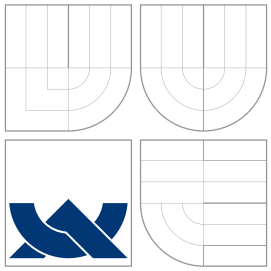
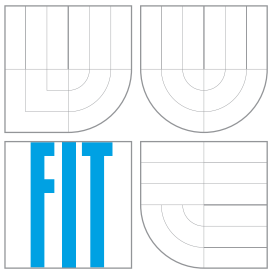


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

MONITOROVACÍ PROGRAM PRO ZÁZNAM PROVOZU NA SBĚRNICI RS485

PROGRAM FOR MONITORING RS485 BUS COMMUNICATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MAREK ZUKAL

VEDOUCÍ PRÁCE
SUPERVISOR

Dr. Ing. PETR PERINGER

BRNO 2008

Abstrakt

Cílem této práce bylo vyřešit záznam a zobrazení dat z řídící sběrnice světelného signalizačního zařízení. Pro záznam dat byl vytvořen program určený k provozu na vestavěném PC. Pro zobrazení signálního obrazu ze zanamenaných dat byla navržena a implementována přenositelná aplikace s grafickým uživatelským rozhraním.

Klíčová slova

sběrnice, RS-485, Linux, záznam dat, světelné signalizační zařízení, signální obraz

Abstract

The goal of this project was to solve recording and displaying the data from the control bus of the traffic light controller device. A program for recording the data was created for the usage on an embedded PC. A portable application with a graphic user interface was implemented to display singal image from the recorded data.

Keywords

bus, RS-485, Linux, data recording, traffic light, signal image

Citace

Marek Zukal: Monitorovací program pro záznam provozu
na sběrnici RS485, bakalářská práce, Brno, FIT VUT v Brně, 2008

Monitorovací program pro záznam provozu na sběrnici RS485

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Dr. Ing. Petra Peringera. Další informace mi poskytli zaměstnanci firem CROSS Zlín s. r. o. a PATRIOT spol. s r. o. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Marek Zukal
6. května 2008

Poděkování

Poděkování patří všem zaměstnancům firem CROSS Zlín a PATRIOT, kteří vstřícně a zasvěceně odpovídali na moje otázky a tím maximálně zpříjemnili naši spolupráci.

Obsah

Kapitola 1

Úvod

Řízení dopravy je obecně složitá problematika a její složitost neustále roste se zvyšující se hustotou provozu ve městech i mimo ně. Dnes už si lze jen těžko představit, že by nejfrekventovanější městské křižovatky byly řízeny pomocí naohýbaných plíšků na otáčejícím se válci, jako tomu bylo v minulosti. Aby bylo dosaženo co nejlepších výsledků a řízení bylo co nejefektivnější, signalizační zařízení dospěla do stavu, kdy je každá křižovatka schopna dynamicky reagovat na aktuální provoz a synchronizovat signalizaci s okolními křižovatkami tak, aby na sebe navazující signály *Volno* umožnily vozidlům plynulý průjezd několika křižovatkami (tzv. Zelená vlna) nebo přidělovat preference vozidlům hromadné dopravy v závislosti na čase jejich příjezdu vzhledem k jízdnímu řádu.

S rostoucí složitostí celého systému také rostou požadavky na kontrolu a monitorování jednotlivých komponent a komunikace mezi nimi. V současné době je sice možné sledovat stav řadiče a aktuální signální obraz pomocí připojení přes GSM, ale tímto způsobem lze komunikovat jen s některými typy řadičů a nelze pořizovat dlouhodobý záznam. Tato práce se snaží přijít s řešením právě tohoto problému, tedy vytvořit možnost zaznamenat a poté kontrolovat komunikaci jednotlivých komponent řadiče. Práce je zaměřena na současně používaný typ RS3 firmy CROSS Zlín s. r. o.

První požadavky na řešení se začaly formovat během srpna 2007, kdy bylo nasazeno prvotní řešení a já jsem byl osloven firmou PATRIOT spol. s r. o., která se podílí na vývoji řadičů, abych se vyjádřil k některým jeho nedostatkům. Vzhledem k tomu, že některá omezení byla v operačním systému MS DOS prakticky nepřekonatelná, navrhl jsem nasazení Linuxu jako operačního systému a vývoj nové aplikace, která by již podobnými nedostatky netrpěla. Tím se mi naskytla možnost zpracovat řešení v rámci bakalářské práce, což jsem jednoznačně uvítal.

Kapitola Přehled obsahuje stručný úvod do problematiky, shrnuje nastudovaný materiál, použité prostředky a nástroje a popisuje současné řešení. Oběma aplikacím, jejichž implementace byla hlavní náplní této práce pak byla věnována jedna kapitola, kde je blíže popsána jejich specifikace, návrh a implementace. Závěr poté shrnuje odvedenou práci a hodnotí její přínos.

Kapitola 2

Přehled

Tato kapitola obecně popisuje cíle práce, kterých mělo být dosaženo a definuje odbornou terminologii užívanou v oboru řízení dopravy na pozemních komunikacích. Dále shrnuje problematiku řízení dopravy, vysvětluje základy jejího technické řešení, a popisuje použité a dostupné prostředky.

2.1 Cíle práce

Zpracováno by mělo být kompletní řešení zaznamenávání a analýzy dat spojených se světelnou signalizací řízenou řadičem typu RS3. Pro vlastní záznam dat je nutné vytvořit program pro záznam komunikace komponent sběrnice řadiče, který by byl provozovatelný na vestavěném PC. V záznamu komunikace musí být obsažena všechna data přesně v takové podobě, v jaké byla přijata ze sběrnice. Data v této podobě budou sloužit pro diagnostické účely při odhalování chyb v komunikaci jednotlivých komponent a pro zpětnou rekonstrukci signálního průběhu, čehož lze využít například při vyšetřování dopravních nehod nebo ověřování toho, že řadič je schopen přizpůsobit trvání jednotlivých signálů vzhledem k aktuálnímu provozu.

Dále pro vizuální analýzu zaznamenaných dat navrhnout a implementovat aplikaci schopnou zobrazit signální obraz ze záznamů pořízených výše zmíněným programem tak, aby bylo možné provozovat ji pod operačním systémem MS Windows XP. Účelem aplikace není přesná analýza dat, ale kontrola trvání jednotlivých signálů a možnost zpětně dohledat signální průběh nebo opticky rychle vyhledat okamžik poruchy, jehož příčina pak může být vyhledána v původních odposlechnutých datech.

2.2 Odborná terminologie

Jako každý obor, i řízení dopravy na pozemních komunikacích užívá svých specifických odborných termínů. V této práci se vyskytují poměrně často a proto je zde uveden seznam použitých termínů spolu s jejich významy tak, jak jsou definovány podle platných českých norem. [10, 11]

Řízení dopravy Usměrňování dopravy světelnými signály nebo dopravními značkami.

Monitorování Metoda shromažďování informací o řadiči světelného signalizačního zařízení a o diagnostických kontrolách sloužících ke zjištění poruchového stavu.

Systémy silniční dopravní signalizace Zahrnují systémy a zařízení, za předpokladu, že jsou vzájemně propojeny elektrickými obvody a jsou na sobě závislé.

Světelné signalizační zařízení (SSZ) Řadič světelného signalizačního zařízení, spolupracující přes připojená rozhraní s řízenými návěstidly.

Návěstidlo Zařízení sestávající z jedné nebo více optických jednotek včetně komory, se všemi držáky, upevněním, slunečními clonami a kontrastními rámy, jehož úlohou je předávat vizuální informace účastníkům silničního provozu. [1]

Světelný signál Návěst pro účastníky silničního provozu.

Řadič Elektrické zařízení k řízení signálů.

Detektor/Čidlo Termíny označující prostředky detekce přítomnosti nebo průjezdu vozidla úsekem silnice nebo tratě, detektory chodců nebo tlačítka pro zadání požadavků.

Signální skupina Sled podmínek pro soubor signálních ploch, které dostávají vždy stejné povely.

Signální plán Popis časových intervalů jednotlivých signálů pro jednotlivé skupiny a definice reakcí na vstupy z detektorů, preference vozidel MHD a rychlých zásahových vozidel a synchronizaci s okolními křižovatkami.

Signální obraz Záznam trvání signálů jednotlivých signálních skupin v čase.

2.3 Úvod do problematiky

2.3.1 Světelné signalizační zařízení (SSZ)

SSZ je soustava návěstidel, detektorů a ostatních prvků pro řízení dopravy na pozemních komunikacích. Řídící jednotkou světelného signalizačního zařízení je *řadič*, který bývá zpravidla umístěn v blízkosti křižovatky. Odtud jsou vedeny kabely k jednotlivým návěstidlům, detektorům, ostatním součástem systému a okolním křižovatkám. Klíčovou komponentou řadiče je pak *procesorová deska*, která řídí provoz, podle nahraného *signálního plánu*. Signální plán popisuje délky jednotlivých signálů a reakce na vstupy z detektorů pro dané signální skupiny. Signální plán tedy umožňuje procesorové desce podle množství vozidel přijíždějcích z daného směru prodloužit trvání příslušného signálu *Volno*. Aby bylo dosaženo ještě vyšší flexibility, procesorová deska obsahuje několik signálních plánů, které jsou přepínány v závislosti na denní době nebo uvážení dispečinku. V případě nutnosti je možné řídit signály také ručně prostřednictvím panelu pro ruční ovládání.

2.3.2 Komunikace komponent

Procesorová deska komunikuje s ostatními komponentami řadiče po sběrnici typu RS-485. K této sběrnici jsou mimo jiné komponenty připojeny i spínačové desky, které pak ovládají jednotlivé žárovky návěstidel. Každá spínačová deska má přiřazenou adresu podle slotu(patice), ve kterém je zapojena a ovládá až čtyři žárovky. Procesorová deska neustále obvolává všechny spínačové a sděluje žádaný stav jednotlivých žárovek. Spínačová deska na to odpovídá chybovým stavem jednotlivých žárovek. Stav je vyhodnocován a pokud by

[1] Obecně vžitý termín pro návěstidlo užívaný neodbornou veřejností je „semafor“.

došlo například k poruše na důležité žárovce, například červené, dojde k přepnutí křižovatky do chybového stavu, blikavé žluté.

2.3.3 Komunikační rozhraní RS-485 (EIA-485)

Toto sériové komunikační rozhraní se na poli osobních počítačů nikdy nerozšířilo, ale stalo se obecným průmyslovým standardem. Na rozdíl od běžně známého RS-232 neobsahuje žádné řídící vodiče, datový spoj je diferenciální, tzv. half-duplex a umožňuje, aby spoj sloužil jako sběrnice – tedy umožňuje více různým zařízením účastnit se komunikace. Počet vodičů je redukován na dva, tedy pouze datový spoj, ke kterým se někdy přidává ještě třetí vodič, který souží jako společný zemnící potenciál a stínění kabelu. Společným oběma rozhraním bývá typ UARTu(Universal Asynchronous Receiver Transmitter), kterým je zpravidla čip 16550 nebo kompatibilní [9]. To umožňuje celkem bez problémů vytvářet převodníky mezi oběma rozhraními a podobný přístup k ovládání z řídícího programu nebo operačního systému.

Kvůli výše zmíněnému half-duplexu je nutné před vysláním dat přepnout rozhraní do vysílacího režimu. Protože rozhraní nemá řídící vodiče, je nutné koordinovat všechna zařízení tak, aby vysílalo vždy pouze jedno zařízení. Ve většině případů je jeden prvek označován jako *master* a všechny ostatní prvky *slave*. *Master* pak adresuje ostatní zařízení bytem s jedničkovou paritou, vyšle příkaz nebo odchozí data a po předem určenou dobu čeká na odpověď. U jiných než adresových bytů je parita nulová. Tento typ parity se nazývá *tuhá parita*, anglicky *stick parity*. Jedničková parita pak *mark parity* a nulová *space parity*. Pro kontrolu správnosti dat pak slouží kontrolní byte, jehož spolehlivost bývá vyšší než parity jako takové.

2.3.4 Stávající řešení

- Původní program pro účely záznamu komunikace byl napsán pro operační systém MS DOS. Provozován byl na vestavěném PC typu mitePC-L s 16 MB interní flash paměti, 16 MB RAM a CompactFlash kartou pro ukládání záznamů. Data byla ukládána v prostém textu. Program byl velice jednoduchý a omezený možnostmi operačního systému, na kterém byl provozován.
- Pro zobrazení záznamů neexistovala žádná aplikace.

2.4 Studium dostupných prostředků

2.4.1 Vestavěné PC

V průběhu realizace byl nalezen výkonnější typ vestavěného PC za nižší cenu a proto bylo rozhodnuto, že se v budoucnu bude využívat právě tento typ. Konkrétně jde o model ET-205-128 [8]. Jde o PC s procesorem Vortex86 pracujícím na frekvenci 166MHz s instrukční sadou kompatibilní s Pentiem. Dále je vybaven 128 MB RAM, VGA konektorem, dvěma sériovými porty (jedním přepínatelným na RS-485), síťovým rozhraním, LPT, USB a PS/2 konektorem. Jako úložné zařízení je použita CompactFlash karta.

2.4.2 GNU/Linux

GNU/Linux je pro tento typ zařízení jistě nejvhodnějším kandidátem na operační systém. Jako open-source software s podporou multitaskingu, širokými možnostmi použití, ma-

ximální přizpůsobivostí, relativně nízkými HW nároky a nulovými licenčními náklady proniká do stále více vestavěných zařízení. Protože společnost vyrábějící použité PC dodává i linuxovou distribuci postavenou na míru, nebyl důvod poohlížet se po jiném řešení.

Konkrétně jde o distribuci *X-Linux* dodávanou společností DM&P. [6] Základním prvkem je *BusyBox*, který zastává funkci shellu, řádkových utilit, telnet serveru a DHCP klienta. Dalšími aplikacemi jsou *WN Server*(HTTP server) a *vsftpd*.

Systém je distribuován jako obraz diskového oddílu ve formátu programu *Norton Ghost*, „disk dump“ pořízený aplikací *dd* nebo zkomprimovaný obsah kořenového svazku společně s několika jádry zkompilovanými na míru jednotlivým typům procesorů. Takto již připravená jádra jsou řady 2.4. Pro potřebu kompilace jiného jádra nebo s jinými parametry je dodáváno i výchozí nastavení kompilace, soubor *.config*.

Výsledný systém nepřesahuje velikostí 8 MB a na vestavěném PC s procesorem *Vortex86* je schopný nabootovat v čase kratším než deset vteřin. Veškerá inicializace je umístěna v jediném startovacím skriptu umístěném ve standardním adresáři `/etc/init.d/`. Odtud jsou připojeny diskové oddíly a aktivováno síťové rozhraní. Adresář `/tmp` pro dočasné soubory je přesměrován na oddíl vytvořený v operační paměti, aby nedocházelo ke zbytečnému opotřebení úložného zařízení typu Flash, na které je možné zapsat „jen“ několikset tisíckrát.

2.4.3 Sériová komunikace

Základní princip a pojmy sériové komunikace

Podstatou sériové komunikace je přenos pouze jednoho bitu v daném časovém okamžiku. Díky tomu odpadají požadavky na synchronizační vodič a komunikaci je tak možné zrychlit, nehledě na nižší náklady a prostorové požadavky spoje. Na druhou stranu je nutné zajistit synchronizaci přijímače a vysílače. Toho je dosaženo rozdělením komunikace na rámce, které začínají startbitem, obsahují 7 nebo 8 datových bitů, 1 paritní bit a 1 nebo 2 stopbity. Startbit je vždy hodnoty 0 a stopbit 1. To zaručí alespoň jednu změnu úrovně signálu na každý rámec a touto hranou je pak synchronizován přijímač.

Základní rychlost komunikace, stejně jako její parametry je možné nastavit pomocí příslušných bitů v příznakovém registru UARTu. Mezi tyto parametry patří například počet stopbitů, typ parity, počet datových bitů atd.

V každém operačním systému se k HW prostředkům přistupuje jinak a sériová komunikace není výjimkou. Přestože princip zůstává stejný, je třeba použít platformě závislé knihovny a volání operačního systému. V Linuxu, který je implementačním prostředím programu, se k sériovým portům přistupuje velice podobně jako k souborům. Na rozdíl od obyčejných souborů je však možné pomocí deskriptoru portu nastavovat parametry UARTu.

Přístup k portům v Linuxu

Ovládání portů je rozděleno do tří úrovní.

První úroveň umožňuje pohled na port jako na obyčejný soubor ze kterého je možné číst, zapisovat. Deskriptor portu je získán voláním funkce `open()` na soubor představující sériový port (typicky `/dev/ttyS[0-n]`). Příchozí data lze číst funkcí `read()` a zapisovat funkcí `write()` jako by šlo o obyčejný soubor. Čtení se pak chová podle příznaků (flagů) nastavených pomocí deskriptoru. Z pohledu aplikace a její logiky je nejdůležitější rozhodnout, zda používat blokující nebo neblokující čtení.

Při neblokujícím čtení funkce `read()` zjistí, zda jsou k dispozici příchozí data, která by mohla přečíst. V případě, že taková data nenalezne, okamžitě vrací hodnotu `-1` a nastavuje globální příznak chyby *errno* na hodnotu *EAGAIN*. Na rozdíl od tohoto přístupu při blokujícím čtení vrací funkce `read()` řízení programu až po té, co jsou nějaká data k dispozici. Při přerušení blokování signálem je proměnná *errno* nastavena na hodnotu `EINTR`.

Implicitně je deskriptor v režimu blokujícího čtení. Nastavení příznaku je možné pomocí funkce `fcntl()`, jejímiž argumenty jsou deskriptor portu, konstanta `F_SETFL` označující příkaz nastavení příznaků a konstanta `O_NONBLOCK` označující vlastní nastavovaný příznak.

Druhá úroveň je již zaměřena striktně na sériovou komunikaci a umožňuje nastavení parametrů komunikace. Veškerá nastavení tohoto typu je možné provádět přes strukturu `termios` definovanou ve stejném hlavičkovém souboru. Ta obsahuje čtyři registry příznaků:

`c_cflag` registr režimu řízení nastavující počet datových bitů, typ parity, počet stopbitů atd.

`c_lflag` registr lokálního režimu, který je zaměřen na komunikaci sériových terminálů. Pomocí hodnot je možné zapnout např. odesílání „ozvěny“ (echo) některých nebo všech přijatých znaků, interpretaci netisknutelných znaku apod.

`c_iflag` registr režimu vstupu – nastavení kontroly parity, označování bytů s chybnou paritou, mapování znaku `CR` na `NL` nebo nulování nejvyššího bitu.

`c_oflag` registr režimu výstupu – nastavení mapování znaků konců řádků, výplňových znaků nebo zpoždění po znaku `TAB`.

Popsané registry sice umožňují i nastavení rychlosti komunikace, ale pro vlastní nastavení je doporučováno použít samostatných funkcí `cfsetispeed` a `cfsetospeed` namísto přímé úpravy hodnot registrů.

Kromě výše zmíněných příznakových registrů struktura obsahuje pole `cc_t`, které obsahuje popis znaků se speciálním významem (typicky escape sekvence pro kontrolu toku dat). Ty jsou ale v platnosti pouze pokud je nastavena jejich interpretace příznakem `IEXTEN` v registru `lflag`. Výjimkou jsou pouze hodnoty `VMIN` a `VTIME`, které se vztahují k nastavení doby blokování blokujícím čtením. `VMIN` udává minimální počet znaků, který musí být dostupný, aby mohlo dojít k úspěšnému vrácení dat, `VTIME` pak maximální čas v desetinách vteřiny, po který může funkce `read()` blokovat provádění programu před návratem řízení programu.

Třetí úroveň je nejnižší systémovou úrovní, ze které lze port ovládat. Nastavení je opět svázáno s deskriptorem otevřeného portu, tentokrát je ale prováděno funkcí `ioctl()`. Pomocí této funkce lze přistupovat přímo k úrovním jednotlivých signálů UARTu a jiným systémovým nízkoúrovňovým prostředkům. Pro práci na této úrovni je třeba vkládat hlavičkový soubor `sys/ioctl.h`. Jako příklad použití funkce `ioctl()` poslouží následující úsek kódu, který demonstruje způsob ovládání úrovně signálu *RTS* (Ready To Send). V tomto případě jde o vynulování tohoto příznaku.

```
...
int pflags;//příznaky portu
int port;// file descriptor portu
port=open("/dev/ttyS0",O_RDWR);//otevření portu
```

```
if(ioctl(port, TIOCMGET, &pflags)==-1)//získání současných příznaků
{
  //chyba, nepovedlo se získat příznaky portu
}
else
{
  pflags &= ~TIOCM_RTS;//změna požadovaného příznaku
  if(ioctl(port, TIOCMSET, &pflags)==-1)//nastavení nových příznaků
  {
    //chyba, nepovedlo se nastavit nové příznaky
  }
}
...
```

2.4.4 Komprese dat

Vzhledem k množství příchozích dat a jejich povaze se přímo nabízí možnost komprese před zápisem na úložné zařízení. Komprimačních knihoven a formátů je veliké množství, ze kterého byla vybrána knihovna *zlib*[5], pracující s formátem *gzip* běžně používajícím příponu *gz*. Tento formát je velice jednoduchý a neumožňuje kompresi více souborů v jednom archivu. Proto se v běžné praxi používá s v kombinaci programem *tar*. V tomto případě ale není nutné komprimovat více souborů a naopak je výhodou, že je každý soubor zkomprimován samostatně. Samotná knihovna má velice jednoduché aplikační rozhraní poskytující požadovanou funkcionalitu.

Funkce pro práci se soubory jsou velice podobné standardním funkcím. Na rozdíl od nich ale pracují nad zkomprimovaným souborem. Komprimovaný soubor je možné otevřít funkcí `gzopen()`, číst a zapisovat funkcemi `gzread()` a `gzwrite()` a uzavření souboru je možné pomocí funkce `gzclose()`. Při ukončení programu bez volání funkce `gzclose()` není soubor správně ukončen a výsledný soubor není čitelný. Další funkcionalita je popsána v manuálu na stránkách projektu. [5]

2.4.5 Knihovna grafického rozhraní

Pro tvorbu oken aplikace byla vybrána knihovna *GTKmm* [2], což je objektová nadstavba nad GTK [1]. Tato nadstavba poskytuje drtivou většinu funkčnosti GTK. Pro případy, kdy by bylo třeba využít funkčnosti, kterou neposkytuje, umožňuje pomocí metody přistoupit k zapouzdřené struktuře i funkcemi GTK. Při běžné práci je však pravděpodobnost takovéto potřeby minimální.

Aplikační rozhraní je zpracováno velice kvalitně a čistě, což velice zpříjemňuje práci. Pro další zjednodušení je možné využít programu Glade, který umožňuje WYSIWYG tvorbu oken a dialogů. Popis oken je po ukončení práce uložen do *xml* souboru, který je možné načíst za běhu aplikace pomocí knihovny glade a jednoduše instancovat jednotlivá okna. Tento přístup umožňuje měnit rozložení uživatelského rozhraní bez nutnosti opětovného překladu aplikace a je platformově nezávislý.

2.4.6 MinGW

Aby bylo možné zkompilovat aplikaci pro zobrazování záznamů pod Windows, je nutné nainstalovat aplikaci *MinGW* [3], která poskytuje funkcionalitu systémů kompatibilních s normou POSIX. To je nezbytné při použití systémů *Automake*, *Autoconf* a dalších.

2.4.7 Návrhové vzory

Návrhové vzory (Design patterns) jsou obecná, znovupoužitelná a ověřená řešení běžně se vyskytujících problémů v objektově orientovaných jazycích. Nejdná se o konkrétní znovupoužitelné komponenty nebo algoritmy, ale o popis logických vztahů jednotlivých tříd a instancí objektů mezi sebou navzájem v konkrétní problematice řešené daným vzorem. Snaží se tak vytvořit jednotný pohled na často řešené problémy a vytvořit tak obecně srozumitelné pojmy, které pojmenovávají jednotlivá řešení. Při znalosti těchto pojmů je pak mnohem snažší dané řešení zdokumentovat, vysvětlit nebo vůbec aplikovat. Pokud je například v dokumentaci programu uvedeno, že některá komponenta vychází z určitého návrhového vzoru, čtenář se základní znalostí návrhových vzorů mnohem snadněji pochopí činnost a smysl konkrétního řešení.

Návrhové vzory je možné rozdělit do 3 skupin. *Vytvářející vzory(Creational Patterns)* se zabývají tvorbou objektů nebo skupin objektů, kdy je dbáno na výběr správného typu objektu (většinou z hierarchie tříd se společným předkem) a jeho konstrukci a inicializaci v závislosti na stavu programu nebo logickém kontextu. *Strukturální vzory (Structural Patterns)* řeší skládání jednotlivých tříd do celků, které umožní patřičně rozšířit jejich chování nebo provádět hromadné operace nad celou strukturou jako jedním celkem bez nutnosti explicitně provádět operaci nad jejím každým prvkem zvlášť. Poslední skupinu tvoří *Behaviorální vzory (Behavioral Patterns)* zaměřující se na chování objektů a jeho změny v závislosti na pozici v hierarchii tříd. Jiné vzory z této kategorie řeší kooperaci objektů ve skupině při řešení problémů nebo zprostředkování a zapouzdření funkcionality uvnitř objektu. Další informace o návrhových vzorech a použití lze nalézt v literatuře [7].

V souvislosti s návrhovými vzory je dobré zmínit také *architektonické návrhové vzory (Architectural patterns)*, které sledují obdobné cíle jako návrhové vzory, ale narozdíl od nich představují o jeden stupěň vyšší abstrakci a zaměřují se na aplikaci jako celek. Často se při tom odkazují na návrhové vzory, které popisují jejich vnitřní organizaci.

V následujícím textu jsou blíže popsány pouze ty návrhové vzory, které byly použity při implementaci aplikace pro zobraznení záznamů.

Model – View – Controller Architektonický návrhový vzor, který dělí aplikaci na tři ucelené komponenty. Komponenta *Model* obsahuje data představující model systému. Tato data jsou čtena komponentou *View*, která je zobrazuje uživateli v pro něj srozumitelné podobě. Zároveň uživateli poskytuje rozhraní pro vyžádání akcí, které se mají s modelem provést. Tyto akce však sama neprovádí, ale předává komponentě *Controller*, která se stará o řízení běhu aplikace a zpracovávání uživatelských požadavků. Po provedení žádané akce nad modelem je již komponentou *View* zobrazena nová podoba dat.

Tento přístup umožňuje izolovat jednotlivé části aplikace a vytvořit stabilní rozhraní, za nímž je možné měnit implementaci aniž by byly ovlivněny zbývající části aplikace. V některých případech je také používáno více nezávislých pohledů na stejná data, čehož je pomocí tohoto přístupu snadné docílit.

Factory Tento vytvářející návrhový vzor se využívá ke konstrukci objektů, jejichž typ nebo hodnota je závislá nakontextu nebo předaných parametrech. Výhodou tohoto přístupu je předejítí duplikace kódu a možnost využít při konstrukci objektu informace, ke kterým by samotný konstruovaný objekt neměl přístup. Při použití dědičnosti pak volající nemusí ani určit jaký typ objektu se má zkonstruovat, což umožňuje držet podstatnou nezávislost mezi tvorbou objektu a jeho následným použitím.

Kapitola 3

Program pro záznam komunikace komponent řadiče

Jak vyplývá z cílů práce, tento program by měl být schopen zaznamenat všechna data komunikace mezi procesorovou deskou a spínačovovými deskami na řídící sběrnici řadiče světelného signalizačního zařízení. Pro účely provozu tohoto programu je k dispozici vestavěné PC vybavené příslušným komunikačním rozhraním. V následujícím textu jsou blíže popsány požadavky na program, návrh, implementace, testování a nasazení na vestavěné PC s operačním systémem GNU/Linux.

3.1 Specifikace

- Data přicházejí na sériový port připojený ke sběrnici RS-485 rychlostí 57 600bps.
- Adresové byty mají paritu nastavenou na 1, datové na 0.
- Data se ukládají do souborů na úložné zařízení v hexadecimálním formátu.
- Soubory jsou vytvářeny v nekonečné smyčce – při zaplnění kapacity, je smazán nejstarší soubor.
- Po přijetí adresy, tedy bytu s paritou 1, je třeba přejít na nový řádek.
- Na začátku souboru je nutné vypsat aktuální čas.
- Každý řádek musí být uveden minutou, vteřinou a milisekundou ve které byl datový paket přijat.
- Aplikace musí umět reagovat na čas řadiče, který je zasílán sa sběrnici a synchronizovat systémové hodiny.
- Data je nutné před zápisem do souboru komprimovat.
- Při přerušení toku dat je nutné uzavřít aktuální soubor aby s násilným odpojením nebo výpadkem elektřiny nebyla poškozena data bezprostředně předcházející chybě, která způsobila toto „zaseknutí sběrnice“.

3.2 Návrh

Vzhledem k přímosti aplikace a její nízkoúrovňové orientaci nebyl nalezen žádný důvod pro objektový návrh. Navíc bylo vzato do úvahy, že objektové dynamické knihovny často nebývají v operačních systémech určených do vestavěných počítačů vůbec přítomny. Byl tedy využit strukturovaný návrh, jehož části byly rozděleny do těchto modulů:

Obsluha portu Modul sdružující funkce, které nastavují parametry sériové komunikace a čtou příchozí data.

Obsluha souborů Funkce starající se o komprimovaný zápis do souboru.

FIFO adresář Funkce zajišťující nahrazování starých souborů novými.

Správa času Modul pro práci s časem, načítání času ze sběrnice a synchronizaci HW hodin PC.

Modul CRC Funkce pro ověření kontrolních součtů paketů.

Zpracování argumentů Modul pro zpracování argumentů z příkazového řádku.

Hlavní modul Obsahuje funkci `main()` ze které volá funkce ostatních modulů.

3.3 Implementace

I přesto, že celý program je napsán v jazyce C, v dané podobě je přeložitelný pouze na Linuxu nebo jiných Unixových systémech. To je dáno využitím některých pro Linux specifických funkcí, které například na platformě MS Windows nenajdeme. Použití v jiném Unixovém operačním systému než Linuxu nebylo testováno ani zamýšleno. Nicméně takové přenesení programu by nemuselo vyžadovat velké změny. S největší pravděpodobností by bylo třeba přepsat pouze modul přistupující k sériovému portu.

3.3.1 Obsluha portu

Jak již bylo řečeno, k sériovým portům (stejně jako k veškerému jinému HW) se v Linuxu přistupuje podobně jako k souboru. Po jeho otevření je získána struktura termios, kde je nutné upravit příznaky pro uvedení komunikace do požadovaného režimu. Pro účely odposlechu je nutné vypnout všechna systémová zpracování, která jsou většinou spojena s komunikací sériových terminálů, aby byla načítána stejná data, která dorazila po sběrnici na port. Dále je třeba nastavit komunikaci s osmi datovými bity, jedním stopbitem a tuhou nulovou paritou (space parity). „Chyba parity“ pak bude označovat byty s adresou zařízení. V případě tuhé parity je situace trochu ošemetná. Tuhá parita, která je podporována všemi UARTy řady 16550, není popsána ve standartu POSIX a proto není ani řádně dokumentována v referenčních příručkách o sériovém programování v Linuxu. Navzdory tomu v hlavičkových souborech existuje příznak `CMSPAR`, který je určen právě pro přepnutí do režimu tuhé parity. Pak je možné pomocí kombinace `(PARENB | CMSPAR)` nastavit *space parity* nebo pomocí `(PARENB | CMSPAR | PARODD)` *mark parity*. Tyto příznaky se pochopitelně vztahují k registru `c_cflag` struktury `termios`. Pokud je zapnutá kontrola parity, což je v tomto případě nutné, byte s chybnou paritou je označen tak, že jeho hodnotu přechází dva byty s hodnotu `0xFF` a `0x00`. Pokud je načten pouze znak `0xFF` je zdvojen, aby nemohlo dojít k záměně.

Dalším důležitým krokem je přepnutí portu do příjmacího režimu. Pokud by se tak nestalo, nemohl by program číst data z portu a navíc by sběrnici zablokoval, takže by znemožnil komunikaci i všem ostatním připojeným zařízením. Toho je dosaženo pomocí funkce `ioctl()` s příslušnými argumenty, která vynuluje registr signálu RTS(Ready To Send) a nastavit registr signálu DTR(Data Terminal Ready), které odpovídají řídícím vodičům rozhraní RS-232.

3.3.2 Obsluha souborů

Tento modul popisuje strukturu `fileControl`. Ta obsahuje deskriptor právě otevřeného souboru, způsob zápisu, čas otevření souboru atd. Dále modul implementuje funkce vytvoření jména souboru z aktuálního času ve formátu `rrmmddHHMM.ext` (významy zkratek v pořadí: rok, měsíc, den, hodina, minuta, přípona), otevření nového souboru, zápisu do souboru a jeho zavření. Soubory jsou implicitně otevírány v režimu *append*, tedy přidávání nového obshahu na konec souboru. Tím je předcházeno situaci, kdy by vytvoření dvou souborů v krátkém časovém okamžiku (v rámci jedné minuty – dáno rozlišením jména souboru) způsobilo přepsání staršího souboru tím novějším. Otevření souborů je pak prováděno funkcí `fopen()` respektive `gzopen()` v závislosti na zvoleném formátu výstupu.

Zápis do souboru může probíhat v několika režimech, které je možno nastavit při módu otevření souboru. Funkce `PrintFormatted()` pak do souboru vypisuje data různým způsobem v závislosti na módu. Tato funkce je implementována pomocí funkce `vsnprintf()`, která vypíše data do staticky alokovaného bufferu, který je po té zapsán do výstupního souboru. Soubory lze zapisovat úplně bez komprese v prostém textu nebo komprimovaně pomocí knihovny zlib. To vše ale závisí pouze na nastaveném módu a je plně transparentní.

3.3.3 FIFO adresář

Zde jsou umístěny funkce pro práci s adresářem. Označní *FIFO* pochází z anglického zkratkového slova, které vzniklo ze slov *First In, First Out*. Tedy jde o typ fronty, kdy po zaplnění kapacity „opouští systém“ nejstarší soubor.

Otevřený adresář je popsán strukturou `fifodir`. Pomocí této struktury lze zjistit maximální délku jména souboru, volné místo v cílovém umístění nebo název nejstaršího souboru. Stáří souboru se zde vyhodnocuje pouze podle jeho jména. Předpokládá se, že v cílovém adresáři se budou vyskytovat pouze soubory se jmény odpovídajícími datu jejich vytvoření.

Struktura je naplněna při otevření a požadovaného adresáře pomocí funcke statvfs() a cyklu, který projde všechny soubory obsažené v adresáři, porovná jejich jména a ukazatel na řetězec se jménem nejstaršího uloží do příslušné proměnné struktury.

3.3.4 Správa času

Modul zabezpečuje načítání času z přijatých dat, synchronizaci HW hodin a získání formátovaného aktuálního času pomocí funkce `strftime()`. Pokud se na sběrnici objeví paket s aktuálním časem, je zkontrolováno, zda je rozdíl vteřin alespoň dvě a pokud ano, je podle něj sesynchronizován HW čas. Tento přístup byl zvolen, protože hodiny v řadiči běží pouze ve vteřinovém rozlišení a interval mezi časovými pakety není přesně definován. Steně tak není známo ve kterém časovém okamžiku v rámci jedné vteřiny bude časový paket přijmut. Synchronizace s každým časovým paketem by pak zapříčinila časté časové „skoky“.

3.3.5 Modul CRC

Funkce z tohoto modulu jsou využity ke kontrole správnosti CRC jednotlivých datových paketů. CRC je počítáno postupnou operací `XOR` průběžného výsledku s hodnotou z tabulky na indexu aktuálního bytu.

3.3.6 Zpracování argumentů

V tomto modulu je definována funkce pro zpracování argumentů z příkazového řádku a jsou zde definovány všechny výchozí hodnoty, které jsou použity, pokud dané vlastnosti nejsou explicitně zadány při spuštění aplikace. Funkce nejprve naplní strukturu argumentů výchozími hodnotami a po té prochází řetězec argumentů, kde hledá známé přepínače a načítá za nimi následující hodnoty, které zapisuje do předinicializované strukutury.

3.3.7 Hlavní modul

Tento modul zabezpečuje chod celého programu, k čemuž využívá všech výše popsaných modulů. Nejprve jsou zpracovány argumenty, dále je nastaven systémový časovač, který zabezpečuje periodické vypisovnání stavových informací a nakonec jsou nastaveny handlery systémových signálů a vlastní otevření a nastavení portu. Systémové signály `SIGINT` a `SIGTERM` jsou odchytávány, aby běžnou cestou nemohlo dojít k násilnému ukončení programu, při čemž by došlo k neplatnému ukončení současně otevřeného souboru a jeho obsah by poté nebyl čitelný. Dalším odchytávaným signálem je `SIGALRM`, který je doručován po každém vypršení časovače. Tím je zaručeno periodické vypisování aktuálního stavu programu. Signály jsou odchytávány pomocí struktury `sigaction` definované v souboru `signal.h` a obsahující masku blokovaných a handlery příchozích signálů. Kvůli použití tohoto platformově závislého rozšíření je nutné program kompilovat s parametrem překladače gcc `std=gnu99`.

Dále modul zabezpečuje zpracování příchozích dat, která zapisuje do bufferu dokud není přijmut byte s „chybnou“ paritou, nebo není překročena velikost bufferu. Pokud se tak stane, je zkontrolován obsah, zda odpovídá paketu, který je nutné dále zpracovat a po případném zpracování je zapsán v příslušném formátu do výstupního souboru. Stane-li se, že se funkce `read()` vrátí aniž by načetla nějaká data, je zaznamenán čas a nastaven příznak selhání. Při každém dalším selhání čtení je zkontrolován čas a poté, co doba opakovaného selhávání přesáhne stanovenou dobu, je uzavřen výstupní soubor a přiznak selhání je nastaven na hodnotu, která tento stav reflektuje. Při obnovení toku dat je otevřen nový soubor, jehož název bude vytvořen podle aktuálního času.

3.4 Vestavěné PC

Vestavěná PC jsou speciální kategorií „osobních“ počítačů. Jejich výkon, rozměry, spotřeba, odolnost a další parametry se mění v závislosti na prostředí, ve kterém mají být nasazeny. Oba dále zmíněné modely spadají do kategorie průmyslových počítačů, které se většinou vyznačují zvýšenou mechanickou odolností montáží v odolném šasi a minimem mechanických součástí. Například pevné disky jsou nahrazeny elektronickou pamětí typu Flash nebo nejsou přítomny vůbec. Pracovní teploty se většinou pohybuj v rozmezí -20 – 60 °C.

3.4.1 Přístup k datům

Po tom co jsou data zaznamenána by také měla být pohodlně dostupná. Na původním vestavěném PC od firmy MITE bylo možné jednoduše vyjmout CompactFlash kartu a na ní zobrazená data zkopírovat. Model ET-205-128 však tuto možnost nenabízí a proto bylo nutné nalézt jiné řešení. Vzhledem k tomu, že oba použité modely jsou vybaveny ethernet rozhraním, byl zvolen přístup přes FTP. Zařízení má pak staticky přidělenou IP adresu, na které běží FTP server nabízející všechna zaznamenaná data. To narozdíl od předchozího přístupu umožňuje stažení jen některých dat a to bez nutnosti PC před vlastním stahování odpojit od řadiče nebo vypnout.

Pro složitější práci s daty nebo servisní úkony je možné se k zařízení připojit pomocí telnetu.

3.4.2 Linux

K distribuce X-Linux je dodávána krátká dokumentace shrnující obecné informace ohledně instalace a konfigurace Linuxu obecně. Jako zavaděč je doporučen *Syslinux* nebo *LILO*, nicméně vzhledem k předchozím kladným zkušenostem a neznalosti zbylých dvou byl použit *GRUB*.

Dále bylo třeba ve startovacím skriptu zařídit automatické spuštění vlastního programu. Ten není spouštěn přímo, ale přes startovací skript, který teprve spouští binární soubor s příslušnými argumenty.

3.5 Testování

Testování probíhalo dvěma způsoby podle toho, v jaké fázi byl zrovna vývoj programu. Nejprve byl program testován pomocí samostatné procesorové desky, která neustále obvolávala neexistující spínačové desky. Na tom byl odladěn přístup k portům a ověřeno, že program je vůbec schopen zaznamenat všechna příchozí data. V druhé fázi bylo PC připojeno na funkční řadič zapojený ve firemní dílně, který poskytoval data totožná s daty, která by se vyskytovala v reálném provozu. Během této fáze bylo objeveno a opraveno mnoho chyb, které se týkaly synchronizace systémového času podle dat načtených ze sběrnice, cyklického ukládání souborů a zpracování argumentů. Dále bylo testováno, zda při výpadku proudu nedojde při obnovení jeho dodávky ke konfliktu monitorujícího PC s procesorovou deskou na sběrnici. Žádné problémy tohoto druhu však nebyly zaznamenány.

Ppřesto že příchozí datový tok je velice objemný, paměťová náročnost aplikace je menší než 1 MB. Deset minut zaznamenané komunikace představuje asi 500 000 datových paketů, které v nekomprimované podobě představují cca. 15 MB dat. Při použití komprese je velikost zmenšena 10 až 20 krát.

Další informace o implementaci, překladu a použití aplikace je možné nalézt v příručce k programu, která je k této práci přiložena.

Kapitola 4

Aplikace pro zobrazení záznamů

Podobně jako tomu bylo v předchozí kapitole, i zde jsou popsány jednotlivé fáze vývoje aplikace. Tentokrát však týkající se aplikace pro zobrazení záznamů. Tyto záznamy, pořízené výše popsaným programem obsahují všechna data, která se objevila na sběrnici. Největší část těchto dat je právě komunikace spínačových desek, což je z hlediska zobrazovacího programu to podstatné. Na základě příkazů spínačovím deskám je pak vykreslován pásový diagram odpovídající signálnímu průběhu v daném časovém intervalu.

4.1 Specifikace

4.1.1 Základní požadavky

- Vzhled a ovládání aplikace by mělo být podobné již používané aplikaci pro vzdálené monitorování řadiče.
- Aplikace musí být schopná načíst záznamy pořízené odposlouchávacím programem a zobrazit signály ve vteřinové mřížce.
- Definice signálních skupin jsou načítány ze souboru radic.xxx.
- Aplikaci musí být možné provozovat pod operačním systémem MS Windows XP.

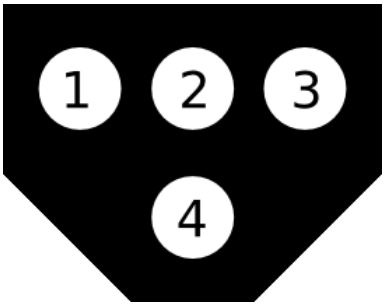
4.1.2 Význam odposlechnutých dat

Jak už bylo zmíněno dříve v textu, každá spínačová deska ovládá až čtyři žárovky, jejichž stav je daný povelem procesorové desky odeslaným na sběrnici. Na tento povel spínačová deska odpovídá chybovým kódem, který se ovšem nezpracovává. (Jeho význam převyšuje požadavky na implementovanou aplikaci.) Naproti tomu signální skupina obsahuje pět žárovek s náseldujícími významy: Červená 1, Žlutá, Zelená, Červená 2 a Červená 3. To, které žárovky musí mít definovány své stavy, je dáno typem signální skupiny. Například skupna *Chodec* využívá pouze první červenou a zelenou žárovku jak jsme zvyklí z chodeckých návěstidel. Naopak signální skupiny tramvají používají všech tří červených žárovek pro reprezentaci žárovek 1-3 návěstidla a zelenou pro žárovku 4. Pořadí žárovek tramvajového návěstidla je znázorněno na obrázku 4.1.

4.1.3 Typy signálních skupin

Popis typů signálních skupin z pohledu definičního souboru řadiče a jim příslušících návěstidel podle platné vyhlášky [4].

1. Vozidlo – **S1**, **S2**, **S3**
2. Chodec – **S9**
3. Cyklista – **S10**, **S11**
4. Doplňková šipka – **S5**
5. Vyklizovací šipka (Signál pro opuštění křižovatky) – **S6**
6. Blikač blikající při provozu křižovatky – **S4**, **S6**
7. Blikač signální skupiny – bliká v intervalech definovaných signálním plánem – **S4**, **S6**
8. Trvalý blikač – bliká i pokud je křižovatka v pohotovostním nebo chybovém režimu (blikavá žlutá) – **S4**, **S6**
9. Tramvaj sm. přímo – **S15**
10. Tramvaj sm. vlevo – **S15**
11. Tramvaj sm. vpravo – **S15**
12. Odezva
13. Opakovač tramvajového signálu
14. Opakovač blikače
15. Signál dvou vedle sebe umístěných střídavě přerušovaných červených světel – **S13**



Obrázek 4.1: Schéma tramvajového návěstidla - **S15**

4.2 Návrh

Vzhledem k povaze a určení aplikace se přímo nabízí využití objektového návrhu, který nejlépe vyjadřuje samostatnost jednotlivých částí. Při návrhu bylo zváženo využití návrhových vzorů, z nichž přímo vychází některé části aplikace. Všechny použité prostředky byly voleny s ohledem na bezproblémovou přenositelnost mezi operačními systémy GNU/Linux a MS Windows.

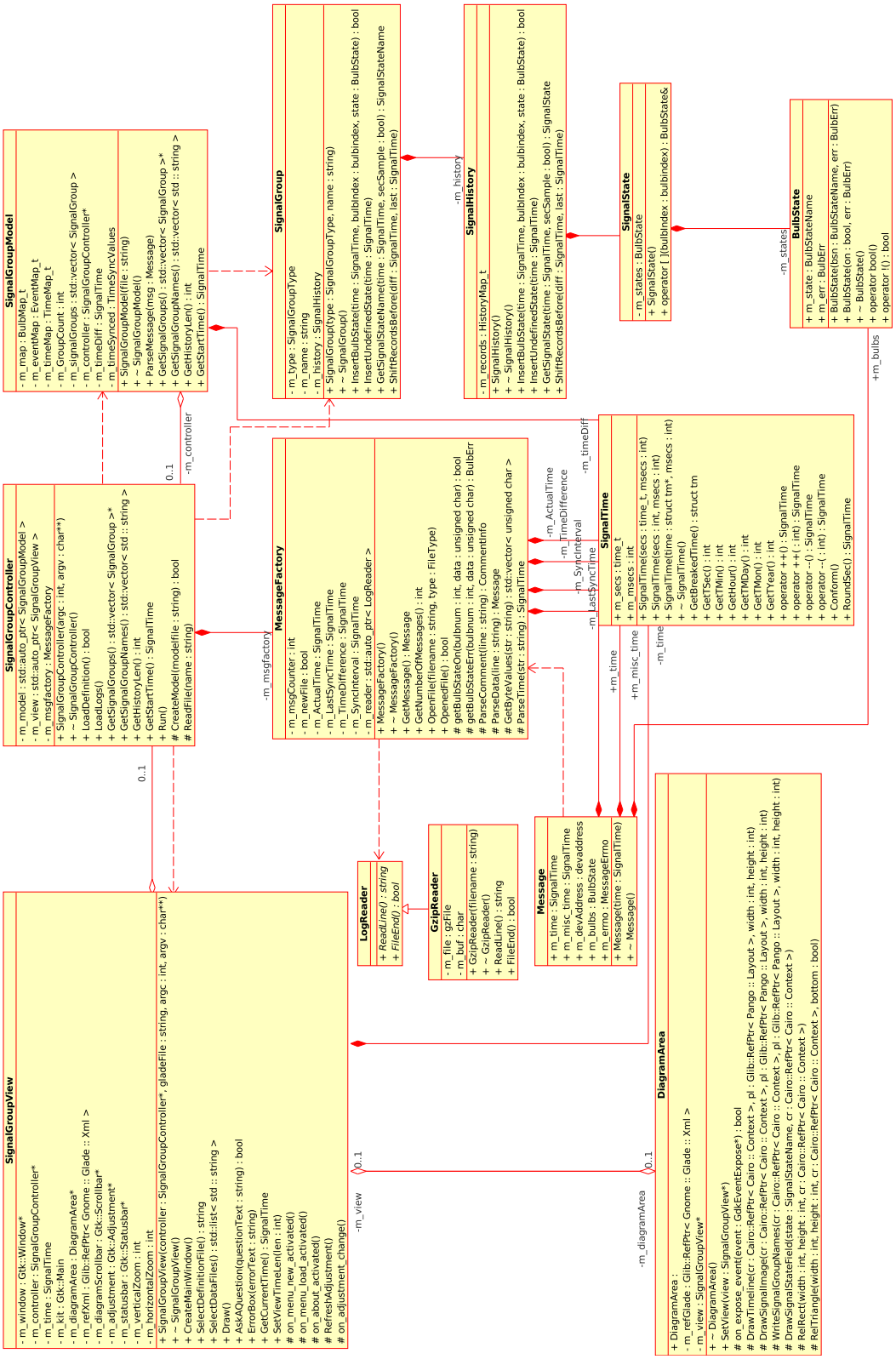
4.2.1 Objektový návrh

Základní struktura je odvozena od návrhového vzoru *Model-View-Controller*. To dělí aplikaci na tři celky, kdy běh a řízení aplikace je obsluhován komponentou *Controller*, data reprezentující definice signálních skupin a průběh signálního obrazu jsou uložena v datových strukturách komponenty *Model* a veškerá funkcionalita související se zobrazením dat a obsluhou oken je implementována v komponentě *View*. Pokud tedy uživatel požaduje změnu datového modelu, tedy chce načíst definiční soubor řadiče nebo soubor záznamu komunikace, je tento požadavek zpracován komponentou *Controller* a teprve ta změní data podle požadavku. V situacích, kdy je třeba se uživatele zeptat na dodatečné informace, například vybrat příslušný soubor nebo potvrdit nahrazení stávajícího modelu jiným, komponenta *Controller* využije metodu komponenty *View*, která „se zeptá“ uživatele a výsledek vrátí prostřednictvím návratové hodnoty, která pak může být dále zpracována.

Dalším užitím návrhového vzroru je třída `MessageFactory`, která na základě pozice v souboru záznamu komunikace vrací objekty typu `Message`, které vyjadřují analyzovaný stav daného datového paketu. Doba jejich existence končí při zaznamenání v nich obsažených informací v objektu třídy `SignalGroupModel`, představující komponentu *Model* předchozího návrhového vzoru.

Pro reprezentaci času napříč všemi moduly aplikace byla navržena třída `SignalTime`, která reprezentuje čas dvojicí datových typů a to standardním UNIXovým časem `time_t` a počtem milisekund v dané vteřině. Nad touto třídou byla přetížena většina standardně používaných operátorů, aby se dala použít jako klíč v mapách STL a bylo zjednodušeno veškeré použití. Dva objekty tohoto typu lze od sebe například odečíst pro získání intervalu mezi těmito dvěma časy, sečíst pro získání času posunutého o daný interval, porovnat nebo použít unární operátory `++` a `--` pro inkrementaci a dekrementaci času o jednu vteřinu.

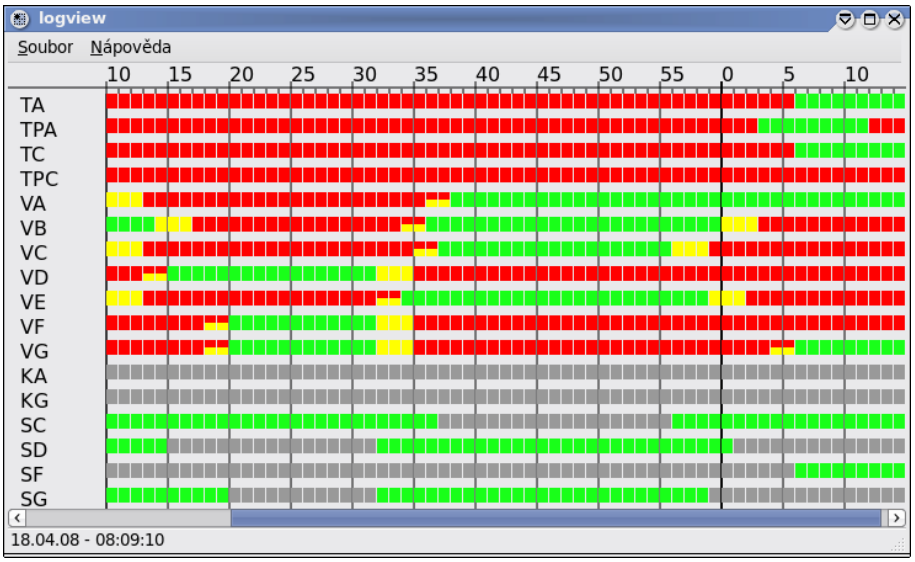
Celý diagram tříd lze zhlédnout na obrázku 4.2.



Obrázek 4.2: Diagram tříd

4.2.2 Grafické uživatelské rozhraní - GUI

Návrh uživatelského rozhraní sledoval dva hlavní cíle. Vzhledem by se aplikace měla podobat současně používané aplikaci pro vzdálené monitorování a vlastní plocha diagramu by měla umožňovat zobrazit co největší část signálního obrazu. Základní rozvržení bylo zvoleno standardním způsobem tedy v horní části okna hlavní nabídka nástrojů, níže hlavní kreslící plocha, pod ní posuvník a úplně dole stavový řádek. Snímek aplikace zobrazující signální průběh je k vidění na obrázku 4.3.



Obrázek 4.3: Snímek grafického uživatelského rozhraní

Hlavní nabídka je velice jednoduchá a obsahuje pouze nutné položky pro otevření definičního souboru, načítání záznamů a zobrazení informací o aplikaci. V případě, že by měla být doprogramována ještě nějaká další funkcionalita, je vpravo volná většina řádku pro umístění nástrojové lišty nebo dalších nabídek.

Pro vykreslování vlastního diagramu, tedy části signálního obrazu, byl zvolen přístup kreslící plochy s nezávislým posuvníkem. To umožní přesně kontrolovat, která část záznamu má být vykreslena a přispívá tak k rychlejšímu běhu programu. Při spuštění programu je posuvník automaticky neaktivní a zaktivuje se až po přidání prvního záznamu, kdy pak dle obecných zvyklostí šířka vlastního posuvníku znázorňuje poměr právě zobrazeného úseku vzhledem k délce celého záznamu.

Sama kreslící plocha je logicky rozdělena do tří částí. U levého okraje je vyčleněn sloupec pro názvy signálních skupin, jimž náleží jednotlivé řádky. Zbývající část je rozdělena horizontálně, kde horní pruh obsahuje časovou osu s popisy označující každý násobek pěti vteřin, a hlavní část, kam je vykreslován pásový diagram odpovídající danému signálnímu obrazu.

Názvy signálů signálních skupin jsou pak v diagramu zobrazovány barevně vyplněnými poli, jejichž barva intuitivně udává význam signálu. Pro tříbarevnou soustavu tedy *Stůj* – červená, *Pozor* (červeno–žlutá) – červenožlutě vyplněné a horizontálně rozdělené pole, *Volno* – zelená, *Pozor* (žlutá) – žlutá. V případě signálů, které blikají v půlvteřinových intervalech, je jedna z polovin pole vyplněna tmavě šedou barvou. Pokud v daném čase není definován stav všech relevantních žárovek signální skupiny, žádná nesvítí nebo jejich

stav neodpovídá žádnému platnému signálu, je pole vyplněno šedou barvou.

V nejnižší části okna je pak umístěn stavový řádek, který zobrazuje datum a čas první zobrazené vteřiny. Bez toho by byla orientace v záznamu jen těžko možná.

4.3 Implementace

Implementačním jazykem celé aplikace je C++. Jako objektově orientovaný programovací jazyk s nespočetným množstvím knihoven, které je možné použít při tvorbě programů byl vyhodnocen jako nejvhodnější. Díky tomu, že k implementaci byly využity pouze standardní funkce jazyka a platformově nezávislé knihovny, bylo možné vyvíjet aplikaci v operačním systému GNU/Linux s příležitostným testováním funkčnosti pod MS Windows.

Zde následuje detailnější popis některých podstatných a implementačně zajímavých tříd. Podrobnosti vlastní implementace je možné zhlédnout v programové dokumentaci na přiloženém médiu.

4.3.1 SignalGroupController

Objekt této třídy zabezpečuje běh celé aplikace. Ve funkci `main()` je vytvořena jedna instance a je zavolána její metoda `Run()`. Od této chvíle získává objekt kontrolu nad během programu a návrat z této metody znamená ukončení programu. Nicméně řízení je po několika málo inicializacích předáno hlavní smyčce okna, skryté v komponentě *View*, jejíž instanci před tímto předáním objekt vytvoří. Pak už se objekt dostane „ke slovu“ jen při předávání informací mezi komponentami *View* a *Model* a při reakci na některé uživatelské akce.

Komponenta se také stará o správnou reakci na výjimky vyvolané v běhu různých aplikačních částí. Takto jsou odchytávány výjimky vyvolané v třídách `SignalGroupModel` a `MessageFactory`. Po odchycení výjimky je program opět uveden do konzistentního stavu a pomocí komponenty *View* je uživateli zobrazena zpráva oznamující, že nastala chyba.

4.3.2 SignalGroupView

Tento objekt při aktivaci svojí metodou `Run()` spustí hlavní smyčku okna vytvořeného pomocí knihovny GTKmm. Převážná část jeho funkcionality je implementace handlerů signálů na uživatelské vstupy a vykreslování signálního obrazu do hlavního okna. Okno je vytvořeno instanciováním hlavního okna z xml souboru vytvořeného programem *Glade-3*. Součástí okna je hlavní kreslící plocha, která je implementována jako samostatná třída. Pro její správnou funkčnost je nutné jí při inicializaci předat ukazatel na objekt, který ji vlastní, protože potřebuje volat některé jeho metody.

Dále třída obsahuje funkce, prostřednictvím kterých umožňuje ostatním komponentám programu komunikovat s uživatelem. Takovou funkcí je například metoda zobrazující chybové hlášení nebo funkce pokládající otázku s možnostmi *Potvrdit* a *Zrušit*.

4.3.3 SignalGroupModel

Objekt této třídy je vytvářen až na uživatelovo přání načíst definiční soubor řadiče(typicky radic.xxx). Umístění tohoto souboru je předáno konstruktoru, který z něj načte informace o počtu, jménech a typech signálních skupin spolu s mapováním adres jednotlivých spínačových desek a na nich umístěných žárovek na žárovky jednotlivých skupin. Pro každou signální skupinu je vytvořen objekt třídy `SignalGroup`, který obsahuje jméno a typ skupiny

spolu s historií signálního průběhu. Mapování žárovek je uloženo v kontejneru `multimap` z STL. Položky mapy jsou pak dvojice typů, z nichž první udává zdrojovou fyzickou žárovku, tedy adresu spínačové desky a číslo žárovky, a druhá číslo signální skupiny a název žárovky v rámci skupiny. Mapa s více hodnotami pro stejný klíč je použita z důvodu nutnosti mapovat jednu fyzickou žárovku na několik signálních skupin. To je nutné například pro tramvajová návěstidla, kde až tři signální skupiny (tramvaj sm. vlevo, tramvaj sm. rovně a tramvaj sm. vpravo) sdílejí jedno fyzické návěstidlo.

V případě, že je během konstrukce souboru nalezena nějaká nekonzistence, je vyvolána výjimka, která uvozuje neplatný nebo poškozený definiční soubor. Bohužel, definiční soubory neobsahují žádnou signaturu nebo jiný prostředek, pomocí kterého by šlo s jistotou určit, že jde právě o definiční soubor. Z toho důvodu se může stát, že při načítání neplatného souboru budou všechna načtená data „odpovídat představám“ analyzátoru a i takový soubor bude načten. Chování programu je pak nespecifikované.

Zápis do historie signálních skupin pak provádí metoda `ParseMessage()`, která podle mapování postupně ukládá stavy jednotlivých žárovek do historie objektů signálních skupin voláním příslušných členských funkcí.

Dále objekt obsahuje mapu událostí, do které jsou zaznamenávány časy ve kterých začíná a končí záznam souboru a okamžiky „vteřinových tiků“. Jde o časy, ve kterých mění svůj stav signální skupiny, u kterých je změna zaručena pouze v celou vteřinu, tedy neblikají v půlvteřinových intervalech. Tyto záznamy se následně využívají při dopočítávání přesného času. Důvodem k tomuto řešení je fakt, že datový paket obsahující informace o čase může být odeslán v libovolném časovém okamžiku a jeho přesnost je omezena na jednotky vteřin. Pokud je tedy ke zpracování doručena zpráva obsahující takový časový paket, je v historii událostí vyhledána poslední změna úrovně signálu a doba od této změny vzhledem k času přijetí paketu je použita k dopočítání přesného času. Při přijetí prvního časového paketu není pouze vypočítána odchylka od času řadiče, ale zároveň jsou o tuto odchylku posunuty i všechny dosud zaznamenané stavy. Další časové údaje pak můžou být vzhledem času řadiče zaznamenány včetně tohoto časového posunu, aniž by se to projevilo na jednotlivých časových intervalech signálního obrazu.

Za běžných podmínek by doba od poslední změny signálu neměla být větší než několik desítek vteřin, což umožní relativně přesné určení času. V situaci, kdy by signální plán obsahoval dlouho se neměnící signály (například trvalý signál *Volno* pro vozidla přerušený pouze v případě stisknutí „chodeckého tlačítka“), by se sice mohlo stát, že se hodiny rozejdou o větší časový úsek, ale v neměnném signálu se tento rozdíl nijak neprojeví.

Protože uživatel by mohl chtít načítat soubory záznamů v různém pořadí, tedy k již načtenému záznamu načíst jiný, pocházející z předchozího časového úseku, byla zavedena ještě mapa časových synchronizací, která umožňuje podle úvodního času v souboru načíst také rozdíl oproti reálnému času, který byl vypočítán v průběhu předchozího zpracování záznamů.

4.3.4 DiagramArea

Tato třída dědí vlastnosti třídy `Gtk::DrawingArea`. Definuje několik nových členských proměnných a přetěžuje její metodu `on_expose_event()`, která slouží k vykreslení plochy při jejím odkrytí nebo při změně její velikosti. Metoda pak pomocí komponenty *Controller* získá seznam jmen signálních skupin, které vypíše do sloupce při levém okraji, začátek a délku historie pro zobrazení časové osy a jména stavů signálních skupin v časech, které vykreslí na hlavní část plochy.

4.3.5 SignalGroup

Objekty této třídy popisují jednotlivé signální skupiny. Obsahují jméno a typ skupiny a historii úrovní signálů. Členská funkce `GetSignalStateName()` pak umožňuje na základě času získat jméno právě aktivního signálu. Jméno signálu je logicky závislé na tom, které žárovky signální skupiny svítí a na typu signální skupiny.

4.3.6 SignalHistory

Každý objekt třídy `SignalGroup` obsahuje právě jeden objekt této třídy pro uchovávání historie signálu. Hlavní datová struktura je implementována jako mapa času na stavy signálu. Do této mapy jsou ukládány pouze změny signálu, čímž je dosažena poměrně veliká úspora potřebné paměti. Stav signálu v daném čase je definován jako stav uložený v záznamu s nejbližším nižším nebo rovným časovým údajem.

Přidat stav do historie je možné pomocí metody `InsertBulbState()`. Stav se vkládá po jednotlivých žárovkách, protože jedna signální skupina může být poskládána z několika fyzických žárovek, jež jsou rozmístěny na několika různých spínačových deskách. Z tohoto důvodu je také nastaven toleranční čas 50 ms, který udává maximální časový interval, po který se vkládaný signální stav považuje za součást předchozího.

Pro přístup ke stavům signálu v jednotlivých časech je nutné použít členskou funkci `GetSignalState()`. Ta vrací stav všech žárovek signální skupiny buď přesně v daném časovém okamžiku nebo agregovaný stav vzhledem k celé vteřině. V agregovaném stavu se mimo běžných hodnot(Zapnuto, Vypnuto a Nedefinováno) může vyskytovat i hodnota Rozsvícení nebo Zhasnutí, pro reprezentaci stavů blikajících skupin, kdy žárovka v první polovině vteřiny nesvítí a v druhé svítí nebo naopak.

4.3.7 MessageFactory

Instance této třídy je obsažena v objektu představujícím komponentu *Controller*. Vnitřní stav objektu je mimo jiné dán pozicí v souboru záznamu. Po zavolání metody `OpenFile()`, se objekt pokusí zjistit typ souboru z jeho přípony. Pokud přípona nepatří známému souborovému typu, je vyvolána výjimka. V opačném případě je pro čtení souboru zkonstruován objekt dědící vlastnosti třídy `LogReader`, jehož konstruktoru je předána cesta k souboru. Ten se jej pokusí otevřít a v případě neúspěchu vyvolá výjimku. Při zavolání metody `GetMessage()` je ze souboru přečten jeden řádek představující jeden paket datové komunikace nebo poznámku uvozenou znakem `'#'`. Prázdné řádky jsou ignorovány. Podle dat je pak vytvořen objekt třídy `Message`, jehož členské proměnné obsahují informace načtené z příslušného paketu.

4.3.8 Message

Objekty této třídy jsou konstruovány třídou `MessageFactory` a nesou data z přijatých datových paketů nebo chybový kód. To, které členské proměnné jsou inicializovány závisí právě na chybovém kódu, který udává proměnná `m_errno`. Pokud jde o běžný datový paket, je inicializován čas přijetí paketu, adresa spínačové desky a stavy jednotlivých žárovek. V případě, že došlo k chybě v komunikaci, tedy například spínačová deska nezaslala odpověď nebo jsou data jiným způsobem porušená, je nastavena hodnota odpovídající chybě dat nebo nekorektní hodnotě CRC. Při synchronizaci systémového času odposlouchávajícího PC, je nastaven čas přijetí podle posledního přijatého paketu a proměnná `m_MiscTime` obsahuje

rozdíl o který byl čas posunutý. V případě přijetí paketu obsahujícího informaci o čase, je opět čas přijetí nastaven na čas přijetí paketu a proměnná `m_MiscTime` na čas, který obsahuje přijatý paket.

Kapitola 5

Závěr

Oba hlavní cíle práce byly zpracovány v plném rozsahu zadání i přes mírné úpravy vedlejších požadavků během implementace. Tím se zaplnila mezera v možnostech sledování řízení provozu a kontroly komunikace komponent řadiče. Pro nasazení odposlechové aplikace se v současné době zkouší ještě jiný typ vestavěného PC než na kterých byla vyvíjena a testována, nicméně na předchozích dvou typech je vše již plně funkční, otestované a čekající na první ostré nasazení v provozu.

Díky modulárnímu zpracování je v budoucnu možné obě aplikace rozšířit o další funkce. U aplikace pro záznam komunikace je možné doplnit možnosti širší analýzy datového toku popř. využít většinu modulů pro záznam dat z jiných částí řadiče, komunikujících podobným způsobem. Takto již byla zvažována možnost zaznamenávat napěťové úrovně a diagnostická data ze spínačových desek, které je odesílají na sběrnici RS-232.

Aplikace pro zobrazování záznamů by v případě potřeby mohla být doplněna například o možnost zobrazení vstupů z detektorů nebo jiných dat souvisejících s vlastním signálním obrazem. Například by mohla zobrazovat časy, kdy došlo k přepnuí signálního plánu atd.

Jako autor jsem získal mnoho nových zkušeností se zajímavou a ne zcela běžně dostupnou elektronikou. Řízení dopravy je perspektivní obor, do kterého stále více pronikají informační technologie a zkušenosti nabyté zpracováním této práce se mohou v budoucnu ukázat jako velice cenné. Dalším osobním přínosem je pro mne zkušenost s využitím sériové komunikace a bližší seznámení s nízkoúrovňovými funkcemi aplikačního rozhraní Linuxu. Za nejlepší ale považuji to, že moje práce má reálné praktické využití. Dalším přínosem mé práce je to, že firma CROSS Zlín s. r. o. se rozhodla nasadit Linux v jednom ze svých budoucích produktů a zvažuje využití v některých dalších projektech.

Literatura

[1] GTK+: Oficiální webové stránky projektu.
URL http://www.gtk.org/

[2] GTKmm: Oficiální webové stránky projektu.
URL http://www.gtkmm.org/

[3] MinGW: Oficiální webové stránky projektu.
URL http://www.mingw.org/

[4] Vyhláška č. 30/2001 Sb.: příloha č. 5 – světelné signály. Znění platné od 1.7.2006.

[5] Zlib: Oficiální webové stránky projektu.
URL http://www.zlib.net/

[6] DM&P: *DM&P X-Linux Developer's Manual*. 2006.

[7] GAMMA, E.; HELM, R.; JOHNSON, R.; aj.: *Návrh programů pomocí vzorů*. Grada, 2003, ISBN 80-247-0302-5, překlad: Pavel Makovec.

[8] ICOP Technology Inc.: *VX86–6042 – Embedded Vortex86^{TM} Half-Size AIO SBC – User's manual*. 2004.

[9] National Semiconductor: *PC16550D Universal Asynchronous Receiver Transmitter with FIFOs datasheet*. 1995.

[10] Český normalizační institut: *ČSN EN 12675 Řízení dopravy na pozemních komunikacích – Řadiče světelných signalizačních zařízení – Funkčně bezpečnostní požadavky*. 2002.

[11] Český normalizační institut: *ČSN EN 36 5601 Systémy silniční dopravní signalizace*. 2002.

Seznam příloh

A Příručka k programu pro záznam komunikace

B Příručka k aplikaci pro zobrazení záznamů

Přiložené CD Zdrojové kódy a programová dokumentace implementovaných programů

Dodatek A

Pokyny k programu pro odposlech záznamů

A.1 Překlad

Pro překlad je vytvořen soubor *Makefile*, který umožní vytvoření spustitelného souboru použitím standardního nástroje *make*. Pro překlad je nutné mít nainstalovánu vývojářskou verzi knihovny *zlib*. Jak již bylo uvedeno v textu, při kompilaci programu překladačem *gcc* je nutné uvést přepínač `-std=gnu99`, kvůli nutnosti zpracovávat systémové signály.

A.2 Provoz

Program je nutné spouštět s **právy superuživatele**, protože program přistupuje k nízkoúrovňovým systémovým prostředkům, které nejsou zpřístupněny jiným uživatelům. Konkrétně jde o synchronizaci HW hodin a přístup k portu.

Program je možné spustit úplně bez parametrů, nicméně ve většině případů je třeba nastavit alespoň základní parametry záznamu jako je soubor vstupního portu, výstupní adresář a počet řádek na soubor. Výstupní adresář nesmí obshahovat jiné soubory než soubory záznamu. Výjimkou jsou pouze skryté soubory, které program ignoruje.

Pokud je nastaven argument infotick na hodnotu jinou než 0, po uplynutí tohoto časového intervalu je na standardní chybový výstup vypsán aktuální stav programu.

A.3 Seznam přepínačů a jejich význam

`-h,--help` Přenosová rychlost

`-b <baudrate>` Přenosová rychlost

`-i <input>` Vstupní soubor (blokový soubor portu)

`-d <directory>` Pracovní adresář

`-l <linecount>` Počet záznamů v jednom souboru

`-s <space>` Minimální velikost volného místa v bytech

`-m <mode>` Mód výstupu dat

tout Pouze výpis na stdout

text Výstup do souboru, textový mód

bnc Výstup do souboru binárně, bez komprese

bh Výstup do souboru binárně, hufmanovo kódování

brl Výstup do souboru binárně, nízká paměťová náročnost

brh Výstup do souboru binárně, vysoká paměťová náročnost

`-e <extension>` Přípona souboru (bez prvního znaku '.')

`-to <timeout>` Doba klidu na sběrnici po které je uzavřen soubor

`-it <infotick>` Časový interval ve vteřinách po kterém bude program vypisovat info

`-nst` Neprovádět synchronizaci času

A.4 Formát zaznamenaných dat

```
#T <timestamp> - DD.MM.RR - HH:MM:SS
#X poznámka začínající znakem '#' a ihned nasledujicí
#X písmeno určijící typ poznámky

#C záznam datového paketu uvozený časem přijetí
#C data jsou vypsána v˜hexadecimální soustavě
MM:SS.mmm XX XX XX XX XX XX

#C libovolný počet datových paketů

#C záznamu může dojít k˜synchronizaci času
#C po synchronizaci bude vypsán nejprve paket, který nesl informaci o˜čase
#C po té komentář oznamující sychnronizaci času
#C tedy formátu stejného jako první řádek

#I Closing file
```

Dodatek B

Pokyny k programu pro zobrazení zánamů

B.1 Překlad

Program je možné přeložit pomocí standardních nástrojů *Automake* a *Auotoconf* na platformách GNU/Linux i MS Windows. Pro překlad je nutné mít nainstalovány vývojářské verze knihoven GTK+ a GTKmm. Pod MS Windows je nutné překládat program v prostředí MinGW. [1] Po spuštnění příkazů `./configure` a `make` je výstupem spustitelný soubor s názmem `logview`, který pro svůj běh potřebuje nalézt ve svém aktuálním adresáři soubor `logview.glade` s definicí uživatelského rozhraní.

B.2 Použití

Po spuštění aplikace je zobrazeno s prázdnou plochou diagramu. V prvním kroku je nutné načíst definiční soubor řadiče pomocí položky *Nový* v nabídce *Soubor* hlavního menu. Po výběru souboru definice je ze souboru načten seznam typů a jmen signálních skupin a mapování žárovek. Po té je možné načíst libovolný počet souborů záznamu pomocí položky *Otevřít* umístěné rovněž v nabídce *Soubor*. Při načítání záznamů z delšího období může načítání na pomalejších počítačích trvat dlouhou dobu. (10 minut záznamu představuje cca 500 000 datových paketů.)

Kdykoliv v průběhu prohlížení signálního obrazu je možné načíst další soubory se záznamy, opět pomocí položky *Otevřít*. Při načtení nového definičního souboru budou zapomenuty všechny načtené záznamy je možné načíst jiné, související s daným definičním souborem.

B.3 Legenda: Význam barev polí

Červená Signál *Stůj*

Červeno-žlutá – horizontálně rozdělená Signál *Pozor* (červeno-žlutá)

Zelená Signál *Volno*

[1] Při potížích s překladem na MS Windows může pomoci vymazání adresáře `autom4te.cache` a spuštění příkazu `aclocal -I C:\GTK\share\aclocal`.

Žlutá Signál *Pozor* (žlutá)

Vertikálně rozdělené pole blikající signál dané barvy – význam závisí na typu signální skupiny

Šedá Žádný, nesmyslný nebo nedefinovaný signál