeb. Podrobný popis struktury konfiguračního souboru je uveden v nápovědě k aplikaci S8 Project Configurator.

5.3.2 Překlad

Na základě uživatelem provedených nastavení je vygenerován makefile soubor zahrnující všechny konfigurační údaje. Přímo v aplikaci S8 Project Configuratoru je pak možné spustit program make (či jiný zvolený sestavovací program) pro provedení překladu. Jeho výstup bude zobrazen v dolní části okna aplikace.

5.4 Další podpůrné nástroje

Kromě programu S8 Project Configurator byly vyvinuty další dva podpůrné nástroje pro překlad S8 aplikací. Jedná se o program fsbuilder pro zabalení adresáře do binárního souboru ve formátu čitelném pro souborový systém S8 OS a program bin2c pro převod binárních souborů do pole znaků ve zdrojovém souboru jazyka C. Oba tyto nástroje jsou konzolovými aplikacemi a nejsou nijak integrovány do prostředí programu S8 Project Configurator.

5.4.1 fsbuilder

fsbuilder (File System Builder) je konzolová aplikace pod Windows, jejíž vstupem je adresář a výstupem je binární soubor s obsahem tohoto adresáře včetně všech jeho podadresářů. Tento binární soubor je čitelný S8 souborovým systémem, což lze využít hlavně u aplikací typu HTTP server, kdy je adresář obsahující HTML stránky, obrázky či další dokumenty zabalen do binárního souboru, ten je pak např. exportován do externí EEPROM paměti S8 Serveru odkud jsou dokumenty načítány a zasílány HTTP klientům.

5.4.2 bin2c

bin2c (Binary To C) je jednoduchá konzolová aplikace pod Windows pro převod binárního souboru do pole znaků ve zdrojovém souboru jazyka C. Byla vytvořena hlavně pro podporu nástroje fsbuilder pro ty případy, kdy je žádoucí, aby výsledný binární soubor vygenerovaný aplikací fsbuilder byl přímo součástí kódu aplikace. V takovém případě jej stačí převést do zdrojového souboru jazyka C a zkompilovat spolu s aplikací.

5.5 Instalace aplikace do AVR

Po úspěšném přeložení aplikace do spustitelného souboru již zbývá pouze instalace tohoto souboru do interní Flash paměti AVR procesoru, po jejímž dokončení je aplikace spuštěna. Pro tyto účely byla využit program avreal [1], který po připojení AVR procesoru přes ISP konektor paralelním kabelem do klasického PC zajistí stažení aplikace do Flash paměti a její spuštění. Vstupem tohoto programu je obraz aplikace ve formátu Intel Hexadecimal Object File, což je jeden z podporovaných výstupních formátů překladače avr-gcc.

Kapitola 6

Podobné projekty

V prostředí Internetu se dnes můžeme setkat s celou řadou komerčních i nekomerčních projektů realizujících TCP/IP protokoly pro malá zařízení podobného charakteru jako je S8 Server. Tato kapitola přiblíží některé z těchto projektů a také poskytne srovnání s vytvořenou TCP/IP implementací.

6.1 PicoWeb

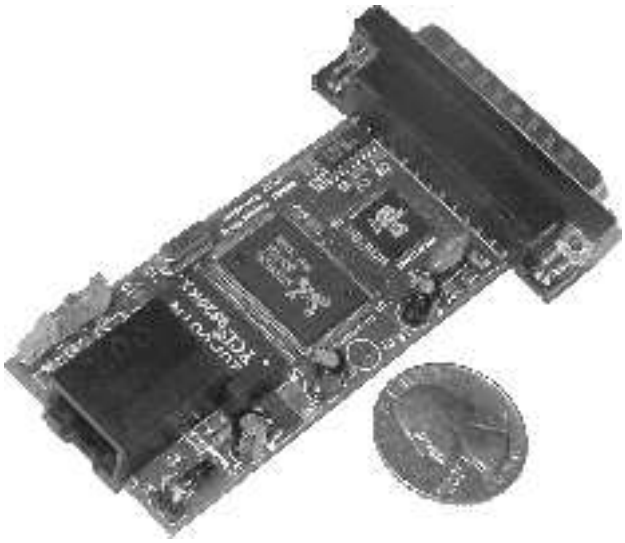
PicoWeb [25] je projekt, jehož snahou je být nejmenším prakticky použitelným webovým serverem na světě. Je realizován na AVR procesoru AT90S8515 rychlosti 8 MIPS, ke kterému je připojen ethernetový řadič a 32 kilobytová EEPROM paměť (viz obrázek 6.1). Jelikož tento procesor nabízí pouze 8 kilobytů Flash paměti pro kód, rozhodli se autoři projektu umístit kód aplikací do externí EEPROM paměti a jeho provedení zajistit interpretrem umístěným v rámci 8kilobytové Flash paměti. Kód aplikace je v EEPROM uložen v instrukcích tzv. p-kódu, který narozdíl od nativního kódu AVR procesoru podporuje 16bitovou aritmetiku a byl navržen s ohledem na efektivní běh PicoWeb aplikací.

TCP/IP knihovna PicoWebu nabízí jednoduchou implementaci HTTP a SMTP protokolu, což dovoluje aplikacím fungovat jako webový server a zasílat e-mailové zprávy. Pro podporu webového serveru zahrnuje tento projekt i jednoduchý souborový systém podobně jako S8 Server.

Hlavní nevýhodou projektu je načítání kódu aplikací z EEPROM paměti, jejíž čtení je řádově pomalejší než čtení z klasické Flash paměti. Načtené p-kód instrukce musejí být navíc ještě interpretovány, takže celková rychlost aplikací a síťové komunikace vůbec se pohybuje hluboko pod výkonem 8 MIPS, které AVR procesor nabízí. Nevýhodou je také nepřenositelnost TCP/IP knihovny, která je psána přímo v instrukcích p-kódu.

Oproti PicoWebu tedy S8 Server nabízí poměrně velmi rychlou komunikaci [1], což může hrát roli u aplikací náročných na časově efektivní přenos dat.

[1] Při připojení S8 Serveru přímo k PC síťovým kabelem byla naměřena přenosová rychlost kolem 2 Mbit/s.



Obrázek 6.1: Hardware PicoWeb projektu

6.2 lwIP

lwIP (Lightweight TCP/IP Stack) [11] [12] je přenositelná a na platformě zcela nezávislá TCP/IP knihovna psaná v jazyce C navržená pro účely vestavěných systémů s kódovou pamětí v řádu desítek kilobyte a RAM pamětí v řádku jednotek kilobyte. Jedná se o open source projekt, takže zdrojové kódy jsou volně přístupné a sloužily také jako hlavní inspirace pro vlastní TCP implementaci na S8 Serveru. lwIP podporuje protokoly ARP, IP verze 4 i verze 6 , ICMP, UDP, TCP, PPP a DHCP. Hlavní důraz zde byl kladen na TCP protokol, jehož modul zabírá ve výsledném kódu také největší podíl. Kromě všech algoritmů pro zamezení zahlcení, řízení toku dat a výpočtu RTT podporuje samozřejmě i úschovu odchozích dat a také příchozích TCP segmentů mimo pořadí.

Srovnání velikosti implementace protokolů knihovny lwIP a S8 Serveru je uvedeno v tabulce 6.1.

Modul	**Velikost kódu (ATmega161)**	
	lwIP	**S8 Server**[2]
IPv4	1 460	1 126
ICMP	712	156
TCP	**14 696**	**4 902**
UDP	1 676	482
DHCP	5 362	1 762

Tabulka 6.1: Srovnání implementací lwIP a S8 Serveru

[2]TCP modul byl nakonfigurován s plnou podporou výpočtu RTT, algoritmů pro zamezení zahlcení a úschovy odchozích dat ve vyrovnávací paměti.

Propastný rozdíl mezi velikostí TCP modulů je způsoben tím, že autor lwIP vytvořil spíše robustnější implementaci, která nabízí rozhraní na bázi BSD socketů, což umožňuje aplikacím nad ní pracovat s TCP proudem dat jako se souborem, a to pouze s jednoduchými operacemi čtení a zápisu. V případě S8 Serveru si aplikace musí být vědoma, že používá pro přenos dat protokol TCP a komunikace s TCP modulem je pro ni z tohoto pohledu náročnější než pouze zapsaní či přečtení dat.

6.3 uIP

uIP (mikro IP) [13] je velmi odlehčená verze lwIP pocházející od stejného autora. Jedná se rovněž o open source projekt, jehož cílem je minimální implementace protokolů ARP, IP, ICMP a TCP v jazyce C pro 8bitové mikroprocesory s kódovou pamětí v řádu jednotek kilobytů a RAM pamětí v řádu stovek bytů. Zvláštností této implementace je fakt, že realizace IP, ICMP a TCP se nacházejí v jediném zdrojovém modulu a také není používána žádná dynamická alokace paměti. Samotná implementace TCP je oproti lwIP nebo S8 Serveru velmi zjednodušená a umožňuje zasílání TCP segmentů pouze metodou jednotlivého potvrzování, což je při TCP spojeních, kde obě komunikující strany leží v sítích s rozdílnými přenosovými rychlostmi, velmi nevýhodné a umožňuje pouze velmi pomalý datový přenos. Na druhou stranu v rámci lokálních sítí, kde se předpokládá velmi rychlá přenosová rychlost, je tato technika vyhovující.

Modul	**Velikost kódu (ATmega161)**	
	uIP	**S8 Server**
Alokace paměti, časovač	0	408
Checksum	502	204
ARP	1 314	782
IP,ICMP, TCP	3 256	4 416
Celkem	**5 072**	**5 810**

Tabulka 6.2: Srovnání implementací uIP a S8 Serveru

I když tabulka 6.2 srovnávající uIP a S8 Server svědčí ve prospěch uIP implementace, rozdíl 738 bytů již není tak markantní, uvážíme-li, že S8 Server nabízí v rámci TCP protokolu techniku kontinuálního potvrzování – tj. v jeden okamžik může mezi příjemcem a odesílatelem putovat v síti větší počet TCP segmentů.

Kapitola 7

Závěr

Cílem této práce bylo splnění dvou úkolů: implementovat funkce pro řízení hardware S8 Serveru a vytvoření TCP/IP knihovny pro síťovou (internetovou) komunikaci s důrazem na její co nejmenší velikost, protože paměťová kapacita obou AVR procesorů, které byly k dispozici, je velmi omezená.

Splnění prvního úkolu vyžadovalo hlavně seznámení se se základními principy fungování hardware jako je sériový port, čítač/časovač či ethernetový řadič a napsání ovladačů pro ně, čehož výsledkem je jednoduchý operační systém nazvaný S8 OS, který aplikacím S8 Serveru tento hardware přímo zpřístupňuje.

V rámci druhého úkolu byla vytvořena knihovna vybraných protokolů z rodiny TCP/IP, která umožňuje aplikacím S8 Serveru fungovat jako webový server, zasílat e-mailové zprávy, provádět překlad doménových jmen na IP adresy či automaticky získávat IP adresu a další informace o síti z dostupného DHCP serveru. Díky modularitě celé knihovny lze navíc pro každou aplikaci před kompilací vybrat pouze ty moduly TCP/IP protokolů, které budou skutečně využity, čímž se ušetří paměťový prostor ve prospěch aplikace. Velikost implementace jednotlivých protokolů TCP/IP knihovny dokonce předčila původní očekávání (viz sekce 4.4), jelikož se ukázalo, že např. aplikace plně funkčního webového serveru si vystačí pouze s 8 kilobyty kódové paměti, což je ve srovnání s ostatními plnohodnotnými implementacemi TCP/IP velmi uspokojivý výsledek (viz kapitola 6).

Tato práce nezahrnuje testy korektnosti a výkonnosti vytvořené implementace a nebylo ani provedeno řádné testování jednotlivých protokolů kromě jednoduchých testů prováděných při jejich vývoji. Některé nástroje pro tyto účely lze najít v [24]. Jejich využití k ověření vytvořené TCP/IP knihovny by bylo dalším přirozeným krokem této práce.

Námětem na budoucí práci může být také rozšíření TCP/IP knihovny o další užitečné protokoly jako např. NTP (Network Time Protocol) [23] pro nastavení přesného času modulu hodin reálného času, který je součástí S8 Serveru a jehož nastavení se musí provádět „ručně“. Na zvážení by byla také podpora mechanismů pro zajištění bezpečnosti S8 Serveru, což by zahrnovalo hlavně testování odolnosti proti možným vnějším útokům a následné provedení opatření, která by eliminovala vliv těchto útoků na správnou činnost serveru, či implementaci protokolů pro bezpečnou komunikaci S8 Serveru s vybranými klienty (např. HTTPS protokol [31]).

Literatura

[1] avreal ISP programátor, http://ln.ua/∼real/avreal/index_e.html.

[2] avr-gcc překladač, http://gcc.gnu.org/.

[3] Berners-Lee T., Fielding R., Frystyk H.: *Hypertext Transfer Protocol - HTTP/1.0*, RFC 1945, IETF, 1996.

[4] Berners-Lee T., Fielding R., Gettys J., Mogul J., Frystyk H., Masinter L., Leach P.: *Hypertext Transfer Protocol – HTTP/1.1*, RFC 2616, IETF, 1999.

[5] Braden R.: *Requirements for Internet Hosts – Communication Layers, RFC 1122*, Internet Engineering Task Force, 1989.

[6] Claffy K., Shannon C. , Moore D.: *Beyond folklore: Observations on Fragmented Traffic*, IEEE/ACM Transactions on Networking, 2002.

[7] Clark D. D.: *IP Datagram Reassembly Algorithms*, RFC 815, IETF, 1992.

[8] Cooper G. H.: TinyTCP, http://www.unusualresearch.com/tinytcp/.

[9] Croft B., Gilmore J.: *Bootstrap Protocol (BOOTP)*, RFC 951, IETF, 1985.

[10] Droms R.: *Dynamic Host Configuration Protocol*, RFC 2131, IETF, 1997.

[11] Dunkels A.: Lightweight TCP/IP Stack, http://www.sics.se/∼adam/lwip/.

[12] Dunkels A.: *Minimal TCP/IP implementation with proxy support*, Swedish Institute of Computer Science, 2001.

[13] Dunkels A.: uIP projekt, http://www.sics.se/∼adam/uip/.

[14] *The Ethernet – A Local Area Network*, Version 1.0, Digital Equipment Corporation, Intel Corporation, Xerox Corporation, 1980.

[15] Ethernut projekt, http://www.ethernut.de/.

[16] FreeRTOS operační systém, http://www.freertos.org/.

[17] Jacobson V.: *Congestion avoidance and control*, konference SIGCOMM '88, Stanford, California, 1988.

[18] Mockapetris P.: *Domain Names - Implementation and Specification*, RFC 1035, IETF, 1987.

[19] Kadak KwikNET TCP/IP Stack, http://www.kadak.com/tcp_ip/tcpip.htm.

[20] Karn P., Partridge C.: *Improving Round-trip Time Estimates in Reliablie Transport Protocols*, konference SIGCOMM '87, Stowe, Vermont, 1987.

[21] Klensin J.: *Simple Mail Transfer Protocol*, RFC 2821, IETF, 2001.

[22] Liquorice projekt, http://liquorice.sourceforge.net/.

[23] Mills D. L.: *Network Time Protocol (Version 3) Specification, Implementation and Analysis*, RFC 1305, IETF, 1992.

[24] Parker S., Schmechel C.: *Some Testing Tools for TCP Implementors*, RFC 2398, IETF, 1998.

[25] PicoWeb projekt, http://www.picoweb.net/.

[26] Plummer D. C.: *An Ethernet Address Resolution Protocol*, RFC 826, IETF, 1982.

[27] Postel J.: *Internet Control Message Protocol*, RFC 792, IETF, 1981.

[28] Postel J.: *Internet Protocol*, RFC 791, IETF, 1981.

[29] Postel J.: *Transmission Control Protocol*, RFC 793, IETF, 1981.

[30] Postel J.: *User Datagram Protocol*, RFC 768, IETF, 1980.

[31] Rescorla E., Schiffman A.: *The Secure HyperText Transfer Protocol*, RFC 2660, IETF, 1999.

[32] Sollins K.: *The TFTP Protocol*, RFC 1350, IETF, 1992.

[33] Stone M. S., Wilson P. R.: *The Memory Fragmentation Problem: Solved?*, ACM, 1998.

Příloha A

Obsah CD-ROM

Součástí této práce je i přiložený CD-ROM obsahující zdrojové kódy TCP/IP knihovny, S8 operačního systému a ukázkových aplikací S8 Serveru. Adresářová struktura CD-ROM je následující:

- `/conf` – konfigurační soubory pro překlad S8 aplikací
- `/doc` – veškerá dokumentace zahrnující text této práce, programátorskou příručku a katalogové listy k jednotlivým komponentám S8 Serveru
- `/doxy` – soubory programu Doxygen pro generování programátorské dokumentace
- `/examples` – ukázky S8 aplikací
- `/include` – hlavičkové soubory TCP/IP knihovny a S8 operačního systému
- `/libc` – zdrojové kódy některých funkcí z knihovny jazyka C
- `/install`– instalační soubory avr-gcc překladače verze 3.4.5 a avreal ISP programátoru
- `/net` – zdrojové kódy TCP/IP knihovny
- `/os` – zdrojové kódy S8 operačního systému
- `/tools` – podpůrné nástroje pro překlad S8 aplikací
- `doxyfile` – konfigurační soubor pro program Doxygen
- `readme.txt` – soubor s popisem obsahu CD-ROM