UNIVERZITA PARDUBICE

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Ovládací software modemu GSM-R pro embedded systém EXM32

DIPLOMOVÁ PRÁCE

2009 **Bc. Josef BAČKOVSKÝ**

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Ovládací software modemu GSM-R pro embedded systém EXM32

Bc. Josef Bačkovský

Diplomová práce
2009

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Katedra softwarových technologií
Akademický rok: **2008/2009**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Josef BAČKOVSKÝ**
Studijní program:	**N2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Ovládací software modemu GSM-R pro embedded systém EXM32**

Zásady pro vypracování:

* Úvodní část diplomové práce se bude zabývat problematikou komunikace drážních vozidel s využitím komunikačního zařízení MTR10 (RADOM, s.r.o., Pardubice) a popis tohoto zařízení. * Dále bude uveden přehled a popis použitých technologií (Sqlite, Embedded systém EXM32, modem MT2) a uvedení do překladu kódu pro EXM32. * Implementační část diplomové práce bude návrh a vytvoření ovládacího software pro modem GSM-R (Kapsch MT2). Tento software bude rozdělen na serverovou a klientskou část. Komunikace obou částí bude realizována technologií Sqlite. Dále bude vytvořen diagnostický program pro testování ovládacího softwaru. Komunikační software bude vyvinut v programovacím jazyku C++ a přeložen pro EXM32. Testovací software pro platformu PC bude vyvinut v programovacím jazyku C++ s použitím Framework QT.

Rozsah grafických prací:

Rozsah pracovní zprávy:

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

Seznam odborné literatury:

1. MATTHEW, Neil, STONES, Richard. Linux Začínáme programovat. 2000. 912 s. ISBN 8072263072.

2. PRATA, Stephen. Mistrovství v C++. 2007. 1120 s. ISBN 978-80-251-1749-1.

3. ECKEL, Bruce. Myslíme v jazyku C++ – knihovna programátora. 2000. 556 s. ISBN 80-247-9009-2.

Vedoucí diplomové práce: **Mgr. Tomáš Hudec**
Katedra informačních technologií

Datum zadání diplomové práce: **31. října 2008**

Termín odevzdání diplomové práce: **22. května 2009**

Univerzita Pardubice Česká republika

L.S.

doc. Ing. Simeon Karamazov, Dr.
děkan

doc. Ing. Antonín Kavička, Ph.D.
vedoucí katedry

V Pardubicích dne 4. listopadu 2008

Prohlašuji:

Tuto práci jsem 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 28. 5. 2009

Josef Bačkovský

PODĚKOVÁNÍ

Rád bych poděkoval společnosti RADOM, s. r. o., Pardubice, za možnost tvorby této práce na jejich prostředcích a za odborné konzultace, které mi poskytli zaměstnanci této firmy. Dále bych rád poděkoval Mgr. Tomáši Hudcovi za odborné vedení této práce.

ANOTACE

Práce se zabývá komunikací drážních vozidel v síti GSM-R. Cílem práce je vytvořit aplikaci pro ovládání modemu GSM-R, která se skládá ze serverové a klientské části. Primární komunikace obou části je realizována technologií SQLite. Aplikace je tvořena pro OS Linux a serverová část je kompilována pro vestavěný systém EXM32.

KLÍČOVÁ SLOVA

GSM-R; MT2; SQLite; EXM32; Linux; RS-232; Sériový port; MTR10

TITLE

GSM-R modem controlling software for EXM32 embedded system

ANNOTATION

The dissertation is about railway communication in GSM-R net. The target of the project is to create an application to control GSM-R modem. The application uses a server-client architecture. The primary communication is realized by SQLite technology. The application is made for the Linux OS and the server part is compiled for embedded system EXM32.

KEYWORDS

GSM-R; MT2; SQLite; EXM32; Linux; RS-232; Serial Port; MTR10

OBSAH

Seznam obrázků

Seznam tabulek

ÚVOD

Tato diplomová práce se zabývá problematikou komunikace drážních vozidel. Cílem této práce je vytvoření ovládacího softwaru modemu GSM-R MT2 od společnosti Kapsch, s. r. o. Software, který je vyvíjen pro komunikační zařízení MTR10 společnosti RADOM, s. r. o., Pardubice. Testování a vývoj aplikace je prováděn na prostředcích této společnosti.

Úvodní část této diplomové práce obsahuje úvod do problematiky komunikace drážních vozidel. Popisem používaných komunikačních technologií, jaká jiná zařízení se v současné době na našich tratích používají a jaké technologie se budou využívat do budoucna.

Druhá část popisuje zařízení MTR10, příčiny jeho vzniku, strukturou a nároky, které jsou na toto zařízení kladeny, a to jak z hlediska hardwaru tak softwaru.

Třetí část se zabývá popisem a opodstatněním výběru použitých technologií při vývoji zařízení MTR10.

Čtvrtá a hlavní část popisuje konfigurace použitých technologií a samotné implementace ovládacího softwaru pro modem GSM-R. Obsahuje popis klíčových částí softwaru a ukázky kódu.

1. KOMUNIKACE DRÁŽNÍCH VOZIDEL

V současné době je na tratích nejen v České republice, ale především v Evropě používáno velké množství více či méně nekompatibilních komunikačních systémů. Tento fakt znemožňuje volný pohyb hnacích vozidel po tratích, jak tuzemských tak především zahraničních. Tato kapitola se zabývá přehledem těch nejpoužívanějších v České republice a popisem systémů, které se v Evropě budou využívat v budoucnu. Zdroje použité v této kapitole: [1], [11], [12], [13], [14], [15].

1.1 Analogové rádio (simplex)

Tento systém byl vyvinut v šedesátých letech minulého století. Pracuje v pásmu VKV 150 MHz, v současnosti je to nejstarší používaný komunikační systém na tratích v České republice. Umožňuje pouze jednosměrnou komunikaci (simplex), to znamená, že v daný okamžik může po zaklíčování kanálu hovořit pouze jeden z účastníků hovoru a ostatní účastníci hovoru (jeden nebo více) mohou pouze poslouchat. Jelikož se jedná o systém, který nevyužívá žádného systému adresování, tedy „všichni slyší vše", není vhodný pro použití na úsecích s větším provozem. Tento systém se již dále nerozvíjí.

1.2 Tesla TRS (duplex, pásmo 450 / 460 MHz)

Obr. 1) Tesla TRS Ovládací skříňka [Zdroj: RADOM]

1.2.1 Pojmy používané v souvislosti se systémem TRS

- **TRS** – traťový rádiový systém,
- **stuhová síť** – skupina základnových radiostanic, umístěných podél traťového úseku,
- **kmitočtová čtveřice** – mezinárodně koordinované duplexní kanály v pásmu 450 MHz,
- **simplexní kanál** – komunikační kanál v pásmu 450 MHz nebo 160 MHz,
- **generální volba** – přenos hlasu ve směru od dispečera nebo výpravčího ke strojvedoucímu hnacího vozidla,
- **nouzové volání** – prioritní tísňové volání.

1.2.2 Charakteristika systému

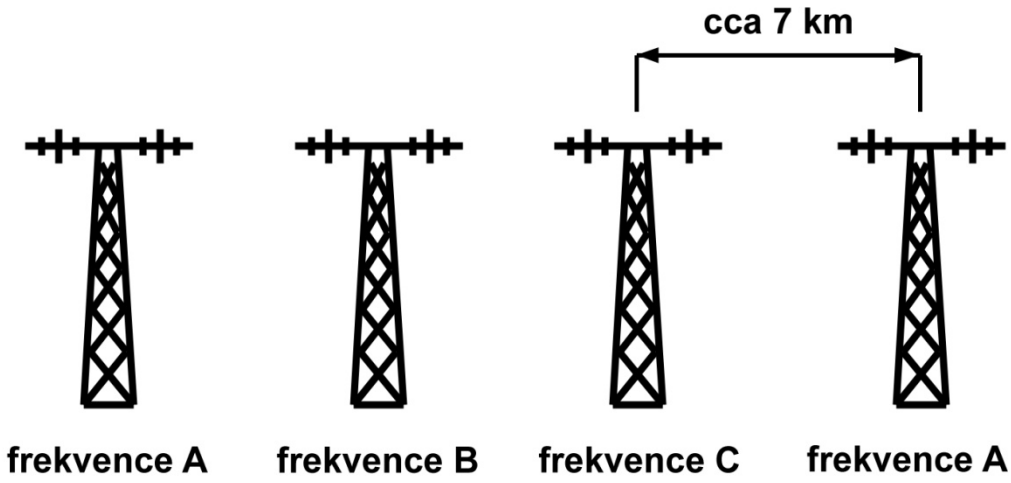
Traťový rádiový systém (TRS) byl vyvinut v Tesle Pardubice počátkem devadesátých let a je určen pro duplexní spojení dvou účastníků (strojvedoucího s dispečerem nebo výpravčím). Tento systém je v současné době nejrozšířenější na tratích v České republice. Spojení strojvedoucího jedoucího vlaku s dispečerem je realizováno po celé délce trati, která je vybavena systémem TRS. Pro spojení strojvedoucího s výpravčím je dočasně vyčleněna základnová stanice stuhové sítě, v jejímž dosahu se nachází strojvedoucí. Systém TRS umožňuje také přenos identifikace, selektivní volbu lokomotivy, přenos rutinních informací, navázání spojení s jiným hnacím vozidlem s využitím selektivní retranslace a simplexní spojení hnacího vozidla s dalšími účastníky v prostoru nádraží.

TRS zajišťuje základní kompatibilitu provozu s jinými systémy dodržováním základních doporučení UIC 751-3 a OSŽD 875/1.

Dodržovaná doporučení umožňují v kombinaci s jinými systémy:

- navázaní hovorového spojení,
- generální volbu mobilních prostředků,
- vysílání signálu NOUZE.

Systém TRS realizuje spojení na mezinárodně koordinovaných kmitočtových čtveřicích v pásmu 450 MHz. Kmitočtová čtveřice je tvořená symbolicky označenými kmitočty A, B, C (kmitočty vysílačů základnových radiostanic) a D (kmitočet vysílače soupravy hnacího vozidla). Pro simplexní provoz je na soupravě hnacího vozidla použita samostatná radiová část v pásmu 450 MHz nebo 160 MHz.



Obr. 2) Schéma sítě TRS

Stuhová síť je tvořená řadou základnových radiostanic, které jsou rozmístěny podél traťového úseku. Princip rozmístění radiostanic je zobrazen na obrázku 2 (Zdroj: autor). Stuhová síť je řízena stanovištěm dispečera. Mikropočítač vyhodnocuje naměřenou sílu signálu na jednotlivých kmitočtech A, B, C a nastaví přijímač hnacího vozidla na kmitočet s nejlepším signálem. Základnové stanice automaticky vyhodnocují nejsilnější signál na kmitočtu D a k dispečerovi se propojí základnová stanice s nejlepším signálem.

1.2.3 Komunikační možnosti systému TRS

- Duplexní spojení mezi strojvedoucím a dispečerem, nebo výpravčím umožňující přenos hovoru, rutinních informací, selektivní nebo generální volbu a nouzového volání. V případě spojení strojvedoucího s výpravčím je toto spojení realizováno prostřednictvím dočasně vyhrazené základnové radiostanice.
- Semiduplexní spojení mezi strojvedoucími pomocí retranslací v dosahu základnové stanice umožňující hovor.

- Simplexní spojení mezi strojvedoucími a dalšími účastníky na předem zvoleném kanále.

Pro digitální komunikaci je TRS vybaven formou přenosu telegramů FFSK 1200 bit/s v obou směrech.

1.3 ERTMS

Rozvoj vysokorychlostních tratí zapříčinil, že pohyb vlakových souprav po tratích Evropské unie omezují národní komunikační systémy. Z tohoto důvodu UIC (International Union of Railways), ve snaze unifikovat dopravní technologii v Evropském prostoru, zahájilo definování tzv. pan-evropské železniční sítě, jedná se o síť vysokorychlostních železničních koridorů.

V roce 1992 zahájilo UIC práce na řešení projektu pod názvem ERTMS (European Rail Traffic Management System), který má za cíl vytvořit jednotný komunikační standard pro Evropské železnice. Základní stavební kameny tohoto projektu jsou ETCS a GSM-R (podrobný popis je uveden v kapitole 2).

1.3.1 ETCS

ETCS (European Train Control System) je zkratka pro evropský vlakový zabezpečovací systém. Jeho cílem je postupně nahradit asi 20 národních vlakových zabezpečovacích systémů a tím umožnit pohyb vlakových souprav po evropských tratích bez nutnosti vybavování hnacích vozidel větším množstvím nekompaktních systémů či výměně hnacích vozidel na hranicích států. Více podrobností se lze dozvědět například ze zdroje [11].

2. GSM-R

Další součástí ERTMS je komunikační standard GSM-R, v této kapitole jsou použity zdroje [13], [14], [15]. Jak již sám název napovídá standard GSM-R (Global System for Mobile Communications - Railway) vychází ze standardu GSM. Tento standard lze použít ke vzájemné komunikaci mezi všemi účastníky drážního provozu (např. strojvedoucí, dispečer, výpravčí, průvodčí atd.).

Technologie GSM byla zvolena jako základ pro moderní železniční komunikační sytém pro své úspěšné dlouholeté používání. GSM-R umožňuje komunikaci účastníků bez výpadků do rychlosti 500 km/h. Tyto dva standardy se od sebe liší jednak používanými kmitočty a také specifickými funkcemi standardu GSM-R, které jsou popsány dále v textu.

Pásmo	**Kmitočet (MHz)**
GSM veřejných operátorů	880 - 915 / 925 - 960
GSM-R	876 - 880 / 921 - 925

Tab. 1) Kmitočty sítí GSM a GSM-R [Zdroj: 18]

Na úsecích tratí, které nedisponují pokrytím GSM-R, mohou systémy používat sítě veřejných poskytovatelů služeb GSM. Specifické funkce jsou zde řešeny emulací.

2.1 Specifické funkce GSM-R

2.1.1 eMLPP (Multi-Level Precedence and Pre-emption Service)

První ze specifických funkcí GSM-R je víceúrovňová priorita a preempce. Víceúrovňová priorita garantuje dostupnost spojení pro kritické hovory, jako například REC (Railway Emergency Call), který je popsán dále v textu.

Preempce umožňuje ukončení hovoru dvou či více účastníků pokud například brání spojení hovoru s vyšší prioritou nebo je-li sít přetížena a je zapotřebí uvolnit přenosovou kapacitu pro tyto hovory.

Priority:

- A, B – nejvyšší priorita vyhrazená pro systémové zprávy (používá se pro řízení sítě),
- 0 – nejvyšší priorita pro hlasové služby,
- 1 až 4 – další priority (čím větší číslo tím nižší priorita), implicitní priorita hlasových služeb je 4.

2.1.2 VGCS (Voice Group Call Service)

Zkratku VGCS lze do češtiny přeložit jako skupinové hovory. Tato funkce umožňuje spojení většího množství účastníků a jedná se v podstatě o digitální náhražku analogové funkce PMR (Private Mobile Radio). Stejně jako v analogové podobě i u VGCS může v jednom okamžiku hovořit pouze jeden účastník. Pro řízení toho, který z účastníků má hovořit, se využívá PTT (Push To Talk – Stiskni a mluv). Po stisknutí PTT zařízení odešle do sítě požadavek k zaklíčování hovoru. Pokud je linka volná síť umožní zaklíčování, pokud není linka volná, síť o tom informuje zařízení a k zaklíčování nedojde.

VGCS je obdobou konferenčních hovorů, které jsou součástí standardu GSM, oproti konferenčním hovorům je efektivněji využívaná přenosová kapacita sítě. Díky faktu, že hovoří vždy pouze jeden účastník, VGCS využívá pouze jedné frekvence pro vysílání hlasu všem účastníkům a jedné frekvence pro přenos od účastníka, který má zaklíčovanou linku. Klasické GSM při konferenčním hovoru využívá dvě frekvence pro každého účastníka.

Pro používání VGCS jsou nadefinované skupiny, každá skupina má nastavené územní oblasti. Pokud jeden z účastníků zahájí hovor ve skupině je okamžitě spojen se všemi účastníky, kteří jsou zařazeni do dané skupiny a nachází se ve stejné územní oblasti jako zakladatel skupinového hovoru. Při zahájení hovoru ve skupině jsou ostatní hovory účastníků patřící do dané skupiny ukončeny, pokud mají nižší prioritu.

V současné době na tratích v České republice jsou územní oblasti pro většinu skupin nadefinovány na celou oblast ČR, z tohoto důvodu zatím nejsou VGCS v reálném provozu hojně používány. Výjimkou jsou nouzové hovory.

2.1.3 VBS (Voice Broadcast Service)

VBS je obdoba VGCS s tím rozdílem, že hovořit smí vždy pouze zakladatel skupiny.

2.1.4 REC (Railway Emergency Call)

Nouzový hovor je speciálním typem VGCS s nejvyšší prioritou 0. Nejvyšší priorita zaručuje nouzovému hovoru spojení hovoru se všemi účastníky v dané skupině a územní oblasti.

2.1.5 Functional number management

Adresování pomocí funkčních čísel umožňuje dispečerům a ostatním účastníkům provozu komunikovat bez nutnosti znalosti volacích čísel konkrétních hnacích vozidel a dalších účastníků provozu.

Jak již napovídá název, podstata systému funkčních čísel spočívá v přihlašování jednotlivých zařízení do systému dle jejich funkce. Například každý vlak pohybující se na tratích v ČR má své vlastní funkční číslo nezávislé na hnacím vozidle. Pomocí systému funkčních čísel se může strojvedoucí jednoduše přihlásit jako strojvedoucí daného vlaku a ostatní účastníci provozu ho mohou volat bez znalosti čísla hnacího vozidla.

Struktura funkčního čísla je `<RAC><CallType><Uin><FK>`:

- `<RAC>` – národní předvolba sítě GSM-R, pro ČR `420`,
- `<CallType>` – typ volání, používá se k rozlišení jednotlivých druhů zařízení a jejich použití, například hodnota `2` odpovídá vlaku,
- `<Uin>` – identifikační číslo, například číslo vlaku `90200`,
- `<FK>` – funkční kód, například `1` odpovídá strojvedoucímu jedna.

3. POPIS ZAŘÍZENÍ MTR10

Větší množství používaných komunikačních systémů, jejichž přehled je uveden v předchozím textu, zapříčinil nutnost vytvoření nového typu digitálního traťového systému (DTS), který by umožňoval komunikaci s využitím stávajících (analogový TRS) i perspektivních (GSM-R) drážních radiových systémů.

Vývojem takovéhoto zařízení se samozřejmě nezabývá pouze firma RADOM, s. r. o., ale i další firmy jako například Kapsch, s. r. o. Firma Kapsch jako specialista na dopravní komunikační a zabezpečovací systémy má díky svým dlouholetým zkušenostem velký podíl na Evropském trhu se systémy DTS.

Firma RADOM, s. r. o., již vyvinula zařízení FXM20, toto splňuje všechny funkcionality podle zadání Českých drah, a. s. Systém FXM20 je modulární a skládá se ze dvou hlavních částí.

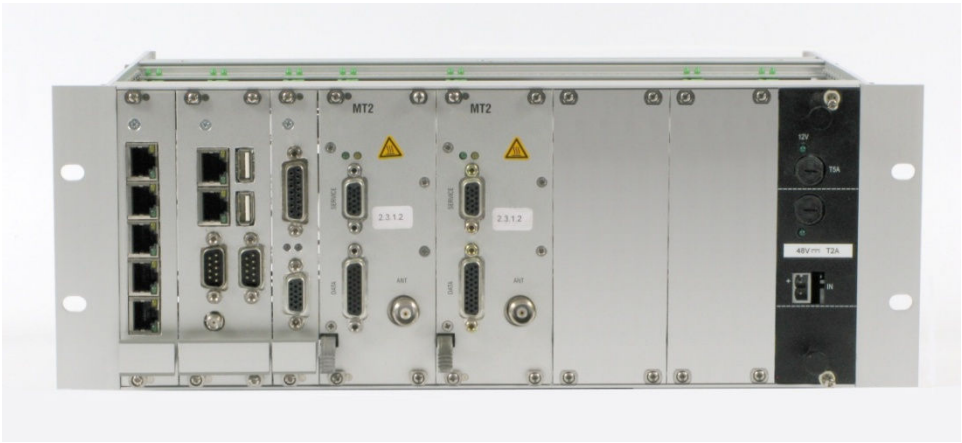
Obr. 3) FRB20 [Zdroj: RADOM]

První částí je blok rádia označovaný FRB20 a druhou částí je ovládací skříňka FCB20, která je umístěna na obou čelech hnacího vozidla. Tato modularita umožňuje vybavení hnacího vozidla systémem dle konkrétních potřeb, a tím umožňuje snížení nákladů. Zařízení FXM20 je otestováno a schváleno pro provoz.

Obr. 4) FCB20 [Zdroj: RADOM]

Vedení společnosti RADOM, s. r. o., si uvědomuje potenciál tohoto odvětví trhu, a proto neustále inovuje své systémy, v souladu s tím vyvíjí zcela nový DTS pod názvem MTR10 založený na nových technologiích.

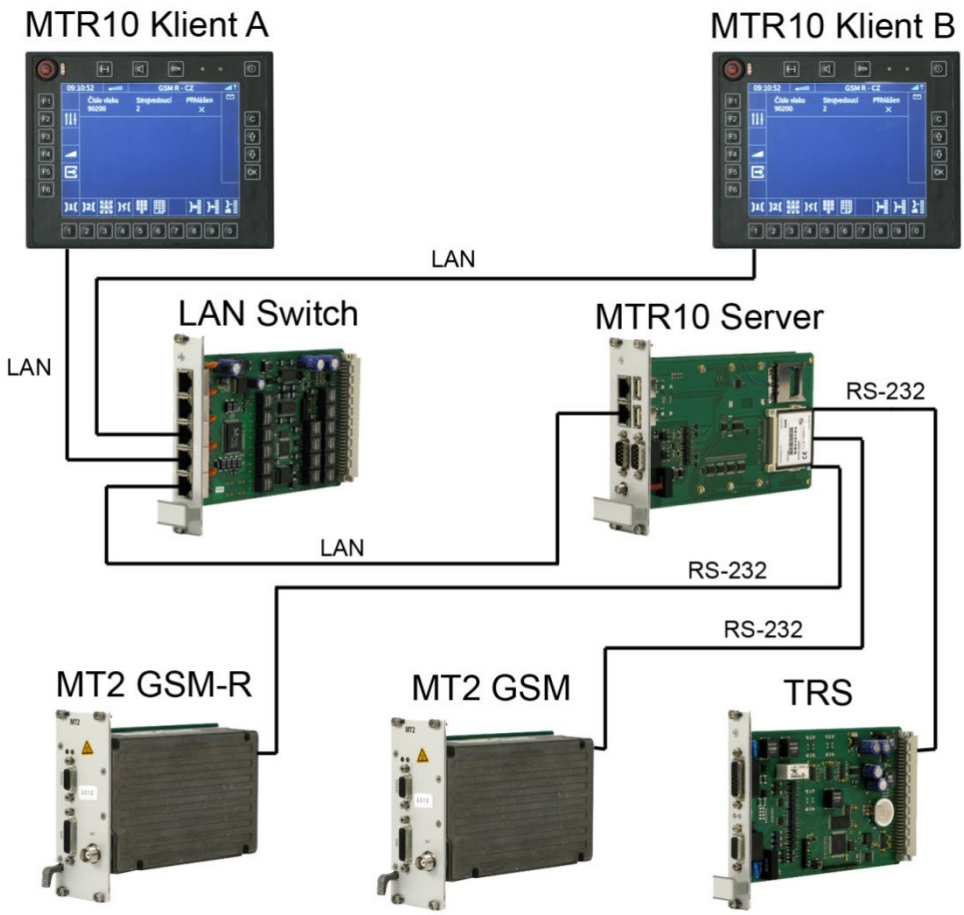
MTR10 se skládá ze dvou hlavních částí. První částí je MTR10 Server, který obsluhuje modem GSM-R, GSM a systém TRS. Druhou částí je MTR10 Klient, který zprostředkovává interaktivní ovládání obsluze, touto částí je vybaveno přední a zadní čelo hnacího vozidla.



Obr. 5) MTR10 Server [Zdroj: RADOM]

Obr. 6) MTR10 Ovládací skřínka [Zdroj: RADOM]

Obě části jsou propojeny prostřednictvím sítě LAN. Toto je jedna z klíčových změn o proti zařízení FXM20, které pro komunikaci používá sériový port RS-485 a pro přenos hlasu samostatné vodiče. Další významnou inovací oproti zařízení FXM20 je využití embedded systémů EXM32 s operačním systémem Linux. Tato inovace značně rozšiřuje možnosti zařízení, například přenos hlasu prostřednictvím IP telefonie, ale také s sebou nese celou řadu komplikací, například delší doby startování systému nebo delší doby obsluhy sériového portu atd.



Obr. 7) Schéma zapojení MTR10 [Zdroj: autor]

K serverové části jsou připojené prostřednictvím sériového portu RS-232:

- modem MT2 se SIM-kartou sítě GSM-R,
- modem MT2 se SIM-kartou sítě GSM veřejných operátorů,
- obslužná karta komunikačního systému TRS.

Další možnou částí systému MTR10 může být i snímač polohy GPS. Modemy MT2 se ovládají pomocí AT-příkazů. Podrobnější vysvětlení systému AT-příkazů je uveden v kapitole Modem MT2.

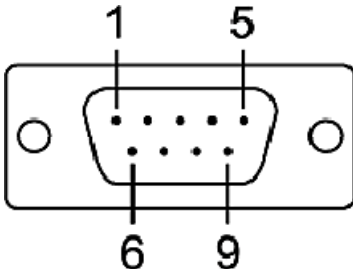
4. POUŽITÉ TECHNOLOGIE

4.1 Sériový port

Tato část práce se zabývá popisem komunikace po sériovém portu, v této části jsou použity zdroje [16], [17], [18]. Sériový port (nebo také sériová linka) RS-232 respektive jeho poslední verze z roku 1969 RS-232-C byl původně určen pro komunikaci s periferiemi a také pro propojování osobních počítačů. Sériový port RS-232 umožňuje plně duplexní komunikaci mezi dvěma navzájem propojenými zařízeními. Některé jiné verze sériové komunikace jako například RS-485 umožňují komunikaci mezi větším počtem zařízení. U RS-485 smí v daný okamžik vysílat pouze jedno zařízení, v opačném případě dojde ke kolizi na lince.

Přesto, že je tento komunikační port již v mnoha ohledech překonán a jeho použití v osobních počítačích převzaly novější typy portů jako například USB (Universal Serial Bus), je stále hojně využíván v průmyslu pro svou jednoduchost. Často bývá využitý jako komunikační rozhraní pro nízkoúrovňové aplikace na specifickém hardwaru, rozhraní pro konfiguraci serverů a mnoho dalších účelů.

Sériový port byl původně navržen pro komunikaci s modemem, tomu odpovídá také zapojení jeho konektorů. Pro připojení se standardně používají dva typy konektorů, první z nich má devět pinů a druhý dvacet pět. Ve většině případů se dnes používá pouze první konektor. Jeho podrobné zapojení a význam jednotlivých pinů je uveden na obrázku 8 (Zdroj: autor) a v tabulce 2, která vychází ze zdroje [18]. Hlavní tři vodiče pro komunikaci jsou TxD – vysílaní, RxD – příjem, GND – zem, ostatní vodiče nemusí být zapojeny.



Obr. 8) Konektor RS-232 9pinů

Pin č.	Název	Směr	Popis
1	**CD**	IN	**Carrier Detect** Modem indikuje, že na telefonní lince byl detekován nosný kmitočet.
2	**RXD**	IN	**Receive Data** Přijímaná data.
3	**TXD**	OUT	**Transmit Data** Vysílaná data.
4	**DTR**	OUT	**Data Terminal Ready** Pokud je na tomto pinu logická jednička, terminál oznamuje modemu, že je připraven komunikovat.
5	**GND**		**Ground** Zem.
6	**DSR**	OUT	**Data Set Ready** Pokud je na tomto pinu logická jednička, modem oznamuje terminálu, že je připraven komunikovat.
7	**RTS**	OUT	**Request To Send** Pokud je na tomto pinu logická jednička, modem oznamuje terminálu, že komunikační cesta je volná.
8	**CTS**	OUT	**Clear To Send** Pokud je na tomto pinu logická jednička, terminál oznamuje modemu, že komunikační cesta je volná.
9	**RI**	IN	**Ring Indicator** Logická jednička na tomto pinu znamená, že modem detekoval na lince signál zvonění.

Tab. 2) Popis funkce pinů RS-232

Tato práce se nadále zabývá komunikací pouze s využitím pinů RXD, TXD a Gnd.

4.1.1 Popis komunikace

Komunikace po sériovém portu je asynchronní. Jednosměrný sériový přenos probíhá po jednom páru vodičů. Data jsou přenášena v rámcích, popis struktury rámce je uveden dále v textu. Pro logické hodnoty 0 a 1 se využívá symetrických hodnot napětí, dle použitého zařízení nabývá napětí hodnot ± 5 V, ± 12 V nebo ± 15 V, záporná hodnota napětí standardně reprezentuje logickou úroveň 1 a kladná úroveň 0. Nejrozšířenější variantou je ± 12 V. V době, kdy na lince neprobíhá přenos, je v tak zvaném klidovém stavu (IDLE).

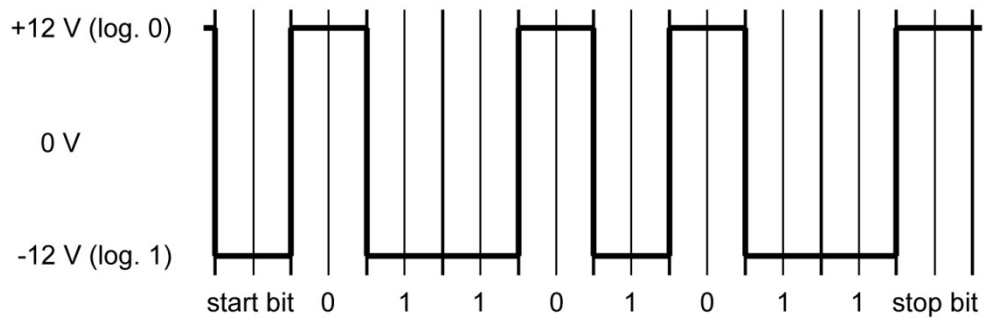
Základní vlastnosti asynchronního přenosu:

- asynchronní přenos nevyužívá k synchronizaci vysílače a přijímače samostatný vodič, synchronizuje se na začátku každé zprávy vysíláním předem stanovené sekvence (start bitu),
- je vhodný pro použití na velké vzdálenosti, kde by použití samostatného synchronního vodiče bylo nákladné,
- nevhodné pro velké objemy dat,
- je nutné předem stanovit přenosovou rychlost a další vlastnosti přenosu,
- má vyšší náklady na komunikační elektroniku, především je nutné použít kvalitní krystalové oscilátory,
- z důvodu nutnosti přidávání startovacích a paritních bitů do zpráv je efektivita přenosu až o 20 % menší oproti synchronnímu přenosu.

Struktura rámce:

- **start bit** – sestupná hrana z log. 1 na log. 0,
- **datové bity** – volitelný počet, posloupnost od nejméně významného bytu po nejvíce významný,
- **paritní bit** – volitelný, podrobně popsán dále v textu,
- **stop bity** – volitelný počet, poskytují čas pro zpracování přijatých dat.

Nejčastěji používaná varianta rámce má celkovou délku 10 bitů, 1 start bit, 8 datových bitů, žádný paritní bit a jeden stop bit. Ukázka takového rámce je na obrázku 9 (Zdroj: autor).



Obr. 9) Struktura rámce RS 232 – přenos hodnoty 01101011

Rámce mohou následovat bezprostředně za sebou. Rychlost přenosu po sériové lince se udává v baudech (Bd) a určuje počet změn stavů za sekundu. Tato rychlost sama o sobě nevypovídá, jak velké množství reálných dat za sekundu linka přenese, k tomu je zapotřebí vzít v potaz ještě strukturu rámce. Máme-li například sériovou linku s rychlostí 57 600 Bd a strukturu rámce, která je popsána v předchozím textu, tedy rámec o celkové délce 10 bitů, z čehož 8 je datových, je reálná přenosová rychlost linky 5 760 bajtů za vteřinu.

4.1.2 Parita

Parita je jednoduchý způsob zabezpečení přenosu dat bez nároků na výpočetní výkon. Princip parity (neboli paritního bitu) spočívá v přidání informace o sudém či lichém počtu logických jedniček v přenášených datech do přenosového rámce. Nevýhodou paritního bitu je schopnost detekovat pouze lichý počet chyb v přenášených datech.

Druhy parity:

- **lichá parita** – má hodnotu `1`, pokud počet jedničkových bitů + paritní bit je liché číslo,
- **sudá parita** – má hodnotu `1`, pokud počet jedničkových bitů + paritní bit je sudé číslo,
- **mark parity** – paritní bit je vždy nastaven na `1`,
- **space parity** – paritní bit je vždy nastaven na `0`, používá se při komunikaci ze sedmibitových zařízení na osmi bitové.

V tabulce 3 (Zdroj: autor) je uvedena ukázka výpočtu paritního bitu pro sudou a lichou paritu.

8bitová data	**Sudá parita**	**Lichá parita**
00000000	**0**00000000	**1**00000000
11101010	**1**11101010	**0**11101010
10101010	**0**10101010	**1**10101010
11111111	**1**11111111	**0**11111111

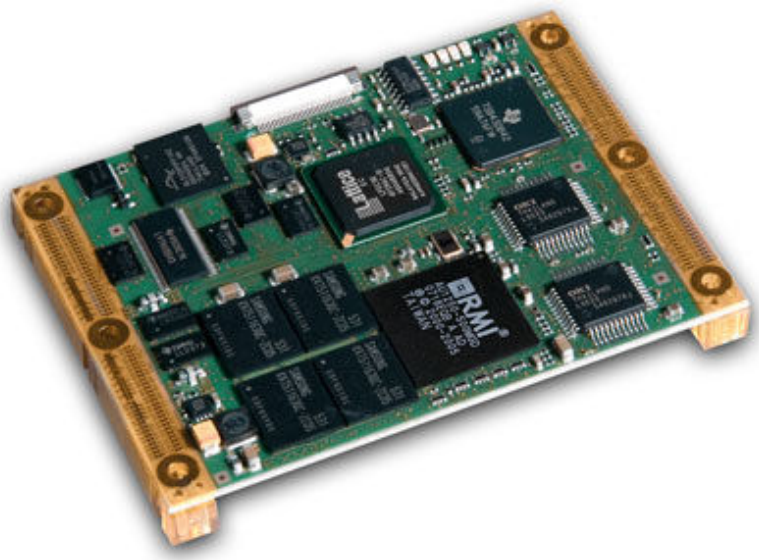
Tab. 3) Příklad použití paritního bitu

4.2 Embedded modul

Pro vytvoření komunikačního zařízení MTR10 bylo nutné zvolit vhodný procesorový modul, který je srdcem celého zařízení.

Původně byl zvolen modul HITACHI SH7750 (série SH-4), který je osazen mikroprocesorem s RISC instrukční sadou a disponuje 64 MB operační paměti. Tento modul byl používán během několika prvních měsíců vývoje. Když kolega Bc. Pavel Pavlík začal vyvíjet IP telefonii pro zařízení MTR10 zjistil, že tento modul není dostatečně výkonný pro dosažení potřebného maximálního tolerovaného zpoždění při komunikaci. Proto byl pro další vývoj zvolen výkonnější modul EXM32-Au1250.

4.2.1 Popis modulu EXM32-Au1250



Obr. 10) Modul EXM32-Au1250 [zdroj 6]

Modul EXM32-Au1250 vyrábí a dodává společnost MSC. Jedná se o robustní, kompaktní modul, který je napojen na základní desku odolným elastomerovým konektorem. Ten díky své konstrukci umožňuje napojení dalších rozšiřujících modulů jednoduchým přidáním na modul CPU.

Tento modul odpovídá standardu EXM32 SOM (system-on-module), který je definovaný společností MSC pro použití v průmyslových systémech. Moduly tohoto standardu nabízejí vysoký výpočetní výkon a jsou určeny pro velký teplotní rozsah

odpovídající jejich využití. Modul je také vhodný pro využití v mediálních systémech díky integrované hardwarové podpoře kodeků MPEG1/2/4, WMV a DIVX.

Vybavení modulu EXM32:

- Au1250 RISC 500 MHz CPU s 32kB Cache,
- 256 MB DDR2 SDRAM,
- 128 MB NOR Flash a víc,
- 512 MB NAND Flash a víc,
- SD Card, CompactFlash a IDE podpora,
- 2 x UART,
- MPEG 1/2/4, H.263, WMV a DivX Video Decoder,
- 10/100 BaseT Ethernet interface,
- USB 2.0 HighSpeed interface,
- Pracovní teploty -40°C až +85°C.

4.3 Gentoo Linux

Vzhledem k licenčním podmínkám Microsoft Windows pro embedded systémy byl jako OS pro zařízení MTR10 zvolen Linux. Distribuce Gentoo byla vybrána na základě doporučení výrobce modulu EXM32.

Jedná se o distribuci, která je stejně jako například Debian vyvíjena komunitou. Na rozdíl od většiny distribucí Linuxu, které jsou založeny na předkompilovaných balíčcích, je Gentoo založena na zdrojových kódech. Díky tomu si každý uživatel může přeložit svůj vlastní systém, který odpovídá přesně jeho potřebám a je plně optimalizován pro daný hardware. Gentoo dosahuje velkého výkonu a stability.

Nevýhodou tohoto přístupu je vysoká náročnost na výpočetní výkon při kompilaci zdrojových kódů. Pokud srovnáme Gentoo s některou z distribucí založených na balíčcích (například Mandriva), zabere instalace Linuxu Gentoo mnohonásobně větší čas. Některé programy jsou proto pro Gentoo dostupné již v předkompilované verzi, například balíček nástrojů OpenOffice, jehož kompilace by byla extrémně náročná na výpočetní výkon a nepřinesla by téměř žádné zvýšení výkonu. V této části práce jsou použité zdroje [6], [7].

4.3.1 Systémový správce balíčků Portage

Systémový správce balíčků, který využívá Gentoo Linux se jmenuje Portage je podobný správci balíčků v BSD, jež se nazývá FreeBSD Ports. Portage byl vyvinut v programovacím jazyce Python s přihlédnutím právě k již zmiňovanému FreeBSD Ports. Portage obstarává mnoho klíčových úloh, jako například instalaci nového softwaru, aktualizace atd. Přesto že se tento systém jmenuje Portage, ovládá se příkazem `emerge`.

4.3.2 Příkaz emerge

Příkaz `emerge` se používá velice jednoduše a jeho ovládání je intuitivní. Pro potřeby této práce je zde uvedeno několik základních použití příkazu `emerge`:

- `emerege --search jméno-aplikace -` prohledá Portage a vypíše všechny dostupné balíčky zdrojových kódů, jejichž názvy obsahují námi hledané `<jméno-aplikace>`,
- `emerge --pretend jméno-aplikace`, nebo `emerge -p jméno-aplikace` – těmito dvěma zápisy lze vypsat seznam balíčků, které jsou nezbytné pro instalaci aplikace,
- `emerge jméno-aplikace` – program emerge nainstaluje danou aplikaci,
- `emerge --sync` – tento příkaz synchronizuje lokální databázi Portage s aktuální databází z internetu.

4.3.3 USE Flag

Při instalaci aplikací pomocí příkazu emerge můžeme narazit na komplikace. Některé aplikace je nutné kompilovat s podporou volitelných součástí a k tomuto účelu slouží USE Flag.

Pokud nastavíme například parametr USE="alsa", budou všechny aplikace, kde je tato podpora nastavena jako volitelná, kompilovány s podporou pro ALSA (Advanced Linux Sound Architecture). Nechceme-li, nebo nám nejde přeložit některá námi požadovaná aplikace právě kvůli volitelně zapnuté podpoře (například pro již

zmiňovanou ALSU), můžeme použít parametr USE="-alsa". V takovém případě se aplikace bude kompilovat bez podpory pro ALSA.

Existují dva druhy USE proměnných:

- globální proměnné USE – platí pro všechny instalované aplikace,
- lokální proměnné USE – pokud potřebujeme nainstalovat pouze konkrétní aplikaci s podporou pro danou funkci, nebo naopak pro danou aplikaci podporu funkce vypnout, použijeme lokální proměnou USE.

4.3.4 Gentoo Linux pro embedded systém EXM32

Vývoj aplikací pro moduly jako je EXM32 se obvykle neprovádí přímo na těchto systémech, protože neumožňují použití kvalitních vývojových nástrojů a jejich výkonové možnosti jsou omezené. Samozřejmě na embedded systémech nelze spustit aplikace přeložené pro standardní PC. Důvodem je odlišná architektura CPU. Procesory odlišných architektur používají jiné základní instrukce. Aby bylo možné na běžném PC provádět vývoj aplikací a překlad kernelu pro embedded systémy, je nutné mít nainstalován příslušný takzvaný toolchain. Ten obsahuje překladač, asembler, linker a knihovny pro příslušnou architekturu. Vytvořit toolchain pro distribuci Gentoo je možné pomocí emerge, ale je to velice časově náročné.

Postup tvorby toolchainu lze shrnout do následujících kroků:

1. Nejprve je nutné nainstalovat skript crossdev pro tvorbu toolchainu.
   ```
   emerge --sync
   emerge crossdev
   ```

2. Po instalaci tohoto skriptu se vytvoří kompletní toolchain příkazem:
   ```
   USE="nptl" crossdev -b -v -t mipsel --b 2.17 --g 4.1.2 --l 2.5
   ```

   Ten vytvoří toolchain skládající se z binutils-2.17, gcc-4.1.2, glibc-2.5. Také vytvoří předkompilované balíčky, které lze na jiný systém instalovat příkazem:
   ```
   PKGDIR=/usr/portage/packages/cross/mipsel-softfloat-linux-gnu/
   emerge -K cross-mipsel-softfloat-linux-gnu/binutils cross-
   mipsel-softfloat-linux-gnu/gcc cross-mipsel-softfloat-linux-
   gnu/mips-headers cross-mipsel-softfloat-linux-gnu/glibc
   ```

Výsledkem těchto příkazů může být např. toolchain s názvy součástí začínajících `mipsel-softfloat-linux-gnu`.

4.4 Vlákna

Tato část práce se zabývá problematikou vláken a byly v ní použity zdroje [2], [3], [4], [9], [10]. Proces se skládá z kódu programu, dat programu, informaci o otevřených souborech, zásobníku a informací o zpracování signálů. Všechny tyto prvky má každý proces privátní a nemůže je sdílet (mimo datových oblastí) s jinými procesy.

Vlákno je vlastně odlehčený proces, který má privátní pouze zásobník a stav, ostatní části jsou sdílené s dalšími vlákny procesu. Vlákno samo o sobě nemůže existovat bez procesu. V OS Linux se používá model vláken `one-to-one`, jedná se o implementaci vláken na úrovni jádra. Pro plánovač se každé vlákno tváří jako samostatný proces a plánovač při přepínání nedělá rozdíly mezi vlákny a procesy. Všeobecně má tento model nevýhodu velké režie při přepínání vláken jako procesů avšak OS Linux má přepínání procesů velice efektivní, a proto jej může využívat.

4.4.1 Vytvoření vlákna

```
int pthread_create(pthread_t* vlakno, pthread_attr_t * attr,
      void* (*funkce_vlakna)(void*), void* argument);
```

Tato funkce vytvoří nové vlákno s atributy specifikovanými parametrem `attr`, parametrem `attr` může být hodnota `NULL`, v takovém případě se použijí implicitní parametry pro vytvoření vlákna. Vlákno bude vykonávat kód `funkce_vlakna()`. Funkci, kterou má vlákno vykonávat, lze předat jeden argument typu `void*`. Návratová hodnota funkce je typu `int` a vrací hodnotu `0`, pokud vytvoření vlákna proběhlo v pořádku, jinak vrátí kód chyby. Funkce se volá v rodičovském vláknu (procesu).

4.4.2 Ukončení vlákna

```
void pthread_exit(void *retval)
```

Vlákno lze ukončit dvěma způsoby, prvním z nich je dokončení kódu `funkce_vlakna()` a druhým je ukončení zavoláním funkce `pthread_exit()`. Tato funkce ukončí vlákno, ve kterém byla zavolána, prostřednictvím parametru `retval` můžeme předat hodnotu rodičovskému vláknu.

4.4.3 Připojení vlákna

```
int pthread_join(pthread_t th, void **thread_return);
```

Po ukončení vlákna funkcí `pthread_exit()`, musíme v rodičovském vlákně zavolat funkci pro připojení vlákna. Pokud by pro naše použití bylo připojování vlákna nežádoucí, můžeme mu při vytváření nastavit atribut „detachable (odpojitelný)“. V takovém případě není nutné volat funkci `pthread_join()`.

Ukázka použití vlákna:

```
//Deklarace funkce vlákna
static void *thread_function(void *args){
    //...
    //Tělo funkce vlákna
    //...
    pthread_exit(0);
}

int main() {
    //Deklarace vlákna
    pthread_t m_ptread;
    //Vytvoření vlákna
    pthread_create(&m_ptread,0,thread_function,0);
    //...
    //Tělo funkce
    //...
    //Připojení vlákna
    pthread_join(m_ptread,0 );
    return 0;
}
```

4.5 Mutex (MUTual EXclusion)

Při použití vláken je nutné zamezit současný přístup k sdíleným proměnným a objektům z více vláken. K tomuto účelu slouží kritické sekce. Kritickou sekci lze řešit pomocí semaforů a mutexů. Semafory umožňují přístup do kritické sekce i více vláknům. Tato práce využívá mutexů.

Mutex má dva stavy `locked` (některé vlákno je právě v kritické sekci a ostatní vlákna do ní nemohou vstoupit) a `unlocked` (v kritické sekci se právě nenachází žádné vlákno).

4.5.1 Inicializace mutexu

```
int pthread_mutex_init(pthread_mutex_t *mutex,
      const  pthread_mutexattr_t *mutexattr);
```

Výše uvedenou funkcí se inicializuje `mutex`, jako parametr mutexattr lze zadat hodnotu `NULL`. Inicializace mutexu je nutná, bez ní není možné s mutexem korektně pracovat.

4.5.2 Uzamčení mutexu

```
int pthread_mutex_lock(pthread_mutex_t *mutex);
```

Tato funkce uzamkne mutex, který je zadaný jako parametr `mutex`. Po provedení této funkce je mutex uzamčen pro vlákno, ze kterého byla zavolána. Pokud se nepodařilo vstoupit do kritické sekce, protože je mutex již uzamčen jiným vláknem, pozastaví se běh vlákna do doby, než předchozí vlákno opustí kritickou sekci.

4.5.3 Odemčení mutexu

```
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

Tato funkce odemkne mutex, který je zadán jako parametr `mutex` a umožní tak vstup dalšímu čekajícímu vláknu. Odemčení mutexu je nutné, aby ostatní vlákna nečekala do nekonečna.

4.5.4 Pokus o uzamčení mutexu

```
int pthread_mutex_trylock(pthread_mutex_t *mutex);
```

Pokud chceme v případě, že se nám nepodařilo vstoupit do kritické sekce, provádět jinou činnost, můžeme použít tuto funkci. V případě, že se nepodařilo vstoupit do kritické sekce, tato funkce nepozastaví vlákno až do doby kdy je vstup možný, ale vrátí se s chybou `EBUSY`.

4.5.5 Ukončení mutexu

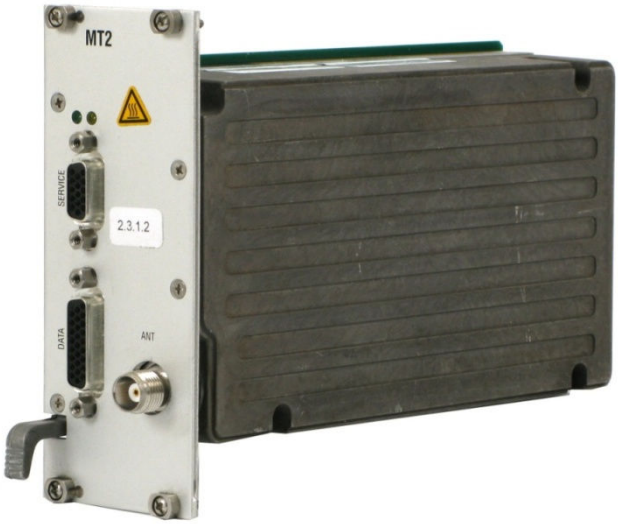
```
int pthread_mutex_destroy(pthread_mutex_t *mutex);
```

Volání této funkce uvolní veškeré zdroje, které využívá mutex zadaný parametrem `mutex`.

Ukázka použití mutexu je uvedena v příloze C.

4.6 Modem MT2

Modem MT2 je výrobkem rakouské společnosti Kapsch, s. r. o. Modul modemu MT2 pracuje ve frekvenčních rozsazích 876-915 / 921-960 MHz, z čehož vyplývá, že je určen jak pro komunikaci v síti GSM, tak i v GSM-R. V síti GSM-R podporuje modem MT2 všechny rozšířené hlasové funkce, které jsou popsány v kapitole 2. Maximální vysílací výkon modemu je 8 W. V této části práce je použit zdroj [5].



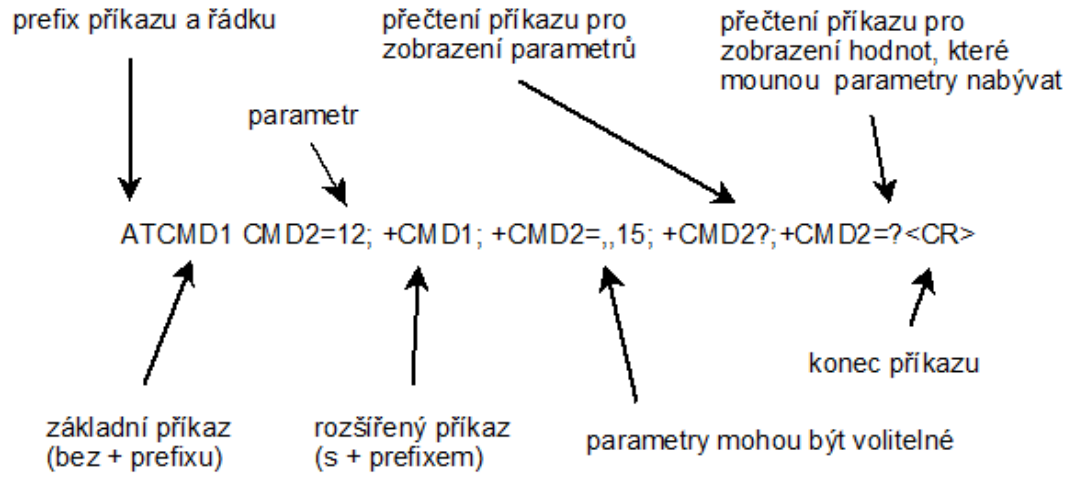
Obr. 11) Modem GSM-R Kapsch MT2 [Zdroj: RADOM]

Modul má tři rozhraní:

- uživatelské rozhraní v přední části modemu,
- servisní rozhraní v přední části modemu,
- TDMA rozhraní v backplane sběrnici v zadní části modemu,

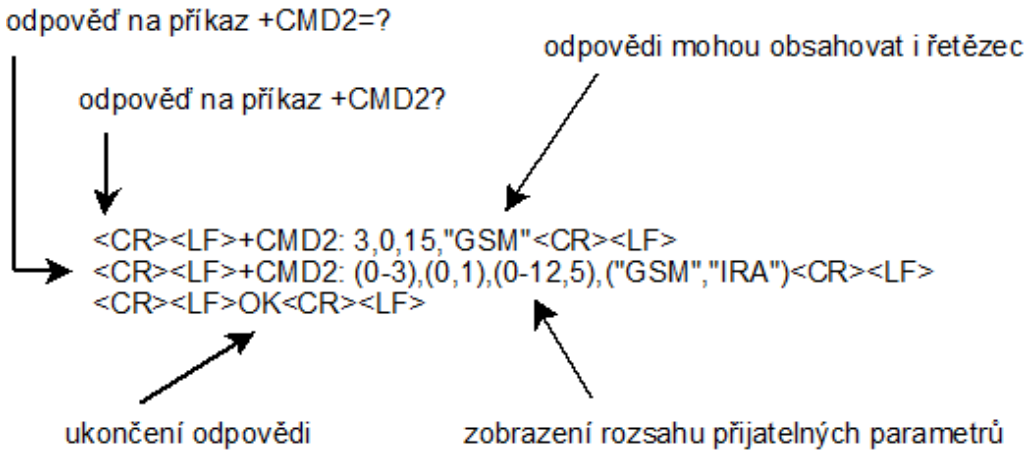
- v přední části modemu je také umístěn konektor pro anténu.

K ovládání modemu slouží uživatelské a backplane rozhraní. Servisním rozhraním nelze ovládat modem, používá se ke konfiguraci a aktualizaci softwaru. Modem se ovládá pomocí AT-příkazů. Komunikace s modemem probíhá kódování ASCII.



Obr. 12) Syntaxe AT-příkazu [Zdroj: 5]

Od modemu přicházejí dva druhy zpráv, vyžádané a nevyžádané. Vyžádané zprávy jsou odpovědi na příkazy, zaslané modemu. Nevyžádané zprávy modem posílá při změnách v síti, o kterých je nezbytné informovat obsluhující program, příkladem nevyžádané zprávy je `+CRING`, který informuje o příchozím hovoru.



Obr. 13) Struktura odpovědi na AT-příkaz [Zdroj: 5]

Odpověď na AT-příkaz, tedy vyžádaná zpráva se skládá z několika částí v závislosti na jejím typu. Jednotlivé části odpovědi jsou odděleny znaky nového řádku `<CR><LF>`, každá zpráva je zakončena jedním z finálních řetězců (například `OK`, `ERROR`, `BUSY`, `CONNECT` atd.). Dle typu příkazu může odpověď před finálním řetězcem

také obsahovat vlastní informace oddělené řádky, každý takový řádek začíná informací o příkazu, ke kterému odpověď náleží ve formátu `+příkaz` (například `+CLIP` atd.).

4.6.1 Inicializace modemu MT2

Komunikace s modemem MT2 po sériovém portu probíhá rychlostí 19 200 Bd, přenos využívá osm datových bitů, žádný paritní bit a jeden stop bit.

Před použitím modemu je nutné nastavit ještě další parametry ovlivňující chování modemu a přihlášení do sítě GSM-R.

Prvním z příkazů, které je nutné použít při inicializaci modemu je `ATE<mode>`. Tento příkaz nastavuje chování modemu po příjmu AT-příkazu, pokud je parametr `<mode> 0`, modem neodesílá zpět příkazy, které přijal. Nastavením parametru na `1` se zapne zpětné odesílání přijatých příkazu (echo).

Dalším příkazem je `AT+CKPD=<keys>,[<time>]`. Tento příkaz emuluje stisknutí kláves a používá se pro zapínání a vypínání modemu. Parametr `<keys>` má hodnotu `P/p` (Power), pokud je hodnota `<time>` menší než `1000` modem se zapne, v opačném případě se vypne. Defaultní hodnota parametru `<time>` je nula.

Příkaz `AT+CPIN?` kontroluje, je-li vyžadován kód PIN. Vrací `+CPIN: READY` pokud kód pin není vyžadován.

Dále lze nastavit možnost zobrazování chybových hlášení pomocí čísel nebo řetězců. K tomu slouží příkaz `AT+CMEE=<mode>`, pokud je parametr `<mode>` nastaven na `1` jsou chybové hlášení zobrazováni jako čísla chyb, v případě, že je nastaven na `2`, jsou zobrazovány jako celá chybová hlášení.

Pro registraci do sítě slouží příkaz `AT+CREG=<n>`. Protože registrace může trvat dlouhou dobu, je návratová hodnota příkazu `OK`. Po provedení registrace či deregistrace přijde nevyžádaná zpráva ve formátu `+CREG: <n>,<stat>`. Díky tomu lze během registrace provádět další příkazy. Pro zjištění aktuálního stavu registrace lze použít zápis `AT+CREG?`, odpověď má stejný formát jako nevyžádaná zpráva o stavu registrace.

- `<n>` – registrace zapnutá nebo vypnutá:
  - `0` – vypnutá,
  - `1` – zapnutá,
- `<stat>` – stav registrace:
  - `0` – neregistrováno,
  - `1` – registrováno,
  - `2` – neregistrováno modem vyhledává operátora,
  - `3` – registrace byla zamítnuta,
  - `4` – neznámý stav,
  - `5` – registrováno s aktivním roamingem.

Příkaz `AT+CSQ` vrací sílu signálů, odpověď má formát `+CSQ: <rssi>`, hodnoty `<rssi>` jsou následující:

- `0` – -110 dBm nebo méně,
- `1` – od -110 dBm do -109 dBm,
- `2 až 30` – od -109 dBm do -49 dBm,
- `31` – -48 dBm nebo více,
- `99` – neznámá úroveň signálu.

4.6.2 Volání

Pro volání slouží příkaz `ATD` a to nejen pro volání přímého telefonního čísla, tento příkaz se využívá také v souvislosti s ostatními typy hovorů. Příkaz `ATD` má několik možných způsobu zápisu dle typu využití.

Plnohodnotný zápis `ATD*<services><priority>#<number>;` se používá pro běžné hovory s využitím priority hovoru, skupinové hovory a jednosměrné skupinové hovory (příklad: `ATD*754#+420123456;`).

- `<services>` – určuje o jaký typ hovoru se jedná:
  - `75` – běžný hovor,
  - `17` – skupinový hovor,
  - `18` – jednosměrný skupinový hovor,

- <priority> – určuje prioritu hovoru 0 až 4, implicitní hodnota je 4 (nejnižší),
- <number> – telefonní číslo, které chceme volat, v případě typu 17 a 18 se jedná o číslo skupiny,
- ; – určuje, že se jedná o hlasový hovor, pokud není součástí příkazu, jedná se o datový hovor.

Zkrácený zápis ATD<number>; se používá pro běžné hovory s implicitní prioritou. Pro volání čísla z telefonního seznamu SIM-karty se požívá zápis ATD><string>; nebo ATD><n>; parametr <string> udává jméno, pod kterým je v seznamu číslo uloženo, parametr <n> je pozice záznamu v telefonním seznamu.

V době od odeslání příkazu pro vytočení hovoru ATD a příjmu finální zprávy o spojení či nespojení, nelze odeslat modemu jakýkoli jiný příkaz, ten by totiž způsobil ukončení pokusu o vytvoření spojení.

4.6.3 Příchozí hovory

Příchozí hovor je indikován nevyžádanou zprávou +RING nebo +CRING, což je rozšířený +RING o další podrobnosti. Jaká nevyžádaná zpráva má být modemem odeslána lze nastavit pomocí příkazu AT+CRC=<mode> (mód 0 pro +RING a mód 1 pro +CRING).

Formát +CRING: VOICE,[<priority>,[<subaddr>,<satype>]][1] (například +CRING: VOICE,"01",128) modem používá pro informování o běžném příchozím hovoru.

- <priority> – priorita příchozího hovoru,
- <subaddr> – určuje funkční identitu na kterou je hovor směrován, problematika funkčních identit je popsána dále v textu, parametr je volitelný,
- <satype> – typ volaného čísla:
    - 128 – číslo je zadáno bez národní identifikace,

[1] Hranaté závorky v zápisu AT příkazu označují volitelný parametr.

    - 145 – číslo je zadáno s národní identifikací (například +420).

Pro skupinové hovory a jednosměrné skupinové hovory se používá formát +CRING: <type> <GCA>,<GId>,<ackflag>,[<priority>] (například pro skupinové hovory +CRING: VGC 99999,178,0,2 a pro jednosměrné skupinové hovory +CRING: VBC 99999,178,0,3).

- <type> – typ hovoru:
    - VGC – skupinový hovor (viz 2.1.2),
    - VBC – jednosměrný skupinový hovor (viz 2.1.3),
- <GCA> – identifikátor lokality volání,
- <GId> – identifikátor skupiny,
- <ackflag> – informuje, zda je po ukončení hovoru vyžadována potvrzovací procedura,
- [<priority>] – priorita hovoru.

Pro příjem běžného hovoru slouží příkaz ATA, přijmutí hovoru příkaz indikuje návratem odpovědi OK.

Pro příjem skupinového a jednosměrného skupinového hovoru slouží příkaz AT+CAJOIN=<services>,<GId>,<GCA> (například AT+CAJOIN=18,178,99999).

- <services> – určuje, o jaký typ hovoru se jedná:
    - 17 – skupinový hovor,
    - 18 – jednosměrný skupinový hovor,
- <GCA> – identifikátor lokality volání,
- <GId> – identifikátor skupiny.

4.6.4 Ukončení hovorů

Pro ukončení příchozího i odchozího běžného hovoru slouží příkaz AT+CHUP, tento příkaz ukončí právě aktivní hovor. Další možností jak ukončit hovor je příkaz ATH, ten ukončí všechny hovory.

4.6.5 Podrobné informace o hovorech

Pro přehlednější zobrazení probíhajících příchozích i odchozích hovorů lze využít příkaz `AT+CLCC` pro běžné hovory a `AT+CALCC` pro skupinové a jednosměrné skupinové hovory. Tyto příkazy vracejí přehlednou tabulku o všech probíhajících hovorech.

Příkaz pro tabulkový výpis běžných hovorů `AT+CLCC` vrací odpověď ve formátu:

```
+CLCC:
<id1>,<dir>,<stat>,<mode>,<mpty>[,<number>,<type>,[<alpha>[,<priorit
y>]]][<CR><LF>
+CLCC:
<id2>,<dir>,<stat>,<mode>,<mpty>[,<number>,<type>,[<alpha>[,<priorit
y>]]][...]]
<CR><LF>OK<CR><LF>
```

- `<idn>` – číslo hovoru,
- `<dir>` – směr hovoru:
  - `0` – odchozí hovor (MO),
  - `1` – příchozí hovor (MT),
- `<stat>` – stav hovoru:
  - `0` – aktivní,
  - `1` – odložený,
  - `2` – vytáčí se (MO),
  - `3` – vyzvání (MO),
  - `4` – příchozí (MT),
  - `5` – čekající (MT),
- `<mode>` – mód hovoru:
  - `0` – hlasový,
  - `1` – datový,
  - `3` – fax,
- `<mpty>` – hovor je součástí konference:
  - `0` – není,
  - `1` – je,
- `<number>` – telefonní číslo,

- <type> – typ zápisu telefonního čísla,
- <priority> – priorita hovoru.

Příklad odpovědi: `+CLCC: 1,1,0,0,0,"+420959800182",145,,4`

Příkaz pro tabulkový výpis skupinových a jednosměrných skupinových hovorů má zápis `AT+CALCC=<mode>`.

- <mode> – jaké skupinové a jednosměrné skupinové hovory chceme zobrazit:
  - 0 – všechny hovory pro danou oblast <GId>, které jsou povelené na SIM-kartě,
  - 1 – pouze aktivní hovory.

Formát odpovědi:

```
+CALCC:
<GId>,<GCA>,<service>,<stat>,<dir>,<ack_flag>[,<priority>][<CR><LF>
+CALCC: <GId>,<GCA>,<service>,<stat>,<dir>,<ack_flag>[,<priority>]
[...]]
<CR><LF>OK<CR><LF>
```

- <GId> – identifikátor skupiny,
- <GCA> – identifikátor lokality volání,
- <services> – určuje typ hovoru:
  - 17 – skupinový hovor,
  - 18 – jednosměrný skupinový hovor,
- <stat> – stav hovoru:
  - 0 – aktivní,
  - 1 – odložený,
  - 2 – příchozí, zatím nepřijatý,
- <dir> – směr hovoru:
  - 0 – odchozí hovor (MO),
  - 1 – příchozí hovor (MT),
- <ackflag> – informuje, zda je po ukončení hovoru vyžadována potvrzovací procedura,
- <priority> – priorita hovoru.

Příklad odpovědi: `+CALCC: 178,99999,17,2,1,0,2`

4.6.6 Telefonní seznam

Před samotným vyčítáním seznamu ze SIM-karty je třeba zvolit, který ze seznamů chceme vyčítat. K tomu slouží příkaz `AT+CPBS=<storage>[,<passwd>]` (Například `AT+CPBS="SM".`)

- `<storage>` – volba seznamu:
  - `"SM"` – telefonní seznam (SIM),
  - `"SN"` – telefonní seznam servisních čísel (SIM),
  - `"ON"` – vlastní telefonní seznam (MT2),
  - `"LD"` – seznam posledních volaných čísel (SIM),
  - `"EC"` – kódy nouzových hovorů (SIM),
  - `"FC"` – seznam fixních čísel (SIM),
- `<passwd>` – heslo pro přístup do paměti "FC".

Po zvolení seznamu všechny ostatní příkazy pro práci se seznamem obsluhují seznam, který byl zvolen tímto příkazem.

Po zvolení seznamu SIM-karty je třeba vyčíst kapacitu tohoto seznamu příkazem `AT+CPBS?`, odpověď na tento příkaz má formát `+CPBS: <storage>[,<used>,<total>]`.

- `<used>` – počet použitých záznamů,
- `<total>` – celkový počet záznamů.

Pro samotné vyčtení seznamu ze SIM-karty slouží příkaz `AT+CPBR=<index1>[,<index2>]` (například `+CPBR=1,100`).

- `<index1>` – počáteční index, od kterého chceme seznam vyčítat, pokud není zadaný parametr `<index2>`, příkaz vrátí položku seznamu na pozici zadané tímto parametrem,
- `<index2>` – koncový index, po který se má seznam vyčíst, pro výčet celého seznamu využijeme znalost o celkové kapacitě seznamu z odpovědi na příkaz `+CPBS?`, vzhledem k tomu že na SIM-kartě je seznam uchová-

ván nesetřepaný (některé pozice mohou být prázdné), nelze využít položku <used> jako parametr <index2>.

Příkaz pro čtení vrací položky seznamu ve formátu:

```
[+CPBR: <index1>,<number>,<type>,<text>
[<CR><LF>+CPBR: <index2>,<number>,<type>,<text>[<CR><LF>[...]]]
<CR><LF>OK<CR><LF>
```

- <indexn> – pozice v seznamu,
- <number> – telefonní číslo,
- <type> – typ zápisu telefonního čísla,
- <text> – text záznamu.

Pro vyhledávání přímo v paměti SIM-karty lze využít příkaz AT+CPBF=<findtext>, kde parametr <findtext> udává hledaný řetězec. Formát odpovědi je stejný jako u příkazu pro čtení seznamu. Tato práce tento příkaz nevyužívá, pro vyhledávání se využije struktura v paměti programu, která obsahuje celý seznam vyčtený ze SIM-karty.

Pro zápis do seznamu SIM-karty zvoleného příkazem AT+CPBS slouží příkaz AT+CPBW=[<index>][,<number>,[<type>],<text>].

- <index> – pozice na kterou má být záznam zapsán, pokud tento parametr není zadán, modem by měl záznam uložit na první volnou pozici, ale ve verzi softwaru modemu MT2 Rel_2.1.2.0 tento zápis příkazu není funkční a je nutné nejprve vyhledat volnou pozici ručně; pokud je jako parametr zadán pouze index, modem smaže záznam na zadané pozici ze seznamu SIM-karty,
- <number> – telefonní číslo, které má být zapsáno,
- <type> – typ zápisu telefonního čísla,
- <text> – text záznamu.

V rámci projektu MTR10 a této diplomové práce není zápis do seznamů SIM-karty řešen. Zápisy do SIM-karty lze provádět prostřednictvím sítě GSM-R, díky tomu není nutný přístup přímo k zařízení. Tato vlastnost usnadní centrální zprávu čísel a možnost přepisu seznamu zařízením MTR10 je tedy nežádoucí.

4.6.7 Funkční čísla

Pro zprávu funkčních čísel slouží příkaz `AT+CUSD=[<n>[,<str>[,<dcs>]]]`. Tento příkaz vrací odpověď `OK` nebo `ERROR`. O výsledku prováděného požadavku informuje nevyžádaná zpráva ve formátu `+CUSD: <m>[,<strOut>,<dcs>]`.

- `<n>`:
  - `0` – vypnutí návratových kódů příkazu,
  - `1` – zapnutí návratových kódů příkazu,
- `<m>`:
  - `0` – jedná se o informaci sítě bez předchozího vyžádání,
  - `1` – jedná se o odpověď na příkaz,
  - `2` – funkce funkčních čísel byla ukončena sítí,
- `<str>` – tento parametr určuje jakou akci má síť provést:
  - `"*#214*<funkční číslo>***#"` – dotaz na aktuální stav funkčního čísla, výsledek operace určuje parametr `<strOut>`:
    - `"03"` – funkční číslo je již registrováno, odpověď obsahuje také informaci o tom kým,
    - `"62"` – funkční číslo je volné,
  - `"**214*<funkční číslo>***#"` – pokus o registraci funkčního čísla, výsledek operace určuje parametr `<strOut>`:
    - `"01"` – funkční číslo bylo úspěšně registrováno,
    - `"41"` – chybný formát čísla,
    - `"61"` – funkční číslo je již registrováno jiným účastníkem,
    - `"98"` – blíže nespecifikovaná chyba,
  - `"##214*<funkční číslo>***#"` – deregistrace funkčního čísla, tuto funkci lze použít pouze na deregistraci námi používaného funkční čísla, v případě že jiný účastník řádně neuvolnil své funkční číslo a my jej potřebujeme, lze ho odhlásit takzvanou nucenou deregistrací, která je popsána dále, výsledek operace určuje parametr `<strOut>`:
    - `"02"` – funkční číslo bylo deregistrováno,

- "##214*<funkční číslo>*88*<MSISDN>*#" – nucená deregistrace funkčního čísla, je nutné uvést aktuálního vlastníka čísla, k jeho zjištění lze použít dotaz na aktuální stav čísla, zjištěný majitel se zadá jako parametr <MSISDN>, výsledek operace určuje parametr <strOut>:
  - "02" – nucené odhlášení proběhlo v pořádku,
  - "62" – funkční číslo je volné,
  - "63" – chyba při nuceném odhlašování.

4.6.8 Přímý přístup do paměti SIM-karty

Některé informace jako například seznam povolených operátorů, seznam všech operátorů s příslušnými národními předvolbami a dalšími informacemi, které jsou potřebné pro přihlášení do sítě, je nutné vyčíst z adresářové struktury SIM- karty. Pro přímý přístup do paměti SIM-karty slouží příkaz AT+CRSM, který umožňuje zápis i čtení.

AT+CRSM=<command>[,<fileid>[,<P1>,<P2>,<P3>[,<data>,<dirid>]]]

- <command> – příkaz který má být vykonán:
  - 176 – čtení binárních dat,
  - 178 – čtení záznamu,
  - 192 – získání informací o vybraném adresáři, slouží také pro změnu adresáře,
  - 214 – úprava binárních dat,
  - 220 – úprava záznamu,
  - 242 – získání informací o aktuálním adresáři,
- <fileid> – určení souboru (adresáře), pro který má být příkaz vykonán, je mandatorní pro všechny příkazy s výjimkou příkazu 242,
- <P1>,<P2>,<P3> – parametry, které se používají pro vyčítání velkého množství dat naráz, řádek AT-příkazu může mít maximálně 200 znaků (včetně hlavičky), pokud potřebujeme vyčíst či zapsat větší data musíme je rozdělit na menší části pomocí těchto parametrů, implicitně je není nutné používat,
- <data> – přenášená data,

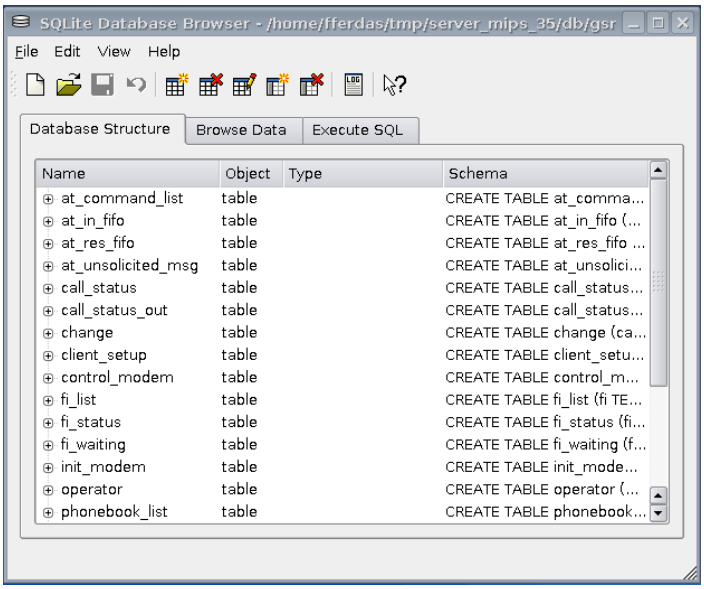
- `<dirid>` – v některých případech není možné adresář zadat jako `<fileid>` a musí být zadán jako tento parametr, toto je způsobeno vnitřní posloupností příkazů.

4.7 SQLite

SQLite je souborově orientovaný databázový nástroj, který je vytvořen s ohledem na minimalizaci nároků na výkon hardwaru. Nevyužívá klasické architektury klient-server, takže hardware nezatěžuje žádný trvale běžící démon. SQLite je malá knihovna napsána v jazyce C. Každá databáze je uložena v samostatném souboru s příponou `dbm`, v souboru se data ukládají do stejně velkých bloků s využitím primárního klíče. Při vyhledávání SQLite využívá hashovacích technik pro rychlé vyhledávání dle klíče. Implementuje většinu standardu SQL92.

Je určena pro využití v aplikacích, které nevyžadují uchovávání velkého množství dat, ale přesto chtějí využít přehlednosti a jednoduchosti SQL. SQLite hojně využívají například webové prohlížeče, softwary pro zálohování a mnoho dalších aplikací.

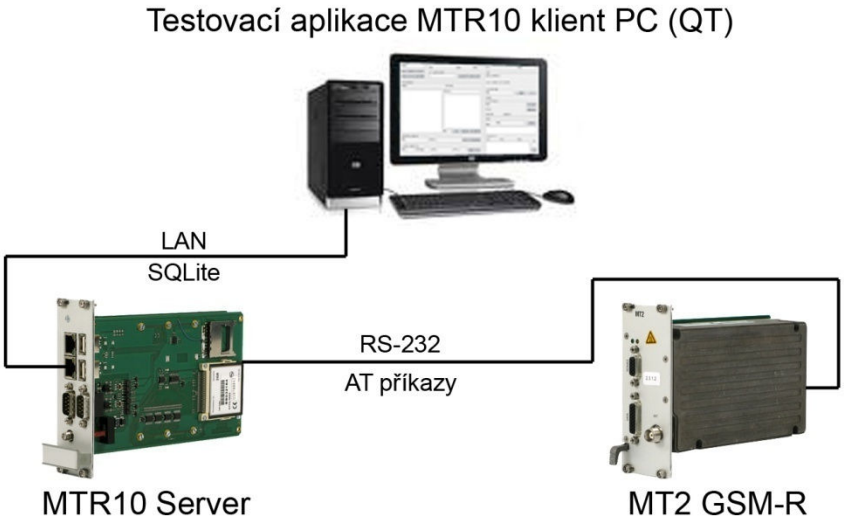
Pro práci s databází SQLite v operačním systému Linux lze využít konzolový nástroj sqlite3 nebo grafickou nadstavbu SQLiteBrowser.



Obr. 14) SQLiteBrowser [Zdroj: autor]

5. ANALÝZA OVLÁDACÍHO SOFTWARU MODEMU MT2

V rámci této diplomové práce má být dle zadání implementován ovládací software pro modem MT2, který má architekturu klient-server. Serverová část je napsána v jazyce C++ a přeložena pro EXM32. Klientská část je také napsána v jazyce C++, pro grafické rozhraní byl zvolen Framework QT od společnosti Nokia (dříve Trolltech). Serverová a klientská část je propojena prostřednictvím LAN. K serverové části je prostřednictvím sériového portu RS-232 připojen modem MT2, komunikace s modemem se provádí pomocí AT-příkazů.



Obr. 15) Schéma zapojení MTR10 [Zdroj: autor]

5.1 Požadavky na systém

Systém má umožňovat ovládání modemu MT2 v síti GSM-R prostřednictvím sítě LAN. Musí umožňovat ovládání modemu na malé síti, která není veřejná a její součástí jsou moduly systému MTR10, díky tomu nehrozí velká ztrátovost přenášených informací. Podstatným faktorem je vhodná volba frekvence provádění zjišťování stavu sítě, na frekvenci závisí doba zpoždění v ovládání a zatížení modulů EXM32. S klesající dobou odezvy stoupá zatížení EXM32.

Vyžadované funkcionality systému:

- přímé volání telefonního čísla,
- vyčtení telefonního seznamu ze SIM-karty,

- volání čísla z telefonního seznamu,
- skupinové hovory,
- registrace (deregistrace) funkčního čísla,
- volání funkčního čísla,
- identifikace funkčního čísla,
- SMS nejsou zatím v síti GSM-R SŽDC povolené a proto nejsou součástí systému MTR10 v rámci této diplomové práce.

5.2 Analýza propojení MTR10 Server a MTR10 Klient

Serverová a klientská část je propojena sítí LAN. V souladu s tím bylo nutné zvolit vhodný způsob komunikace. Vzhledem k tomu, že velká část komunikace s modemem MT2 má odpovědi ve formě tabulek nebo je možné pro větší efektivitu tyto informace do formy tabulky přeložit, bylo rozhodnuto v rámci projektu MTR10 pro komunikaci využít databázi.

Bylo tedy nutné zvolit vhodnou databázi. Mezi hlavní požadavky na databázi spadaly:

- malá náročnost na výpočetní zdroje,
- diskový prostor,
- bezplatná dostupnost databáze pro embedded systémy.

Těmto kritériím vyhovuje databáze SQLite. Jak již bylo uvedeno v kapitole zabývající se použitými technologiemi, tato databáze je napsána jako knihovna v jazyce C, nevyužívá trvale běžícího démona na straně serveru, díky tomu její nároky na výpočetní zdroje i diskový prostor jsou minimální. SQLite je také k dispozici pro embedded systémy zdarma. Vzhledem k tomu, že databáze nevyužívá trvale běžícího démona, bylo nutné zvolit vhodný způsob sdílení souboru s příponou `dbm`, který obsahuje databázi po síti LAN.

První variantou bylo využití NFS (Network File System), po prvních pokusech na PC vypadal tento způsob sdílení nadějně. Avšak při použití na modulu EXM32 vznikala velká zpoždění, při prvním přístupu do databáze zhruba 50 vteřin. Během testů bylo vyzkoušeno množství konfigurací NFS pro odstranění tohoto pro-

blému. Testy také ukázaly, že občas dochází k poškození souboru obsahujícího databázi. Hledání řešení odhalilo fakt, že na některých systémech není možné použít databázi SQLite ve spojení se sdílením prostřednictvím NFS a to díky chybě v implementaci NFS. Tato chyba souvisí s uzamykáním přístupu do databáze. Bylo tedy nutné zvolit jiný způsob sdílení.

Další možnou variantou sdílení souboru po síti LAN je Samba. Testy ukázaly, že přístupové doby k databázi s využitím sdílení prostřednictvím Samby jsou přijatelné a také že nedochází k poškozování souboru obsahující databázi.

Dle průběhu vlastní implementace je možný vznik potřeby minimalizace odezvy některých částí systému, v takovém případě lze využít oddělené komunikace prostřednictvím UDP.

5.3 Analýza MTR10 Server

Úkolem serverové části je zprostředkování ovládání modemu MT2 klientské části. Z hlediska této diplomové práce by bylo možné, aby veškerou logiku prováděla klientská část systému a však z hlediska projektu MTR10 a samotné logiky architektury klient server by to nebylo vhodné. Vzhledem k tomu, že tato práce vychází z mé účasti na projektu MTR10, bude serverová část obstarávat téměř veškerou logiku s výjimkou podpůrných činností jako je například vyhledávání v telefonním seznamu atd.

Funkcionality serveru lze rozdělit do tří základních části:

- pravidelné vykonávání dotazů na stav sítě a modemu, jako např. kontrola síly signálu, stavu registrace do sítě a stavy hovorů,
- provádění uživatelských požadavků, na jejichž základě se odešlou příkazy či sekvence příkazů modemu a vyhodnotí se jejich odpovědi, uživatelské příkazy budou z klientské části předáváni prostřednictvím databáze SQLite, tato část by se dala také označit jako směr komunikace do sítě GSM-R,
- pravidelné kontrolování příchodu nevyžádaných zprav od modemu MT2 a jejich vyhodnocování, tuto část lze označit jako směr komunikace ze sítě GSM-R.

Výsledky všech tří základních části se budou předávat serveru prostřednictvím databáze SQLite.

5.4 Analýza MTR10 Klient

Základní funkce klientské aplikace budou zapouzdřeny do jedné třídy, která bude obsluhovat serverovou část a bude zprostředkovávat data k interpretaci grafickému uživatelskému prostředí. Pro projekt MTR10 je vytvářeno uživatelské prostředí pro X server na embedded systému EXM32.

V rámci této diplomové práce bude vytvořena grafická aplikace v QT Frameworku, která bude sloužit jako ukázková, testovací a diagnostická. Diagnostická část aplikace bude realizována rozšířenými logovacími soubory na straně serveru, průběžnými konzolovými výpisy pro potřeby testování a zobrazováním pomocních informací v GUI aplikaci, jako například tabulky VGCS/VBS hovorů atd.

6. IMPLEMENTACE MTR10 SERVER

První částí, kterou bylo třeba naimplementovat je třída pro komunikaci po sériovém portě. Provedená analýza ukázala, že třída pro komunikaci po sériovém portu musí být napsána jako více vláknová. Obsahuje:

- hlavní vlákno, ve kterém se provádějí metody pro zařazování dat do bufferu k odeslání a vybírání z bafferu příjmu,
- druhé vlákno čeká na příjem dat ze sériového portu a tato data ukládat do výstupního bufferu,
- třetí vlákno očekává příkaz k odeslání dat z vstupního bufferu na sériový port.

Díky tomuto konceptu je také nutné vytvořit třídu bufferu, která bude přizpůsobena používání ve více vláknové aplikaci. Tedy bude zajištěno, že při přístupu do bufferu z jednoho vlákna je ostatním vláknům dočasně zamezen přístup, aby nedošlo k poškození dat.

6.1 CSerialBuffer

Pro komunikaci po sériovém portu s využitím více vláknového přístupu je nejprve nutné vytvořit třídu bufferu s ošetřenou kritickou sekcí.

Jako základ byla použita obousměrná fronta `deque` z STL (Standard Template Library). STL poskytuje velké množství struktur a algoritmů, které se při programování v C++ velice dobře používají a zefektivňují práci programátora. Více informací o knihovně STL lze získat např. ze zdroje [20].

`CSerialBuffer` je v podstatě pouze `deque` typu `unsigned char`, jejíž metody jsou ošetřeny jako kritické sekce pomocí mutexů.

6.1.1 Klíčové metody CSerialBuffer

- `void PushDataBack(const unsigned char *lpbBuf, const unsigned int uiCount)` – vložení dat na konec bufferu, parametr `lpbBuf` obsahuje data a `uiCount` udává jejich počet,

- `unsigned int PopDataFront(const unsigned int uiCount, unsigned char * lpbBuf)` – odebrání dat ze začátku bufferu, parametr `uiCount` určuje, kolik dat chceme odebrat, data se uloží do `lpbBuf`, návratová hodnota udává, kolik dat bylo skutečně odebráno,
- `bool IsInBuffer(string str)` – metoda vrací `true` pokud se v bufferu nachází řetězec zadaný parametrem `str`.

Ukázka klíčové metody:

```
void PushDataBack(const unsigned char *lpbBuf, const unsigned int
uiCount)
{
	//uzamčení mutexu
	pthread_mutex_lock(&mutex_buffer);
	//vložení dat do bufferu
	m_buffer.insert(m_buffer.end(),lpbBuf,lpbBuf+uiCount);
	//odemčení mutexu
	pthread_mutex_unlock(&mutex_buffer);
}
```

6.2 Ovládání sériového portu

V OS Linux jsou porty reprezentovány soubory (stejně jako další systémové části), tento přístup umožňuje velice efektivní práci.

6.2.1 Otevření portu

Pro otevření sériového portu se stejně jako pro otevření standardního souboru používá funkce `int open(const char *pathname, int flags)`. Parametr `pathname` určuje cestu k souboru, který má být otevřen, v tomto případě se tedy jedná o cestu k speciálnímu souboru zpřístupňující sériový port. V OS Linux jsou soubory sériových portů umístěny v adresáři `/dev/` a jejich názvy jsou `ttyS0` až `ttySx` podle počtu sériových portů, které jsou v OS k dispozici. Celá cesta tedy je například `/dev/ttyS0`. V souvislosti se soubory sériového portu ještě upozorňuji na nutnost úpravy oprávnění k těmto souborů.

Parametrem `flags` nastavujeme vlastnosti otevření souboru. Parametry `flags` lze kombinovat pomocí bitového součtu `|`, existuje velké množství `flags`, pro

potřeby této práce zde nebudou všechny uvedeny, lze je najít v manuálové stránce funkce `open()`. `Flags` musí obsahovat jeden z parametrů:

- `O_RDONLY` – pouze pro čtení,
- `O_WRONLY` – pouze pro zápis,
- `O_RDWR` – pro čtení i zápis.

Návratová hodnota funkce je typu `integer` a vrací file deskriptor otevřeného souboru, pokud proběhlo vše korektně a -1 pokud došlo k chybě.

6.2.2 Parametry komunikace

Po otevření sériového portu je nutné nastavit parametry komunikace po sériovém portu. K tomu slouží struktura `termios`, ta popisuje vlastnosti asynchronní komunikace portů, má velké množství nastavitelných parametrů. V této práci jsou uvedeny pouze ty použité v rámci projektu.

```
struct termios
{
      tcflag_t c_iflag;      /* input modes */
      tcflag_t c_oflag;      /* output modes */
      tcflag_t c_cflag;      /* control modes */
      tcflag_t c_lflag;      /* local modes */
      cc_t     c_cc[NCCS];   /* control chars */
};
```

Hodnoty nastavené c_flags:

- `B57600` – rychlost přenosu, tato hodnota odpovídá přenosové rychlosti 57600 Bd, rychlost se zadává ve formátu `B<rychlost>`,
- `CLOCAL` – ignorování kontrolních řádků modemu,
- `CREAD` – zapnutí přijmu,
- `CS8` – počet datových bitů,
- `CSTOPB` – nastaví dva stop bity,
- `IGNPAR` – ignorování chyb synchronizace bloku a parity.

Hodnoty c_cc:

- `c_cc[VTIME] = 0` – nepoužívá se mezi znakový časovač,
- `c_cc[VMIN] = 1` – blokuj čtení, dokud nepřijde jeden znak.

K zjištění aktuálního nastavení sériového portu slouží metoda `int tcgetattr(int fd, struct termios *termios_p)`. Tato metoda uloží do parametru `termios_p` nastavení portu určeného file deskriptorem `fd`. Návratová hodnota je typu `integer` a vrací -1, pokud došlo k chybě. Aktuální nastavení portu před změnou se používá k obnovení nastavení po ukončení komunikace. Přesto, že většina aplikací využívajících komunikaci po sériovém portu nastavuje před samotným využíváním portu parametry přenosu, je nepsaným pravidlem uvádět port do původního stavu.

Pro nastavení nových parametrů se používá funkce `int tcsetattr(int fd, int optional_actions, const struct termios *termios_p)`, první parametr `fd` opět určuje o jaký se jedná port. Parametr `optional_actions` muže nabývat tří hodnot:

- `TCSANOW` – změny nastavené ve struktuře `termios` se projeví okamžitě,
- `TCSADRAIN` – změny se projeví až po odeslání všech dat uložených v bufferu portu,
- `TCSAFLUSH` – změny se projeví poté co všechna data zapsaná do objektu, na který se odkazuje file deskriptor `fd`, jsou odeslána a všechna přijatá data která nebyla přečtena jsou zahozena.

Třetím parametrem funkce jsou nová nastavení uložená ve struktuře `termios`. Návratová hodnota je `-1` pokud došlo k chybě.

6.2.3 Vyprázdnění bufferů portu

Před začátkem používání portu je ještě vhodné vyprázdnit vstupní a výstupní buffer. K tomu slouží metoda `int tcflush(int fd, int queue_selector)`, parametr fd určuje port a parametr `queue_selector` buffer, který chceme vyprázdnit, ten může nabývat hodnot:

- `TCIFLUSH` – vyprázdnění vstupního bufferu,
- `TCOFLUSH` – vyprázdnění výstupního bufferu,
- `TCIOFLUSH` – vyprázdnění obou bufferů.

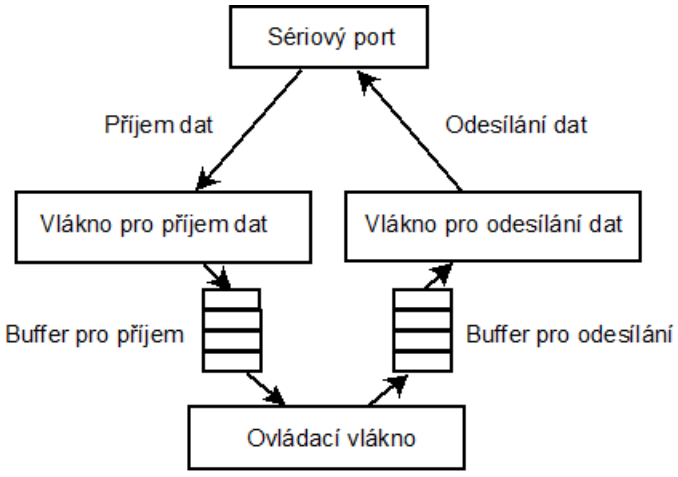
6.2.4 Odeslání a přijímání dat po sériovém portu

Tomu, že se komunikace po sériovém portu v OS Linux ovládá prostřednictvím souboru, odpovídá také způsob odesílání a přijímání dat. K čemuž se využívají standardní funkce pro zápis/čtení do/z souboru.

Funkce `ssize_t write(int fd, const void *buf, size_t count)` zapíše do souboru, určeného file deskriptorem `fd` data z bufferu `buf` o délce `count`. Návratová hodnota je typu `ssize_t` a může být přetypována na `integer`. Vrací počet zapsaných dat, pokud vše proběhlo v pořádku. Pokud se vyskytla chyba, vrátí funkce `-1` a systémová globální proměnná `errno` je nastavena na hodnotu odpovídající dané chybě.

Pro načtení dat ze souboru slouží funkce `ssize_t read(int fd, void *buf, size_t count)`. Tato metoda načte ze souboru, určeného file deskriptorem `fd`, počet dat zadaný parametrem `count` do bufferu určeného parametrem `buf`. Vrací počet přečtených dat. Hodnota přečtených dat může být menší, než kolik jich bylo požadováno (např. vyžadovaný počet dat v souboru není). V případě chyby vrátí `-1` a systémovou globální proměnnou `errno` opět nastaví na hodnotu odpovídající chybě, která se vyskytla.

6.2.5 CSerialPort



Obr. 16) Schéma CSerialPort [Zdroj: autor]

Třída `CSerialPort` používá pro ovládání sériového portu tři vlákna, hlavní vlákno vykonává řídící metody. Druhé vlákno slouží pro příjem dat ze sériového

portu, data ukládá do instance `CSerialBuffer` (viz 6.1) pro příjem, z tohoto bufferu se vybírají pomocí metod v ovládacím vláknu. Třetí vlákno slouží k odesílání dat na sériový port. Data k odeslání se ukládají pomocí metod řídícího vlákna do instance `CSerialBuffer` pro odesílání.

Pro zahájení komunikace po sériovém portu slouží metoda `void OpenCom(const char *szPortName, int iBaudRate = 19200, bool invertRTS = false, int stopBits= 1, int dataBits = 8, CSParity parity= None)`.

Parametry metody `OpenCom`:

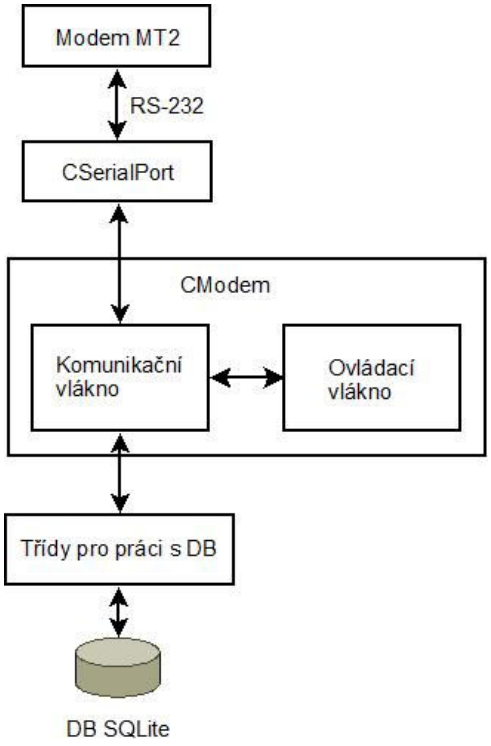
- `szPortName` – cesta k souboru sériového portu (například `/dev/ttyS0`),
- `iBaudRate` – přenosová rychlost, typ `integer` (například `19200`),
- `invertRTS` – invertní hodnoty řízení toku RTS, slouží k ovládání modemů či převodníků, pokud má parametr hodnotu `1`, má RTS obrácené logické úrovně,
- `stopBits` – počet stop bitů,
- `dataBits` – počet datových bitů,
- `parity` – používaná parita.

Další klíčové metody jsou:

- `bool IsAnyDataReceived(unsigned int timeOut = 500000, unsigned int minBytes = 1)` – kontrola zda byla přijata data, metoda kontroluje, zda byla přijata data ze sériového portu, počet dat, který je potřebný k vyhodnocení příjmu, jako dostačujícího je určen parametrem `minBytes`, parametr `timeOut` určuje dobu kontrolování v mikrosekundách,
- `void Send(const char * data)` – odesílání ASCII řetězce,
- `string Receive()` – příjem ASCII řetězce,
- `void Send(const unsigned char* data, unsigned int length)` – odesílání binárních dat, metoda odešle binární data zadaná parametrem `data` o délce `length`,
- `unsigned int GetReceivedBytes()` – metoda vrátí počet přijatých dat v bufferu,

- `int Receive(unsigned int count, unsigned char* buffer)` – metoda příjme počet dat daný parametrem `count` z bufferu příjmu do proměnné `buffer`, návratovou hodnotu je počet reálně vyzvednutích dat v bajtech, pro velikost alokace bufferu lze využít výše popsanou metodu `GetReceivedBytes()`.

6.3 Ovládání modemu MT2

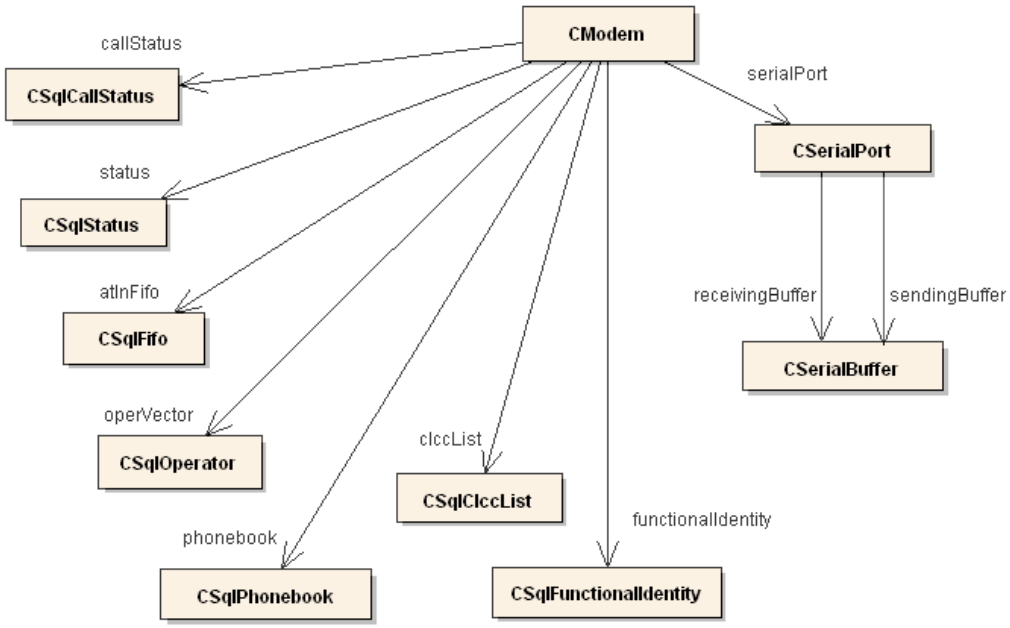


Obr. 17) Schéma ovládání modemu MT2 [Zdroj: autor]

K ovládání modemu MT2 slouží třída `CModem`. Tato třída využívá `CSerialPort` ke komunikaci s modemem po sériovém portu. Pro ukládání do DB SQLite využívá množství tříd, které v rámci této práce nebudou všechny uvedeny. Jejich základem je třída `CSqlite`, která je popsána v následující kapitole. Jednotlivé třídy jsou vytvořené s ohledem na informace, které je potřeba ukládat do DB, například je vytvořena samostatná třída pro telefonní seznam a další. Diagram klíčových tříd je zobrazen na obrázku 18.

Samotná třída `CModem` využívá dvě vlákna, hlavní vlákno k ovládání (spuštění, ukončení komunikace atd.) a komunikační vlákno, které kontroluje DB, zda neob-

sahuje nové příkazy pro modem, provádí tyto příkazy a jejich výsledky ukládá do DB.



Obr. 18) Diagram tříd [Zdroj: autor]

6.3.1 CSQLite3

Po nainstalování databáze SQLite lze využít pro ovládání z programovacího jazyka C/C++ programové rozhraní z knihovny `sqlite3`. Tato knihovna umožňuje základní práci, ale pro snazší práci bylo vhodné vytvořit třídu `CSqlite3`, která využívá funkce základní knihovny `sqlite3`. Důvod pro vytvoření nadstavbové knihovny bylo především snazší a přehlednější vykonávání dotazů.

Struktura SQLResult slouží k přehlednějšímu reprezentování výsledků SQL dotazu.

```
struct SQLResult{
   //Dotaz který byl proveden
   string sql;
   //Počet ovlivněných řádků, dotazy pro vložení, mazání či úpravu
   int affected_rows;
   //Počet vybraných řádků příkazem SELECT
   int selected_rows;
   //Hlavička dat vybraných dotazem SELECT
   vector<string> header;
   //Vektor řádků vybraných dotazem SELECT
   vector< vector<string> > data;
```

```
    //Doba vykonání příkazu
    double time;
};
```

Klíčové metody třídy SQLite:

- `void OpenDb(string dbName)` – otevření DB,
- `SQLResult ExecuteSQL(string sql)` – vykonání SQL dotazu,
- `void CloseDb()` – uzavření DB.

Třída `CSQLite3` slouží pro základní práci s DB, vzhledem k tomu, že většina informací, které je třeba zprostředkovat ovládací aplikaci má charakter tabulek, jako například telefonní seznam, jsou pro jejich reprezentaci vytvořené samostatné třídy založené na třídě `CSQLite3`. Těchto tříd je větší množství a nebudou zde podrobně popsány.

Přehled klíčových tříd pro reprezentaci dat:

- `CSqlFifo` – fronta AT-příkazů,
- `CSqlStatus` – stav modemu,
- `CSqlCallStatus` – stav hovorů,
- `CSqlOperator` – tabulka povolených operátorů,
- `CSqlClccList` – tabulka hovorů.

6.3.2 CModem

Třída `CModem` zastřešuje ovládání modemu MT2 (viz Obr. 17). Hlavní částí třídy `CModem` je komunikační vlákno které zajišťuje funkcionality popsané v kapitole 5.3. Popisem tohoto vlákna se zabývá několik následujících odstavců.

Pro návrh komunikačního vlákna je nutné počítat s nevyžádanými zprávami od modemu MT2. Z hlediska vytížení systému bylo nutné zvolit vhodný způsob uspávání vlákna s ohledem na přijatelné prodloužení odezvy aplikace. Vhodnou variantou je využít kontrolu příjmu nevyžádaných zpráv od modemu po sériovém portu. Vlákno tedy kontroluje sériový port po dobu timeoutu, jelikož se jedná o čekání blokující, komunikační vlákno nevytěžuje v době kontroly portu systémové zdroje. Pokud před vypršením timeoutu příjme komunikační vlákno nevyžádanou zprávu od modemu, provede její vyhodnocení. V opačném případě, tedy došlo-li k vypršení

timeoutu a jsou-li požadavky od klienta, které je nezbytné zpracovat, provede vlákno odeslání těchto požadavků modemu a následné vyhodnocení odpovědí. Toto se opakuje do doby, než přijde od řídícího vlákna příkaz k ukončení komunikace.

Funkce komunikačního vlákna tedy lze rozdělit na kontrolu nevyžádané komunikace po sériovém portu, k čemuž se využívá třída `CSerialPort` a kontrolu požadavku klientské části systému, k čemuž se používá třída `CSqlFifo` založená na třídě `CSQLite3`.

Jelikož veškerá komunikace s modemem probíhá v kódování ASCII (struktura AT-příkazů je popsána v kapitole 4.6), bylo z hlediska vyhodnocování zpráv nutné vytvořit metody pro rozdělování textových řetězců a další zpracování.

Kromě komunikačního vlákna obsahuje `CModem` také vlákno ovládací. Díky tomu že, veškeré komunikační funkce provádí komunikační vlákno, slouží ovládací vlákno pouze ke spouštění a ukončování serverové části systému.

Vzhledem k rozsáhlosti třídy `CModem` a faktu, že zdrojové kódy jsou vlastnictvím firmy RADOM, s.r.o., se tato diplomová práce nebude zabývat podrobným popisem této třídy.

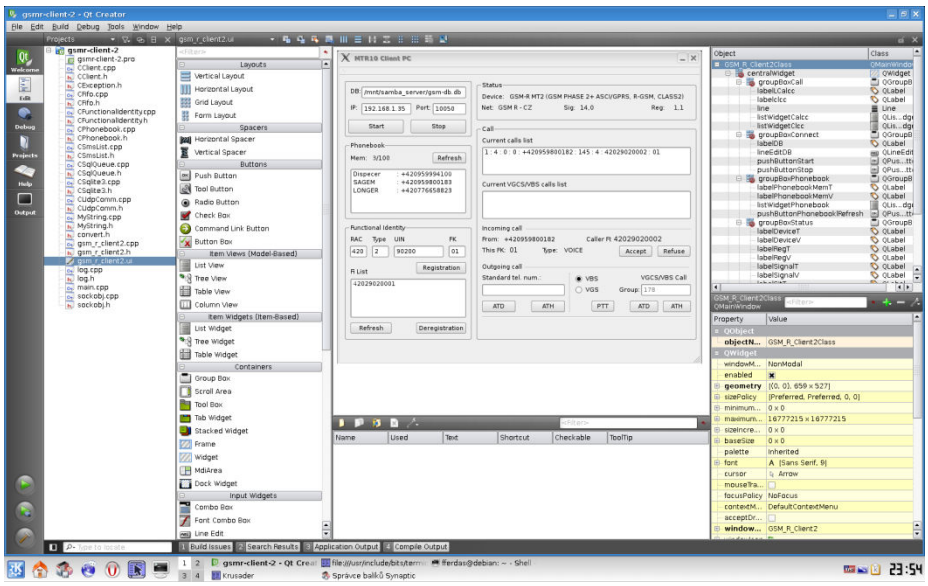
7. IMPLEMENTACE MTR10 KLIENT

7.1 CClient

`CClient` je klientská část systému, která obsahuje převážnou většinu požadovaných funkcí (viz kap. 5.1), zbývající požadované funkce jsou zapouzdřeny pomocí tříd pro komunikaci prostřednictvím databáze SQLite, které jsou popsány v kapitole 6.3.1 zabývajícím se implementací serverové části.

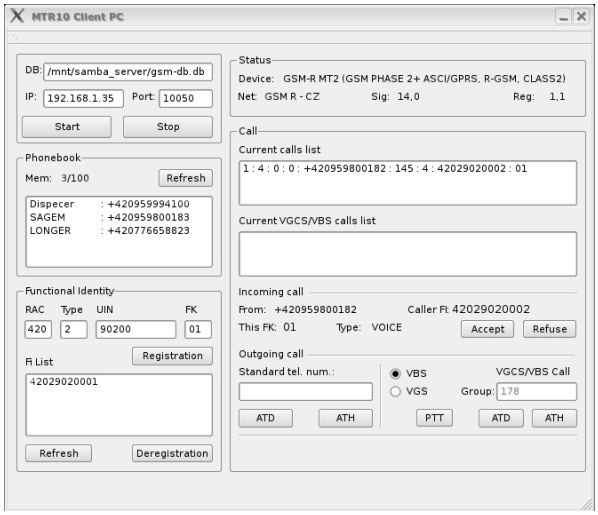
7.2 Testovací a ukázková aplikace MTR10 klient PC

Tvorba klientské testovací aplikace pro architekturu PC je prováděna s využitím Frameworku QT pro jeho velmi dobrou použitelnost. Nové prostředí QT Creator umožňuje jeho snazší využití, ovšem jedná se o nový vývojový nástroj, který není dosud úplně odladěný.



Obr. 19) QT Creator [Zdroj: autor]

Klientská aplikace se skládá ze dvou vláken. Hlavní vlákno obsluhuje uživatelské rozhraní a vedlejší vlákno je použito pro periodické kontroly změn stavu jednotlivých částí, jako například status volání, funkční hodnoty a další informativní části.



Obr. 20) MTR10 Klient PC [Zdroj: autor]

Popis ovládání aplikace Klient PC je uveden v příloze A.

V souvislosti s použitím QT Frameworku, je nutné upozornit na systém používání vláken. V nově vytvořených vláknech nelze využít volání metod pro změnu stavu a vlastností komponent (například Line Edit, Push Buton atd.). V případě nedodržení tohoto pravidla, dojde k častému pádu programu odůvodněného více násobným přístupem do komponent.

Řešením této situace je využití `QMetaObject::invokeMethod`. s použitím metody propojení objektu `QT::QueuedConnection`.

```
bool QMetaObject::invokeMethod ( QObject * obj, const char * member,
Qt::ConnectionType type, QGenericReturnArgument ret,
QGenericArgument val0 = QGenericArgument( 0 ), QGenericArgument val1
= QGenericArgument(), ... )
```

- `QObject * obj` – objekt který chceme napojit,
- `const char * member` – jméno signálu nebo slotu, který obsluhuje napojený objekt,
- `Qt::ConnectionType type` – způsob napojení, pro více vláknový přístup slouží `QT::QueuedConnection`, požadované změny se ukládají do fronty a systém je provede, jakmile vstoupí do hlavní smyčky událostí,

- `QGenericReturnArgument ret` – návratová hodnota slotu, pokud je použit asynchronní způsob zobrazování tak návratovou hodnotu nelze použít,
- `QGenericArgument val 0...9 = QGenericArgument()` – až deset parametrů, které lze předat slotu (signálu) specifikovanému parametrem `member`,
- metoda vrací `true` pokud lze vyvolat slot, jinak vrací `false`.

ZÁVĚR

Cílem této práce bylo vytvoření ovládací aplikace modemu MT2 s využitím architektury klient-server pro embedded systém EXM32, dále pak také vytvoření grafické testovací aplikace pro PC.

Přes problémy, které vznikly během tvorby této diplomové práce, se na prostředcích společnosti RADOM, s. r. o., podařilo úspěšně vytvořit testovací a ukázkovou aplikaci pro ovládání modemu MT2.

Tato problematika svým rozsahem zcela jistě přesahuje rozsah diplomové práce. Existuje velké množství funkcí, které ještě bude nutné doimplementovat před tím, než bude moci být systém MTR10 dokončen a nasazen do reálného provozu. Především se předpokládá dodatečná implementace:

- nouzových hovorů,
- SMS zpráv,
- dálkového zastavení vlaku.

POUŽITÁ LITERATURA

1. TRS – *Traťový radiový systém.* 1993. 83 s.

2. MATTHEW, Neil, STONES, Richar. *Linux – Začínáme programovat.* 2000. 912 s. ISBN 8072263072.

3. PRATA, Stephan. *Mistrovství v C++*. 2007. 1120 s. ISBN 978-80-251-1749-1.

4. ECKEL, Bruce. *Myslíme v jazyce C++ – knihovna programátora.* 2000. 556 s. ISBN 80-247-9009-2.

5. *User Manual GSM-Railway Mobile Termination MT2.* Austria: Kapsch Traffic-Com AG, 2005. 250 s.

6. *MSC Vertriebs GmbH* [online]. 2006 [cit. 2009-03-27]. Dostupný z WWW: <www.msc-ge.com>.

7. *Gentoo: Handbook* [online]. 2001 [cit. 2009-03-27]. Dostupný z WWW: <http://www.gentoo.org/doc/cs/handbook/>.

8. SKULINA, Ivan . RADOM představil digitální traťový systém jako univerzální komunikační prostředek. *Týdeník Českých drah – ŽELEZNIČÁŘ* [online]. 2005, roč. 2005, č. 37 [cit. 2009-03-27]. Dostupný z WWW: <http://www.cd.cz/static/old/NEW/TCD2005/5_37rado.htm>.

9. *POSIX Threads Programming* [online]. 200? [cit. 2009-03-27]. Dostupný z WWW: <https://computing.llnl.gov/tutorials/pthreads/>.

10. PODHOLA , Martin. *Signály a procesy 2* [online]. 14. prosinec 2005 [cit. 2009-05-09]. Dostupný z WWW: <http://www.linuxexpres.cz/praxe/signaly-a-procesy-2>.

11. *European Rail Traffic Management System* [online]. *Wikipedia,* 2009 [cit. 2009-04-27]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/European_Rail_Traffic_Management_System>.

12. *European Train Control System* [online]. *Wikipedia,* 2009 [cit. 2009-04-27]. Dostupný z WWW: < http://cs.wikipedia.org/wiki/European_Train_Control_System>.

13. Přechod od analogové k digitální komunikaci. *Vědeckotechnický sborník ČD* [online]. 2005, roč. 2005, č. 20 [cit. 2009-04-27]. Dostupný z WWW: <http://www.cdrail.cz/VTS/CLANKY/vts20/2003.pdf>.

14. SNÁŠEL, Jaroslav. *Technologie GSM-R: mobilní sítě ve službách železnice* [online]. 2006 [cit. 2009-04-27]. Dostupný z WWW: <http://www.mobilmania.cz/clanky/technologie-gsm-r-mobilni-site-ve-sluzbach-zeleznice/sc-3-a-1112292/default.aspx>.

15. *GSM-R* [online]. *Wikipedia,* 2009 [cit. 2009-04-27]. Dostupný z WWW: <http://en.wikipedia.org/wiki/GSM-R>.

16. OLMR, Vít . *HW server představuje – Sériová linka RS-232* [online]. 2005 [cit. 2009-04-30]. Dostupný z WWW: <http://hw.cz/rs-232#konektory>.

17. TIŠNOVSKÝ , Pavel . *Sériový port RS-232C.* [online]. Root.cz, 2008 [cit. 2009-04-30]. Dostupný z WWW: <http://www.root.cz/clanky/seriovy-port-rs-232c/>.

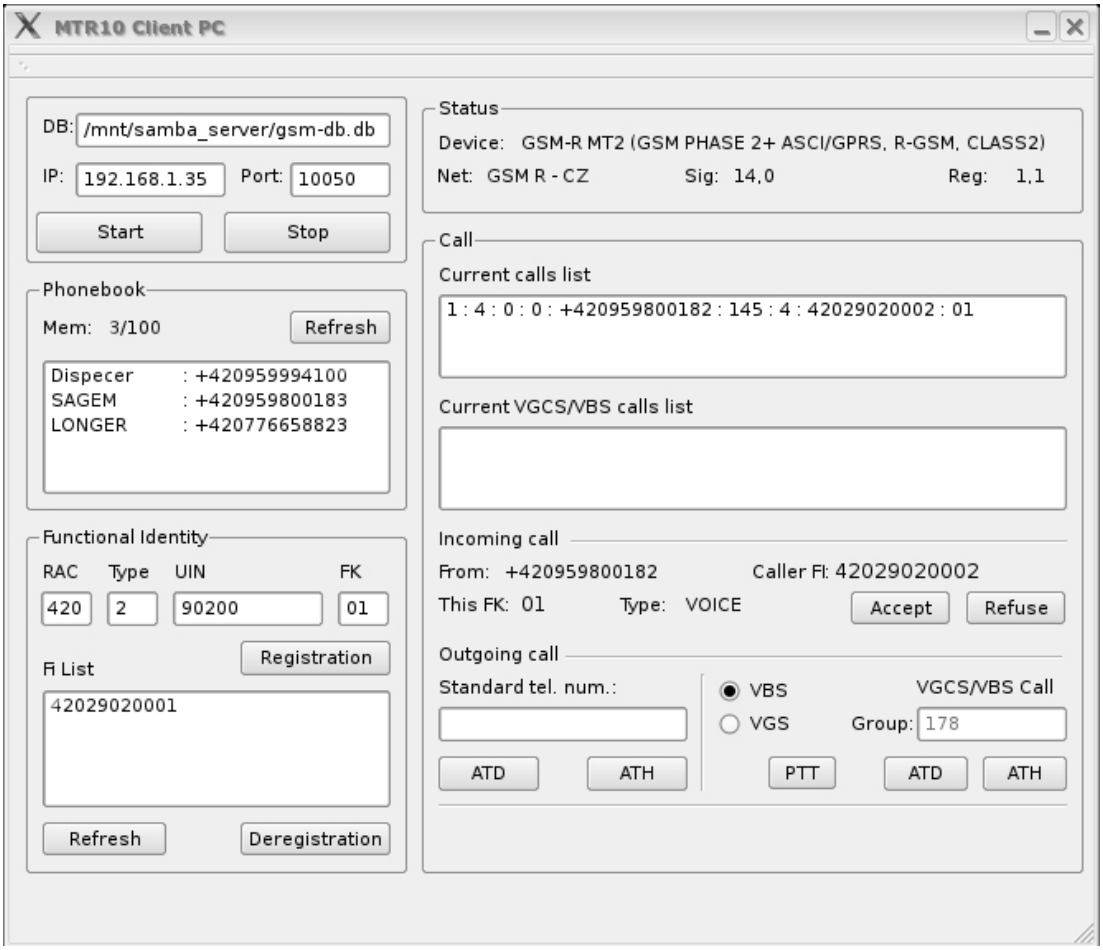
18. *Analýza dějů na rozhraní RS 232 C* [online]. 2006 [cit. 2009-04-30]. Dostupný z WWW: <http://www.fit.vutbr.cz/study/courses/ITP/public/texty_lab/uloha9-rs232.pdf>.

19. MARTINOVSKÝ, Jiří . Evropský digitální rádiový systém pro železnice – EIRENE. *Vědeckotechnický sborník ČD* [online]. 1998, roč. 1998, č. 5 [cit. 2009-04-27]. Dostupný z WWW: <http://www.cdrail.cz/VTS/CLANKY/505.pdf>.

20. *C++ Reference* [online]. 2005 [cit. 2009-05-05]. Dostupný z WWW: <http://www.cppreference.com/wiki/>.

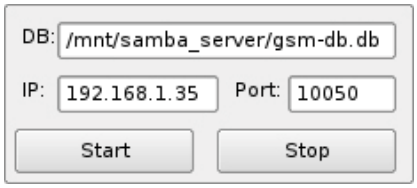
PŘÍLOHA A – UŽIVATELSKÝ MANUÁL MTR10 KLIENT PC

Aplikace MTR10 Klient PC, se ovládá prostřednictvím hlavního dialogového okna. Výsledky prováděných akcí, jsou oznamovány v tomto okně, případně prostřednictvím příslušných MessageBoxů.



Obr. 1) Aplikace MTR10 Klient PC [Zdroj: autor]

KONFIGURACE A SPUŠTĚNÍ OVLÁDÁNÍ MODEMU GSM-R



Obr. 2) Konfigurace a spuštění ovládání modemu GSM-R [Zdroj: autor]

Hlavním komunikačním rozhraním mezi serverovou a klientskou částí systému je databáze SQLite. Editační pole s názvem DB obsahuje cestu k databázi, tato databáze je připojená prostřednictvím Samby.

Pro ovládání tlačítka PTT (Push to talk) a přenosu stavu kanálu se využívá komunikace prostřednictvím UDP. Pro konfiguraci této komunikace slouží editační pole s názvy IP (IP adresa serveru) a Port (Komunikační port). Pro funkci této komunikace je nutné nastavení příslušných hodnot i na serverové části.

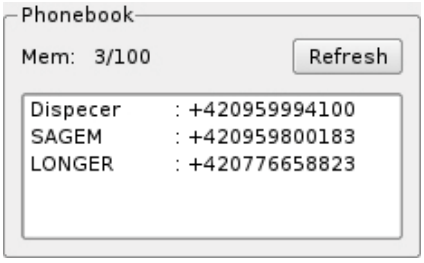
Tlačítko Start

Tlačítko start spouští ovládání modemu GSM-R, v případě, že komunikace je již spuštěna neprovede se žádná akce. V opačném případě se klientská část pokusí připojit k sdílené databázi. V době spouštění aplikace musí být serverová část již v aktivním provozu, pokud tomu tak není, klientská část se pokouší opakovaně připojit k serverové po dobu deseti vteřin, poté zobrazí zprávu o neúspěšném připojení. Komunikace prostřednictvím UDP pro obsluhu stavu kanálu VGCS hovoru a ovládání PTT se spouští teprve v okamžik, kdy je aktivní hovor VGCS, po jeho ukončení se ukončí i tato komunikace.

Tlačítko Stop

Tlačítko stop ukončí aktivní komunikaci a to jak prostřednictvím databáze tak i UDP spojením.

TELEFONNÍ SEZNAM



Obr. 3) Telefonní seznam [Zdroj: autor]

Telefonní seznam je automaticky načten po připojení klientské části k serveru, kde je vyčítán z SIM-karty. Ukládání do seznamu není implementováno, protože není požadováno, provádí se prostřednictvím sítě GSM-R centrálně.

Informační text `Mem`, ukazuje stav využití paměti telefonního seznamu SIM-karty.

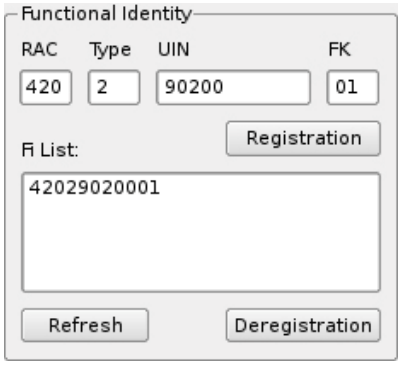
Tlačítko Refresh

Tlačítko Refresh slouží k znovunačtení telefonního seznamu ze serveru.

Výběr záznamu

Dvojklik na příslušný záznam, vloží jeho telefonní číslo do části odchozích hovorů, kde lze provést volání na toto číslo.

FUNKČNÍ IDENTITY

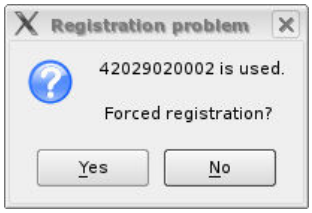


Obr. 4) Funkční Identity [Zdroj: autor]

Group box Functional Identity obsahuje funkce pro spravování funkčních identit.

Tlačítko Registration

Toto tlačítko slouží k registraci nové funkční identity, která je zadána v editačních polích RAC, Type, UIN, FK. Výsledek registrace je zobrazen po jejím provedení v MessageBoxu s příslušným textem, pokud proběhla registrace v pořádku je funkční číslo přidáno do přehledu FI List.



Obr. 5) Dotaz na nucené přihlášení [Zdroj: autor]

Pokud je funkční identita, kterou chceme registrovat již obsazená, je zobrazen dotaz, zda ji chceme registrovat násilně.

Přehled FI List

V tomto přehledu jsou zobrazené funkční identity, které jsou registrované na danou SIM-kartu. Přehled se aktualizuje při každé požadované operaci s identitami. Aplikace také periodicky kontroluje, zda nedošlo k nucenému odhlášení registrované funkční identity jiným účastníkem sítě, pokud ano je zobrazen příslušný MessageBox a funkční identita je odebrána ze seznamu.

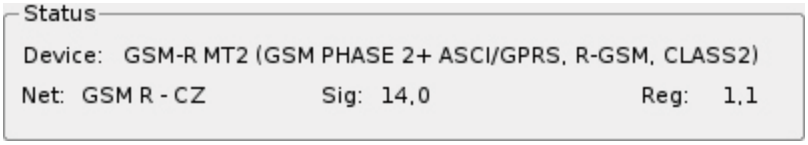
Tlačítko Refresh

Provede znovunačtení registrovaných identit ze serverové části.

Tlačítko Deregistration

Provede odregistrování funkční identity, která je zvolena v jejich seznamu.

STATUS



Obr. 6) Status [Zdroj: autor]

GroupBox status zobrazuje základní informace o stavu systému.

Device

Zobrazuje identifikaci modemu GSM-R, která je vyčtena AT-příkazem z modemu.

Net

Zobrazuje síť, do které je modem přihlášen, identifikace sítě se vyčítá AT-příkazem jako číselná hodnota, tento identifikátor je poté přeložen na textové pojmenování sítě dle povoleného seznamu sítí vyčteného ze SIM-karty pomocí přímého přístupu do její adresářové struktury.

Sig

Zobrazuje aktuální sílu signálu:

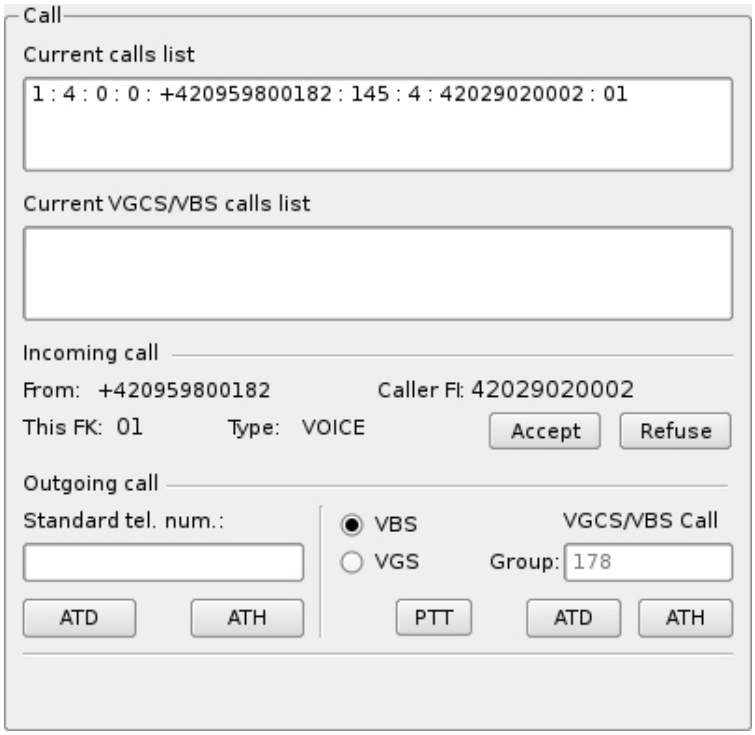
- `0` – -110 dBm nebo méně,
- `1` – -110 dBm do -109 dBm,
- `2 až 30` – -109 dBm až -49 dBm,
- `31` – -48 dBm nebo více,
- `99` – neznámá úroveň signálu.

Reg

Stav přihlášení do sítě ve formátu `<n>,<stat>`.

- `<n>` – registrace zapnutá nebo vypnutá:
  - `0` – vypnutá,
  - `1` – zapnutá,
- `<stat>` – stav registrace:
  - `0` – neregistrováno,
  - `1` – registrováno,
  - `2` – neregistrováno modem vyhledává operátora,
  - `3` – registrace byla zamítnuta,
  - `4` – neznámý stav,
  - `5` – registrováno s aktivním roamingem.

HOVORY



Obr. 7) Ovládání a přehled hovorů [Zdroj: autor]

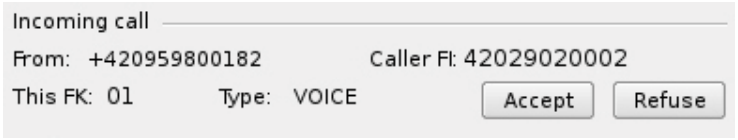
Current calls list

Pomocná přehledová tabulka standardních příchozích a odchozích hovorů. Struktura zobrazovaných záznamů je popsána v podkapitole 4.6.5.

Current VGCS/VBS calls list

Pomocná přehledová tabulka VGCS/VBS příchozích a odchozích hovorů. Struktura zobrazovaných záznamů je rovněž popsána v podkapitole 4.6.5.

PŘÍCHOZÍ HOVORY



Obr. 8) Příchozí hovory [Zdroj: autor]

From

Informace o čísle volajícího, jedná se o klasické telefonní číslo, například +420959800184.

Caller FI

Pokud volající má registrovanou funkční identifikaci, zobrazuje jí toto pole v plném formátu, například 42029020002, v opačném případě je pole prázdné.

This FK

Pokud je registrována funkční identita, zobrazuje se v tomto poli poslední část, neboli FK. V reálném provozu se této informace využívá k určení příjemce hovoru v rámci vlaku, kdy je možné, aby server měl registrován větší počet funkčních identit lišících se pouze poslední složkou, tato složka určuje, o jakou pozici se jedná (strojvedoucí, průvodčí, vlakový rozhlas atd.).

Type

Určuje typ příchozího hovoru (VOICE, VBS, VGS).

Tlačítko Akcept

Přijme příchozí hovor

Tlačítko Refuse

Odmítne příchozí hovor

ODCHOZÍ HOVORY

Obr. 9) Odchozí hovory [Zdroj: autor]

Ovládání odchozích hovorů je rozděleno do dvou částí, první z nich obsluhuje standardní hovory a druhá část obsluhuje skupinové a jednosměrné skupinové hovory.

Standardní hovory se provádí vyplněním čísla, které chceme volat do editačního pole `Standard tel. num.` a vytočení pomocí tlačítka `ATD`, pro ukončení hovoru slouží tlačítko `ATH`. Informace o stavu hovoru se zobrazují ve společné části pod odchozími hovory. Do editačního pole pro zadání čísla lze zadat také funkční číslo, které chceme volat.

ATD VGCS/VBS

Dle vybraného typu hovoru (VBS/VGS) a čísla skupiny provede vytvoření skupiny, číslo skupiny nelze v rámci testování měnit, mohlo by dojít k narušení provozu drážních vozidel.

Příchozí skupinové a jednosměrné hovory jsou dle požadavků na systém přijímaný automaticky.

PTT

Pokud se jedná o příchozí či odchozí hovor VGS, je vedle tohoto tlačítka zobrazen stav kanálu (0 – kanál je obsazen, 1 – kanál je volný). V případě že je kanál volný dojde stisknutím tohoto tlačítka k zakličování kanálu pro vysílání.

ATH VGCS/VBS

Ukončí probíhající skupinový hovor.

PŘÍLOHA B – UŽIVATELSKÝ MANUÁL MTR10 SERVER

Aplikace se spouští příslušným binárním souborem `gsmr-server`, veškeré parametry se nastavují v konfiguračním souboru `server.conf`, který musí být umístěn ve stejném adresáři jako spouštěný soubor.

Pro ukončení aplikace je nutné stisknout klávesu `q` a následně `Enter` v konzole, ve které je aplikace spuštěna.

KONFIGURAČNÍ SOUBOR SERVER.CONF

Jednotlivé parametry konfigurace musí být na samostatných řádcích. Každý parametr se skládá z klíče a hodnoty, ty jsou odděleny bílým znakem (mezera nebo tabulátor). Klíč musí být umístěn na začátku řádku. Řádky začínající znakem „#" nejsou brány v potaz a slouží jako komentáře. Na pořadí konfiguračních řádků nezáleží.

Možné klíče:

- `DB` – cesta k souboru databáze, která slouží pro komunikaci mezi serverovou a klientskou částí,
- `LOG` – cesta k souboru, do kterého se loguje komunikace s modemem, tento log je určen jako diagnostický, je do něj uložena každá komunikace s modemem,
- `LOG_U` – do tohoto souboru se ukládají příchozí nevyžádané zprávy modemu,
- `LOG_Error` – cesta k souboru, do kterého se ukládají případné chyby aplikace,
- `PORT` – cesta k souboru obsluhy sériového portu,
- `BAUDRATE` – komunikační rychlost modemu v Baudech,
- `UDP_ADDR` – IP adresa klientské aplikace,
- `UDP_PORT` – port vysílání UDP komunikace.

Cesty logovacích souborů jsou doplněny o datum a příponu `log`, například `Msq_09_02_19.log`.

Ukázka konfiguračního souboru:

```
#Komentář
DB    /home/server/db/gsm-db.db
LOG   /home/server/log/Msq
LOG_U /home/server/log/U_msq
LOG_Error /home/server/log/Error
PORT  /dev/ttyS4
BAUDRATE    19200
UDP_ADDR    192.168.1.37
UDP_PORT    10050
```

PŘÍLOHA C – UKÁZKA POUŽITÍ MUTEXU

```
//Deklarace vláken
pthread_t m_ptread1;
pthread_t m_ptread2;
//Deklarace mutexu
pthread_mutex_t m_mutex;
//Deklarace proměnné ošetřené kritickou sekcí
int iCount;

//Deklarace funkce prvního vlákna
static void *thread_function1(void *args){
    for(int i=1;i<4;i++){
        //...
        pthread_mutex_lock(&m_mutex); //Uzamčení mutexu
        iCount++; //Operace s proměnnou
        cout <<"Thread function 1 count:"<< iCount << endl;
        pthread_mutex_unlock(&m_mutex); //Odemčení mutexu
        usleep(100);
        //...
    }
    pthread_exit(0); // Ukončení vlákna
}

//Deklarace funkce druhého vlákna
static void *thread_function2(void *args){
    for(int i=1;i<4;i++){
        //...
        pthread_mutex_lock(&m_mutex); //Uzamčení mutexu
        iCount--; //Operace s proměnnou
        cout <<"Thread function 2 count:"<< iCount << endl;
        pthread_mutex_unlock(&m_mutex); //Odemčení mutexu
        usleep(100);
        //...
    }
    pthread_exit(0); // Ukončení vlákna
}

int main() {
    iCount=0;
    pthread_mutex_init(&m_mutex,0); // Inicializace mutexu
    // Vytvoření vlákna 1
    pthread_create(&m_ptread1,0,thread_function1,0);
    // Vytvoření vlákna 2
    pthread_create(&m_ptread2,0,thread_function2,0);
    //...
    sleep(1); // Tělo
    //...
    pthread_join(m_ptread1,0 ); //Připojení vlákna 1
    pthread_join(m_ptread2,0 ); //Připojení vlákna 2
    pthread_mutex_destroy(&m_mutex); //Uvolnění zdrojů mutexu
    return 0;
}
```