UNIVERZITA PARDUBICE

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

BAKALÁŘSKÁ PRÁCE

2010 Martin Mariška

UNIVERZITA PARDUBICE
Fakulta elektrotechniky a informatiky

Komunikační systém jednoduchého průmyslového řídicího systému

Martin Mariška

Bakalářská práce
2010

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2009/2010**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Martin MARIŠKA**
Osobní číslo:	**I07711**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Komunikační systém jednoduchého průmyslového řídicího systému**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování:

Řídicí systém se skládá z počítače, na kterém běží řídicí proces a z připojených periferií. Systém musí umožňovat obousměrnou komunikaci mezi PC a periferiemi. To znamená, že řídicí modul musí jednak umožnit posílání řídicích příkazů do periferií, jednak číst a reagovat na požadavky přicházející z periferií. Pro výměnu informací mezi PC a periferiemi je použit textový formát s volitelným kódováním (ASCII, Win1250, Unicode). Na PC je spuštěn modul obsahující server TCP. Server čeká na definovaném portu na přihlášení se periferie. Na periferii se po zapnutí napájení vytvoří TCP klient a pokusí se na definovaném portu připojit k PC. Protože se předpokládá, že více periferií může být zapnuto ve stejný okamžik, je třeba vyřešit konflikty, kdy se k serveru pokouší připojit více klientů najednou. Na periferii je zároveň vytvořen TCP/IP server na pevně nastaveném portu. Tento server umožňuje serveru na PC oznámit periferii, že po přerušení činnosti je opět připraven a periferie na to reaguje navázáním spojení přes svého klienta. Po navázání spojení mezi klientem a serverem se na serveru vytvoří nové vlákno (thread) pro komunikaci s příslušným klientem, který kromě jiného v určitých časových intervalech ověřuje, zda je klient ještě připojen. Před ukončením činnosti serveru to server pomocí zpráv oznámí připojeným periferiím. Podobně klient si hlídá, zda je server aktivní. Pokud není, pokouší se s ním v určitých časových prodlevách obnovit spojení. Reakce na jednotlivé události na klientech jsou uloženy v databázi Firebird. Jestliže tedy klient pošle informaci o změně stavu v textovém tvaru, server z databáze získá potřebnou odezvu a odešle ji v textovém tvaru zpět klientovi. Server je realizován pomocí standardního PC s operačním systémem WinXP a je naprogramován v jazyce Delphi s použitím komponent Indy pro zajištění TCP/IP přenosů. Data jsou uložena v databázovém systému typu klient-server Firebird. Server je propojen s periferiemi sítí Ethernet 100mb/s s PC pomocí protokolu TCP/IP. Periferie jsou realizovány mikropočítačem na bázi mikroprocesoru rodiny Atmel, předpokládá se max. 200 připojených stanic. Programové vybavení je napsáno v jazyce ANSI C.

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

LITERATURA K PRÁCI:

[1.] CHAD, Z., KUDZU, A.k.a. Indy in depth. , 2005. Dokumentace firmy Atozed software. s. 374.

[2.] ŠIMŮNEK, Milan. SQL kompletní kapesní průvodce. , 1999. 246 s. ISBN 80-7169-692-7 .

DOPORUČENÁ LITERATURA:

[3.] CANTU, Marco. Essential Delphi. , 2002. 156 s.

[4.] KERNIGHAN, Brian , RITCHIE, Dennis M. . The C Programming Language., 1978. 542 s. ISBN 80-251-0897-X.

[5.] CÍSAŘ, Pavel. InterBase/FireBird – Tvorba, administrace a programování databází. , 2001. 453 s.

Vedoucí bakalářské práce: **Ing. Jaromír Junek**
ID system, s.r.o.

Datum zadání bakalářské práce: **15. ledna 2010**

Termín odevzdání bakalářské práce: **14. května 2010**

L.S.

prof. Ing. Simeon Karamazov, Dr.
děkan

Ing. Lukáš Čegan, Ph.D.
vedoucí katedry

V Pardubicích dne 31. března 2010

Prohlášení autora

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 12. 5. 2010 Martin Mariška

Poděkování

Rád bych poděkoval panu Ing. Jaromíru Junkovi za cenné rady a připomínky. Za jeho čas, trpělivost a ochotu při tvorbě této bakalářské práce.

Dále bych rád poděkoval všem kolegům ve společnosti ID system, s. r. o., za ochotu poradit a konzultovat technické záležitosti týkající se nového i starého systému.

Anotace

Práce zahrnuje vytvoření programového vybavení pro propojení řídicího počítače s periferiemi na bázi mikroprocesoru Atmel a navržení simulačního programu na PC pro ověření a testování funkcionality. Součástí práce je ověření doby odezvy při různém zatížení systému.

Klíčová slova

řídicí systém, komunikace, periferie, multitasking

Title

Simple communication system of industrial control system

Annotation

The work includes creation of software for interfacing the control computer with peripherals based on a microprocessor architecture Atmel, and creation of the simulation program on a PC for verification and testing of the functionality, testing the response time for different loads of application.

Keywords

control system, communications, peripherals, multithreading

Obsah

Seznam zkratek **8**

Seznam obrázků **9**

Seznam tabulek **9**

1 Úvod **10**

2 Teoretická část **11**

2.1 TCP/IP protokol 11

2.1.1 TCP protokol 12

2.1.2 IP protokol 13

2.2 Blokované a neblokované metody programování 14

2.3 Databáze 14

2.3.1 Firebird 16

2.4 Multiprogramming – vláknové programování 18

3 Architektura aplikace **20**

3.1 Popis současného systému 20

3.2 Požadavky společnosti na nový systém 21

3.3 Příklad fungování sytému 21

3.4 Vymezení systému 21

3.5 Navržení systému 22

3.5.1 Řízení periferií 23

3.5.2 Rozeznávání periferií 24

3.5.3 Plánování událostí 24

3.6 Softwarová část 24

3.6.1 Řízení toku dat 24

3.6.2 Sledování plánovaných událostí 25

3.6.3 Minoritní řízení periferií 25

3.6.4 Sledování pohybu dat 25

3.7 Databázová část 25

3.7.1 Správa a úložiště dat 25

3.7.2 Řízení periferií 26

4 Realizace aplikace **27**
4.1 Komunikační protokol 28
4.2 VCL komponenta mmTCPServer 30
4.2.1 UML diagram tříd 30
4.2.2 Popis implementovaných tříd 30
4.3 Serverová aplikace 36
4.3.1 Popis aplikace 36
4.3.2 Realizace přímého řízení periferií 38
4.3.3 Nerozpoznané zařízení 38
4.4 Databáze 38
4.4.1 ER diagram 40
4.4.2 Popis databázových tabulek 40
4.4.3 Popis PSQL funkci a procedur 41
4.4.4 Popis ostatních použitých objektů 42
4.5 Problémy spojené s realizací 42
5 Požadavky aplikace **44**
6 Uživatelská příručka serveru **45**
7 Ověření funkcionality **46**
7.1 Simulační program 46
7.1.1 UML diagram tříd 47
7.1.2 Popis implementovaných tříd 47
7.2 Ověření doby odezvy 48
7.2.1 Výsledky 49
8 Závěr **52**
Literatura **53**
Příloha A – UML diagram tříd pro TmmTCPServer **54**
Příloha B – Zdrojový kód databázových pohledů **55**
Příloha C – Přiložené CD **56**

Seznam zkratek

ASCII	American Standard Code for Information Interchange
API	Application Programming Interface
ARP	Address Resolution Protocol
ACID	Atomicity, Consistency, Isolation, Durability
IETF	Internet Engineering Task Force
DDL	Data definition language
DHCP	Dynamic Host Configuration Protocol
DLL	Dynamic-link library
DNS	Domain Name System
FAQ	Frequently asked questions
FDDI	Fiber Distributed Data Interface
FTP	File Transfer Protocol
HTTP	Hypertext Transfer Protocol
ICMP	Internet Control Message Protocol
ID	Identifikační číslo
IGMP	Internet Group Management Protokol
IGRP	Interior Gateway Routing Protocol
IP	Internet Protokol
LAN	Local area network
ODBC	Open Database Connectivity
OLEDB	Object Linking and Embedding, Database
PSQL	Procedural SQL
SMTP	Simple Mail Transfer Protocol
SSL	Secure Socket Layer
SQL	Structured Query Language
RARP	Reverse Address Resolution Protocol
TCP	Transport Kontrol Protocol
UDP	User Datagram Protokol
VCL	Visual Component Library

Seznam obrázků

Obrázek 1 – Určení hranic systému 22
Obrázek 2 – Diagram pohybu dat v systému 23
Obrázek 3 – Ukázka stavového diagramu 24
Obrázek 4 – Struktura posílaných dat v komunikačním protokolu 28
Obrázek 5 – Pravidla komunikace mezi severem a klientem 29
Obrázek 6 – Zpracování dat v databázové části 39
Obrázek 7 – ER diagram 40
Obrázek 8 – UML diagram tříd pro Simulační program 47
Obrázek 9 – Graf časové odezvy serveru 50
Obrázek 10 – Graf pro porovnání časové odezvy v režimu bez vizualizace (1 z./s) 51
Obrázek 11 – Graf pro porovnání časové odezvy v režimu bez vizualizace (5 z./s) 51

Seznam tabulek

Tabulka 1 – Přehled výsledků časové odezvy serveru v [s] 50

1 Úvod

Mezi hlavní cíle mé bakalářské práce patří zhotovení komunikačního systému, který umožní spojení mezi jednotlivými periferiemi a řídící serverovou aplikací. Dále pak vytvoření jednoduchého simulačního programu pro ověření funkcionality serverové aplikace. Serverový program bude sloužit k asynchronně paralelní obsluze a řízení periferií po síti LAN. V praxi bude nahrazovat stávající systém pro odbavování lidí, jako jsou například aquaparky, kryté bazény, anebo velké haly s kontrolovaným vstupem a výstupem.

Teoretická část práce má za úkol vysvětlit základní teorii použitých technologií pro zhotovení práce.

Druhá část se věnuje představení architektury, rozdělení a teoretickému popisu systému pro lepší pochopení, jak se která část systému chová a jak řeší jednotlivé problémy.

Vývoji a realizaci je věnována třetí část této práce. Na požadavek společnosti byla vyvinuta komponenta pro zapouzdření funkcionality serveru a jeho následné znovupoužití. Jako vývojové prostředí bylo zvoleno Delphi 2010, protože jím disponuje i společnost a vyvíjí v něm své produkty. Pro realizaci připojení k databázovému serveru Firebird jsou využity komponenty FibPlus a pro síťovou komunikaci komponenty Indy 10.

Na konci práce je představena architektura a realizace simulačního programu použitého pro ověření časové odezvy serveru při různém zatížení. Výsledky měření časové odezvy jsou v poslední části této práce.

2 Teoretická část

2.1 TCP/IP protokol

TCP/IP je odvozeno od anglických slov Transmission Control Protocol / Internet Protocol. Většina aplikací na internetu v dnešní době používá protokol TCP. Tyto aplikace se spoléhají na jeho mechanismy, které zajišťují bezpečné dodávky dat přes protokol IP, který nezaručuje doručení paketu. Přestože se stal standardním souborem protokolů teprve před několika lety, je starý již více než dvacet let. Úplně na začátku byl navržen pro použití ve vládních počítačích na žádost armády (v historii byla předchůdcem dnešního Internetu síť ARPANET). Nyní se hlavně využívá v síti Internet, jenž se díky otevřenosti a podpoře stala největší sítí na světě. Od doby, kdy byl masivně implementován do systémů typu UNIX a linux, čemuž se tak dělo v 80 létech, se TCP/IP protokol začal považovat za standard. Díky této podpoře a zároveň díky jeho kompatibilitě vůči velkému množství hardwarových a softwarových systémů je dnes rozšířen po celém světě.

Z důvodu neustálého nárůstu internetových adres se adresový prostor tak zužuje, že hrozí vypotřebení všech IP adres. V současné době se připravujeme na přechod k novému typu protokolu IPv6, který nahradí IPv4.

Architektura TCP/IP

Aby se tak složitý systém jako TCP/IP dal realizovat, tak je síťová komunikace rozdělena do tzv. vrstev. Každá vrstva se stará o určitý požadavek systému (specializuje se na svoji činnost). Výměna informací mezi vrstvami je přesně definována a každá vrstva využívá služeb vrstvy nižší a poskytuje své služby vrstvě vyšší.

Přenos informací a komunikace mezi stejnými vrstvami dvou různých systémů je řízena komunikačním protokolem za použití spojení, které vytvořila nižší vrstva. Tato architektura dovoluje záměnu protokolů v dané vrstvě bez dopadu na ostatní. Znamená to, že když použijeme jiná pravidla v jedné vrstvě, nižší ani vyšší vrstvy to neovlivní. Příkladem může být nezávislá komunikace na různých fyzických médiích – ethernet, optické vlákno, sériová linka nebo již zmiňovaný přechod z IPv4 na IPv6.

Architektura TCP/IP je rozdělena na čtyři vrstvy:

- aplikační vrstva (application layer)
- transportní vrstva (transport layer)
- síťová vrstva (network layer)
- vrstva síťového rozhraní (network interface)

Vrstva síťového rozhraní

Vrstva síťového rozhraní zajišťuje přístup k přenosovému médiu a řeší problém připojení fyzického zařízení do fyzické sítě. Tato vrstva se může hodně lišit pro různé implementace. Odvíjí se od použité technologie například: Ethernet, Token ring, FDDI, X.25.

Síťová vrstva

Síťová vrstva řeší 2 problémy. První funkce je odesílání dat od zdroje sítě, do cílové sítě. Tento proces se nazývá směrování. Druhá funkce je adresování a identifikace zařízení v síti.

Protokoly podporované síťovou vrstvou jsou: IP, ARP, RARP, ICMP, IGMP, IGRP, IPSEC. Tato vrstva je implementována ve všech prvcích sítě (směrovače, routery, koncové zařízení).

Transportní vrstva

Transportní vrstva se implementuje až v koncových zařízeních (počítačích) a zajišťuje aplikaci přizpůsobit si chování sítě k vlastním potřebám. Poskytuje dva druhy přístupu. Spojovaný přístup (protokol TCP jako spolehlivá varianta, navazuje se spojení s cílem), anebo nespojovaný přístup (protokol UDP jako nespolehlivá možnost, nenavazuje spojení a nepotvrzuje doručení paketu).

Aplikační vrstva

Aplikační vrstva vytváří protokoly na vyšší úrovni, které se používají ve většině aplikací pro rozšířenou síťovou komunikaci. Příklady protokolů v aplikační vrstvě jsou například File Transfer Protocol (FTP) a Simple Mail Transfer Protocol (SMTP). Data kódovaná v aplikační vrstvě jsou pak zapouzdřeny do jednoho nebo občas i více dalších protokolů, jako jsou již zmiňované Transmission Control Protocol (TCP) nebo User Datagram Protocol (UDP), které využívají zase služeb z nižších vrstev a mohou tak provést požadovaný přenos dat. Mezi další příklady patří: Telnet, HTTP, DHCP, DNS.

2.1.1 TCP protokol

Transmission Control Protocol (TCP) standard je definován v Request For Comment (RFC) normy číslem 793 podle Internet Engineering Task Force (IETF). Původní specifikace popsaná v roce 1981 byla založena na dřívějších výzkumech a experimentování v původní síti ARPANET.

Mezi protokoly v Internetu je TCP prostřední vrstvou mezi IP protokolem a aplikací. Předtím, než si mohou dvě zařízení vyměňovat data přes TCP, se nejprve musí dohodnout na ochotě komunikovat. Zde je jistá analogie k telefonnímu hovoru. Obě strany musejí nejdřív zvednutím sluchátka potvrdit, že budou komunikovat. Pokud aplikace odesílá data pomocí TCP, činí tak v 8-bitových proudech bajtů. TCP předá vzniklé pakety

IP protokolu k přepravě internetem do TCP modulu na druhé straně spojení. Poté je zas na protokolu TCP, aby byl přenos dat schopný spolehlivě dosáhnout cíle a to v pořadí, v jakém byla data odeslána.

Spolehlivost zajišťuje několik mechanizmů. První z nich je *kontrolní součet* tzv. „checksum“ (ověřuje se, zda přenesená data nebyla poškozena šumem, a proto se před odesláním vypočítává kontrolní součet, ukládá se do odesílaného paketu a příjemce podle tohoto součtu znovu zpětně ověří jeho správnost). TCP garantuje spolehlivé doručování a to ve správném pořadí (samotná data mohou jít různými cestami). Dosahuje toho pomocí potvrzování o doručení. Aby se zbytečně nepřetěžoval přenos, tak mohou odpovědi dorazit v určitém časovém úseku (round-trip time, RTT), ale dorazit musí, jinak bude protokol na straně odesílatele daná data posílat znovu, dokud nedorazí.

TCP porty

„K rozlišení komunikujících aplikací používá TCP protokol čísla portů. Každá strana TCP spojení má přidruženo 16bitové bezznaménkové číslo portu (existuje 65535 portů) přidělené aplikaci. Porty jsou rozčleněny do třech skupin: dobře známé, registrované a dynamické/privátní. Seznam dobře známých portů je přiřazován organizací Internet Assigned Numbers Authority (IANA) a jsou typicky používané systémovými procesy. Dobře známé aplikace běžící jako servery a pasivně přijímající spojení typicky používají tyto porty. Několik příkladů: FTP (port 21 a 20), SMTP (port 25), DNS (port 53) a HTTP (port 80). Registrované porty jsou typicky používané aplikacemi koncových uživatelů při otevírání spojení k serverům jako libovolná čísla zdrojových portů, ale také mohou identifikovat služby. Dynamické/privátní porty mohou být také používány koncovými aplikacemi, ale není to obvyklé.“ [1]

2.1.2 IP protokol

Internet Protocol (IP) je základní datový protokol pro přenos dat přes paketové sítě.

Funkce protokolu

Hlavní funkcí IP protokolu je doručování dat podle adresovacích metod od zdroje k cíli. Mezi nejznámější adresovací metody dnes patří IPv4 nebo IPv6. Ty určují, jak zapisovat adresy zdroje a cíle. Data v síti putují naprosto nezávisle po blocích nazývaných datagramy. Na začátku komunikace není potřeba navazovat spojení a to i v případě, že spolu příslušné stroje nikdy předtím nekomunikovaly.

Tento protokol je *„nespolehlivá“* služba pro doručování datagramů, proto se označuje také jako *best effort* – „nejlepší úsilí“. Znamená to, že všechny stroje na trase od zdroje k cíli se datagram snaží podle svých možností poslat blíže k cíli, ale nezaručuje jeho doručení ani odeslání. Datagram vůbec nemusí dorazit, naopak může být doručen několikrát, anebo ve špatném pořadí.

Adresování a směrování

„Každé síťové rozhraní komunikující prostřednictvím IP má přiřazeno jednoznačný identifikátor, tzv. IP adresu. V každém datagramu je pak uvedena IP adresa odesílatele i příjemce. Na základě IP adresy příjemce, pak každý počítač na trase provádí rozhodnutí, jakým směrem paket odeslat tzv. směrování (routing). Tuto činnost provádí hlavně specializované stroje označované jako směrovače (routery).“ [2]

2.2 Blokované a neblokované metody programování

V dnešní době existují dva nejrozšířenější programovací modely pro vývoj síťových aplikací pod platformou Windows, blokované a neblokované volání. Někdy bývají také nazývány synchronní (blokované) a asynchronní (neblokované).

Systémy rodiny Unix používají pouze blokované volání.

Blokované

Blokované volání se hodně podobá práci se soubory. Pokud chcete číst data, tak funkce se zablokuje do doby, dokud nebude celá operace dokončena. Rozdíl mezi prací se soubory a sokety je ten, že operace se sokety může zabrat podstatně více času, protože data nemusí být v danou chvíli připravená ke čtení nebo k zápisu. Zápis a čtení může probíhat pouze tak rychle, jak dovoluje rychlost přenosového média (např. síť nebo modem).

Neblokované

Neblokované sokety pracují systémem spouštění událostí. Spuštění události je provedeno, pokud je nějaká operace dokončena nebo potřebuje pozornost.

Pro příklad, pokud se pokoušíme připojit k soketu, musíme volat metodu *connect*. Metoda *connect* ihned vrátí hodnotu a to ještě předtím, než je soket doopravdy připojen. Následně, až je soket opravdu připojen, spustí se událost o připojení soketu. Tento typ programování vyžaduje, aby kód byl logicky rozdělen do mnoha procedur, které obsluhují síťovou komunikaci.

2.3 Databáze

„Databáze (datová základna) je určitá uspořádaná množina informací (dat) uložená na paměťovém médiu. V širším smyslu jsou součástí databáze i softwarové prostředky, které umožňují manipulaci s uloženými daty a přístup k nim. Tento software se v české odborné literatuře nazývá systém řízení báze dat, od toho zkratka SŘBD. Běžně se označením *databáze* myslí jak uložená data, tak i software (SŘBD).“ [6]

Relační databáze

Relační databáze je založena na relačním modelu dat. Základem relačního modelu dat je databázová relace (množina). Ta obsahuje jméno relace, jména atributů a popisuje integritní omezení.

Často se pojmem relační databáze označuje nejen databáze samotná, ale i její konkrétní softwarové řešení. Termín relační databáze definoval Dr. Edgar F. Codd (působil ve firmě IBM) roku 1970 v článku: „A relational model of data for large shared databanks".

Integrita databáze

Integrita databáze znamená, že se dodržují pravidla ukládání dat. Nemůže existovat situace, kdy jsou uložena data, která neodpovídají definovaným pravidlům. Lze zadávat pouze ty informace, které patří do databáze a není možné, aby se ztratily informace, které mají být zachovány. K zajištění integrity slouží integritní omezení. Pomocí těchto nástrojů lze zabránit vložení nesprávných dat, ztrátě, nebo poškození stávajících záznamů.

Druhy integritních omezení

Entitní integritní omezení – je povinné integritní omezení, které zajišťuje korektní zadání primárního klíče tabulky. Musí zamezit vložení informací, které nejsou správně vyplněny nebo data, která již existují v nějakém zapsaném řádku tabulky a tím by vznikaly duplicity.

Doménová integritní omezení – je omezení, které zajišťuje korektnost datových typů nebo domén, které jsou definovány u sloupců databázové tabulky

Referenční integritní omezení – zajišťují správné vztahy mezi dvěma tabulkami. Jejich relace je dána vazbou primárního nebo cizího klíče

Aktivní referenční integrita – definuje, jak se má systém chovat, pokud dojde k porušení výše uvedených integritních omezení.

Dodržování integritních omezení

Dodržování se můžeme ve své podstatě rozdělit do tří skupin podle toho, na kterém místě se pravidla hlídají a kontrolují.

1. Deklarativní na straně databázového serveru – z hlediska ochrany dat je to nejlepší způsob, protože pravidla jsou uvedena a implementována přímo v databázovém systému.
2. Aplikační na straně klienta – v případě výskytu chyby kontroly se oprava musí provést na více místech. Respektive v každé klientské aplikaci, která má definována pravidla pro vkládání.

3. Procedurální na straně serveru – v dnešní době jsou pro tento účel zavedeny tzv. spouště (triggery). Jsou to procedury, které se automaticky spouští při definované manipulaci s daty. Výhoda je, že s jejich pomocí lze vytvářet i složitější integritní omezení (kaskádové mazání atd.).

Normální formy

Pod pojmem *normalizace* rozumíme proces odstraňování redundantních (opakujících se) dat. Proces zajistí zjednodušení a optimalizaci navržených tabulek, se kterými bude efektivní práce, budou dále dobře rozšiřitelné a budou obsahovat co nejméně závislých informací.

„Normální formy:

- Nultá normální forma (0NF) - tabulka v nulté normální formě obsahuje alespoň jeden sloupec (atribut), který může obsahovat více druhů hodnot.
- První normální forma (1NF) - tabulka je v první normální formě, pokud všechny sloupce (atributy) nelze dále dělit na části nesoucí nějakou informaci znamená, že prvky musí být atomické. Jeden sloupec neobsahuje složené hodnoty.
- Druhá normální forma (2NF) - tabulka je v druhé normální formě, pokud obsahuje pouze atributy (sloupce), které jsou závislé na celém klíči.
- Třetí normální forma (3NF) - tabulka je ve třetí normální formě, pokud neexistují žádné závislosti mezi neklíčovými atributy (sloupci).
- Čtvrtá normální forma (4NF) - tabulka je ve čtvrté normální formě, pokud sloupce (atributy) v ní obsažené popisují pouze jeden fakt nebo jednu souvislost.
- Pátá normální forma (5NF) - tabulka je v páté normální formě, pokud by se přidáním libovolného nového sloupce (atributu) rozpadla na více tabulek.“ [7]

2.3.1 Firebird

Pro vypracování této práce byl použit databázový systém Firebird. Pro představu dále uvedu některé významné a podstatné informace, které jsou specifické pro databázový systém Firebird.

Firebird je open source projekt, proto náklady na pořízení jsou nulové. Je nenáročný, velice spolehlivý a hodí se pro provozování v praxi i u středně velkých podniků. Jeho možnosti v podporovaných jazycích jsou obrovské (Delphi, C++, .NET, Python, PHP, a mnoho dalších). Firebird plně podporuje interní rozšíření jazyka SQL, zde nazvané jako PSQL jazyk spouští a uložených procedur. Podporuje uživatelské externí funkce UDF, „record versioning“, inkrementální zálohy, monitorovací a dynamické tabulky, triggery na spojení a transakce. Dále velké množství způsobů jak přistupovat k databázi (například nativní/API, dbExpress ovladače, ODBC, OLEDB, .NET provider, JDBC nativní typ 4 ovladač, Python modul, PHP, Perl atd). Dále splňuje podmínky ACID (Atomic, Consistent, Isolated, Durable), plně podporuje cizí klíče, operace JOIN, pohledy

a domény. Obsahuje většinu standardních datových typů, např. INTEGER, NUMERIC, FLOAT, CHAR, VARCHAR, DATE, BLOB, TIME a TIMESTAMP.

Pro Firebird existuje velká komunita a mnoho míst, kde lze dohledat řešení různých problémů a dobré rady. V dnešní době už existuje množství nástrojů třetích stran, obsahujících grafické administrační nástroje, nástroje pro replikaci a další standardní výbavu nástrojů, které ke správě databáze obvykle potřebujeme.

Firebird je multiplatformní a přímo podporuje všechny hlavní operační systémy, zahrnující Windows, Linux, Solaris, MacOS.

Historie Firebirdu

Databázový systém Firebird. byl vytvořen nezávislým týmem vývojářů ze zdrojových kódů, původně komerčního databázového systému InterBase, uvolněného jako open source firmou Borland pod licencí InterBase Public License v.1.0 dne 25. července 2000.

Vývoj ve zdrojovém stromu Firebird 2 započal již v průběhu vývoje Firebirdu 1 převodem zdrojových kódů Firebirdu 1 z jazyka C do C++ a prvním velkým čištěním kódu. Firebird 1.5 je první veřejná verze systému založená na zdrojovém stromu Firebird 2. Ačkoliv jde o důležitý milník pro vývojáře i celý projekt Firebird, nejedná se o konečné stádium vývoje Firebirdu. V současné době už je na trhu dostupná stabilní veřejná verze 2.1.3, kandidát na stabilní verzi 2.5.0 a testovací verze 3.0.0. Všechny se poskytují v instalačních verzích Classic a SuperServer. Vývojáři také vytvořili speciální úpravu pro využití bez instalování serveru do počítače tzv. „embedded“ verzi. Za pomoci embedded verze si může aplikace načíst Firebird server pomocí DLL knihoven a připojit se do databáze bez potřeby instalovat databázový server.

Uložené procedury

„Uložená procedura je především procedura, uložená přímo v databázi a má přístup k datům. Uložená procedura je (nebo by aspoň měla být) jasně funkčně oddělená od svého okolí a má vlastní interface (seznam parametrů) pro komunikaci s jinými moduly programu. Může mít i lokální proměnné, neviditelné pro ostatní části programu.

Uložená procedura je uložená (rozumí se: uložená v databázi). To znamená, že se k ní lze chovat stejně jako ke každému jinému objektu databáze (indexu, pohledu, triggeru apod.). Lze jí založit, upravovat a smazat pomocí příkazů dotazovacího jazyka databáze (v případě relační databáze obvykle pomocí příkazů DDL SQL).

Pro psaní uložených procedur je obvykle používán specifický jazyk konkrétní databáze, který je rozšířením jejího dotazovacího jazyka (názorným příkladem je pro databázi Oracle procedurální jazyk PL/SQL, který je rozšířením klasického dotazovacího jazyka SQL).“ [12]

Firebird disponuje tzv. PSQL, což je procedurální rozšíření jazyka SQL přebraného z InterBase. Myšlenka těchto procedur je, že se uložená procedura může chovat jako funkce (vrací nějakou množinu dat), anebo jen manipuluje s daty v databázi. Pokud procedura něco vrací, můžeme jí zavolat pomocí SQL příkazu SELECT. Výhodou, kterou využívám i ve své práci je, že jednotlivé dotazy a procedury můžeme skládat v průběhu zpracování uložené procedury pomocí tzv. EXECUTE STATEMENT. Ten umožňuje kompilaci vloženého kódu za běhu a provádět tak výrazy, které mohou být uloženy například po částech v jiných databázových objektech.

2.4 Multiprogramming – vláknové programování

Vláknové programování se ve většině případů využívá pro zpracování náročných činností, které se vyplatí zpracovávat paralelně. Paralelním zpracování těchto činností dosáhneme efektivnějšího využití zdrojů (hardwarových, časových). V dnešní době patří mezi nejnáročnější vstupně/výstupní operace, kdy se čeká na jiný zdroj (např. vstup od lidského zdroje), proto je pro některé činnosti hodně efektivní vykonávat nějakou činnost paralelně. Uvedu příklad na textovém editoru, kde budeme chtít funkci kontroly pravopisu. V sériově zpracovávaném programu by se kontrola musela spouštět po té, co uživatel dopíše svůj zadávaný text a čas spotřebovaný na kontrolu bude uživatele zdržovat. V případě implementace vlákna pro kontrolu pravopisu může vlákno kontrolovat text ve stejnou chvíli, kdy ji uživatel zrovna zadává, tím se efektivněji využije jak procesorový čas, tak i čas uživatele.

Vlákno je vždy potomkem hlavního procesu, který ho vlastní a všechna vlákna jednoho procesu sdílejí stejný paměťový prostor jako hlavní proces. To znamená, že všechna vlákna mohou ovlivňovat jakákoliv data v tomto paměťovém prostoru. Zde může být problém, který je obecnou problematikou při použití více vláknové aplikace. Při použití této techniky si musíme dát pozor, k čemu a kam má vlákno přístup. Musíme se snažit předejít situaci, kdy budou mít dvě a více vláken přístup ke stejné globální proměnné, která bude ve výsledku obsahovat třeba výpočet nějaké rovnice. Vlákna mohou způsobit, že budou zapisovat data do proměnné ve stejnou chvíli a výsledek bude špatný. Tahle situace je jenom pro ukázku. V praxi se vlákna mohou tímto způsobem chytit například do smyčky nebo tzv. „smrtelného objetí“ (znamená to, že čekají na stejná data, která nemůže dostat ani jeden z nich).

Pro shrnutí výhody vláknového programování jsou ve větší efektivnosti při činnosti, která je časově náročná a nevyužívá se efektivně procesor (například V/V operace). Na druhé straně pak nevýhody jsou, že se vlákna při přístupu ke globálnímu prvku musí mezi sebou synchronizovat.

Druhy paralelního zpracování

‚**Funkční paralelismus** – složitá činnost (popsaná nějakou funkcí) se rozdělí na více jednodušších, dílčích činností. Každou z těchto dílčích činností vykonává jedno vlákno. Funkční paralelismus se typicky používá pro složité úlohy nad jednoduchými daty.

Analogie se životem: při stavbě rodinného domku se zároveň zavádí elektřina i plyn. Tento paralelismus má smysl i na jednoprocesorových počítačích.

Datový paralelismus – používá se v případě relativně jednoduché činnosti nad rozsáhlými daty. Tato data se rozdělí „na kousky“ a nad každým kouskem dat provede jedno vlákno tutéž činnost. Analogie se životem: dlouhý výkop hloubí parta kopáčů, každý je zodpovědný za jeden úsek výkopu. Tento paralelismus prakticky nemá smysl provozovat na jednoprocesorovém stroji (až na výjimky, které jsme si popsali výše).

Zřetězené zpracování dat – vlákna si „předávají“ nějaký datový záznam, každé z vláken s ním udělá nějaký „kus práce“ a předá jej dál. V tom okamžiku je již připraveno opět přijmout další záznam.‘ [14]

Delphi

Delphi je grafické vývojové prostředí s programovací jazykem Object Pascal od firmy Borland. Je určen pro vývoj aplikací v prostředí Microsoft Windows. Delphi obsahují systém pro vizuální návrh výsledného programu (používají tzv. „VCL knihovnu“). Na základě vizuálního podkladu je generován kód aplikace a tím se velmi urychlí její vývoj.

Delphi jsou dostupné v tzv. „personal“ verzi zdarma, ale ostatní verze jsou komerční. Jako nekomerční náhrada může posloužit např. projekt Lazarus.

Použití vláken v prostředí Delphi

V prostředí Delphi je použití vláken jednoduché. Pro naše potřeby je zde připravena třída TThread, od které stačí vytvořit potomka a zavedením instance třídy se vytvoří v aplikaci nové vlákno. Jedinou podmínkou je, že se musí předeklarovat abstraktní metoda „execute“ (tu vlákno implicitně spouští) u každého potomka této třídy.

3 Architektura aplikace

Navržení architektury systému pro komunikaci mezi řídícím počítačem a ostatními periferiemi podléhalo požadavkům společnosti ID systém, s.r.o. Tato firma se zabývá vývojem a výrobou systémů pro identifikaci osob a řízení procesů. Tato práce bude nahrazovat stávající systém, který je zastaralý a již nevyhovuje požadavkům společnosti.

Společnost vlastní spoustu typů zařízení, které mají specifické funkce. Pro příklad to může být „turniket“ (odbavuje lidi dle různých kritérií) nebo „mincovník“ (dobíjení zákaznických účtů).

Důvodem výběru databázového systému Firebird je kompatibilita s ostatními produkty společnosti. Jeho použitím následně umožníme připojení pokladového systému společnosti, který je součástí jejich produktů. Všechna data pokladny i periferií, jsou sdílena v databázovém prostoru. Proto je možné, aby serverová aplikace poskytovala informace ostatním periferiím. Tato část není součástí této práce, pouze poukazuji na důvod použití tohoto typu databázového stroje.

Periferie jsou realizovány mikropočítačem na bázi mikroprocesoru rodiny Atmel ATmega. Disponují základní deskou dodávanou s mikroprocesorem, dotykovým displejem, modulem pro komunikaci po síti LAN a množinou vstupů a výstupů v podobě ovládacích tlačítek.

3.1 Popis současného systému

Současný systém používaný v praxi je založen na komunikaci serverové aplikace s periferiemi pomocí standardu RS-232/RS-422 (sériová linka). Systém byl nastaven tak, že po zapnutí se serverová aplikace začne cyklicky dotazovat jednotlivých zařízení, zda nemají nějaký požadavek.

Mezi velké nevýhody tohoto systému patří zbytečná komunikace serveru s periferií v případě, že se nic neděje a zařízení nekladou žádné požadavky na server. V případě požadavku na server musí periferie počkat, až bude dotázána. Dále musí server hlídat, zda se do systému nechce připojit nově zapnuté nebo restartované zařízení. Z tohoto důvodu se v každé smyčce vždy dotazuje i na nepřipojené zařízení. Tento druh dotazu po uplynutí určitého časového úseku pokračuje dál (v případě vypnutého zařízení), ale časová ztráta při čekání je ve výsledku veliká.

Další nevýhodou je, že „business“ logika (systém pro řízení periferií) je zabudována přímo v softwaru, a proto v případě změny řízení (např. zákazník potřebuje přizpůsobit chování zařízení) je nutné změnit kód aplikace, dodat nový software zákazníkovi, vypnout zákazníkův systém a spustit upravený.

3.2 Požadavky společnosti na nový systém

Od nového systému se požaduje vyšší výkon na základě odlišného způsobu obsluhy periferií. Systém by měl být schopen paralelně zpracovávat data posílaná po síti LAN. Zároveň ovládat zařízení, podle jeho typu a to způsobem, který je určený stavovým diagramem daného typu periferie.

Na tento problém se můžeme dívat jako na problém server (náš řídicí systém na pc) / klient (periferní zařízení). Server odpovídá na dotazy klientů a zároveň v určitou dobu řídí jejich činnost. Na druhé straně se mohou klienti v případě potřeby asynchronně dotazovat na server pro potřebné informace.

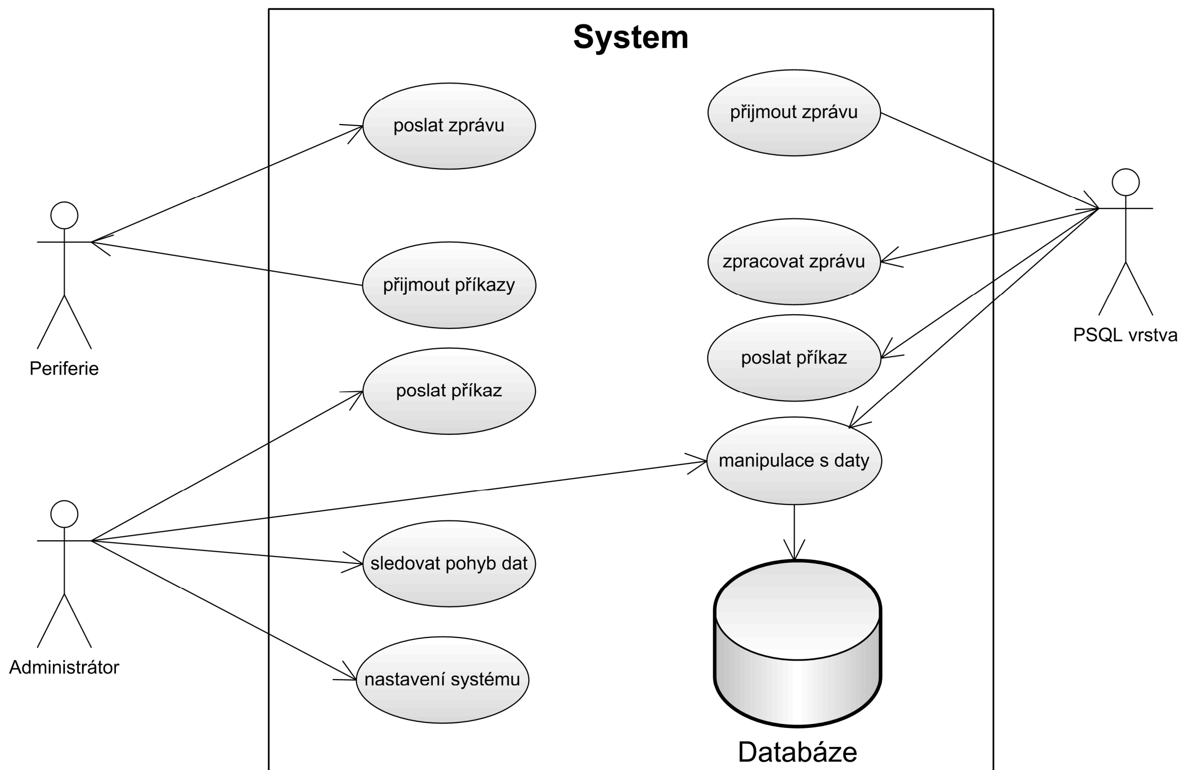
3.3 Příklad fungování sytému

Uvádím příklad komunikace periferie se serverovou aplikací, aby bylo snazší pochopit princip systému.

Přichází zákazník a bude chtít vstoupit do areálu pomocí identifikační karty. Protáhne kartu a v tom okamžiku periferie odesílá zprávu o protažení karty s určitým identifikačním číslem např. ve tvaru [D *(příkaz)*; M100X23 *(data)*]. Klientské vlákno po přečtení zprávy ze soketu, ji takto předá databázové části. PSQL vrstva v databázi (zapouzdřené funkce pro zpracování zpráv), podle typu periferie a jejího aktuálního stavu, vybere potřebnou obslužnou funkci, které se následně předá přijatá zpráva ke zpracování. Obslužná funkce v první fázi rozhodne, zda tato karta může vstoupit do areálu. V případě, že má zákazník dostatek kreditu a karta má povolen vstup do této části areálu, obslužná funkce odečte zákazníkovi z účtu příslušný kredit, v databázi zavolá povel k naplánování vymazání displeje (např. za 2 vteřiny) a vrátí výstupní data např. [P *(příkaz)*; „Aktuální stav účtu: 150 Kč“ *(data)*]. Tu převezme klientské vlákno a pošle zprávu zpět periferii, která podle příkazu „P“ povolí turniket a zobrazí data na displeji pro zákazníka. Zařízení následně vyčkává ve stavu „zobrazení“ na příchod příkazu pro smazání displeje. Po přijetí přechází periferie do základního stavu a je opět připravena vpustit dalšího zákazníka nebo provést další operaci.

3.4 Vymezení systému

Use case diagram pomáhá vizuálně zobrazit vnější závislosti systému.



Obrázek 1 – Určení hranic systému

Jak je vidět z obrázku č. 1, systém není příliš složitý, co se týče závislostí. Hlavním úkolem je zabezpečit přenos informací po síti oběma směry. K tomu nám poslouží protokol TCP, který zaručuje doručení paketu na druhou stranu. Aplikace slouží ke komunikaci bez zásahu uživatele. Proto zde není nutné vyvíjet podporu pro uživatelské účty. Zde bych doporučil implementaci kontrolního hesla, pro ověření identity administrátora. Administrátor bude do systému moci plně zasahovat za účelem nastavení serveru nebo testování, kontrolu a řízení periferií.

3.5 Navržení systému

Systém lze pomyslně rozdělit do 3 částí:

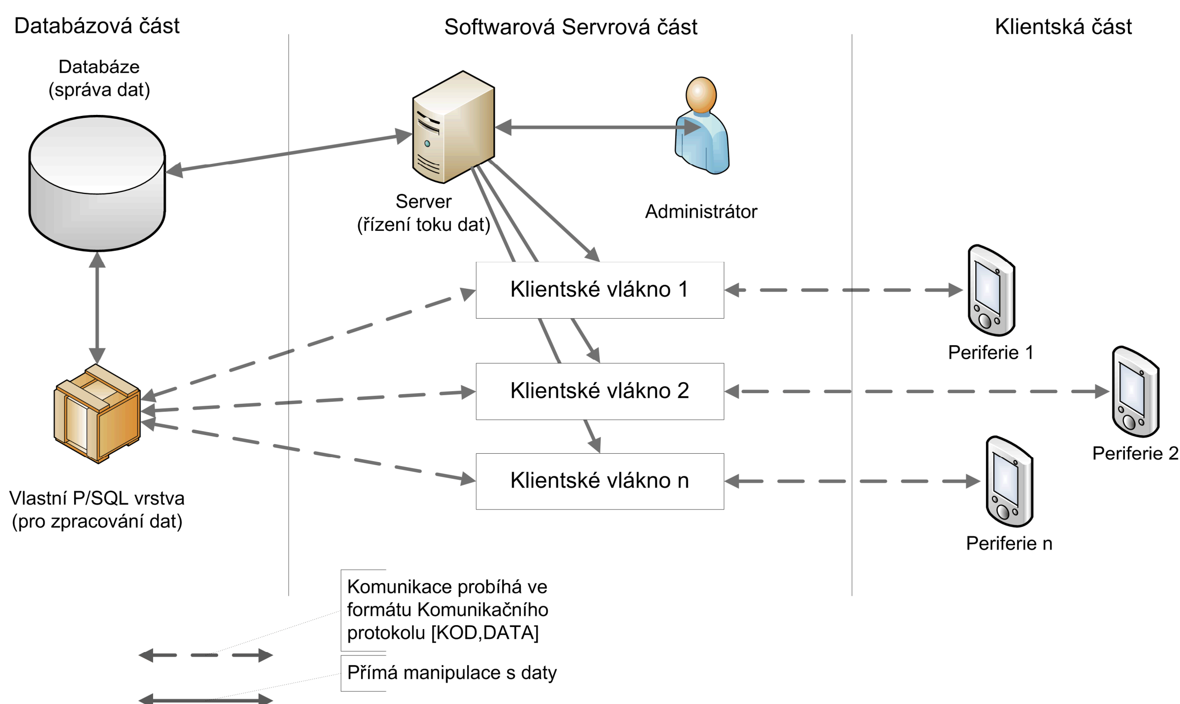
- Správa dat – úložiště
- Řízení toku dat
- Zpracování dat – řízení periferií

Po uvážení, jak rozdělit tyto 3 body mezi databázi a software, je mnohem lepší volba, aby většinu práce zpracování a řízení periferií řídila databázová část. Je optimalizována pro práci s mnoha daty, proto vyhovuje lépe našim potřebám. Navíc jsme tím oddělili správu, zpracování dat a řízení periferií od paralelního řízení toku dat. Řízením toku dat se bude zabývat pouze software a databáze bude umožňovat velice rychlé a dynamické zpracování dat od periferií. Navíc databázová část nebude závislá na operačním systému a bude ji možné použít i v ostatních podporovaných systémech (např. Linux, MacOS, Solaris). V případě použití celé aplikace na jiné platformě je nutné

pouze implementovat novou komunikační vrstvu. Odpadá nutnost vyvíjet celý software znovu, a proto lze považovat program částečně za multiplatformní.

Pro shrnutí je tedy aplikace rozdělena na 2 části: softwarovou a databázovou. Od softwarové části je vyžadován nezávislý (paralelní) přenos dat z periferie do databáze a z databáze zpět do periferie. Databázová část zpracovává příchozí data, manipuluje s daty v databázi a vrací výstupní data do softwarové části. Ta následně odesílá výstupní data zpět do periferie.

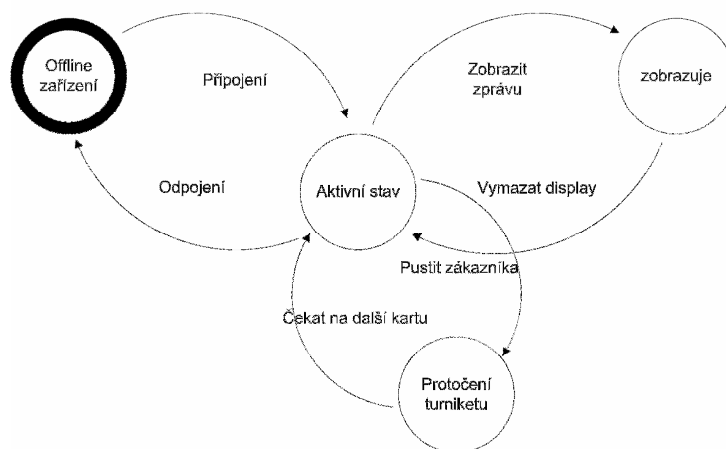
Po lepší pochopení uvádím nákres pohybu dat (viz obrázek č. 2).



Obrázek 2 – Diagram pohybu dat v systému

3.5.1 Řízení periferií

Pro řízení periferií se využívá stavových diagramů, které definují stavy a události při přechodech mezi jednotlivými stavy. Přechody mezi jednotlivými stavy jsou realizovány při příchodu nějakého příkazového kódu do databázové části. Ta podle typu zařízení, aktuálního stavu a poslaného příkazového kódu, vybere z databáze příslušnou obslužnou funkci, která se aplikuje na příchozí data. Tyto funkce mohou převést periferii do jiného stavu, měnit data v databázi, plánovat nové události pro sebe nebo ostatní periferie atd. Následně databázová část generuje výstupní data, která se odešlou do zařízení. Pro příklad uvádím, jak vypadá jednoduchý stavový diagram.



Obrázek 3 – Ukázka stavového diagramu

3.5.2 Rozeznávání periferií

Pro identifikaci zařízení se využívá identifikační číslo zařízení (dále jen zkráceně ID). Na identifikační číslo jsou vyhrazeny 2 bajty, takže ve výsledku to může být číslo od 1 do 65535. Každé identifikační číslo je v dané konfiguraci systému jedinečné. Serverová část podle ID rozpoznává, o jaké jde přesně zařízení (její typ, aktuální stav).

3.5.3 Plánování událostí

Z potřeby ručního i automatického řízení periferií má aplikace zabudovaný systém plánování událostí. Ve své podstatě nahrazuje časovač na straně periferie. Aplikace pomyslně simuluje příjem zprávy od zařízení a pošle ho ke zpracování do databáze, kde už se událost zpracuje standardním způsobem a výsledky se odešlou jako odpověď do zařízení.

3.6 Softwarová část

Tato část zajišťuje paralelní přenos informací z periferie do databáze a zpět do periferie. Nezpracovává příkazy od periferie, tedy neřídí stavy zařízení. Aby mohl administrátor řídit periferie, tak jsou zde dva způsoby. Aplikace umožňuje odeslání příkazu přímo k periferii, anebo lze v databázi naplánovat událost pro dané periferie.

3.6.1 Řízení toku dat

Hlavním požadavkem byla paralelní komunikace, proto jak je vidět z obr. č.2 (kde je znázorněn pohyb dat v systému), bude hlavní proces aplikační části vytvářet paralelní vlákna, která budou individuálně obsluhovat jednotlivé připojené zařízení. Každé vlákno vlastní objekt, který má vlastní připojení do databáze a vlastní lokální data potřebná ke komunikaci dané periferie.

3.6.2 Sledování plánovaných událostí

V hlavním procesu existuje časovač (timer), který se v našem případě periodicky po 1 vteřině dotazuje do databáze, zda některé zařízení nemá nějakou naplánovanou nevyřízenou událost. V případě, že zjistí nějakou naplánovanou událost, oznámí to klientskému vláknu (pokud je připojené) a to ihned individuálně volá PSQL vrstvu v databázi pro výběr dat a následně je odesílá k periferii.

3.6.3 Minoritní řízení periferií

V zadání firmy je požadováno přímé řízení periferií z důvodu rychlé kontroly a testování zařízení (například nový typ periferie). Aplikační část proto umožňuje odeslání dat přes klientské vlákno přímo do periferie.

3.6.4 Sledování pohybu dat

Tato funkce umožňuje vizuální sledování a kontrolu posílaných dat. V nastavení aplikace lze nastavovat úroveň auditu pohybu dat, která jsou v případě potřeby využita znovu ke kontrole nebo analýze vzniklého problému.

3.7 Databázová část

Mezi hlavní funkce databázové části, kterou budeme využívat, patří ukládání dat a rozšíření dotazovacího jazyka SQL. Tzv. „PSQL" rozšíření poskytuje funkci ukládání předdefinovaných procedur a funkcí přímo do databáze.

Pro přístup aplikace k datům je vytvořena PSQL vrstva, která zajišťuje kontrolu vstupu a výstupu informací pro aplikaci a zároveň i pro konkrétní zařízení. Tato vrstva umožňuje nezávislost aplikace nad vrstvou, a proto je do budoucna snaha pokračovat ve vytváření a rozšíření této vrstvy pro univerzálnost přístupu k datům.

3.7.1 Správa a úložiště dat

Návrh databáze, který je nutný k provozu daného systému, není součástí této práce. Součástí je pouze navržení a vytvoření vnitřní sub-architektury, která zajistí funkčnost databázové části, která obsluhuje zařízení. Protože databáze Firebird bohužel neposkytuje balíčkovací systém, jako je tomu například u Oracle databáze, používá se intuitivní předpony pro databázové objekty. Předpona „SYSTEM_" uvozuje tabulky potřebné pro běh vyvíjeného systému. Dále „LOG_" se používá pro tabulky, které obsahují nějaké informace navíc v rámci auditu systému. Pro rozpoznání jsou uložené procedury, které nějaká data vrací (jako funkce), jsou označené předponou „F_" a ty, co jenom vykonávají kód (jako procedury), jsou s předponou „P_"

Je zde možnost vytvářet kontrolní spouště (triggery) pro další kontrolu vkládání dat nebo rozšíření auditu dat.

3.7.2 Řízení periferií

Řízení probíhá voláním dvou funkcí F_PRECHOD (tuto funkci volá klientské vlákno v případě příchozích dat od periferie), anebo F_PLAN_UDALOSTI (tuto funkci volá klientské vlákno v případě, že mu aplikační část oznámí výskyt nějaké naplánované události v databázi). Obě tyto funkce vrací výstupní data pro odeslání. Rozdíl je pouze v tom, že funkce F_PLAN_UDALOSTI může vracet množinu dat k odeslání, ale F_PRECHOD vždy pouze odpovídá na příchozí dotaz.

4 Realizace aplikace

Nejprve si představíme komponenty, které jsou použité při realizaci aplikace a jsou potřebné pro vývoj. Dále jsou popsány jednotlivé prvky systému a jejich konkrétní realizace.

FIB plus

Tyto komponenty podporují připojení a práci s databázemi InterBase, Yaffil a Firebird. Byly navrženy podle požadavků zákazníků a je určen pro řešení každodenních úkolů v běžné praxi. Oproti InterBase komponentám, IBX pracují až 4x rychleji a zacházejí mnohem hospodárněji s pamětí.

Indy 10

Aby mohla aplikační část využívat síťovou komunikaci, využili jsme v prostředí Delphi komponenty Indy 10. Zde bych jen rád upozornil, že v Delphi personál edition 7 jsou standardně předinstalované Indy 9 a je nutno si stáhnout Indy verze 10. Při instalaci je dobré plně odstranit starou verzi nebo separovat tyto dvě verze. Použití těchto dvou verzí zároveň, může způsobit velké problémy, protože Indy verze 10 nejsou plně kompatibilní s verzemi předchozími.

Filozofie a užití komponent Indy 10

Indy komponenty se hodně liší od ostatních soketových komponent, proto uvádím malé vysvětlení a představení. Indy používají tzv. blokované sokety (synchronní způsob), jak je uvedeno výše v teoretické části. Práce s nimi připomíná hodně práci se soubory. Jejich implementace dovoluje vepsat všechen kód do jednoho místa, na rozdíl od událostního zpracování (ansynchrnonní způsob). Indy navíc poskytují kvalitní podporu pro vláknové zpracování, které v našem programu využijeme.

Pokud chceme nějaká data přečíst ze soketu, zavoláme jednoduše metodu „read“ a indy zablokují proces nebo vlákno do doby, než budou data v soketu dostupná. Nesmí se vytvářet smyčky (např. kontrolovat na straně serveru, zda jsou v soketu data ke čtení), protože nejsou potřeba a dokonce jsou nežádoucí. Použitím například kontrolní smyčky, bychom obešli blokované volání metody „read“ a proces by nebyl zablokován, ale zachycen ve smyčce a zbytečně by vytěžoval procesor.

Jediná nevýhoda u blokovaných volání je u klientského procesu např. u volání metody „read“. Proces tzv. „zamrzne“. Tento problém, aby uživatel mohl dále používat program, se řeší komponentou vytvořenou a dodávanou Indy vývojáři. Komponenta TIdAntiFreeze může být v aplikaci pouze jednou. Její funkce spočívá ve vnitřním časovači, který zavolá stack a dovolí zprávám procesu, aby byly zpracovávány mezi intervaly, kdy dojde k vypršení časového limitu u blokovaných metod.

Firebird

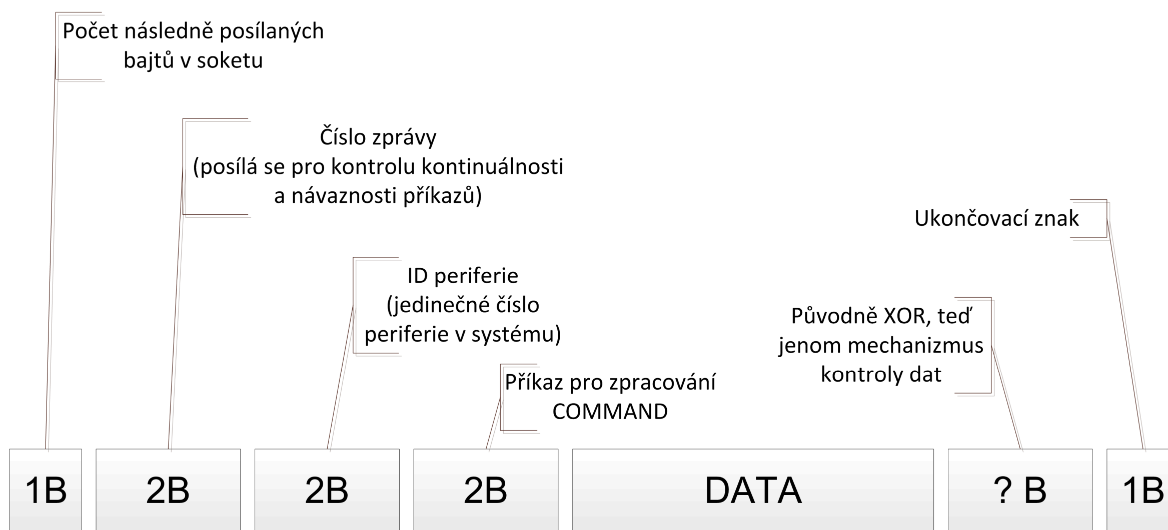
Databázový server Firebird je dostupný ve spoustě verzích. Aktuální oficiální stabilní verze, kterou jsem použil, je v.2.1.3. Jelikož jsou při instalaci dostupné k výběru dvě možnosti („classic“, „super server“), tak jsem zvolil „super server“, protože je vhodnější pro vícevláknové aplikace. V námi zvolené verzi je pro každé připojení vytvořeno separátní vlákno, které má vlastní data. V případě selhání spojení a ukončení vlákna, budou ostatní připojení zachována. Popis, kde je dostupný server ke stažení, je uvedeno v sekci požadavky aplikace.

4.1 Komunikační protokol

Pro komunikaci je vytvořena univerzální třída TCommunicationProtocol, kterou lze použít pro komunikaci jak na straně serveru, tak i na straně klienta. Pro použití stačí pouze vytvořit instanci třídy a objekt je připraven posílat data. Jako vstupní parametr se předává tzv. „IOHandler“, který řídí vstupně výstupní operace Indy komponent typu klient/server.

Struktura posílaných dat

Pro komunikační účely se zavedla struktura posílaných bajtů. Posloupnost informací je vidět na následujícím obrázku.



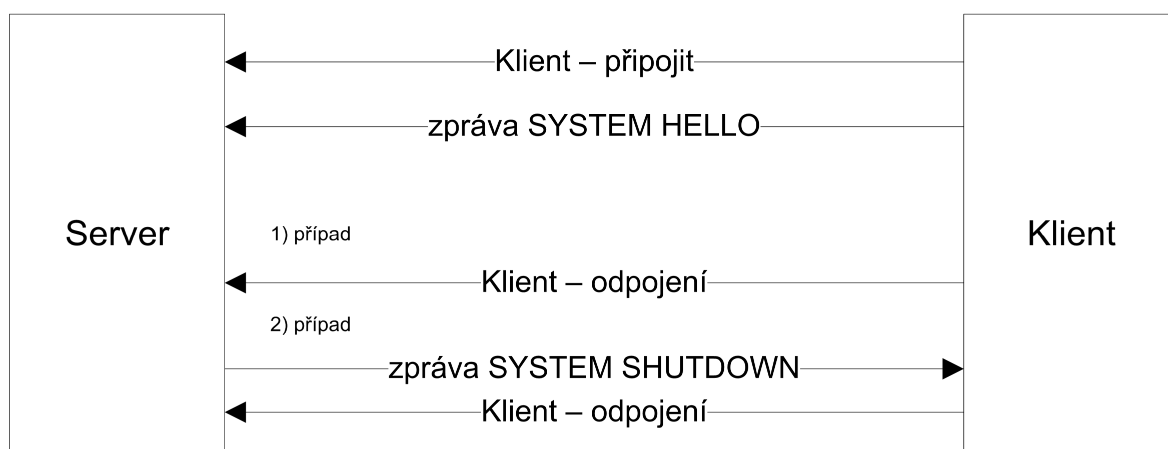
Obrázek 4 – Struktura posílaných dat v komunikačním protokolu

Protože předem není známo, jak velká budou data, tak první byte nese informace o počtu *následujících* bajtů, které se mají ještě přečíst ze soketu. Pro přesnost může

hodnota nabývat až čísla 255, proto je tímto omezena délka posílaných dat. Nepředpokládá se přenos větších celků dat a v případě potřeby je možné jednoduše implementovat podporu přenosu většího objemu dat. Následují informace o čísle zprávy a identifikační číslo, které identifikuje odesílatele (periferie). Samotná zpráva se skládá z 2 bajtového příkazu a dat, která následují za příkazem. V našem případě se pro příkaz používá kódování ANSI, takže příkazy mohou být buď jednoznakové, nebo dvouznakové. XOR kontrola je vynechána, protože protokol TCP již sám zajišťuje kontrolu dat a jejich korektnost po přijetí.

Pravidla komunikace

Pravidla pro komunikaci se musela zavést, protože je systém založený na rozpoznávání ID periferií. Z tohoto důvodu nám nestačí přebrat navazování komunikace z TCP/IP (oznámení o stavu připojení/odpojení klienta). Ve chvíli, kdy se klient přihlásí, tak stále neznáme jeho ID a server neví jaké zařízení je na druhé straně. Vzniká zde potřeba, aby klient hned po navázání spojení poslal tzv. „SYSTEM HELLO“ oznamovací zprávu, kterou oznámí serveru, že je připraven na komunikaci. Zároveň v této zprávě přijde ID zařízení a spojení s periferií je korektně navázáno. Na obrázku jsou zachycena jednoduchá pravidla připojení a odpojení.



Obrázek 5 – Pravidla komunikace mezi severem a klientem

Odpojení může proběhnout dvěma způsoby:

1. Klient se odpojí sám, pak dojde k implicitnímu ukončení komunikace.

2. Server nařídí periferii, aby se odpojila. Poté dojde k implicitnímu ukončení komunikace.

4.2 VCL komponenta mmTCPServer

Komponenta byla vytvořena na požadavek vedoucího práce. Usnadní ve výsledku implementaci serverové části do aplikace. Výhodou je zapouzdření celého serveru do jedné komponenty, která je uživatelsky příjemná a jednoduchá pro použití. Nevyžaduje znalost vnitřní struktury a nastavení se provádí pomocí „public properties“ (veřejné proměnné nastavitelné v grafickém rozhraní).

4.2.1 UML diagram tříd

Viz příloha A. Pro svou rozsáhlost byl diagram zjednodušen a u tříd jsou vypsány pouze nejdůležitější prvky, aby byly dobře zachyceny vztahy mezi jednotlivými třídami.

4.2.2 Popis implementovaných tříd

TmmTCPServer

Je třída serverové komponenty, která má dle zvyklostí vytváření komponent jako předka třídu TComponent. Tato třída zapouzdřuje třídu TIdTCPServer, která umožňuje připojení a komunikaci po síti pomocí protokolu TCP/IP. TIdTCPServer už zapouzdřuje a umožňuje vytváření více vláknové aplikace. Při inicializaci se uvede do instance TIdTCPServeru, na kterých IP adresách a portech má naslouchat (tzv. „bind“). Při spuštění serveru se vytvoří potřebný počet vláken, která naslouchají na daných portech (popřípadě IP adresách). Pro každé nové příchozí připojení server vytvoří další vlákno, které bude obsluhovat komunikaci pro daného klienta na druhé straně.

Přehled a popis důležitých proměnných, procedur a funkcí:

- *property* Active – udává, jestli je server aktivní (připojený a naslouchá)
- *property* AutoHandleToDB – ovlivňuje, zda bude komponenta sama zařizovat přesměrování komunikace rovnou do databázové části
- *property* Bindings – je seznam portů a IP adres k naslouchání. Může obsahovat libovolný počet definic.
- *property* ClientCoreEnableEvents – ovlivňuje, zda bude každý objekt klientského jádra (vlastní ho každé vlákno) spouštět události k případnému odchycení a zpracování. Například události typu: přišla zpráva, odešla zpráva, připojen klient, odpojen klient.
- *property* EchoTimeout – po jaké době se má kontrolovat spojení s klientem
- *property* EchoPokusuDoOdpojeni – jak název napovídá, je to počet neplatných pokusů o připojení ke klientovi. Pokud se n-krát nepovede korektně zkontrolovat spojení, server klienta implicitně odpojí.

- *property* DBparams – jsou informace pro připojení k databázi Firebird (standardně: name = c:\pokus\databaze.fdb, user = SYSDBA, pwd = masterkey, charSet = WIN1250, serverName = localhost, port = 3050)
- *procedura* ClientCoreCreateSettings – je inicializační část pro data, která se připojí do klientského vlákna.
- *procedura* SendDemandToDisconnectClients – pošle všem připojeným zařízením příkaz, po kterém by měla ukončit svou komunikaci.
- *procedura* ServerConnect – procedura, kterou volá zapouzdřený Indy sever při novém připojení klienta. V tuto chvíli už je vytvořena instance třídy TIdContext, která obsahuje informace o daném soketu, připojení a klientovi na druhé straně. V parametru procedury je nám předán ukazatel na konstantní objekt právě vytvořeného kontextu.
- *procedura* ServerDisconnect – procedura, kterou volá zapouzdřený Indy sever při odpojení klienta.
- *procedura* ServerExecute – událost, která je volána pokaždé, když přijde nějaká akce na nějaký kontext (připojení, odpojení a výjimky mají vlastní obsluhu).
- *procedura* ServerException – obsluha výjimek pro zapouzdřený server.
- *procedura* TimerServerExecute – je obsluha události vnitřního časovače komponenty, která má za úkol řídit čítací časovače ve všech aktivních vláknech.

Přehled a popis vlastněných objektů:

- Server – je instance třídy TIdTCPServeru, pro připojení a komunikaci pomocí TCP/IP protokolu.
- FDBGlobal – je objekt pro připojení do databáze pro účely komponenty.
- FTimer – objekt pro interní časovač. Je instancí od standardního VCL TTimer. Vlastníkem je hlavní okno aplikace.
- FmmDeviceBox – tento objekt hlídá a řídí přístup periferií. (popis u třídy TmmDeviceBox)

TmmDeviceBox

Tato třída byla vytvořena pro obsluhu a realizaci požadavků klientských vláken v serverové komponentě, proto je chráněna synchronizačním nástrojem tzv. „mutexem“. Mutex zabraňuje násobnému přístupu do procedur této třídy. Znamená to, že každé vlákno musí počkat, než se zpracuje požadavek vlákna před ním.

Hlavní činností třídy je řešení problému přístupu k serveru. Popis problému a řešení je v kapitole „Problémy spojené s realizací“.

Dále ještě zajišťuje audit zpráv od klientských vláken a zapisuje je do souboru.

Přehled a popis důležitých proměnných, procedur a funkcí:

- *proměnná* FlstDevice – je instance třídy TList a shromažďuje objekty TmmDevice, které udržují informace o periferii, které jsou potřebné k pozdějšímu odhlášení periferie ze systému.
- *proměnná* FlstContexts - je instance třídy TList a shromažďuje ukazatele na připojené instance třídy TIdContext.
- *property* Enable – udává, zda je činnost objektu povolená nebo ne.
- *property* LogPath – je cesta, kde má tento objekt vytvořit logovací soubor.
- *procedura* AddDevice – slouží k registraci zařízení do prostoru serveru.
- *procedura* DeleteDevice – slouží k odhlášení zařízení z prostoru serveru.
- *procedura* AddLog – přidání zprávy do losovacího souboru.

TmmDevice

Je pouze pro účely třídy TmmDeviceBox. V objektu se uchovávají informace o připojených a registrovaných zařízeních v systému, které mají korektní přístup k databázi. Informace, které udržuje, je identifikační číslo periferie a ukazatel na svoji instanci kontextu.

TClientCore

Objekt je určen k obsloužení potřeb a udržování lokálních dat v klientském vlákně. Jeho předek je obecná třída TObject. Po připojení a vytvoření kontextu, který nám vrátí zapouzdřený TIdTCPServer, k němu připojíme novou instanci třídy TClientCore. Připojení vznikne přiřazením ukazatele do instance třídy TIdContext. Instance má proměnou *data*, která je typu TObject, a proto je možné propojit kontext s klientským jádrem.

Třída vlastní další podpůrné objekty pro komunikaci s databází, přístup k vizuálním prvkům hlavního programu a komunikaci po TCP dle pravidel systému. Objekt této třídy existuje ve vlákně po celou dobu od připojení daného zařízení po jeho odpojení. Při odpojení zařízení je tento objekt korektně uvolněn (uzavírá se databázové spojení, zpracovávají se poslední příchozí a odchozí zprávy a odhlašuje se zařízení ze systému).

Přehled a popis důležitých proměnných, procedur a funkcí:

- *property* Context – je ukazatel na instanci svého kontextu.
- *property* CallPlanovaneUdalosti_mamTamZaznam – je důležitá proměnná. Standardně ji nastavuje časovač serveru, poté co provede náhled do databáze, která zařízení mají něco naplánovaného. V případě, že objekt má v databázi nějaké naplánované úkoly, ohlásí mu to tato proměnná a následně si objekt úkoly sám obslouží.
- *property* IDperiferie – je identifikační číslo zařízení na druhé straně.
- *procedura* CallPrechod – metoda se volá vždy v případě příchozí zprávy ze soketu. Předává příchozí data do databázového rozhraní ke zpracování a po vrácení dat z databáze odešle odpověď zpět do zařízení.

- *procedura* CallPrechod_abstractDisconnect – tato metoda slouží pouze k imaginárnímu odpojení zařízení. Aby databázové rozhraní bylo ve správném stavu, tak poslední provedená akce periferie musí být vždy odpojení. Proto je metoda volána implicitně vždy při odpojení zařízení, ať už korektním odpojením nebo při ztraceném spojení.
- *procedura* CallPlanovaneUdalosti – metoda zajišťuje vybrání a přeposlání naplánované události z databáze.
- *procedura* ContextExecute – je hlavní metoda, kterou volá server v případě nějaké události v našem vlákně. Tato metoda přijímá zprávy, směruje data do databáze a kontroluje spojení.
- *procedura* ContextConnect – metoda volaná při připojení zařízení
- *procedura* ContextDisconnect – metoda volaná při odpojení zařízení. Zároveň se stará o korektní uvolnění všech závislých objektů.
- *procedura* HandleReturnedDataFromDB – soukromá metoda, která definuje pravidla, jak zpracovat vrácená data z databázového rozhraní. Např. výstupní kód „ER“ je definován jako chyba v databázi, která se neposílá zpět klientovi, ale pouze se zapisuje do souboru a zobrazí se v grafickém rozhraní.

Přehled a popis vlastněných objektů:

- Visuals – instance třídy TmmClientCoreVisuals. Jsou v ní uložené ukazatele pro přístup na vizuální prvky v hlavním formuláři.
- CountTimer – instance třídy TmmClientCoreCountTimer. Je čítací časovač, řízený hlavním časovačem v komponentě.
- DBcore – instance třídy TDBcore. Tento objekt zapouzdřuje komunikaci s databází.
- CPcore – instance třídy TCommunicationProtocol. Slouží ke komunikaci podle pravidel systému.
- LastPacket – instance třídy TCommunicationProtocolData. Ukazatel na poslední přijatou zprávu od klienta.

TDBcore

Je speciálně určen ke komunikaci s databází. V konstruktoru přebírá data potřebná k připojení do databáze a vytváří lokální připojení do databáze pro dané vlákno. Zapouzdřuje práci s databází, takže v budoucnu není problém vyměnit objekt za jiný a tím se v případě potřeby připojit na jiný druh databáze.

Dotazy do databáze z vláken musejí být synchronizovány, proto je v každé přístupové metodě dotaz chráněn mutexem. Pro zajištění provedení dotazu v databázi, je kvůli tzv. „deadlocku“ implementován kontrolní mechanizmus, který v případě opakovaně nepovedeného pokusu zaznamená chybu a odešle upozornění periferii. Počet opakování lze ovlivnit v nastavení.

Přehled a popis důležitých proměnných, procedur a funkcí:

- *property* DBactive – nastavením proměnné se ovládá zapnutí a vypnutí připojení do databáze.
- *property* MaxPocetPokusuJednohoDotazu – udává počet pokusů opakování v případě, že se nepovede dotázat do databáze.
- *procedura* PlanUdalosti_Query – metoda pro výběr všech aktuálně naplánovaných událostí pro dané zařízení. V parametru se předává odkazem instance třídy TObjectList, která po vrácení obsahuje seznam zpráv k odeslání.
- *procedura* Prechod_Query – je metoda pro zpracování příchozí zprávy od periferie. Data se předají do databázové části, kde dojde k jejich zpracování. V parametru aMsg, předávaný odkazem, se vrací zpracovaná data z databázové části.

TDBGlobalCore

Třída TDBGlobalCore je potomkem třídy TDBCore. Třída je určena pro použití a potřeby hlavního programu.

Přehled a popis důležitých proměnných, procedur a funkcí:

- *funkce* Peripheral_Load – nahraje z databázové tabulky seznam všech zařízení, která by se měla aktuálně k serveru připojit.
- *funkce* CheckPlanUdalosti – tato metoda kontroluje v databázi, zda některé zařízení nemá naplánovanou nějakou událost. Funkce vrací v parametru seznam ID zařízení, která mají připravenou naplánovanou událost.

TCommunicationProtocol

Třída, která byla implementována za účelem zapouzdření pravidel komunikace systému, udává, jakou strukturu udržují posílané zprávy mezi serverem a klientskými zařízeními. Implementuje kontrolní mechanizmy, mechanizmus odpovědi na tzv. „ECHO“ zprávu a obsahuje podporu pro posílání v různém kódování.

Důležité je, že tzv. „ECHO“ zprávy se nezapočítávají do přijatých a odeslaných zpráv.

Přehled a popis důležitých proměnných, procedur a funkcí:

- *property* CountMsgSended – vrací počet odeslaných zpráv
- *property* CountMsgReceived – vrací počet přijatých zpráv
- *statická funkce* Decode – je přetížená funkce, pro převod z textu na bajty, anebo obráceně. První parametr jsou vstupní data a druhý je označení, o jaké kódování se má jednat. Funkce pak vrací buď text, anebo pole bytů.
- *funkce* Send_Msg – metoda pro poslání zprávy
- *funkce* Receive_Msg – blokující metoda pro přijetí zprávy. Při volání je vlákno blokováno do doby, než přijme potřebná data, pak vrací přijatou zprávu.

TCommunicationProtocolData

Je struktura pro potřeby předávání informací mezi instancemi tříd TClientCore a TCommunicationProtocol. Obsahuje: číslo zprávy, ID zařízení, příkaz („cmd“), data zprávy a ukončovací znak.

TmmClientCoreCountTimer

Třída nahrazuje klasický objekt časovače, protože ten není tzv. „thread safe“ znamená to, že není zaručena jeho funkčnost, když ho neřídí hlavní proces. Funguje na principu snižování lokální proměnné až na 0. Ve chvíli, kdy je proměnná rovna nule, je spuštěna metoda pro obsluhu čítacího časovače. Odečítání je řízeno jediným časovačem ze serverové komponenty, kde je funkce časovače správná.

Přehled a popis důležitých proměnných, procedur a funkcí:

- *property* Count – je aktuální hodnota čítacího časovače.
- *property* Enable – povolení/zakázání funkčnosti objektu
- *property* TimeoutInterval – je maximální hodnota, od které se má odečítat.

TmmClientCoreVisuals

Jak název napovídá, tak třída slouží k ukládání ukazatelů na vizuální prvky, se kterými může klientské jádro pracovat. Třída implementuje navíc podporu pro vytváření grafu pomocí instance třídy TmmFileChart.

Přehled a popis důležitých proměnných, procedur a funkcí:

- *property* Enable – povolení/zakázání funkčnosti objektu.
- *property* OwnListBox – je ukazatel na ovládanou VCL komponentu.
- *property* OwnListItem – je ukazatel na ovládanou VCL komponentu.
- *funkce* LocalLog – přidává textovou proměnou na konec seznamu v komponentě OwnListBox
- *funkce* SetStavInList, SetInfoInList, SetIPInList – nastavují textové pole v OwnListItem podle nastavených indexů, které jsou uložené uvnitř objektu.

Přehled a popis vlastněných objektů:

- FileChartStatistic – instance třídy TmmFileChart. Implementuje podporu pro vytváření grafu.

TmmFileChart

Třída je implementována pro možnost grafického výstupu a tvoření statistik. Zatím pouze zapisuje data do souboru v daném formátu, který lze následně načíst do grafu. Z důvodu zdržujících operací se souborem a kvůli přebytečnosti, se zatím tato třída nevyužívá.

TFDbConnectionsParams

Třída využívaná komponentou TmmTCPServer pro nastavení parametrů připojení k databázi. Jejím předkem je TPersistent, aby ji bylo možné použít ve vizuálním prostředí v liště vlastností (tzv. „properties“) komponenty.

TMsgData

Je to struktura, která slouží pro potřeby přenosu informací v databázovém objektu. Drží data pohromadě u sebe a oproti třídě komunikačního protokolu má navíc aktuální stav zařízení. Stav zařízení vždy vrací databázová část po zpracování dat.

4.3 Serverová aplikace

Serverová aplikace implementuje výše zmiňovaný TmmTCPServer. Aplikace poskytuje paralelní přenos informací od klienta do databáze a zpět. Pro uživatele zřizuje přímé řízení připojených periferií a vizuální audit událostí serveru i zařízení.

4.3.1 Popis aplikace

Skládá se ze tří formulářů. Hlavní formulář (TfrmMain), datový modul (TDMnonVisual) a klientského formuláře (TfrmClient).

TfrmMain

Hlavní formulář je osazen komponentou TPageControl, pro vizuální rozdělení okna na skupiny podle druhu nastavení.

Hlavní záložka „Periferie“ je pro kontrolu a řízení periferií. Jako základ byla zvolena komponenta TListView, protože každý řádek je oddělený seznam prvků, který lze přiřadit zvlášť do jednotlivých vláken. Navíc poskytuje tabulkové a přehledné zobrazení periferií a jejich stavů. Ve spodní části je schovaný panel s komponentou TListBox, kde se uchovávají informace o nerozpoznaných periferiích.

Záložka „Informace / Audit“ obsahuje komponentu TListBox pro shromažďování informací o serveru a jeho stavu, stavu klientských vláken a audit událostí, které jsou nastaveny k auditu (připojení, odpojení serveru, výjimky atd.).

Záložka „Nastavení serveru“ je pro nastavení serveru. Na levé straně se definuje, na kterém portu a IP adrese má server po spuštění naslouchat.

Záložka „Grafy“ je jednoduchý grafický výstup, který sleduje vytížení serveru.

Přehled a popis důležitých proměnných, procedur a funkcí:

- *proměnná* FactualListItem – udržuje aktuálně označený řádek. Ten udává aktuální zařízení, se kterým se může pracovat pomocí kontextové nabídky.
- *procedura* BoundIPtoServer – přiřadí informace o IP adresách a portech, na kterých má server naslouchat.

- *funkce* CheckActualListItem – vrací, zda je nastavený ukazatel na daný kontext přiřazený k řádku zařízení, anebo jestli je už nefunkční. Používá se u volání z kontextové nabídky pro daný řádek.
- *procedura* ListOfPeriferalsLoadDB – volá globální připojení do databáze a načte z ní seznam zařízení, která by se měla připojit.
- *procedura* LoadDefaultValues, SaveDefaultValues – načítá a ukládá nastavení do INI souboru.
- *procedura* SetEnvirovment, UnsetEnvirovment – nastavují nebo vymažou ukazatele v datovém modulu na vizuální prvky.

TDMnonVisual

Je datový modul pro pohyb dat v programu. Obsahuje jednu komponentu a to TmmTCPServer.

Přehled a popis důležitých proměnných, procedur a funkcí:

- *property* CountConnection – je aktuální počet připojení.
- *procedura* CreateGlobalDBconnection – vytvoří připojení do databáze.
- *procedura* MakeStatistics – vytváří graf v hlavní aplikaci, který udává počet přijatých zpráv za určitou dobu (než přijde další zpráva), nejkratší interval je jedna vteřina.
- *procedura* WhenIDreceive – obsluha pro událost serveru. Metoda provede kontrolu, zda je zařízení v hlavním ListView. V případě, že se tam nachází řádek, nastaví mu ukazatel na objekt kontextu a příznaky na připojeno. V případě, že nenajde daný řádek, zařadí se do nerozpoznaných zařízení.
- *procedura* WhenContextDisconnect – obsluha pro událost serveru. Když se zařízení odpojí, tak se vymaže ukazatel na kontext a nastaví se textové stavy. Popřípadě se vymaže zařízení ze seznamu nerozpoznaných zařízení.
- *procedura* WhenContextException – obsluha pro událost serveru. Tato metoda využívá zálohovaného ukazatele na kontext, který je uložen na jiné pozici než přístupový. Podle něj může metoda rozeznat, které zařízení vyvrhlo výjimku a nastaví příslušné textové příznaky.
- *procedura* WhenContextMsgReceive – je volána při přijetí zprávy. Metoda volá metodu pro přidání bodu do grafu.

Přehled a popis vlastněných objektů:

- DBcore_DMGlobal – instance třídy TDBGlobalCore. Podpora pro připojení do databáze.

TfrmClient

Jednoduchý formulář na zobrazení informací pouze o jednom zařízení. Lze ho zapnout z kontextové nabídky nad řádkem periferie, které chceme sledovat. Pro každé

zařízení se vytvoří pouze jeden tento formulář. V případě vytvoření se uloží ukazatel na instanci této třídy přímo k řádku zařízení a při dalším vyvolání se už jenom zobrazuje.

Přehled a popis důležitých proměnných, procedur a funkcí:

- *proměnná* FClientContext – ukazatel na kontext zařízení.

4.3.2 Realizace přímého řízení periferií

Řídit periferie je umožněno pomocí kontextové nabídky (při zmáčknutí pravého tlačítka myši) nad zařízením, které chceme ovládat.

V kontextovém menu je nabídka akcí (předdefinovaných zpráv), které lze odeslat periferii. Je k dispozici také univerzální dialog pro odeslání specifické zprávy, definované přímo v dialogu. Ukazatele na dané kontexty se k řádkům připojí automaticky, hned po korektním navázání komunikace.

4.3.3 Nerozpoznané zařízení

V případě, že se k serveru bude hlásit zařízení, které se zrovna nemá hlásit, anebo je špatně nastavené, tak je server sleduje pomocí komponenty ListBox. Pomocí tlačítka nebo z menu, lze zobrazit aktuální seznam takto špatně přihlášených periferií. V komponentě se zobrazí informace, odkud se zařízení připojuje (IP a port) a jeho identifikační číslo.

4.4 Databáze

Databázová část má dvě pomyslné části. První se stará o řízení a zpracování dat od klienta a druhá o správu dat, která do ní ukládáme. Navrhnul jsem PSQL vrstvu nad databází, která na každý klientský *dotaz* následně vrátí *odpověď*.

Důležité je, že každá obslužná funkce v PSQL vrstvě má stejné vstupní a výstupní parametry. Pro potřeby je v databázi uložena F_STAV_SABLONA, která má již předdefinované vstupní a výstupní parametry, a proto lze další obslužné funkce vytvářet podle ní.

Aplikační část má několik implementovaných pravidel pro zpracování odpovědí. Pokud není vyplněn kód, tak server zařízení žádná data neposílá, pouze zobrazí obsah dat v aplikaci serveru. Pokud je kód „ER“, znamená to, že nastala chyba zpracování a klientovi se může poslat oznámení, že došlo k chybě a nebyl splněn jeho požadavek. Pokud kód i data jsou prázdné, tak server nic neprovádí.

Zpracování dotazu od klienta

Jednotlivé zařízení mají podle typu nadefinovaný přechodový diagram. Při zpracování má každá stanice vlastní data, ale sdílí stejný diagram přechodů.

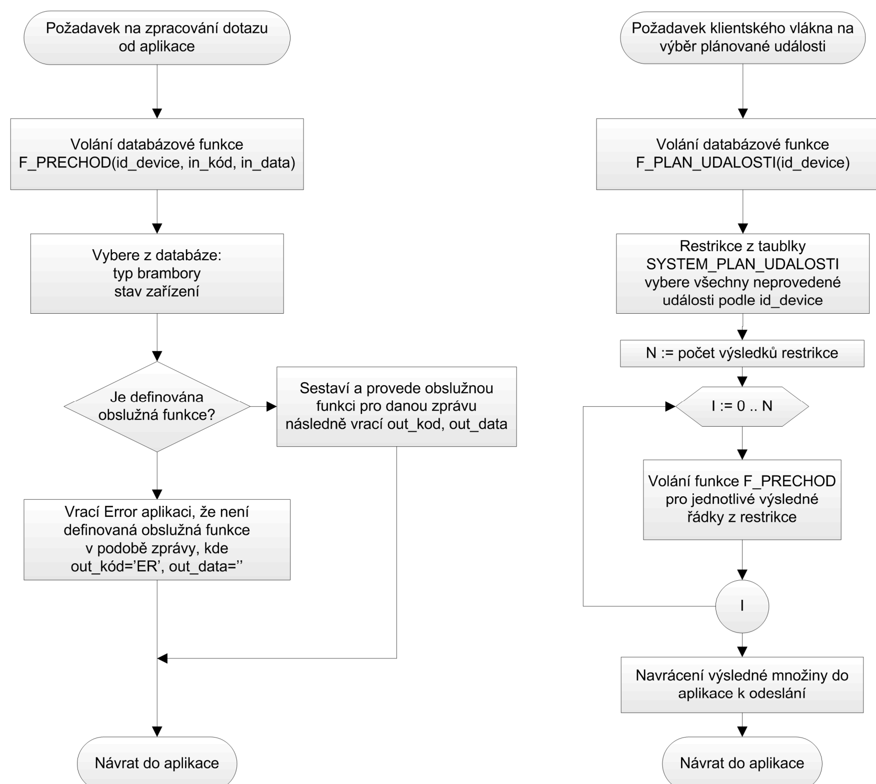
Zpracování probíhá vždy způsobem, že přijde *dotaz* od klienta. Daný dotaz aplikace předá do funkce v PSQL vrstvě zvanou F_PRECHOD. Této funkci se předá *příkaz* (dále

jen *kód*), data a ID zařízení. Přechod následně vybere z tabulky zařízení typ a aktuální stav zařízení. Podle stavu, kódu a typu zařízení, si z tabulky SYSTEM_STAVY_OBSLUHA zjistí příslušnou obslužnou PSQL funkci a spustí ji s daty od klienta. Obslužná funkce následně vrací kód a data k odeslání, která F_PRECHOD pouze předá zpět programu.

Zpracování naplánované události

Pokud má periferie naplánovanou nějakou akci, zavolá PSQL funkci F_PLANOVANE_UDALOSTI. Této funkci se předává v parametru pouze ID zařízení. Funkce provede následně restrikci do tabulky SYSTEM_PLAN_UDALOSTI (záznam obsahuje stručně: id_zařízení, čas_provedení, čas_odeslání, in_kod, in_data, out_kod, out_data) na řádky, které mají nevyplněný čas odeslání a mají příslušné ID zařízení. Všechny výsledky se postupně předávají funkci F_PRECHOD. Přechod vrátí odpověď, ta se nejdřív zapíše zpět do tabulky a vyplní se její čas odeslání a poté se předá aplikaci k odeslání. Takto se předají všechny události v danou chvíli, pokud jsou určeny už k provedení.

Pro názornější pochopení přikládám vývojový diagram zpracování v databázové části. (výraz brambora zavedl vedoucí práce pro jednotlivou definici přechodového diagramu)

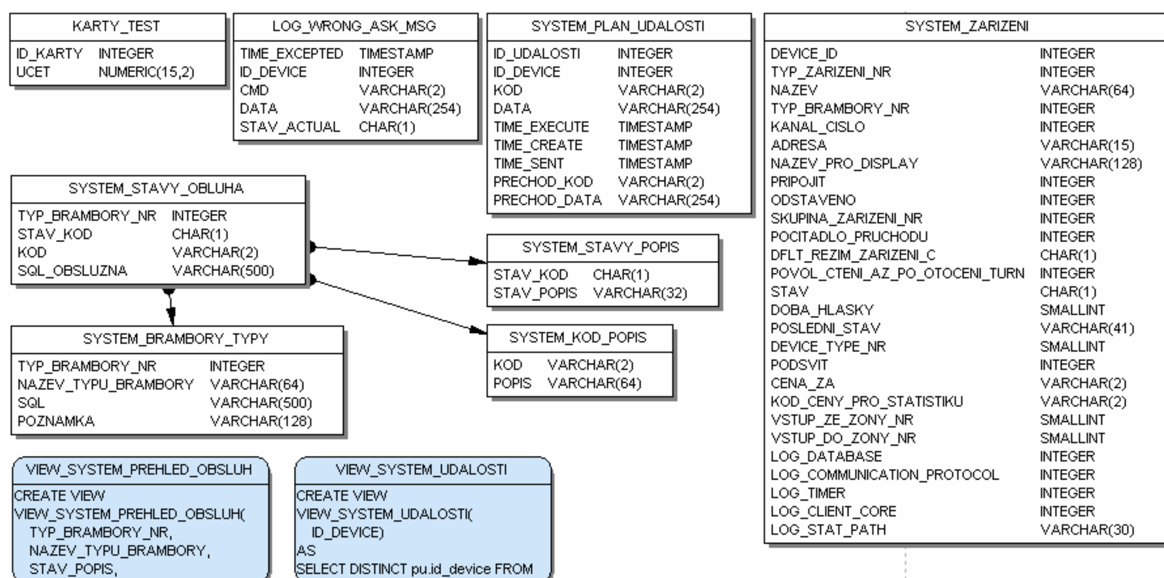


Obrázek 6 – Zpracování dat v databázové části

Integrita dat systémových tabulek

Hlavní systémová tabulka je SYSTEM_STAVY_OBSLUHA, která obsahuje 3 primární cizí klíče. Znamená to, že musí být v databázi nejdříve definovaný kód v tabulce SYSTEM_KOD_POPIS, stav v tabulce SYSTEM_STAVY_POPIS a definovaný typ brambory (typ stavového diagramu) v tabulce SYSTEM_BRAMBORY_TYPY. V rámci dalšího vylepšení je možná ještě implementace triggeru, který bude kontrolovat, jestli název obslužné funkce je mezi názvy systémového katalogu. Takto zůstane vždy zachována integrita dat.

4.4.1 ER diagram



Obrázek 7 – ER diagram

4.4.2 Popis databázových tabulek

KARTY_TEST – pouze pro testovací účely. Obsahuje seznam zákaznických karet a stavy jejich účtů.

LOG_WRONG_ASK_MSG – tabulka, kam se zapisují příchozí kódy, které neprošly databází korektně. Například, když příchozí zpráva není definovaná v daném stavu, anebo není definovaná vůbec.

SYSTEM_BRAMBORY_TYPY – tabulka, kde jsou uloženy názvy a poznámky k jednotlivým typům stavových digramů.

SYSTEM_KOD_POPIS – tabulka, kde jsou definovány kódy (příkazy).

SYSTEM_PLAN_UDALOSTI – tabulka s informacemi o naplánovaných událostech a událostech, které se už provedly.

SYSTEM_STAVY_OBSLUHA – tabulka, kde se definují vztahy mezi jednotlivými kódy a stavy pro daný typ stavového diagramu (brambory). Tabulka obsahuje pro udržení integrity dat 3 primární cizí klíče (TYP_BRAMBORY_NR, STAV_KOD, KOD).

SYSTEM_STAVY_POPIS – tabulka, kde jsou uloženy popisy jednolitých stavů zařízení. Například stav „A“ a popis například „aktivní / klid“.

SYSTEM_ZARIZENI – tabulka s aktuálními informacemi o zařízení.

4.4.3 Popis PSQL funkci a procedur

F_PLANOVANE_UDALOSTI – funkce pro výběr odpovědí na plánované události pro danou periferii. Výstup může být množina výsledků.

F_PRECHOD – hlavní funkce pro zpracování příchozího požadavku od periferie (nebo plánované události). Na jeden příchozí dotaz je vždy jen jedna odpověď.

F_STAV_CLIENT_SAYHELLO – obsluha pro první zprávu, kterou musí každá periferie poslat serveru, aby mohla přejít ze stavu „vypnuto“ do stavu „aktivní“ a dál komunikovat se serverem. V rámci této funkce by mohla periferie odesílat například šifrované heslo pro přístup. Je to jedna z možností dalšího zabezpečení.

F_STAV_INCOMING_MESSAGE – příchozí zpráva pouze pro server. Nevyplňuje se výstupní kód a zpráva se zobrazí pouze v aplikaci serveru.

F_STAV_NOT_ALLOWED_TO_PASS –

F_STAV_PRISLA_KARTA – obsluha pro zpracování příchozího dotazu na vstup karty. V datech zařízení zasílá ID karty. Proto obslužná funkce vybere příslušnou kartu v databázi a odečte jí příslušný počet kreditů, popřípadě odešle klientovi zprávu, že nelze s touto kartou vstoupit do areálu.

F_STAV_SABLONA – šablona pro další obslužné funkce.

F_STAV_SHUTDOWN – funkce pro vypnutí zařízení. Volá se vždy na konci komunikace. Zde v rámci správnosti dat a statistik lze provést určitou kontrolu informaci ,nebo zapsání informací o uskutečněné komunikaci.

F_STAV_TEST_PROPUSTNOSTI – tato obslužná funkce je implementována pro měření časové odezvy. Pouze předá data do výstupního dotazu a posílá ho zpět.

F_STAV_TIMER_DELETE – obsluha pro vymazání obrazovky zařízení. Ve většině případů je spouštěn právě plánovanou událostí.

P_STAV_ZARIZENI – tato funkce mění stav dané periferie a zapisuje předchozí stav do zálohovacího sloupce v tabulce zařízení.

4.4.4 Popis ostatních použitých objektů

Pohledy

VIEW_SYSTEM_PREHLED_OBSLUH – je přehlednější tabulka, využívající popisy z rodičovských tabulek pro zobrazení, jak jsou jednotlivé typy stavových diagramů nastaveny. Zdrojový kód je přiložen v příloze B.

VIEW_SYSTEM_UDALOSTI – je volán z aplikace. Poskytuje seznam zařízení, která mají naplánovanou nějakou akci. Server podle tohoto seznamu zajišťuje jednotlivé zpracování naplánovaných událostí v klientských vláknech. Zdrojový kód je přiložen v příloze B.

Triggery

SYSTEM_PLAN_UDALOSTI_BI – trigger spouštěný před vložením nějaké hodnoty do tabulky *plán událostí* a pokud nemá primární klíč, pak jí přiřadí automaticky novou hodnotu.

SYSTEM_ZARIZENI_BIO – trigger spouštěný před vložením nějaké hodnoty do *tabulky zařízení* a pokud nemá primární klíč, pak jí přiřadí automaticky novou hodnotu.

Generátory

GEN_SYSTEM_PLAN_UDALOSTI_ID – generátor primárních klíčů pro tabulku s *plánovanými událostmi*.

GEN_SYSTEM_ZARIZENI – generátor primárních klíčů pro tabulku s periferiemi.

4.5 Problémy spojené s realizací

Tento odstavec zařazuji z důvodu seznámení ostatních vývojářů s problémy, které se v průběhu implementace a realizace objevily. Je dobré se jich vyvarovat, protože by nebylo efektivní, kdyby se chyby podobného rázu objevily znovu a způsobily tak nějaké problémy, kterým se dá předejít.

Synchronizace přístupu ke společným datům v komponentě TmmTCPServeru

Vláknová aplikace má nebezpečnou část v tom, že některé události se mohou provádět paralelně nebo pseudoparalelně. Znamená to, že může dojít k situaci, kdy v jednu chvíli vlákna přistupují ke stejným datům. VCL jako knihovna vizuálních komponent nepodporuje přímo vláknové programování. Pouze poskytuje funkci *synchronize*, která z vlákna zavolá hlavní proces a ten už sám zajistí synchronizaci a obsloužení obsahu funkce. V našem případě potřebujeme zapisovat informace přicházející z různých vláken do souboru, který je společný pro všechna vlákna. Synchronizaci zajistíme pomocí

tzv. „mutexu," což je synchronizační nástroj, který umí pozastavit proces v případě, že se mutex použije vícekrát než jednou.

Pro vyřešení problému jsem vytvořil třídu TmmDeviceBox. Třída má implementovaný mutex. Všechny funkce, které přistupují ke společným datům, jsou zmíněným mutexem chráněny.

Přístup ke stejným datům v databázi

Databáze nám poskytuje informace, ale sama musí dodržovat integritní pravidla. Proto nám nemůže poskytnout ve stejnou chvíli data, která může měnit zrovna jiná transakce. Z tohoto důvodu je v databázové třídě TDBcore implementován mechanizmus, který hlídá, zda nedošlo k chybě a následně se pokouší provést SQL příkaz znovu. Počet opakování je definován přímo ve třídě a lze ho změnit.

Přístup k serveru v případě velkého vytížení TCP linky

Tento problém se objevil až v průběhu testování a při reálném použití se s velkou pravděpodobností neobjeví. Nicméně je lepší problémům předcházet.

Základ problému je velké vytížení serveru z jednoho zařízení (zprávy posílané za vteřinu v řádech tisíců). Může se stát, že klient vše odešle a odpojí se. Protože protokol TCP zaručuje doručení všech zpráv, server stále zpracovává data od klienta, ale klient se připojí znovu a začne opět komunikovat. V tuto chvíli se objeví problém, protože je v systému 2x zařízení se stejným identifikačním číslem, pod kterým přistupuje do databáze. V jiném případě by to mohlo vypadat tak, že mohou se serverem komunikovat dvě zařízení se stejným ID. Jako řešení byl implementován do třídy TmmDeviceBox v komponentě serveru registrační mechanizmus, který nevpustí do systému další zařízení se stejným identifikačním číslem.

5 Požadavky aplikace

V této kapitole si představíme softwarové vybavení, které je potřeba ke spuštění a provozu aplikace.

Operační systém Windows XP

V rámci této práce byl jako operační systém zvolen Windows XP professional SP3[1]. Aplikace nebyla testována na ostatních systémech a nebylo to ani vyžadováno v rámci této práce.

Databázový server Firebird

Databázový server Firebird je poskytován na domovských stránkách[2] vývojářů. Více v úvodní části práce.

Doplňkový software

IBExpert je software pro správu databází. Součástí tohoto softwaru je kromě standardních nástrojů také možnost vzdáleného přístupu do databáze, monitorování operací nad databází a vytváření metadat pro rychlé vyhledávání v databázi.

FlameRobin je multiplatformní GUI software, jak pro Windows, Linux, tak i UNIXové systémy. Jsou v něm obsaženy všechny standardní nástroje pro správu firebird databáze a měl by být dostupný ve většině základních instalací UNIXových systémů.

Hardware

Obecné omezení nebo požadavky pro hardware nejsou. Ale pro výbornou funkčnost aplikace jsou zde určitá doporučení. Výsledný výkon počítače by měl být zvolen dle potřeby realizovaného systému, ve kterém bude aplikace fungovat. V našem případě se bude aplikace používat pro ovládání zařízení pro hromadné odbavovací systémy, kde se předpokládá maximální vytížení kolem 1-2 dotazů od jedné periferie za vteřinu. Např.: vstupní brána do bazénu, kde lidé procházejí turniketem nebo mincovník.

Pro srovnání uvádím, že v případě 50 připojených periferií, kdy každá z nich se každou vteřinu dotazuje na server, je vytížení na jednoprocesorovém systému, o frekvenci 2 GHz (Turion, AMD) mezi 15% až 25%.

[1] Webová adresa návodu instalace: http://www.pcporadenstvi.cz/pruvodce-instalaci-windows-xp.
[2] Adresa ke stažení serveru Firebird: http://www.firebirdsql.org/index.php?op=files.

6 Uživatelská příručka serveru

Tato kapitola je spíše informativního charakteru, protože samotný systém je bezobslužný a zásah obyčejného uživatele se nepředpokládá. V následujících podkapitolách uvedu, jak lze jednoduše server nastavit a ovládat.

Nastavení serveru

Celé nastavení serveru je uloženo v konfiguračním ini souboru, který má název jako hlavní soubor aplikace. Uvádím přehled jednotlivých direktiv a popis jakých mohou nabývat hodnot.

Kategorie Placement:

- *Top* – pozice hlavního okna (počet pixelů odshora obrazovky)
- *Left* – pozice hlavního okna (počet pixelů odleva obrazovky)

Kategorie Log:

- *Log** – udává, zda se mají zapisovat události z určeného objektu (1 = ano, 0 = ne)

Kategorie IgnoredException:

- *ExCount* – je počet následujících definovaných výjimek
- *Ex1* – je text výjimky, kterou má server ignorovat

Kategorie Settings:

- *Host* – defaultní IP adresa, na které má server naslouchat
- *Port* – defaultní port, na kterém má server naslouchat
- *IPs* – počet následujících definicí IP adres
- *IP1* – IP adresa, na které má server naslouchat

Kategorie Database Settings:

- *Name* – cesta k souboru databáze (neuvádí se v uvozovkách)
- *User* – uživatel pro přístup do databáze
- *Pwd* – heslo pro přístup do databáze
- *charSet* – kódování databáze
- *serverName* – IP adresa pro připojení k databázi (localhost)
- *port* – port pro připojení k databázi

Spuštění serveru

Pro správné spuštění serveru je důležité mít nakonfigurované parametry připojení k databázi (viz. Nastavení serveru) a spuštěný databázový server Firebird. V případě, že je

použitá verze embedded, je aplikace schopná si sama server zajistit a není potřeba instalace serveru do počítače.

Spustíme soubor „SIMserver.exe“ a objeví se hlavní formulář programu. V záložce „nastavení serveru“ můžeme nastavit, na kterých portech bude spuštěný server naslouchat. Samotné spuštění serveru se provádí tlačítkem *spustit.* O stavu serveru nás vizuálně informuje malá ikonka v podobě led diody, která v případě funkčního serveru začne blikat zeleně. Po zapnutí je server schopen plně komunikovat a obsluhovat jednotlivá zařízení.

Ovládání aplikace

Aplikace poskytuje pouze základní ovládací prvky, jako je spuštění, vypnutí, vypnutí vizuálního prostředí, zobrazení nerozpoznaných zařízení a další. Jsou dostupné z menu programu, anebo přehledně v komponentě toolbar.

7 Ověření funkcionality

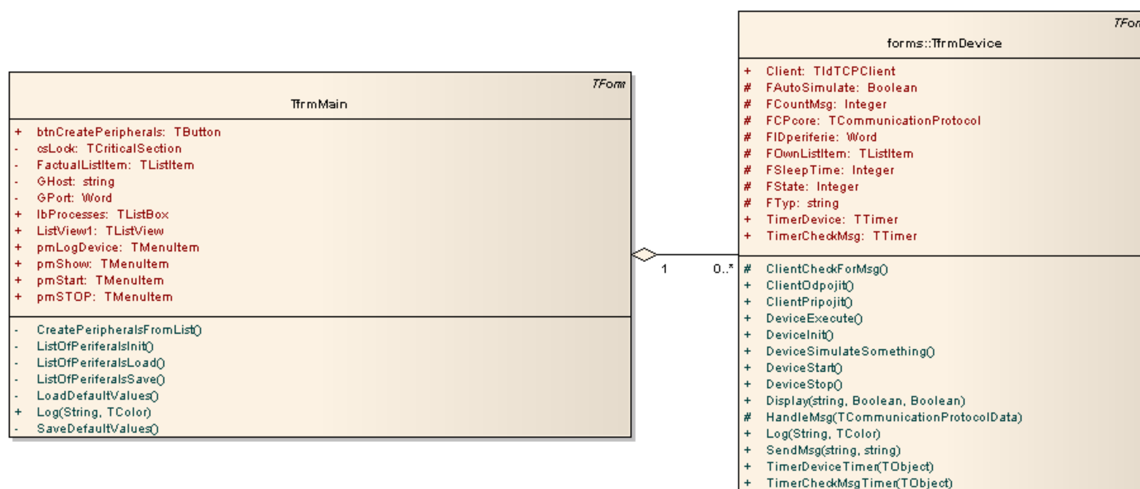
Součástí zadání od firmy bylo vytvoření aplikace, která bude simulovat činnost zařízení. Je pro testovací účely a pro ověření funkcionality serveru. Mezi informace o systému, které firmu nejvíce zajímají, je čas, který potřebuje server pro odpověď periferii při různém zatížení a různém počtu připojených zařízení.

7.1 Simulační program

Aplikace je navržena tak, aby bylo v budoucnu jednoduché implementovat další třídy, které budou zastupovat různé druhy zařízení. Jejich předek je třída TDevice, která definuje a implementuje základní vlastnosti a funkce zařízení, která jsou typově stejná. Ostatní funkce spojené se specializací periferie je už na implementaci v potomkovi.

Program je složen z hlavního formuláře, který načte z INI souboru konfiguraci a podle ní aplikace vytvoří nové formuláře pro nakonfigurované typy. V hlavním formuláři se ukládají ukazatele na všechny vytvořené formuláře do seznamu (realizován pomocí komponenty ListView). Pomocí kontextového menu (vyvolané pravým tlačítkem myši) je možné jednoduše ovládat zařízení, anebo zobrazit jeho formulář pro individuální provádění akcí.

7.1.1 UML diagram tříd



Obrázek 8 – UML diagram tříd pro Simulační program

7.1.2 Popis implementovaných tříd

TfrmMain

Hlavní formulář poskytuje funkce pro vytvoření periferií z načteného seznamu v programu. Seznam se načítá z konfiguračního INI souboru. Při implementaci jiné metody se může seznam načítat například z databáze. Dále spravuje všechny vytvořené formuláře periferií pro následnou obsluhu. Pro větší přehlednost je zde možnost (pomocí kontextového menu), zapnout audit informací z požadovaného formuláře zařízení do hlavního okna.

Přehled a popis důležitých proměnných, procedur a funkcí:

- *proměnná* ActulaListItem – udržuje aktuálně označené zařízení v seznamu.
- *procedura* CreatePeripheralsFromList – vytváří z načteného seznamu formuláře pro dané zařízení.
- *funkce* CheckActualListItemForDeviceForm – kontroluje, jestli je v daném označeném řádku korektní ukazatel na daný formulář.

TfrmDevice

Třída definuje základní chování a funkce pro simulování jednotlivých typů zařízení. Při vytvoření si zařízení generuje tzv. „délku svého života“ ve vteřinách. Po zapnutí se spustí časovač, který spouští proceduru *DeviceExecute*. Ta se na začátku pokusí připojit k serveru a v případě, že se jí to podaří, tak začne implicitně simulovat akci typickou pro periferii. V opačném případě nic nedělá a počká, než vyprší její čas „života“ a pokusí se znovu připojit. V případě, že je režim simulace vypnut, tak se postupuje stejně, jen není spuštěna obsluha pro automatickou simulaci daného typu zařízení.

Pro názornější sledování akcí a událostí zařízení je dostupná záložka *audit / informace*, kde je zaznamenáván průběh „života“ periferie.

Poslední záložka formuláře je pro testování časové odezvy serveru. Lze nastavit kolik zpráv má periferie každou vteřinu odeslat. Výsledky se vypisují do komponenty, která umožňuje textové kopírování a není tedy problém je zkopírovat do nějakého statistického programu k další analýze.

Přehled a popis důležitých proměnných, procedur a funkcí:

- *proměnná* FSleepTime – udává, jak dlouho bude zařízení pracovat v DeviceExecute.
- *property* AutoSimulate – určuje, zda je zařízení v režimu automatické simulace nebo ne.
- *property* IDperiferie – je identifikační číslo periferie.
- *property* OwnListboxLog – je ukazatel na komponentu ListBox v hlavním formuláři. Pokud je nastaven, tak se informace o stavu a událostech zapisují i do této komponenty.
- *procedura* ClientPripojit – korektním způsobem se zkusí připojit k serveru. Před připojením volá interní funkci BeforeConnectClient, která může provádět další kontroly před spuštěním.
- *procedura* ClientOdpojit – před odpojením ještě zkontroluje, zda nejsou nějaká příchozí data a potom se korektním způsobem zkusí odpojit. Před odpojením volá interní funkci BeforeDisconnectClient, která může provádět další kontroly před odpojením.
- *procedura* DeviceStart – spustí činnost zařízení (simulace fyzického zapnutí).
- *procedura* DeviceStop – vypíná činnost zařízení (simulace fyzického vypnutí).
- *procedura* DeviceExecute – definuje průběh činností zapnuté periferie a řídí její připojení a odpojení od serveru.
- *procedura* Display – zobrazí zprávu na obrazovce periferie.
- *procedura* HandleMsg – je metoda pro zpracování zpráv od serveru. Tato metoda definuje akce, které periferie provádí při daných příkazech.
- *procedura* SendMsg – pošle zprávu serveru.

7.2 Ověření doby odezvy

Pro požadavek ověření doby odezvy byl do simulačního programu implementován objekt, který zprostředkovává měření času mezi jednotlivými poslanými dotazy a po navrácení odpovědi poskytne výsledný čas.

Princip měření je založen na odeslání zprávy serveru, kde v datové části se nese pořadové číslo testu. Server následně posílá zpět číslo testu a implementovaný objekt podle něj zjistí začátek odeslání a vypočítá dobu zpracování.

Pro měření času nestačí použít funkci „GetTime“ ani „GetTickCount“. Tyto funkce nejsou pro měření výsledků dostatečně přesné, proto byla použita funkce Windows API „QueryPerformanceCounter“, která vrací aktuální hodnotu časového registru. Protože je vrácená hodnota registru závislá na frekvenci, je nutné ji následně dělit hodnotou frekvence, kterou vrací funkce „QueryPerformanceFrequency“. Tím dosáhneme potřebné přesnosti měření. Výsledky jsou uváděny ve vteřinách s minimální přesností na desetitisíciny.

Měření bylo provedeno postupně pro počty 1, 5, 20, 50 a 100 simulovaných zařízení a to pro extrémní situaci, kdy každé zařízení generovalo a odesílalo definovaný počet dotazů na server. Definované počty byly 1, 5, 10, 100 a 500 zpráv za vteřinu. Pro každý definovaný počet se provedlo 100 pokusů (zhruba po vteřině) a z nich se vypočítala průměrná doba odezvy serveru pro daný simulační scénář.

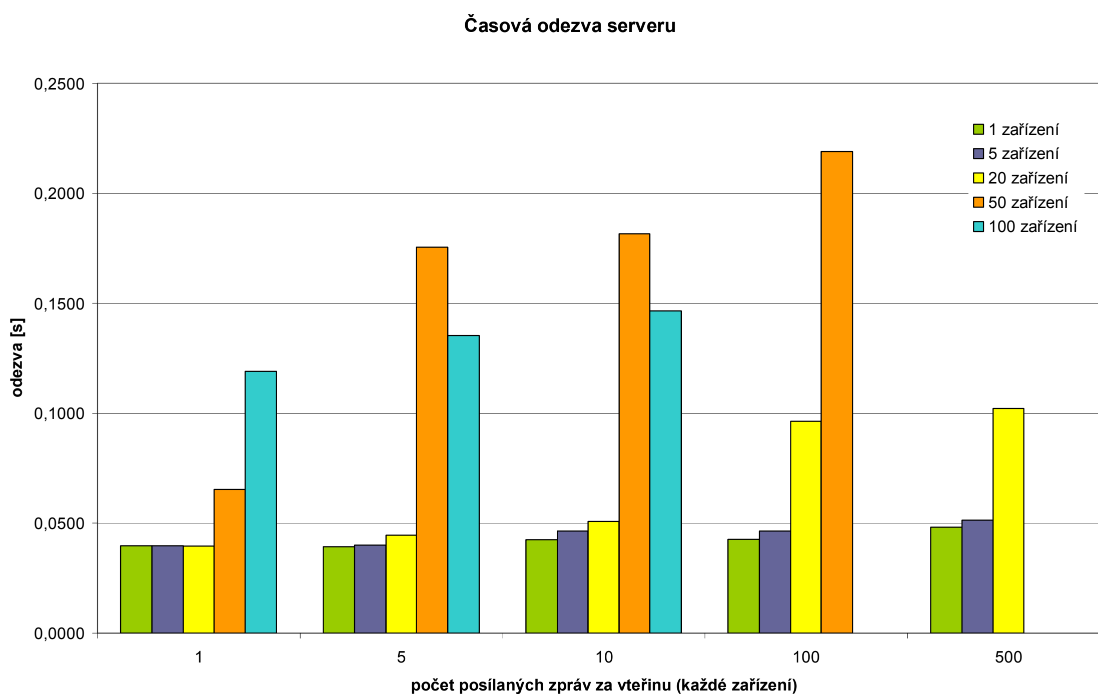
7.2.1 Výsledky

Mezi výsledky uvedu tabulku s přehledem jednotlivých výsledných průměrů časové odezvy. Dále pak grafické znázornění výsledných průměrů a porovnání odezvy serveru v případě, že se na straně serveru vypne podpora vizuálního režimu.

		POČET ZPRÁV ZA VTEŘINU				
		1	5	10	100	500
POČET ZAŘÍZENÍ	1 zařízení	0,0396	0,0393	0,0424	0,0426	0,0482
	5 zařízení	0,0398	0,0399	0,0462	0,0464	0,0513
	20 zařízení	0,0395	0,0443	0,0507	0,0964	0,1022
	50 zařízení	0,0654	0,1754	0,1816	0,2189	-
	100 zařízení	0,1190	0,1354	0,1465	-	-

Tabulka 1 – Přehled výsledků časové odezvy serveru v [s]

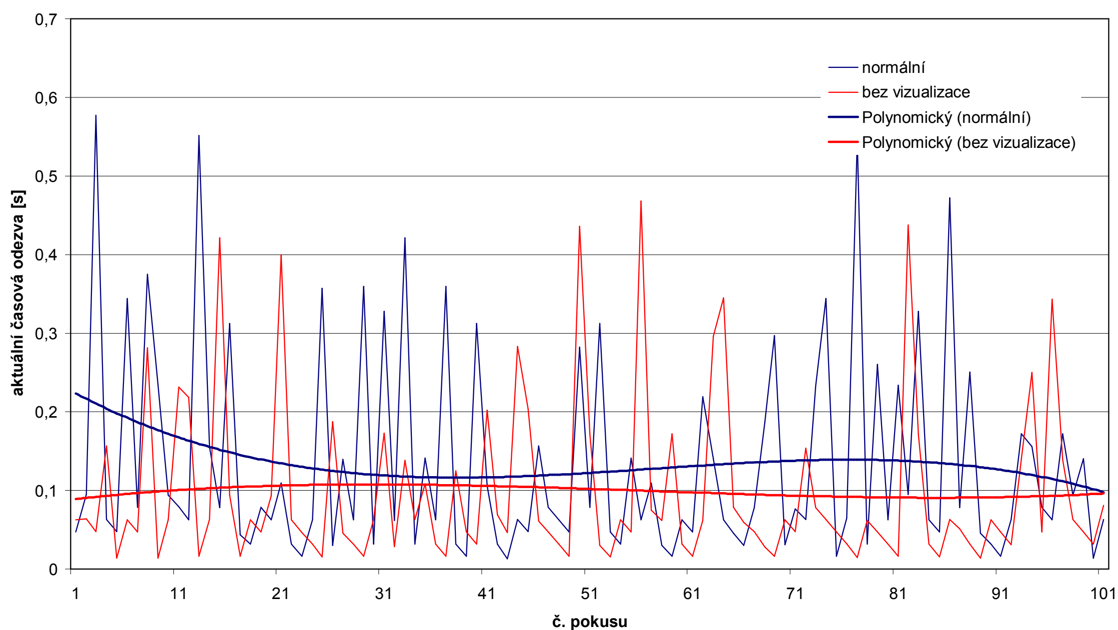
Jednotlivé průměry jsou uvedeny ve vteřinách s přesností na desetitisíciny.



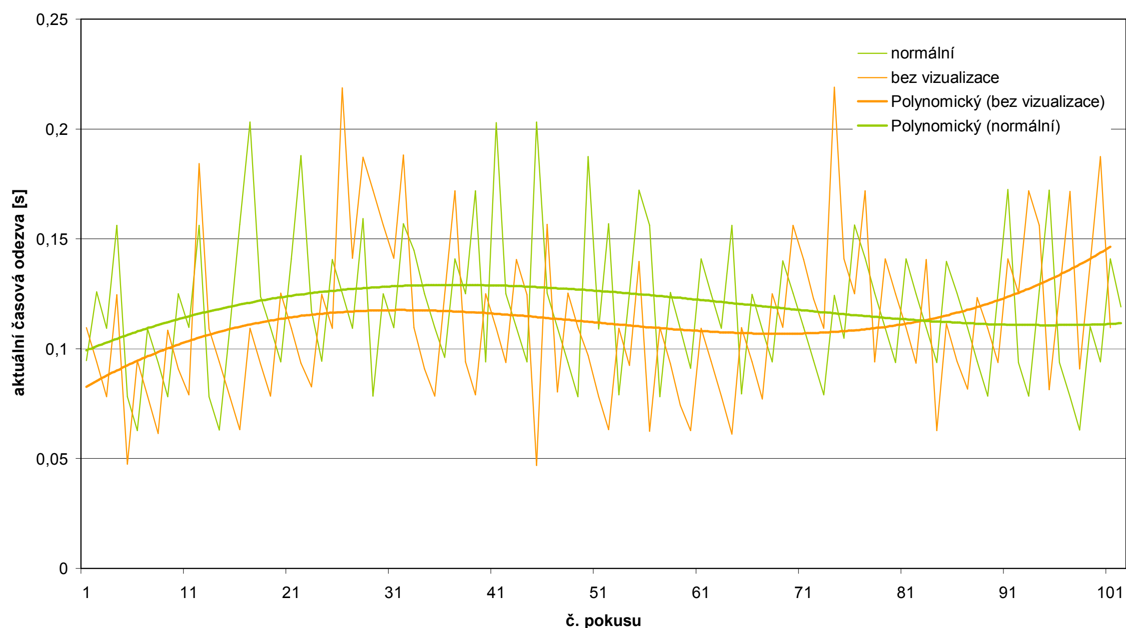
Obrázek 9 – Graf časové odezvy serveru

Následující grafy zachycují rozdíl mezi normálním režimem a režimem, který vypne vizuální prostředí. Při vypnuté vizualizaci server řídí pouze přesměrování dat do databáze a zpět. Test proběhl s 50 simulovanými zařízeními a definovaným počtem zpráv.

Pro lepší odhad chování křivky jednotlivých měření je danou křivkou proložen polynom 3. řádu. V prvním grafu lze vidět případ pro 1 zprávu za vteřinu a v druhém pro 5 zpráv za vteřinu.



Obrázek 10 – Graf pro porovnání časové odezvy v režimu bez vizualizace (1 z./s)



Obrázek 11 – Graf pro porovnání časové odezvy v režimu bez vizualizace (5 z./s)

8 Závěr

Cílem práce bylo vytvořit programové vybavení pro komunikaci a řízení průmyslových periferií ovládaných mikropočítačem typu Atmel a simulačního programu pro následné ověření funkčnosti a časové odezvy serveru při různém zatížení.

Na začátku vývoje, po konzultaci s firmou, byla formulována malá odchylka od zadání. U periferie není potřeba vytvářet TCP/IP server pro oznámení, že je periferie připravena komunikovat. Server bude obsluhovat každou periferii, která se dokáže připojit. V případě výpadku se periferie znovu připojí a pokračuje v komunikaci. Tím odpadá potřeba předem oznamovat serveru stav zařízení. Společnost také momentálně nevyužije ani volitelné kódování textových dat, proto je tento problém řešen pouze na úrovni kódu v podobě kódovací funkce, u které lze nastavit typ kódování při přijetí a odesílání dat.

Účel simulačního programu je testování serveru, ověření časové odezvy serverové aplikace a ověření, že zařízení jsou obsluhovány paralelním způsobem. Po provedení značného počtu testů mohu s jistotou říci, že časová odezva jednotlivých zařízení i při velkém zatížení systému, je stále dobrá a není ovlivněna velkým zatížením do takové míry, že by server přestal odpovídat v rozumném časovém úseku. Testoval jsem, zda se lze dotázat na server při úplném přetížení, abych ověřil, že server pracuje paralelně a je schopen ovládat ostatní zařízení. Dále bylo změřeno, že průměrná doba odezvy při velkém zatížení systému je do 200 ms. V případě použití v praxi, při ovládání např. turniketu, je tato doba odezvy naprosto vyhovující firemním předpokladům. Nakonec jsem ještě otestoval, jaký je rozdíl mezi normálním režimem (s vizuální podporou) a režimem při vypnuté vizualizaci (viz. obrázky č. 10 a č. 11). Podle očekávání se urychlila práce serveru a doba odezvy je o pár desítek milisekund kratší.

Mezi možné diskutované vylepšení systému, by mohlo patřit šifrování komunikace anebo úplné vizuální odlehčení pro vyšší výkon. V případě šifrování je implementace SSL protokolu dostupná za pomoci Indy komponent a lze ji celkem jednoduše doplnit. V případě, že by se měl systém použít na místě, kde bude u zařízení existovat krátký proces komunikace (bude větší frekventovanost jejich připojení a odpojení), bych doporučil implementaci struktury, která bude předem vytvářet a spravovat objekty (instance třídy TClientCore pro jednotlivá klientská vlákna) a řídit jejich následné znovupoužití. Dojde tím ke snížení režie při vytváření a uvolňování objektů a tím zvýšení výkonu serveru.

Chtěl bych dodat, že jsem si vybral externí práci se záměrem vytvořit užitečný a spolehlivý systém pro provoz v praxi. Tím jsem uplatnil nastudované teoretické znalosti, např. vláknového programování a pokročilého užívání databázových procedur. V případě, že se setkáte s tímto systémem v reálném světě (velice pravděpodobné), budu doufat, že s ním budete spokojeni.

Literatura

[1] *TCP* [online]. 11. 3. 2010 [cit. 2010-04-08].
Dostupné z WWW: <http://cs.wikipedia.org/wiki/TCP>. Ověřeno z dostupných zdrojů.

[2] *Internet Protocol* [online]. 18. 3. 2010 [cit. 2010-04-08].
Dostupné z WWW: <http://cs.wikipedia.org/wiki/Internet_Protocol>.

[3] KRISTOFF, John. *The Transmission Control Protocol* [online]. 2000-04-24 [cit. 2010-04-22].
Dostupné z WWW: <http://condor.depaul.edu/~jkristof/technotes/tcp.html>.

[4] MACNAR, Tomáš. *Síťový protokol TCP/IP* [online]. 26.11.1999 [cit. 2010-04-08].
Dostupné z WWW:
<http://www.maturita.cz/referaty/informatika/tcp_ip.htm#_Toc464054837>.

[5] *TCP/IP* [online]. 2. 4. 2010 [cit. 2010-04-08].
Dostupné z WWW: <http://cs.wikipedia.org/wiki/TCP/IP>.

[6] *Databáze* [online]. 12. 2. 2010 [cit. 2010-04-09].
Dostupné z WWW: <http://cs.wikipedia.org/wiki/Datab%C3%A1ze>.

[7] *Relační databáze* [online]. 19. 3. 2010 [cit. 2010-04-09].
Dostupné z WWW:
<http://cs.wikipedia.org/wiki/Rela%C4%8Dn%C3%AD_datab%C3%A1ze>.

[8] *Teorie relačních databází: Relační model dat* [online]. 12.1. 2006 [cit. 2010-04-22]. Dostupné z WWW:
<http://www.manualy.net/article.php?articleID=9>.

[9] *Teorie relačních databází: Normalizace* [online]. 2.8.2007 [cit. 2010-04-22].
Dostupné z WWW: <http://www.manualy.net/article.php?articleID=13>.

[10] *Firebird* [online]. 2010-02-12 [cit. 2010-04-04].
Dostupné z WWW: <http://cs.wikipedia.org/wiki/Firebird>.

[11] ČINČURA, Jiří. *Poznejte Firebird za 2 minuty* [online]. 2010 [cit. 2010-04-04].
Dostupné z WWW: <http://www.firebirdnews.org/docs/fb2min_cz.html>.

[12] *Uložená procedura* [online]. 25. 2. 2010 [cit. 2010-04-09].
Dostupné z WWW:
<http://cs.wikipedia.org/wiki/Ulo%C5%BEen%C3%A9_procedury>.

[13] STĚHULE, Pavel. *PostgreSQL* [online]. 11.4.2010 [cit. 2010-04-22].
Dostupné z WWW: <http://www.pgsql.cz/index.php/PostgreSQL>.

[14] KADLEC, Václav. *Umíme to s Delphi, 52. díl: vlákna a paralelní programování: pokračování* [online]. 8.4.2002 [cit. 2010-04-22].
Dostupné z WWW: <http://www.zive.cz/clanky/umime-to-s-delphi-52-dil--vlakna-a-paralelni-programovani-pokracovani/sc-3-a-106057/default.aspx>.

[15] *Indy in Depth* [online]. [s.l.]: Atozed Software, 2005-07-25 [cit. 2010-05-03].
Dostupné z WWW: <http://www.docstoc.com/docs/8688472/Indy-In-Depth/>.

Příloha A – UML diagram tříd pro TmmTCPServer

TmmTCPServer
TComponent
FAutoHandleMsgToDB: Boolean
FBindings: TIdSocketHandles
FCountConnection: Word
FCriticalSection: TCriticalSection
FdbCParams: TDBConnectionsParams
FDBglobal: TDBGlobalCore
FIDServruPeriferie: Integer
FmmDeviceBox: TmmDeviceBox
FServer: TIdTCPServer
FTimer: TTimer
ClientCoreCreateSettings(TIdContext)
+ CreateGlobalDB()
+ SendDemandToDisconnectClients()
+ SendMsg(TIdContext, string, string)
# ServerConnect(TIdContext)
# ServerDisconnect(TIdContext)
# ServerException(TIdContext, Exception)
# ServerExecute(TIdContext)
# StartServer() : Boolean
# StopServer() : Boolean
+ TimerServerExecute(TObject)

#FmmDeviceBox
1
1

TmmDeviceBox
TObject
- FlstContexts: TList
- FlstDevice: TList
- FMutex: TMutex
+ AddDevice(Word, TIdContext) : boolean
+ AddLog(string) : Boolean
+ DeleteDevice(Word, TIdContext)

1
0..*

TmmDevice
TObject
- FContext: TIdContext
- FID: Word

1
0..*

TIdContext
+ data: TObject

1
1

#FDBglobal
1

TDBGlobalCore
+ CheckPlanUdalosti(TStringList*) : Boolean
+ Peripheral_Load(TListView*) : boolean

+DBcore
1

TDBcore
TObject
ADbConnectionsParams: tFDbConnectionsParams
ConnectDB(Boolean)
+ PlanUdalosti_Query(Integer, TObjectList*) : Boolean
+ Prechod_Query(TMsgData*) : Boolean

#ADbConnectionsParams
1
1

tFDbConnectionsParams
TPersistent
FDatabaseName: string
FDbCharSet: string
FDbPassword: string
FDbPort: Integer
FDbServerName: string
FDbUserName: string

TClientCore
TObject
+ CPcore: TCommunicationProtocol
+ DBcore: TDBcore
# FContext: TIdContext
# FCountTimer: TmmClientCoreCountTimer
# FIDperiferie: Word
# FVisuals: TmmClientCoreVisuals
+ LastPacket: TCommunicationProtocolData
+ CallPlanovaneUdalosti()
+ CallPrechod(TIdContext)
+ CallPrechod_abstractDisconnect()
+ ContextConnect(TIdContext)
+ ContextDisconnect(TIdContext)
+ ContextExecute(TIdContext)
+ ContextIdReceive()
# HandleReturnedDataFromDB(string, Word, string, string, TIdContext)
# KontrolaSpojeni()

#FCountTimer
1..*
1

TmmClientCoreCountTimer
TObject
- FCount: Integer
- FOnCounTimerExecute: TccOnCountTimerExecute
- FTimeoutInterval: Word
+ DoTimerExecute()
+ Restart()

#FVisuals
1
1

TmmClientCoreVisuals
TObject
- FFileChartStatistic: TmmFileChart
- FOwnListBox: Tlistbox
- FOwnListItem: Tlistitem

-FFileChartStatistic

TmmFileChart
TObject
+ Add()
+ SaveToFile(string)

+LastPacket
1..*
1

TCommunicationProtocolData
TObject
+ cmd: string
+ Fkonec: Byte
+ IDperiferie: Word
+ MsgData: string
+ MsgNumber: Word

+CPcore
1
1

TCommunicationProtocol
TObject
FHistoryList: TList
FpocetMsgReceived: Word
FpocetMsgSended: Word
+ Decode(TBytes, string): string
+ Decode(string, string, Boolean) : TBytes
+ Receive_Msg(TIdIOHandler, Integer) : TCommunicationProtocolData
+ Send_Msg(TIdIOHandler, string, Word, string) : TCommunicationProtocolData

Příloha B – Zdrojový kód databázových pohledů

VIEW_SYSTEM_PREHLED_OBSLUH

```
CREATE OR ALTER VIEW VIEW_SYSTEM_PREHLED_OBSLUH(
  TYP_BRAMBORY_NR,
  NAZEV_TYPU_BRAMBORY,
  STAV_POPIS,
  STAV_KOD,
  KOD,
  POPIS,
  SQL_OBSLUZNA)
AS
SELECT so.typ_brambory_nr, tb.nazev_typu_brambory, sp.stav_popis,
  so.stav_kod, so.kod, skp.popis, so.sql_obsluzna
FROM system_stavy_obluha so
  JOIN system_stavy_popis sp ON so.stav_kod = sp.stav_kod
  JOIN system_brambory_typy tb ON tb.typ_brambory_nr = so.typ_brambory_nr
  JOIN system_kod_popis skp ON so.kod = skp.kod
ORDER BY so.typ_brambory_nr, sp.stav_popis, skp.kod asc
;
```

VIEW_SYSTEM_UDALOSTI

```
CREATE OR ALTER VIEW VIEW_SYSTEM_UDALOSTI(
  ID_DEVICE)
AS
SELECT DISTINCT pu.id_device
FROM system_plan_udalosti pu
  WHERE (pu.time_execute <= CURRENT_TIMESTAMP) and (pu.time_sent is null)
ORDER BY pu.id_device, pu.time_execute ASC
;
```

Příloha C – Přiložené CD

Složka *doc* obsahuje textovou část práce ve formátech *doc* a *pdf*.

Složka *src* obsahuje obě aplikace a k nim příslušné zdrojové kódy.

Složka *app* obsahuje tři složky:

- Složku *SIMserver,* ve které soubor serverové aplikace.
- Složku *DB*, která obsahuje příslušnou databázi k testovacímu provozu serverové aplikace.
- Složku *SIM*, která obsahuje soubor pro spuštění simulačního programu periferií.

Složka *firebird* obsahuje instalační balíček pro MS Windows (32 bit).