VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

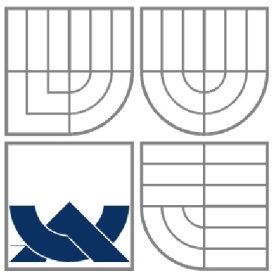
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

Řízení adaptivního dopravního uzlu

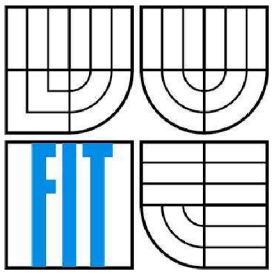
DIPLOMOVÝ PROJEKT
MASTER'S THESIS

AUTOR PRÁCE Bc. Karel Hudec
AUTHOR

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

Řízení adaptivního dopravního uzlu

Adaptive Traffic Junction Controller

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — Bc. Karel Hudec
AUTHOR

VEDOUCÍ PRÁCE — Ing. Josef Strnadel, Ph.D.
SUPERVISOR

BRNO 2009

Abstrakt

Tato práce má za úkol prostudovat inteligentní systémy řízení dopravních uzlů a navrhnout aplikaci, která bude řídit konkrétní dopravní uzel. Od formální specifikace až po implementaci. Bude se jednat o Real-Time aplikaci, což znamená, že bude pracovat v reálném čase. Budou prostudovány také konkrétní Real-Time operační systémy a vybrán nejvhodnější pro danou aplikaci a na něm bude tato aplikace také implementována a odzkoušena.

Klíčová slova

RT aplikace, RT operační systém, inteligentní řízení, dopravní uzel, inteligentní řízení dopravy

Abstract

This work will study intelligent systems for traffic control and will design an application which will control existing traffic junction. From formal specification to an implementation. It will be a real time application. That means it will work in real time. We will also study real time operating systems and we will choose the most suitable for our application and we will implement our application on this operating system.

Keywords

RT application, RT operating system, intelligent control, traffic junction, intelligent traffic control

Citace

Hudec Karel: Řízení adaptivního dopravního uzlu.

Brno, 2009, diplomová práce, FIT VUT v Brně.

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Josefa Strnadela, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Karel Hudec
27. 12. 07

Poděkování

Chtěl bych poděkovat hlavně svému vedoucímu diplomové práce Ing. Josefu Strnadelovi, Ph. D. za výborné vedení a příjemnou spolupráci.

Obsah

1 Úvod

V dnešní době, kdy skoro každý vlastní automobil (někdo i více než jeden) a provoz stále houstne, jsou kladeny čím dál větší nároky na řízení dopravy. Do tohoto systému řízení se zapojují jisté prvky inteligence, což třeba znamená, že systém pozná, že v určitém směru je provoz například hustší, tak změní intervaly zelených na semaforu tak, aby byl v tomto směru provoz plynulejší. Systém má tedy jakési prvky inteligentního chování. Řídicích systémů je v dopravě velice mnoho, křižovatky, železniční přejezdy, různé proměnné dopravní značky nebo informační tabule, řízení provozu v tunelu atd. Do těchto systémů je tedy jistě dobré zabudovat jisté inteligentní prvky, aby systém mohl okamžitě reagovat na dopravní situaci, která nastane a mohl ji jistým způsobem řešit, nebo dokonce třeba inteligentně této nepříznivé dopravní situaci předejít. Pro řídicí systém tohoto typu, tak jak jej hodláme pojmout, je nejvýhodnější použít operační systém (*OS*) pracující v reálném čase (*RT*) – dále *RTOS* – a na něm nám bude řídící aplikace běžet. Bude se tedy jednat o řízení v reálném čase.

V této diplomové práci budu vycházet ze skutečného dopravního uzlu tvořeného dvěma na sebe navazujícími křižovatkami ve městě Brně. Mým úkolem je navrhnout systém, který bude za pomoci různých prostředků a algoritmů přizpůsobovat řízení dopravy dle aktuální dopravní situace tak, aby předešel některým dopravním komplikacím a zlepšil průjezdnost tohoto dopravního uzlu. Systém by měl i nějakým způsobem ulehčovat průjezd uzlem vozidlům s právem přednostní jízdy. Měl by se také monitorovat stav dopravy mezi navazujícími křižovatkami a preventivně působit na dopravu tak, aby nedocházelo k zácpám a tím omezením v provozu mezi křižovatkami, jelikož mezi sebou nejsou moc daleko. Diplomová práce se mimo jiné zabývá současnými trendy v řízení dopravy ve městech a obsahuje i studii dostupných *RT* operačních systémů, které by bylo možno použít pro vlastní aplikaci řídicího systému. Při hledání optimálního jsem jich totiž prošel a vyzkoušel hned několik. Nakonec jsem vybral jeden nejvhodnější a na něm celou navrženou, specifikovanou a verifikovanou aplikaci naimplementoval.

Po implementaci a oživení systému bylo nutné nějak vyzkoušet a ověřit správnou funkci systému a hlavně jeho inteligentních řešení. Proto jsem navrhl a implementoval simulační úlohu, která simuluje provoz dopravního uzlu a ověřuje správnou funkci implementovaných řešení.

2 Inteligentní systémy řízení dopravy ve městech

2.1 Systém řízení dopravy v městské oblasti

Moderní a ekonomický rozvoj měst je bezesporu závislý na rozvoji infrastruktury. Především v centrech regionů - krajských městech - závisí ekonomický rozvoj na vytvoření podmínek pro pohyb zboží a lidí, a tím i rozvoj silniční sítě, případně vhodné využití a zefektivnění silniční sítě stávající. Ve velkých městech dochází k výraznému nárůstu automobilové dopravy, což klade velké nároky na obslužnost města. Neustále se zvyšující provoz má za následek přibývající kongesce a nehody, zároveň prodlužování časových prostojů a zvyšování ekologické zátěže města.

2.1.1 Způsoby řízení dopravy ve městech

Bývají založené na centralizované nebo decentralizované Inteligenci. Oba způsoby jsou blíže popsány níže.

- **Centralizovaná inteligence**

Řízení spočívá ve vyhodnocení všech detektorů v oblasti a optimalizačním výpočtu šíření vozidel. Na základě výpočtů se v reálném čase mění řízené parametry. Jedná se o technicky a ekonomicky velice náročný způsob.

- **Decentralizovaná inteligence**

Řízení využívá pro vyhodnocení jednotlivé dopravní uzly, které okamžitě reagují na stavy dopravy. Na vyšší nadřazené úrovni pracuje řídicí počítač ve funkci koordinátora jednotlivých uzlů sítě. Toto řízení sbírá data ze všech detektorů a podle aktuální dopravní situace mění délky cyklu, skladbu fází, případně délky zelených. K řízení využívá tato inteligence algoritmy MOTION a TASS. Tyto algoritmy budou dále rozebrány v podkapitole inteligentní systémy řízení dopravy.

- **Struktura architektury řízení města**
    - **Nejnižší úroveň** - dopravní řadič světelného signalizačního zařízení (SSZ), parkovací systém, řízení tunelu apod.

- **Střední úroveň** – je úroveň oblastí s pokud možno uzavřenými celky a s minimem vazeb na okolí, obsahující řídicí systém ovládající příslušnou oblast. Může se například jednat o větší dopravní uzly, které jsou ovšem řízeny každý zvlášť a to bez ohledu na ostatní okolní uzly. Uzel zpracováván v této diplomové práci je dalo by se říct, této střední úrovně.
- **Nejvyšší úroveň** – je úroveň města, která integruje systémy řízení jednotlivých oblastí a monitoruje dopravu ve městech. Jedná se například o komplexní systém řízení, který sdružuje a monitoruje jednotlivé dopravní uzly. V globálním měřítku potom provádí například adaptivní řízení dopravy. Takto může být v extrémním případě řízena doprava globálně v celém městě.

- **Progresivní metody řízení křižovatek**
  - **Adaptivní řízení** - reaguje na dopravní situaci adaptováním programů SSZ. Tuto metodu využívám i v této diplomové práci. Jedná se o to, že doprava respektive hustota dopravy je monitorována čidly a následně na základě výsledků jsou upravovány například intervaly zelených na křižovatkách takovým způsobem, aby se zvýšila plynulost dopravy. Například křižovatka, která reaguje na hustotu provozu na příjezdu prodloužením (zkrácením) intervalu zelené v tomto směru.
  - **Řízení při nehodách a kongescích** - nahrazuje manuální řízení operátorem automatizovanými postupy. Například když dojde ve více proudovém tunelu třeba k nehodě nebo zablokování nějakého pruhu, tak se na začátku nebo i v průběhu tunelu na světelných signalizačních zařízeních nad jednotlivými pruhy objeví signál pro včasné přejetí do volného pruhu.
  - **Řízení navigováním a informováním** - využívá informace pro přesměrování dopravy. Může se jednat například informační ceduli nad vozovkou, která zobrazuje dopravní situaci a informace o ní, případně o objízdných trasách na základě dopravní situace, ke které se řidič blíží.

- **Ovlivňování chování účastníků silničního provozu**
  - **Navigace řidičů na alternativní trasy** - podle aktuální dopravní situace jsou řidiči naváděni na optimální trasu s tím, že se vozidlo vyhýbá kritickým místům, jako jsou práce na silnici, nehody, kongesce a jiné. Například se může jednat o velkou proměnnou informační tabuli, která před vjezdem do města upozorňuje řidiče na části města, které jsou ucpané a nabízí nejvhodnější objízdné trasy, případně může jít i o sérii proměnných dopravních značek, které řidiče vedou alternativní trasou.
  - **Využíváním parkovacích navigačních systémů** - v případě dopravních problémů je řidiči nabídnuta alternativní doprava (MHD), včetně cen jízdného a jízdních řádů.

Pro zajímavost zde uvádím **systém, který používá Brněnská MHD [1]** k navigování svých vozidel.

DPMB používá k řízení MHD systém RIS. Ten slouží k okamžitému rozpoznání odchylek v provozu MHD a k jejich rychlé a úspěšné eliminaci. Systém umožňuje okamžitou kontrolu odezvy na rozhodnutí dispečera. RIS je významným prvkem k zajištění bezpečnosti, kvality, hospodárnosti a kultury dopravy.

Základem RIS je datová rádiová síť pro trvalé spojení všech vozidel MHD s dispečinkem. Každé vozidlo předává na dispečink automaticky v intervalu přibližně 20 sekund svoje provozní údaje - mj. skutečný čas odjezdu od zastávky, fyzickou polohu podle družicové navigace GPS a další důležitá provozní data. V případě potřeby se z vozidla na centrálu a opačně posílají textové zprávy a povely pro informační zařízení pro cestující. RIS rovněž zajišťuje hlasovou komunikaci dispečera a řidiče a umožňuje dispečerovi promluvit k cestujícím.

Dispečeři mohou data o provozu zobrazit různým způsobem - jako tabulku vozidel jednotlivých linek a jako obraz provozu na digitální mapě. Poloha vozidel se na mapě zobrazuje v reálném měřítku, každý vůz nese barevnou visačku s nejdůležitějšími provozními údaji, barva přitom reprezentuje okamžitou odchylku od jízdního řádu. Dispečer tak může snadno odhalit žluté značky zpožděných vozidel anebo červená vozidla, která odjela od zastávky dříve. Nejraději ale sledují plnou obrazovku zeleně se hlásících vozidel jedoucích přesně v povolené toleranci. Na mapě mj. rozpoznáme, zda vozidlo odjelo od správného zastávkového sloupku v rozsáhlejším dopravním uzlu anebo na které straně ulice stojí dispečerský vůz. Postižené vozidlo nebo skupinu vozů může dispečer okamžitě zavolat a předat řidičům pokyny k nápravě nepříznivého stavu, zajistit opravu, povolat náhradní autobusy atp.

Výrazně se zvýšila kvalita informací podávaných cestujícím. Projekt RIS zajistil ozvučení všech vozidel MHD. Vyhlašují se názvy zastávek a důležité informace, zajišťuje se automatická odpověď na povel kapesního slepeckého vysílače. Vizuální i akustické informace jsou cílené - cestující dostávají ve správný čas a na správném místě správnou informaci.

Řídící a informační systém umožňuje dokonalejší preferenci vozidel MHD na řízených křižovatkách. Zpožděné vozidlo odešle z předem stanoveného místa prostřednictvím rádiového datového přenosu žádost o preferenci. Řadič semaforů obdrží údaj o trase průjezdu vozidla křižovatkou a o hodnotě zpoždění. Vozidlo poté oznamuje úspěšný průjezd křižovatkou. Tato tzv. dynamická preference dovolí na křižovatce hospodařit s každou sekundou signálního plánu a ve svém důsledku zajistí lepší průjezdnost pro všechna vozidla, nejen pro tramvaje, trolejbusy a autobusy.

2.1.2 Inteligentní systémy řízení dopravy

V následující podkapitole budou uvedeny jednotlivé inteligentní systémy řízení, které se v současnosti v praxi používají a jejich stručné vysvětlení.

- **TASS – Traffic Actuated Signal plan Selection**

[2] Softwarový nástroj, který výběrem řízených signálních plánů na základě aktuální dopravní situace z detektorů reaguje na dopravu. Aktuální dopravní situace je vyhodnocována z měřených dopravních dat ukládaných v systému řízení pomocí seznamu podmínek a prostřednictvím rozhodovacích tabulek se vybírá odpovídající signální plán, což je plán řízení SSZ.
TASS v principu pracuje na dvou úrovních:

  - **Strategická úroveň** – detekování dopravní situace v řízené oblasti a jejím okolí

  - **Taktická úroveň** – výběr signálního plánu pro skupinu řadičů v dostatečné vzdálenosti před oblastí

Na strategické úrovni jsou detekovány různé dopravní situace. Pro každou situaci existuje alespoň jeden (základní) signální plán pro každou křižovatku, který odpovídá charakteru dopravních podmínek dané situace (standardní doba cyklu, koordinace, apod.)

Na taktické úrovni se vybírají signální plány pro všechny řadiče ve skupině TASS v závislosti na aktuálních dopravních podmínkách. Cílem je reagovat rychle na fluktuace dopravního proudu v části řízené oblasti. To je zajištěno výběrem tzv. **alternativního signálního plánu**, jehož základní charakteristiky (doba cyklu, koordinace, apod.) by měly korespondovat s právě aktivním základním signálním plánem. Tím se zabrání vzniku uměle vyvolaných poruch dopravního proudu způsobených přepínáním signálních plánů.

- **MOTION - Method for the Optimisation of Traffic Signals In On-line controlled Networks**

[2] **Systém MOTION** je makroskopický modulární řídicí systém pro optimalizaci řízení dopravních toků v městské silniční síti. Základní koncepcí metody MOTION je schopnost kombinovat výhody účinného modelu dopravní sítě pro nejdůležitější dopravní proudy v síti s možností téměř okamžité reakce na změnu dopravní situace prostřednictvím místního řízení v

křižovatkách. Pro umožnění tohoto pružného řízení se údaje o provozu shromažďují, kompletují a analyzují.
Na základě této analýzy se veškeré části signálních programů (doba cyklu, sled fází, délky zelených) optimalizují pro všechny křižovatky v síti a vytvářejí se nové signální programy.
MOTION pracuje ve třech úrovních:

- **Na strategické úrovni** (každých 10 – 15 minut) je určována:

• doba cyklu
• rozdělení zelených
• základní sled fází
• parametry koordinace (offset)

- **Na taktické úrovni** (cca po 60 - 90 sec) lze základní sled fází ovlivňovat metodou místního řízení pro vypočtenou délku cyklu, například vložením speciální fáze:

  • lokální sled fází, například pro preferování veřejné hromadné dopravy

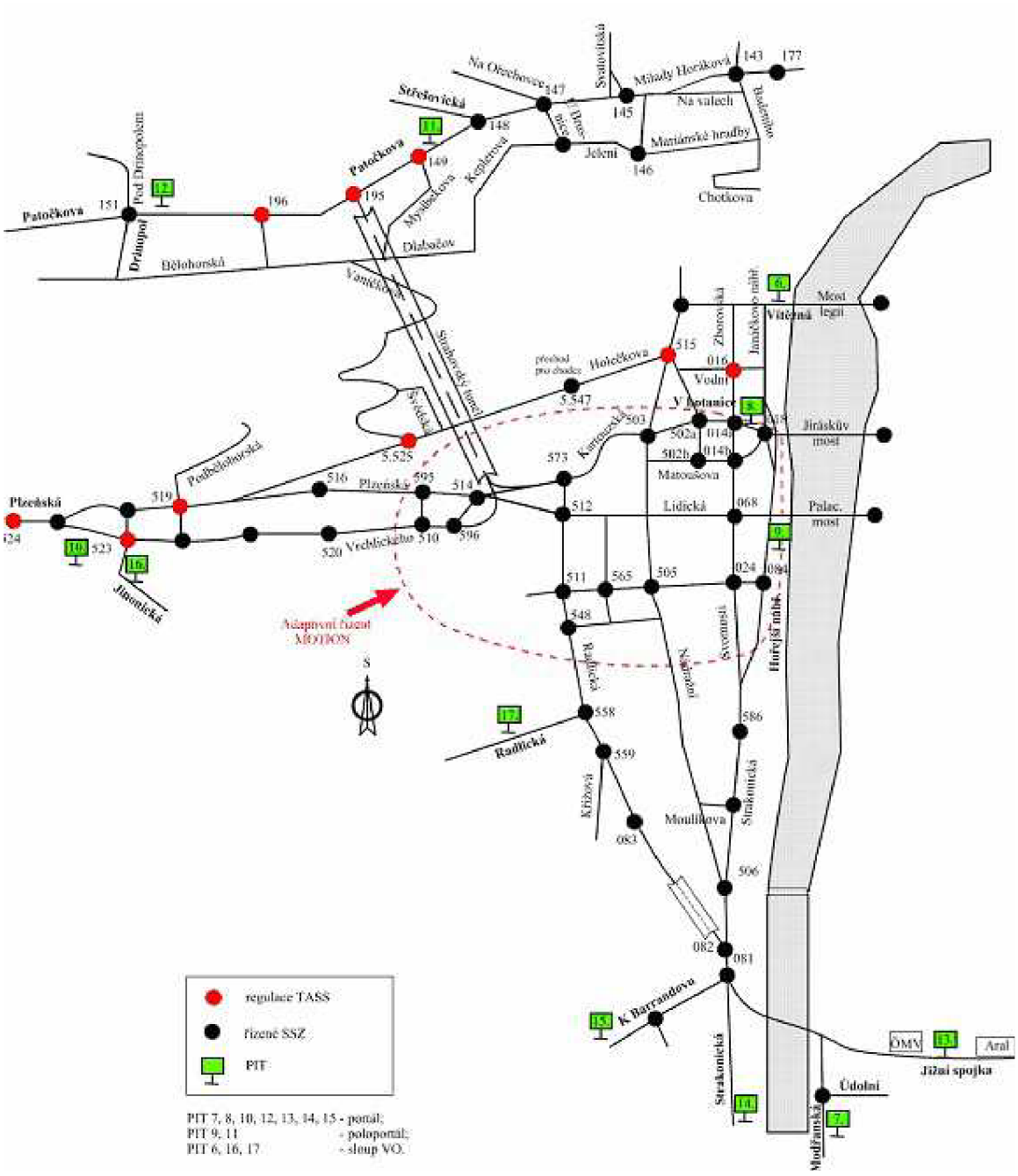
- **Na operační úrovni** (cca 1 sec) lze reagovat metodou na místní řízení pro jednotlivá vozidla:

  • délka zelené (reagováno na jednotlivá vozidla, preferenci MHD apod.)

Ve strategické úrovni lze přidělit veřejné dopravě vyšší priority než individuální automobilové dopravě. Řadiče si při místním řízení zachovávají vysokou úroveň samostatnosti při reakcích na dopravní situace. Povolené meze jsou nastaveny centrálním řízením.

Další zlepšení procesu řízení dopravy řízené systémem MOTION při krizových situacích může být aktivace modulu CIM, který umožňuje provádět management řízení dopravy při nehodách, kongescích a mimořádných událostech s různými strategiemi řízení.
Modul CIM /Congestion and Incident Management/ umožňuje na úrovni sítě výběr reakcí na nehody nebo kongesce v závislosti na dopravní situaci.

Příklad: Oblast řízená systémy MOTION a TASS



Obrázek 2.1 – Oblast řízená systémy MOTION a TASS zdroj[2]

O strategických rozhodnutích v síti rozhoduje MOTION, zatímco taktická a operační rozhodnutí se provádí dle řídicí logiky v řadičích křižovatek. Obecná strategie řízení je založena na optimalizaci šíření vozidel v síti. Síťové řízení omezuje místní řízení pouze do míry potřebné k zajištění dobré koordinace celé sítě.

2.1.3 Informování a navigování řidičů

[2] Nedílnou součástí zlepšení průjezdu vozidel městem je použití vhodného způsobu informování kombinované s navigováním a tím ulehčit momentální situaci v určité části města. Informační systémy je vhodné využít i v systémech MHD, kde informace cestujícím přispívá k psychické pohodě. Provoz tohoto dopravního informačního zařízení značnou měrou přispívá ke zvýšení bezpečnosti a plynulosti provozu na velmi zatížených dopravních sítích ve velkých městech a na příjezdových komunikacích.

Pro informační systém, který má efektivněji působit na dopravní proud, se využívají **proměnné informační tabule** (PIT) Proměnné informační tabule jsou sestaveny z elektromagnetických bistabilních elementů zvýrazněných LED diodami. PIT jsou nedílnou součástí vyšších forem řízení, informace jsou na ně distribuovány z dispečerského centra ručně operátorem na základě vyhodnocení stavu dopravy v příslušných dopravních úsecích systémem ze systému kamerového dohledu anebo automaticky na základě jiných subsystémů řízení města, jako jsou dopravní ústředny nebo řídicí systémy tunelů. Dalšími přednostmi PIT je možnost zpětného hlášení na řídicí ústřednu o stavu zařízení a nízká spotřeba elektrické energie.

Ve městech je především kladen důraz na změnu rychlosti dopravního proudu a přesměrování dopravy v případě vzniku kongesce nebo nehody. Příkladem navigačního systému je systém pro navádění vozidel na parkoviště.

Tento systém poskytuje aktuální, kompletní a přesné informace o místech, volné kapacitě nejbližších záchytných parkovišť P+R a optimálních trasách, a zároveň by měl zůstat účinný i při plném obsazení některého nebo několika záchytných parkovišť. Navrhovaný systém musí být jednotný na dotčeném území a musí být srozumitelný pro řidiče a otevřený pro další rozvoj.

Obrázek 2.2 – Informační proměnná dopravní značka

3 Řízení v reálném čase a RTOS

Tato kapitola je věnována problematice řízení v reálném čase a studii *RT* operačních systémů. Bude zde také vybrán nejvhodnější *RTOS* pro implementaci našeho systému.

3.1 Úvod do oblasti

RT systémy, nebo také reaktivní systémy jsou takové systémy, u kterých se očekává dynamická a správná reakce na podněty z vyvíjejícího se okolí. Správnost reakce těchto systémů je dána nejen správností produkovaných výstupů (odezev), ale rovněž včasností poskytnutí těchto odezev. Tyto, tzv. systémy pracující v reálném čase, zkráceně *RT* systémy, mají široké uplatnění v časově-kritických aplikacích, počínaje např. řídící jednotkou *ABS* v automobilech a konče zařízením pro monitorování životních funkcí pacienta ležícího na jednotce intenzivní péče v nemocnici.

Lze se setkat s argumenty prohlašujícími, že všechny praktické systémy jsou *RT* systémy, a to včetně systémů dávkového zpracování dat – např. systému pro zpracování studijních výsledků za daný semestr či systému pro měsíční generování výplatních pásek. Přestože doba odezvy takového systému (doba od složení závěrečné zkoušky do vydání výsledkového listu či doba od obdržení výplatní pásky do převedení výplaty na účet) může trvat i několik dnů či týdnů, systém musí zajistit odezvu v požadované časové mezi, aby nedošlo k studijní či finanční „katastrofě“. Obdobně se od programu pro zpracování textu očekává, že pokud má přispět k efektivní práci uživatele, tak bude přiměřeně rychle (např. do 1 s) reagovat na jeho podněty. Ve většině literárních pramenů jsou pak takové systémy označovány jako soft *RT* systémy:

Soft RT systém je systém, jehož výkon je v důsledku nedodržení časových omezení degradován, ale jehož funkce není tímto nedodržením dotčena.
Hard RT systém je systém, u něhož nedodržení jediného časového omezení vede k selhání systému jako celku a má katastrofální následky.
Firm RT systém je systém, u něhož nedodržení několika časových odezev nevede k selhání systému jako celku, avšak větší počet nedodržení může k tomuto selhání vést a může mít katastrofální následky.

Důležitým pojmem v RT systémech je **událost**.
Reakce systému na podněty z vnějšího prostředí je často spojena s reakcí na událost přicházející v čase t. Příchod události obvykle vede ke změně toku řízení v systému s cílem reakce na tuto událost.

Událost může být následujících typů:

	Periodická	**Aperiodická**	**Sporadická**
Synchronní	Cyklické volání instrukcí/úloh. Procesy plánované vnitřními hodinami.	Skoková instrukce při běžné činnosti programu	Skoková instrukce řešící zotavení programu z chyby
Asynchronní	Přerušení generované hodinami	Přerušení generované v pravidelných, ale ne periodických intervalech	Výjimka generovaná okolím systému „Náhodné“ události

3.2 Konkrétní RT operační systémy a výběr vhodného pro naši aplikaci

Zvolil jsem několik *RT* operačních systémů, které jsem zběžně prostudoval a vyzkoušel, abych potom mohl zvolit nejvhodnější pro implementaci aplikace. Na každém ze studovaných *RTOS* jsem odzkoušel jednoduchou aplikaci a stručně zhodnotil. Nakonec jsem zvolil jeden, na kterém jsem implementoval celý systém.

3.2.1 RT operační systém BrickOS

[3] *BrickOS* je *RT* operační systém pro Lego Mindstorms *RCX* kontroler [3]. Také poskytuje C/C++ vývojové prostředí pro *RCX* programy používající *gcc a g++* a nezbytné nástroje pro stažení programů do *RCX*.

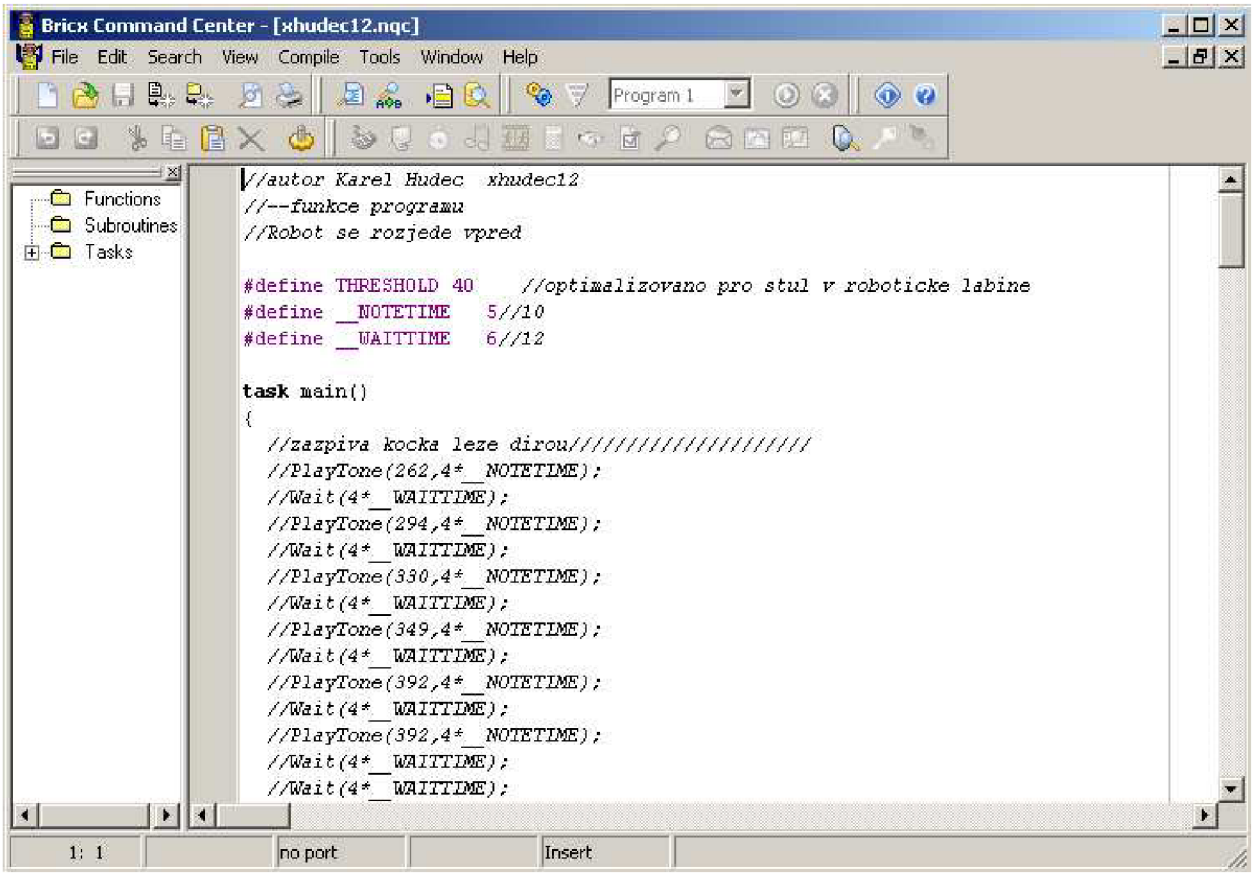
BrickOs je open source. Byl původně vyvinut na Linuxu, nicméně dnes už funguje i na ostatních platformách stejně tak na Windows. *BrickOs* sestává z alternativního operačního systému pro Mindstorms *RCX* a demonstračních programů napsaných v C a C++ . Také obsahuje utility, které umožňují nahrát OS do RCX a také nahrát zkompilované programy do *RCX* kontroléru.
Je také třeba Hitachi H8 cross compiler asembleru a linkeru. Poté lze zkompilovat *OS* a utility a pak už nic nebrání pustit se do vlastních programů.

Mindstorms RCX

Lego mindstorms je řada lego produktů kombinující programovatelné části s elektrickými motory, senzory s klasickými lego kostkami a částmi Lego Technic (pohony, ramena, pneumatické části) ke stavbě robotů a dalších automatických interaktivních zařízení.

Tato řada je poskytována komerčně jako Robotics Invention System (RIS). Je ale také používána a prodávána jako pro výukové účely. Existuje dokonce určitý druh spolupráce mezi Lego a MIT. Výuková verze produktů je nazývána Lego Mindstorms for school a obsahuje i tzv. ROBOLAB což je *GUI* programovací software.

Lego Mindstorms může být použito pro tvorbu modelu vestavěného systému s počítačově řízenými elektromechanickými částmi. Téměř všechny druhy vestavěných systémů z běžného života, od výtahů po průmyslové roboty, mohou být modelovány pomocí Lego Mindstorms.



Obrázek 3.2 – Vývojové prostředí BrickOS

Na obrázku výše je vidět vývojové prostředí, které jsem použil pro vytvoření jednoduchého příkladu. V levé části jsou podskupiny jako funkce, úlohy apod. a v pravé části je editor kódu. Z tohoto vývojového prostředí je také možno přímo kód nahrát do modulu a následně spustit.

Pro ukázku zde uvedu úryvek svého **zdrojového kódu**, který jsem při testování vytvořil.

```
SetPower(OUT_A+OUT_C,2);        //nastaveni rychlosti obou motoru na 2
SetSensor(SENSOR_3,SENSOR_LIGHT);    //nastaveni optickeho senzoru
SetSensor(SENSOR_1,SENSOR_TOUCH);    //nastaveni tlakoveho senzoru
  OnFwd(OUT_A+OUT_C);                //jedeme dopredu
   while (true)
  {
    SetPower(OUT_A+OUT_C,2);   //nastaveni rychlosti obou motoru na 2
    OnFwd(OUT_A+OUT_C);          //jedeme dopredu
    if (SENSOR_3 < THRESHOLD)//kdyz opticky senzor zaznamena zmenu povrchu
    {
      Off(OUT_A+OUT_C);          //zastavime motory
      OnFwd(OUT_B);              //rozsvitime svetla
      PlayTone(5588,4*__NOTETIME);//blikame svetly a pipeme tonem
```

Podařilo se mi zjistit, že právě mikrokotroler Lego Mindstorms *RCX*, pro který je systém *BrickOS* vytvořen se nachází na naší škole v robotické laboratoři. Je mnoho způsobů jakými lze Lego mindstorms *RCX* programovat, od original Lego *RIS* přes různé systémy ve kterých se programuje v jazycích podobných jazyku C, jako například *NQC* (ve kterém jsem si mimochodem taky vyzkoušel vytvořit nějaké aplikace). Systém BrickOS pak přichází s tím, že k programování *RCX* lze použít jazyka C a C++ a tento systém dovoluje pak v aplikacích více a lépe využívat principy reálného času a lze tak budovat aplikace stavěné na *RT* řízení. Bylo by jistě zajímavé vyzkoušet něco praktického v oblasti *RT* aplikací na tomto systému, nicméně pro naši aplikaci inteligentního dopravního uzlu se zcela nehodí.

3.2.2 RT operační systém SHARK

[4] SHARK je zkratka *Soft Hard Real-Time Kernel*. Hlavními rysy tohoto *RTOS* jsou modulární komponentně založený interface pro specifikaci plánovacích algoritmů, ovladače zařízení pro nejběžnější hardware a pokročilá práce s časem. Tento systém si klade za cíle jednoduchost vývoje, flexibilitu v modifikacích plánovací politiky, předvídatelnost a zachování standardu *POSIX*.

Jádro *SHARK* je plně modulární v termínech plánovacích metod, aperiodických serverů a souběžných kontrolních protokolů, které typicky nejsou modulární v tradičních operačních systémech. Modularita je dosažena štěpením aktivit systému mezi generické jádro a množinu modulů, které mohou být registrovány na počátku a konfigurovat jádro podle specifických požadavků aplikace.

Hlavní zisk navrhované architektury jádra je, že aplikace může být vyvíjena nezávisle na zvláštní konfiguraci systému, takže nové moduly mohou být přidány nebo vyměněny ve stejné aplikaci aby vyčíslily účinky specifických plánovacích metod v souladu s předvídavostí a výkonem.
Jádro podporuje plánování zařízení, tudíž umožňuje rozšířit plánovací algoritmy použité pro CPU nebo ostatní hardwarové zdroje.
Modulární souborový systém je dostupný a umožňuje uživateli specifikovat vlastní diskovou plánovací politiku.
Konečně tento systém je vyhovující naprosté většině POSIX 1003.13 PSE52 specifikací ke zjednodušení přenosu aplikačního kódu vyvinutých pro POSIXu vyhovující jádra.

V dnešní době je využití *RT* aplikací v mnoha oborech, od vestavěných řízení procesů až po multimediální systémy. Každá tato aplikace má zvláštní charakteristiku, a právě z tohoto důvodu mnoho rozdílných plánovacích algoritmů a protokolů pro alokaci zdrojů bylo navrženo pro přizpůsobení se rozdílným požadavků těchto aplikací.

Většina nových, přístupů byla analyzována pouze teoreticky a někdy vyhodnocena použitím plánovacího simulátoru. V takovém případě není výkonnost algoritmu vyčíslena na reálných případech, ale byla vyzkoušena pouze synteticky. Toto často souvisí s faktem, že je velice obtížné modifikovat existující operační systém a tak implementovat nové plánovací algoritmy a také je nerealistické vyvíjet nové jádro pokaždé, když se objeví nějaký nový plánovací algoritmus.
Další problém je, že většina nových přístupů v oblasti *RT* plánování je limitována výkonem CPU a zvláštní požadavky mohou být kladeny i na ostatní zařízení v systému. To je hlavně kvůli faktu, že klasické struktury operačních systémů nedovolují přesné plánovaní různých zařízení (kvůli problémům zahrnujícím konflikty zdrojů, převrácení priorit, počtu přerušení, dlouhým nepreemptivním sekcím a podobně).

Cíl *SHARKu* je vystavět výzkumné jádro úmyslně navržené pro pomoc při implementaci a testování nových plánovacích algoritmů.
Toto ale vyžaduje splnění následujících třech podmínek:

- ✓ nezávislost mechanismů jádra a plánovacími politikami pro úlohy a management zdrojů.
- ✓ konfigurovat systém za běhu specifickými algoritmy použitými pro plánování úloh a přístup ke zdrojům.
- ✓ dosáhnout nezávislosti mezi aplikacemi a plánovacími algoritmy.

SHARK je jako množina knihoven, které jsou přidány do *OSLib* knihoven. Každá knihovna pokrývá specifickou část implementace systému: Jsou tu knihovny obsahující funkce jádra, knihovny pro moduly a knihovny ovladačů zařízení.

Aplikace *SHARKu*, stejně tak jádro, moduly a ovladače mohou být provozovány na třech hlavních platformách. DOS, Linux a Windows.

SHARK je *RT* operační systém vhodný pro testování například nových plánovacích algoritmů a podobných věcí jelikož je pro tento účel vytvořen. V rámci tohoto projektu jsem se pokusil tento systém i nainstalovat a rozchodit. Přemýšlel jsem o něm původně jako o docela vhodném pro implementaci, ale bylo by asi obtížné naši výslednou aplikaci dobře simulovat a odzkoušet.

3.2.3 RT operační systém QNX

[5][6] Hlavním cílem *QNX* je dodat otevřený systém s POSIX API v robustní a odstupňované formě, vhodné pro široký rozsah systémů (od malých systémů až po high-end systémy s distribuovaným výpočetním výkonem).
Robustní architektura je nezbytná kvůli aplikacím pracujících v kritických podmínkách, OS pružně a zcela využívá *MMU* (Management Memory Unit).

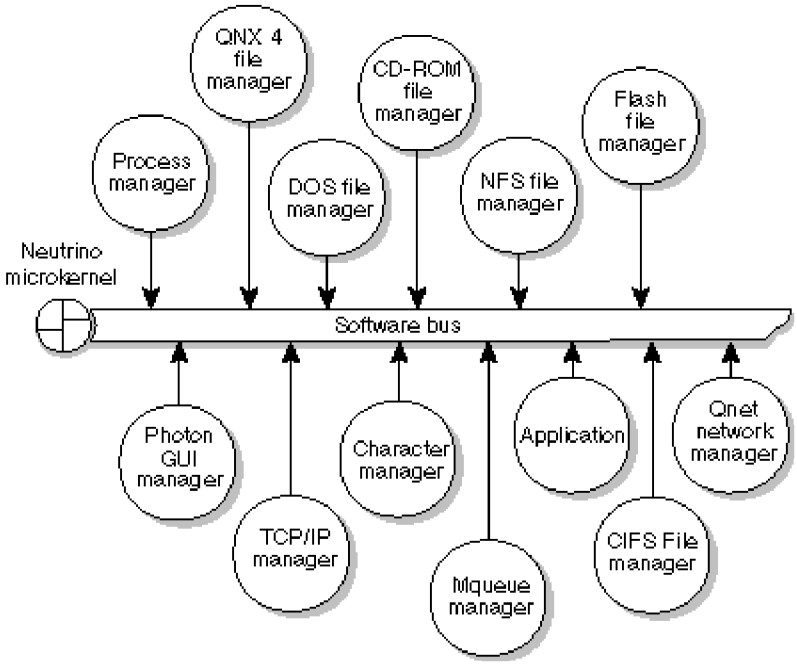
POSIXový operační systém je příliš velký a proto nevhodný pro vestavěné systémy.
QNX nemá UNIXovou architekturu a díky architektuře jádra dodává standardní POSIX API ve zmenšené formě pro *RT* vestavěné systémy a dle požadavků je lze postupně rozšiřovat. *QNX* je ideální *OS* pro vestavěné *RT* aplikace. I v takto malé formě poskytuje základní služby jako multitasking, vlákna, prioritně řízené preemptivní rozvrhování a rychlé přepínání kontextu.
QNX dosahuje jedinečného stupně účinnosti, modularity a jednoduchosti díky dvěma základním principům:

• architektuře mikrojádra

• meziprocesní komunikaci založené na zprávách

Systém *QNX* je opravdu mohutný a plnohodnotný *OS*.

Mikrojádro je strukturováno tak, že poskytuje minimální služby užívané skupinou zvláštních spolupracujících procesů. Tyto procesy zajišťují vyšší stupeň funkčnosti operačního systému. Mikrojádro postrádá systémy souborů a mnoho dalších služeb, které jsou poskytovány pomocí speciálních procesů.



Obrázek 3.2 – Schéma systému QNX zdroj [5]

Ukázka základních funkcí mikro jádra pro práci s vlákny

ThreadCreate()	Vytvoří nové vlákno.
ThreadDestroy()	Zruší vlákno.
ThreadDetach()	Odpojí vlákno (nemusí být připojeno).
ThreadJoin()	Připojí vlákno čekající na exit status.
ThreadCancel()	Zruší vlákno v příštím místě zrušení.
ThreadCtl()	Změní typické chování vlákna.
SyncTypeCreate()	Vytvoří mutex.

SyncDestroy()	Zruší mutex.
SyncMutexLock()	Zamkne mutex.
SyncMutexLock()	Podmíněně zamkne mutex.
SyncMutexUnlock()	Odemkne mutex.
SyncTypeCreate()	Vytvoří condition variable.

Skutečným cílem není vytvořit pouze malé jádro, ale vytvořit modulární operační systém. Velikost jádra je pak pouze vedlejším účinkem tohoto přístupu. Mikrojádro poskytuje služby meziprocesní komunikace, které slouží jako pojítko operačního systému. Výkon a flexibilita těchto služeb určujíce výkon výsledného *OS*.

Jádro *OS QNX* zajišťuje kompletní ochranu paměti, ne pouze pro uživatelské aplikace, ale také pro komponenty OS (ovladače zařízení, systémy souborů, atd.).
Neutrino na rozdíl od vláken není rozvrhováno. Procesor vykoná kód jádra pouze v případě explicitního volání jádra nebo v odpovědi na hardwarové přerušení.

Všechny služby *OS QNX* kromě služeb zajišťovaných modulem procento jsou zpracovávány pomocí standardních procesů, proto je velmi snadné rozšířit *OS* o další služby. Systémové procesy jsou v podstatě nerozeznatelné od některých uživatelských procesů, protože uživatelské procesy používají stejný API a vhodně privilegované uživatelské procesy mají k dispozici i stejné služby jádra.

LAN (Local Area Network) poskytuje ve své nejjednodušší formě mechanismus pro sdílení souborů a periferních zařízení mezi několik vzájemně propojených počítačů. *QNX* jde daleko za tento jednoduchý koncept a zahrnuje celou síť do jediného, homogenního souboru prostředků.
Jakékoliv vlákno běžící na počítači připojeném v síti může přímo využívat prostředky jiného počítače. Z aplikačního hlediska nejsou žádné rozdíly mezi lokálními a vzdálenými prostředky. K používání vzdálených prostředků se nemusí do aplikace vkládat žádné speciální příslušenství.
Uživatel může spouštět aplikace na libovolném počítači v síti, pokud vlastní patřičné oprávnění.

Meziprocesní komunikace umožňuje navrhovat aplikace jako sadu spolupracujících procesů, kde každý proces obsluhuje jednu přesně stanovenou část z celku. *QNX* poskytuje jednoduché, ale výkonné schopnosti meziprocesní komunikace, které značně zjednoduší práci na vyvíjených aplikacích.

QNX byl první komerční *OS* svého druhu, který používá message passing (zasílání zpráv) jako základní prostředek meziprocesní komunikace. Díky kompletní integraci zasílání zpráv do celého systému je *OS QNX* tak jednoduchý a výkonný.

Message passing neslouží pouze k posílání dat mezi procesy, ale také k synchronizaci procesů. Operační systém nepřidává žádná zvláštní data do obsahu posílaných zpráv. Data ve zprávách mají význam pouze pro odesílatele a příjemce.

Se systémem *QNX* jsem se setkal i v zaměstnání, kde jsem na něm dokonce dělal i jeden větší projekt. Tento systém je opravdu mohutný a plnohodnotný *OS*, který zdá se není vhodný pro vestavěné systémy a menší aplikace, ale spíše pro větší projekty a globálnější systémy. I když jsem původně plánoval svůj systém naprogramovat i simulovat na *QNX*, tak jsem od toho nakonec upustil, v praxi by toto řešení zcela jistě nebylo tím správným. Budeme se tedy muset poohlédnout po nějakém „malém“ ale schopném systému, který bude vhodnější pro naši aplikaci.

3.2.4 RT operační systém uC/OS-II

[7] Je to vysoce portovatelné a nastavitelné preemptivní *RT* jádro (*RTOS*) pro mikroprocesory a mikrokontrolery. Poskytuje následující služby:

- Semafory
- Událostní flagy
- Mailboxy
- Fronty zpráv
- Management úloh
- Management času
- A další…

Zdrojové kódy *uC/OS-II* lze pro studijní účely bez problémů získat a zdrojový kód je 100 procentně portovatelné ANSI C a je pravděpodobně jedním z nejčistších a nejkonzistentnějších jader co se týká zdrojového kódu. Bez problémů lze opatřit a rozsáhlou a kompletní dokumentaci, takže se s tímto jádrem dá velmi efektivně a dobře pracovat.

Toto jádro se mi již ze začátku zdálo optimální pro moji aplikaci inteligentního dopravního uzlu, z důvodu své jednoduchosti, ale na druhou stranu velice dobré možnosti použití v široké škále vestavěných systémů.

Abych mohl tento systém vyzkoušet, bylo třeba jej zkompilovat a nainstalovat. Rozhodl jsem se využít školní server *Merlin,* ke kterému mám přes internet a s použitím PuTTy přistup odkudkoliv. Zběžně jsem prostudoval postup kompilace a instalace jádra z manuálu a vyzkoušel pár různých pokusů. Nakonec se mi podařilo za pomoci vedoucího této diplomové práce jádro rozchodit a nahrál

je teda na server *Merlin, kde* bez problémů funguje. Postup kompilace a instalace je v manuálu *uC/OS-II* strana 14.

Po prostudování přehledného manuálu a uživatelské příručky (jež jsem sehnal pouze v angličtině) jsem přišel na to jak v základech s tímto *RT* jádrem pracovat, jak vytvářet a spravovat úlohy a podobně. Později jsem se dostal i k dalším funkcím a složitějším konstrukcím.

Funkce pro management úloh, které jsem použil a vyzkoušel:

OSTaskCreate() – vytvoření úlohy

OSTaskCreateExt() – vytvoření úlohy, ale s možností další specifikace doplňujících informací

OSTackChangePrio() – změna priority

OSTaskDel() – zrušení úlohy

OSTaskDelReq() – požadavek na zrušení úlohy

OSTaskResume() – obnovení suspendované úlohy

OSTaskSuspend() – suspendování (pozastavení) úlohy

Dále jsem také potřeboval něčím řídit čas a zpožďovat některé úlohy. K tomu jsem použil nejčastěji funkce:

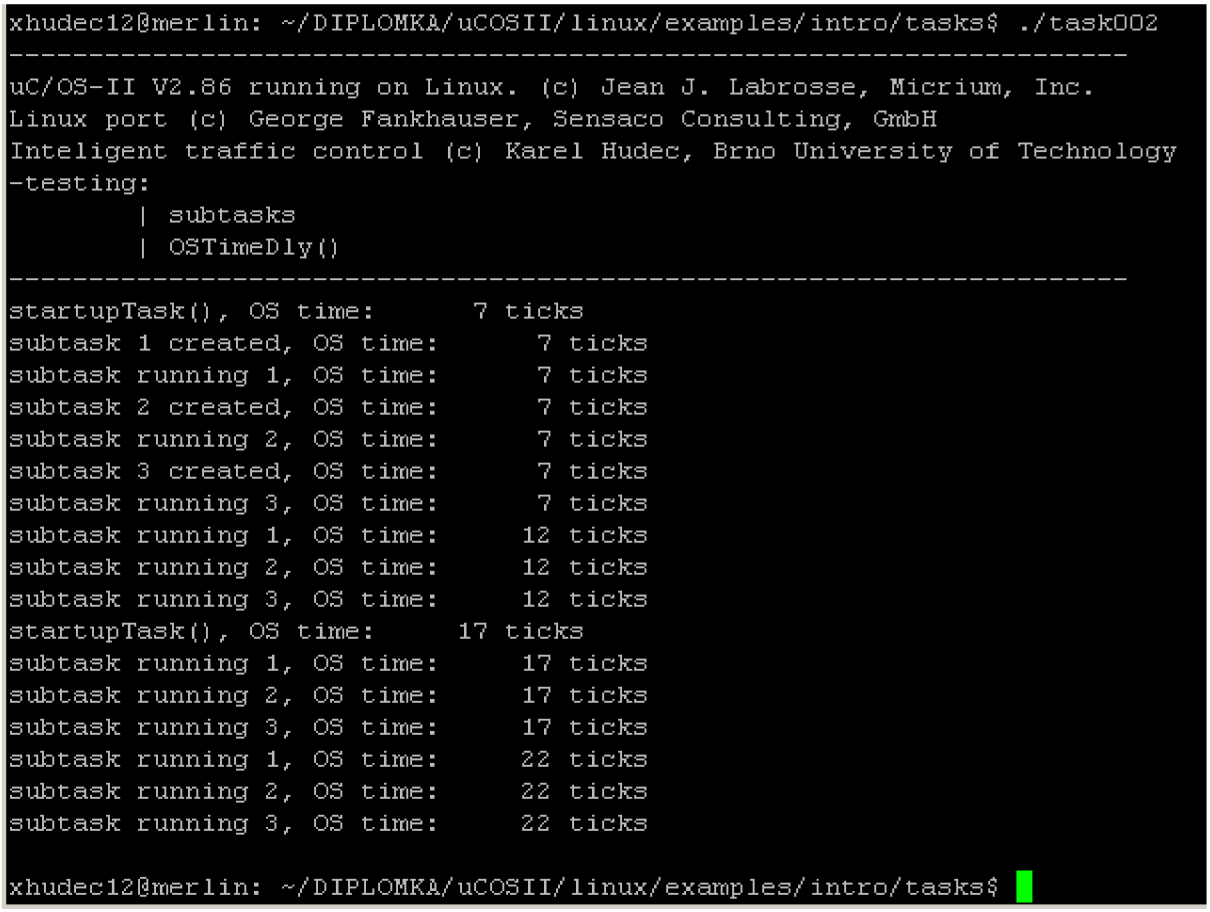
OSTimeDly() – nastavení zpoždění úlohy o určitý počet tiků

OSTimeDlyHMSM() – nastavení zpoždění o určitý čas (hodiny,minuty,sekundy,milisekundy)

Na začátku jsem měl jen triviální příklad demonstrující základní práci s jednotlivými úlohami jako je vytvoření spuštění a podobně. Z tohoto příkladu jsem potom vycházel při implementaci aplikace. Spoustu věcí bylo ale potřeba vyhledat v manuálu nebo na internetu. [7][8]

Na tomto příkladu zde budou demonstrovány základy práce s úlohami. Byla vynechána rozsáhlá hlavička, kterážto není pro demonstrační účely podstatná. Tato aplikace udělá to, že spustí systém (47), vytvoří ***StartupTask (21),*** který začne a který je nadefinován výše ve zdrojovém kódu. Úlohy bývají nadefinovány na jednom místě kódu. Zde je tedy také nadefinován ***SubTask(10)***, který se vytváří a spouští (v tomto případě konkrétně 8x) ve ***StartupTasku(21)***. Program je opatřen výpisy (37,38), takže můžeme sledovat, jakým způsobem se potom naše úlohy chovají. Zdrojový kód je popsán formou komentářů dle ANSI C pro lepší orientaci jinou barvou textu a je označen čísly řádků, na které je odkazováno.

```
#include "includes.h"   //nezbytné includy
#include <stdio.h>

#define TASK_STK_SIZE OS_TASK_DEF_STK_SIZE    /* Size of each task's stacks (# of
bytes) */
#define N_TASKS 3   //počet tasků

OS_STK TaskStartStk[TASK_STK_SIZE];           /* Stack for first task */
OS_STK          TaskStk[N_TASKS][TASK_STK_SIZE];   /* Stacks for subtasks */
char            TaskData[N_TASKS];                   /* Parameters to pass to
each task                   */

void subTask (void *data)    //deklarace subtasku
{
 printf("subtask %c created, ", *(char *)data);
 printf("OS time: %6u ticks\n", OSTime);
 for (;;)
 {
   printf("subtask running %c, ", *(char *)data);  //informační výpis
   printf("OS time: %6u ticks\n", OSTime);
   OSTimeDly(5);           //zpoždění tasku
 }
}

void startupTask(void *data) /* In the very 1st task: init timer and stats */
{                       //deklarace startup taku
 char s[80];
 sprintf(s, "V%1d.%02d", OSVersion()/100, OSVersion() % 100); /* get uC/OS-II's
 version number */


 OSStatInit();     /*    - initialize uC/OS-II's statistics for each host system
*/                          //inicializace statistik

 INT8U   i;
 for (i = 0; i < N_TASKS; i++) /* Create N_TASKS identical tasks
*/                //vytvoří v cyklu určitý počet subtasků
 {
   TaskData[i] = '1' + i;
   OSTaskCreate(subTask, (void *)&TaskData[i], (void *)&TaskStk[i][TASK_STK_SIZE
- 1], i + 1);
 }

 for(;;)
 {
   printf("startupTask(), ");
   printf("OS time: %6u ticks\n", OSTime);
   OSTimeDly(10);
 }
}

int main (void)
{
 OSInit();          //inicializace systému

 OSTaskCreate(startupTask, (void *)0, (void *)&TaskStartStk[TASK_STK_SIZE - 1],
0);                  //vytvoření startup tasku

 OSStart();    /* Start multitasking */   //start multitask systému
 return 0;
}
```



Obrázek 3.3 – Výpis terminálu testovacího programu

Na výpisu z terminálu testovacího programu je názorně vidět jak se postupně pouští jednotlivé úlohy. Je vidět kdy běží StartUpTask i kdy jednotlivé subtasky a vše je doplněno výpisem stavem hodin běhu programu.

Systém *uC/OS-II* mě mile překvapil, protože se jedná o vcelku malý systém, ale řekl bych s velkým potenciálem. Hodí se skvěle pro vestavěné systémy a je opravdu dobře přenositelný. Poté co jsem jej vyzkoušel rozchodit a otestoval pár jednoduchých příkladů, tak jsem se rozhodl vybrat si tento RT systém pro implementaci svého inteligentního systému pro řízení dopravního uzlu. Myslím, že z RT systémů, které jsem v rámci této diplomové práce prošel a odzkoušel je *uC/OS-II* opravdu dobrou volbou.

4 Popis uzlu a zběžný návrh systému

V této kapitole bude neformálně popsán celý uzel a obecně nastíněn návrh jakými způsoby bude systém řízení řešen.

4.1 Popis uzlu

Zde bude popsán uzel, který budeme řídit i zvlášť jednotlivé křižovatky z kterých se uzel skládá.

4.1.1 Popis adaptivního uzlu

Úkolem bylo vytvořit systém pro řízení inteligentního dopravního uzlu. Tento dopravní uzel se bude skládat ze dvou na sebe navazujících netriviálních křižovatek, které jsou řízeny světelným signalizačním zařízením. Jedná se o dopravní uzel, který má předlohu v existujících dvou křižovatkách, které se nachází ve městě Brně. První křižovatkou (řekneme hlavní křižovatkou, dále křižovatka I) je křížení ulic Sportovní a Pionýrská a druhou je navazující světelná křižovatka tvaru T a to křížení ulic Pionýrská a Staňkova (dále křižovatka II).

Obrázek 4.1 - Letecká fotografie dopravního uzlu. Zdroj [9]

Obrázek 4.2 - Dopravní uzel na mapě. Zdroj [9]

Tento dopravní uzel skládající se ze dvou na sebe navazujících křižovatek je místem s velmi frekventovaným provozem a to hlavně při dopravních špičkách. První dopravní špička je kolem osmé hodiny ráno a odpolední špička potom hrozí od 15 do 18 hodin. Na větší hlavní křižovatce hrozí velké kolony ze směru do města (po ulici **S**portovní). Dalším negativním faktorem, který hrozí je to, že mezi těmito dvěma křižovatkami se samozřejmě tvoří fronta, zvláště při dopravních špičkách, a tak se může přihodit, že ulice Pionýrská se od jedné ke druhé křižovatce ucpe a znemožní tak korektní chod provozu. Vzdálenost obou světelných křižovatek totiž není příliš velká (asi **150 m**). Přes tyto dvě křižovatky také docela často projíždí vozidla s právem přednostní jízdy jako je policie, záchranná služba nebo hasiči, což plynulosti dopravy také moc nepřidá.

4.1.2 Popis křižovatky I.

Obrázek 4.3 – Křižovatka I. z letadla. Zdroj[9]

Hlavní křižovatka má tvar klasické křižovatky, kde ale ve dvou směrech A a B (označení směrů později) je odbočování vpravo řešeno extra komunikací, kde se vyhneme světelnému dopravnímu značení, jen musíme dát přednost chodcům, což je zde zvýrazněno světelným signálem blikajícího oranžového panáčka. Jinak je ve třech směrech řízena klasickými semafory, které jsou pro větší přehlednost zdvojené (na pravé straně vozovky a nad vozovkou). V jednom směru je pak zvlášť semafor pro směr doleva a zvlášť pro směr rovně a doprava (doprava doplněn oranžovým panáčkem pro zdůraznění přednosti chodců ve směru doprava). Mimo to ještě ve dvou směrech je křižovatka vybavena zelenou šipkou pro opuštění křižovatky, což umožňuje bezpečné dokončení odbočování vlevo při končícím zeleném intervalu. Přechod pro chodce je v každém směru této křižovatky a každý přechod je vybaven světelným signalizačním zařízením (kromě malých přechodů, které vedou přes odbočující vozovku a kde mají chodci vždy přednost).

Křižovatka je vybavena také dopravními značkami upravujícími přednost v jízdě pro případ nefunkčnosti světelného signalizačního zařízení, nebo pro řízení v nočním provozu, k tomu se ještě podrobně dostaneme později. Intervaly světelné signalizace a další podrobnosti budou uvedeny přesně ve specifikaci.

4.1.3 Popis křižovatky II.

Obrázek 4.4 – Křižovatka II. z letadla. Zdroj [9]

Křižovatka II je tvaru T. Ve směrech hlavní ulice je provoz řízen klasickými semafory, které jsou zdvojené (na pravé straně vozovky a nad vozovkou) pro lepší přehlednost. Ve směru od centra je zde zároveň zelená šipka pro opuštění křižovatky a bezpečné dokončení odbočování vlevo.

Ve směru z boční ulice je jeden semafor klasický, který je doplněn zelenou šipkou, která signalizuje možnost odbočení vpravo. Přechody pro chodce jsou pouze ve dvou směrech a to na hlavní ulici směrem z centra a ve směru z boční ulice. Oba přechody jsou vybaveny světelným signalizačním zařízením.

Křižovatka je vybavena také dopravními značkami upravujícími přednost v jízdě pro případ nefunkčnosti světelného signalizačního zařízení, nebo pro řízení v nočním provozu, k tomu se ještě podrobně dostaneme později. Intervaly světelné signalizace a další podrobnosti budou uvedeny přesně ve specifikaci.

4.2 Stručný návrh systému

Situaci, která nastává na tomto dopravním uzlu jsem delší dobu sledoval a tak jsem mohl z těchto zjištěných informací optimálně reagovat při návrhu inteligentního systému a pokusit se implementovat systém tak aby vyhovoval požadavkům na zlepšení průjezdnosti tímto uzlem.

Intervaly světelného signalizačního zařízení byly vysledovány dle současné reálné situace a místy pozměněny pro lepší výsledek propustnosti systému.

Mým úkolem bylo navrhnout systém řízení dopravy tak aby se umožňoval preventivně a v reálném čase přizpůsobovat aktuální dopravní situaci a operativně měnit parametry řízení a zlepšit tak propustnost dopravního uzlu. Hustota dopravy bude monitorována v problematických směrech, které byly určeny dlouhodobým pozorováním. Systém by měl taky nějakým způsobem zjednodušovat průjezd křižovatkou vozidlům s právem přednostní jízdy. Dále bude systém hlídat dopravní situaci mezi křižovatkami a v případě potřeby na ni reagovat. Systém bude reagovat na nastavený časový výsek pro noční provoz, kdy se od určité do určité nastavené hodiny světelný systém křižovatky automaticky odstavuje a v tento noční interval je se provoz tímto dopravním uzlem řídí výhradně svislými dopravními značkami.

Systém bude možno také manuálně nebo automaticky (například v důsledku nějaké chyby nebo poruchy) přepnout do nouzového režimu, ve kterém se provoz řídí také buď svislými dopravními značkami, nebo případně policistou. V tomto nouzovém režimu fungují světelná signalizační zařízení stejně jako v režimu nočním a to tak, že na nich bliká pouze oranžová barva.

4.2.1 „Inteligentní“ prvky navržené pro řídicí systém

Monitorování hustoty dopravy ze směru do města a reakce na ni.

Obrázek 4.5 – Hustý provoz

Systém bude využívat výstupu z čidla, které nám bude určovat, zda se kolona z určitého směru dostala, přes přednastavenou mez a podle toho bude právě v tomto směru upravovat interval zelené, aby zlepšil propustnost křižovatky. Případně může změnit i intervaly v jiných směrech a tak například omezit příliv dalších aut i z jiných směrů. Od konkrétního čidla je zde zatím abstrahováno, ale půjde jen o to určit, zda kolona aut dosahuje určité délky či nikoliv, což by prakticky neměl být problém zrealizovat.

Monitorování míry ucpání trasy mezi dvěma světelnými křižovatkami dopravního uzlu.

Systém bude využívat vstupu z čidla, které bude udávat, zda míra stojícího provozu mezi křižovatkami nepřesahuje nastavitelnou míru a podle toho bude upravovat parametry řízení jak na první tak na druhé křižovatce aby systém zabránil ucpání pozemní komunikace mezi křižovatkami. Od konkrétního čidla je zde zatím abstrahováno, ale půjde jen o to určit, zda kolona aut dosahuje určité délky či nikoliv, což lze prakticky dobře zrealizovat. Nástin toho, jak by to bylo realizováno může být například kamerovým systémem s detekcí v obraze, případně tlakovým čidlem ve vozovce. Nejjednodušší způsob by pak mohl být třeba infračerveným čidlem snímajícím pohyb vozidel v místě kontroly délky kolony a v případě déle stojícího vozidla detekci kolony (co ale kdyby toto čidlo mířilo přesně náhodou mezi stojící vozidla v koloně). Předchozí návrhy řešení by asi byly spolehlivější.

Inteligentní průjezd vozidel s právem přednostní jízdy.

Obrázek 4.6 – Vozidlo s právem přednostní jízdy

Systém bude umožňovat usnadnění průjezdu vozidel s právem přednostní jízdy, jako jsou záchranná služba, hasiči nebo policie. Bude to zajištěno tak, že systém bude schopen obdržet signál „OOM“ (ochrana osob a majetku), který bude moci vyslat blížící se vozidlo s právem přednostní jízdy. Když dopravní uzel obdrží tento OOM signál, tak zajistí, že se na příslušné křižovatce na semaforech ve všech směrech objeví červená a tím se zastaví provoz křižovatkou, což umožní vozidlu s přednostním právem nerušeně a bezpečně projet křižovatkou. Této změně na červenou by měla předcházet oranžová barva na semaforech, kde se v té době vyskytuje zelená, aby okamžitá změna ze zelené na červenou nezpůsobila řidičům nepříjemnosti. Interval, po který bude všude červená během průjezdu přednostních vozidel bude možno samozřejmě nastavit dle potřeby. Dle pozorování by mělo stačit bez problémů 20 – 30 sekund. Po skončení tohoto intervalu se opět přes oranžovou barvu vrátí semafory do toho stavu, v jakém byly při přijetí signálu OOM. Je také možnost, že by se cyklus semaforů začal opět od počátečního stavu.

Automatické řízení denního a nočního režimu křižovatky.

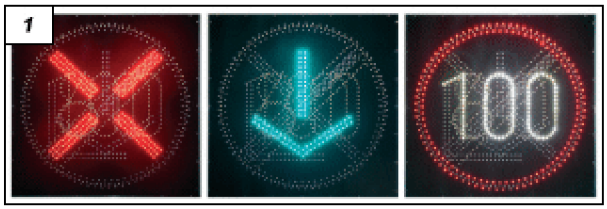
Křižovatka se bude automaticky přepínat na denní a noční režim. Hodnoty času, kdy začíná noční režim lze samozřejmě nastavit a upravovat. V denním režimu křižovatka normálně funguje a v nočním režimu se vypíná světelná signalizace, na semaforech bliká pouze oranžová barva a tím pádem se provoz řídí pouze svislými dopravními značkami. Z pozorování vyplynulo, že nejvhodnější čas pro noční režim by byl od 23:00 do 05:00.

Včasné upozornění řidičů na snížení rychlosti v závislosti na případné tvořící se koloně před křižovatkou za pomoci proměnné dopravní značky.

Obrázek 4.7 – Proměnné dopravní značky. Zdroj [2]

Systém řízení dopravního uzlu bude monitorovat délku kolony, která se bude tvořit před křižovatkou a v závislosti na ní bude schopen ovládat proměnnou dopravní značku, která bude omezovat rychlost vozidel přijíždějících z hlavního směru (kde se tato kolona tvoří). V případě provozu bez delších kolon bude značka ukazovat rychlost 60 km/h. V případě, že kolona v tomto směru překročí určitou hranici, tak systém přepne proměnnou dopravní značku z 60 na 40km/h, čímž omezí rychlost vozidel přijíždějících ke koloně a tím zajistí větší bezpečnost provozu.

Pro tento účel se doporučuje použít proměnnou dopravní značku osazenou vysoce svítivými LED diodami, která zajišťuje dobrou viditelnost i za jasného slunce, nebo při nepříznivých povětrnostních podmínkách.



Obrázek 4.8 – Proměnná dopravní značka PDZ. Zdroj[2]

Automatické přepnutí do nouzového režimu při zjištění chyby v systému a přivolání technika.

Systém řízení křižovatky může obsahovat podsystém, který hlídá správný chod celého hlavního systému a v případě nějaké chyby nastaví příznak, že se systémem není něco v pořádku a systém automaticky přepne dopravní uzel do nouzového režimu, který je shodný s režimem nočním. Zároveň systém také vyšle signál na centrální dispečink řízení provozu, čímž upozorní obsluhu, že není něco v pořádku a tak se může problém začít okamžitě řešit.

5 Specifikace a verifikace systému

V této kapitole bude systém formálně popsán a budou ověřeny jeho vlastnosti za pomocí verifikačních nástrojů *UPPAL* a *TimesTool*.

5.1 Specifikace

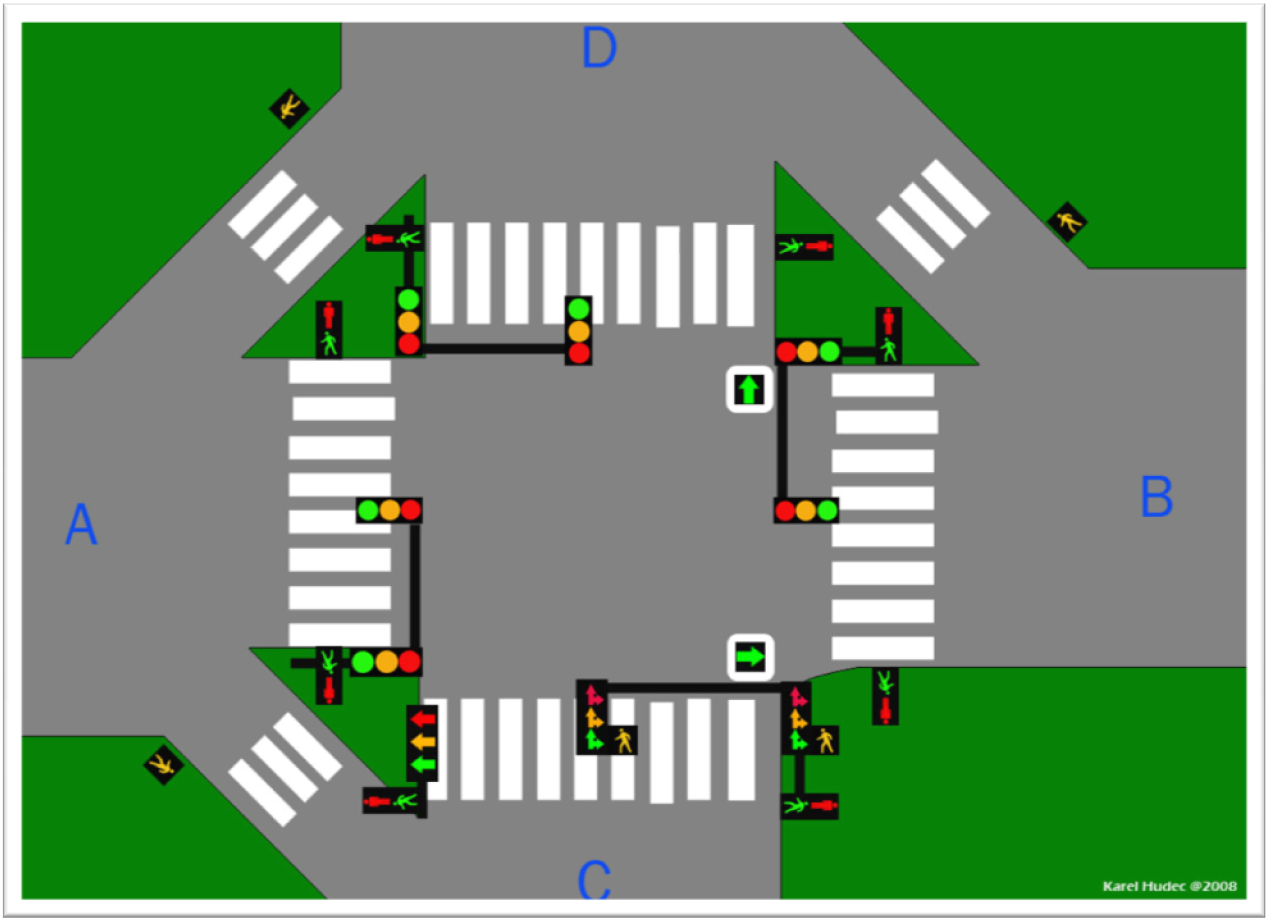
Zde je uvedena kompletní specifikace systému, který jsem navrhl. Jsou specifikovány jak jednotlivé křižovatky uzlu, tak navržená inteligentní řešení.

Obrázek 5.1 – Dopravní uzel na mapě. Zdroj [9]

Dopravní uzel se skládá ze dvou světelných křižovatek.(Každá z nich bude popsána a specifikována dále.) Mezi těmito křižovatkami je spojující komunikace, která je také součástí uzlu. Mezi našimi řízenými křižovatkami se také nachází jedna odbočka do vedlejší ulice, ale ta nemá na chod uzlu dalo by se říci žádný vliv, takže není třeba ji nějakým způsobem zahrnovat. Vzdálenost dvou světelných křižovatek uzlu je 200m. K uzlu také bude patřit proměnná dopravní značka upravující nejvyšší povolenou rychlost, která se bude nacházet na příjezdu ke křižovatce I. A to 250m před ní. Na ulici sportovní. Systém bude dále zahrnovat dvě čidla kontrolující překročení limitu kolony aut. Jedno na příjezdu ke křižovatce I. Na ulici sportovní a druhé mezi křižovatkami I a II na ulici Pionýrská.

5.1.1 Specifikace křižovatky I.

Řízená křižovatka I. bude mít klasický tvar (viz obrázek). To znamená, že bude mít 4 možné příjezdy. Pro jednoduchost si je označíme A, B, C, D.



Obrázek 5.2 – Grafický model křižovatky I.

Příjezd A

Příjezd z tohoto směru bude využívat pro řízení třech klasických semaforů. Jeden na každé straně příjezdové komunikace a jeden umístěný nad vozovkou pro lepší viditelnost. Všechny tyto semafory budou zobrazovat to samé (jako by to byl jeden). Dalším prvkem bude signál pro opuštění křižovatky umístěn naproti přes křižovatku na levé straně. Přes tento směr povede klasicky přechod pro chodce se standardním signalizačním zařízením na každé straně přechodu.

Příjezd B

Příjezd z tohoto směru bude využívat pro řízení třech klasických semaforů. Jeden na každé straně příjezdové komunikace a jeden umístěný nad vozovkou pro lepší viditelnost. Všechny tyto semafory budou zobrazovat to samé (jako by to byl jeden). Přes tento směr povede klasicky přechod pro chodce se standardním signalizačním zařízením na každé straně přechodu.

Příjezd C

Příjezd z tohoto směru bude využívat pro řízení dvou semaforů se světly pro směr rovně a doprava. Tyto budou umístěny na pravé straně vozovky a nad vozovkou pro lepší viditelnost. Oba tyto semafory budou doplněny oranžovým světlem s postavou chodce, které bude blikat frekvencí 1 x za 1 sec. V případě že na těchto semaforech bude svítit zelená. Dále bude tento příjezd využívat jednoho semaforu se světly pro směr doleva, který bude umístěn na levé straně vozovky.(bude sloužit pro odbočování vlevo)

Příjezd D

Příjezd z tohoto směru bude využívat pro řízení třech klasických semaforů. Jeden na každé straně příjezdové komunikace a jeden umístěný nad vozovkou pro lepší viditelnost. Všechny tyto semafory budou zobrazovat to samé (jako by to byl jeden). Dalším prvkem bude signál pro opuštění křižovatky umístěn naproti přes křižovatku na levé straně. Přes tento směr povede klasicky přechod pro chodce se standardním signalizačním zařízením na každé straně přechodu.

Řízení

Ve směru **A** a **B** se bude spouštět interval zelené zároveň. V tuto chvíli se také spustí interval zelené na přechodech pro chodce ve směrech **C** a **D**. Zelená na přechodech pro chodce bude trvat 0:30 min (to znamená 30 sekund, stejně tak i dále)

Ve směru **A** ve standardním režimu budou intervaly následující, zelená bude trvat 1:00 min. 7 sekund před ukončením intervalu zelené se rozsvítí signál pro opuštění křižovatky, který potom zhasne spolu s ukončením intervalu zelené.

Ve směru **B** ve standardním režimu budou intervaly následující, zelená bude trvat 0:50 min.
(po dokončení řízení těchto dvou směrů bude v obou těchto směrech červená na všech semaforech a řízení se předá dalším dvěma směrům).

Bezpečnostní mezera mezi přepnutím směrů je 0:03 min!! Je to interval mezi zastavením jednoho směru pomocí červené a zapnutím zelené ve směru druhém. Slouží k tomu, když někdo projede ještě na hranici červené, aby byla nějaká rezerva, než se rozjedou auta v druhém směru.

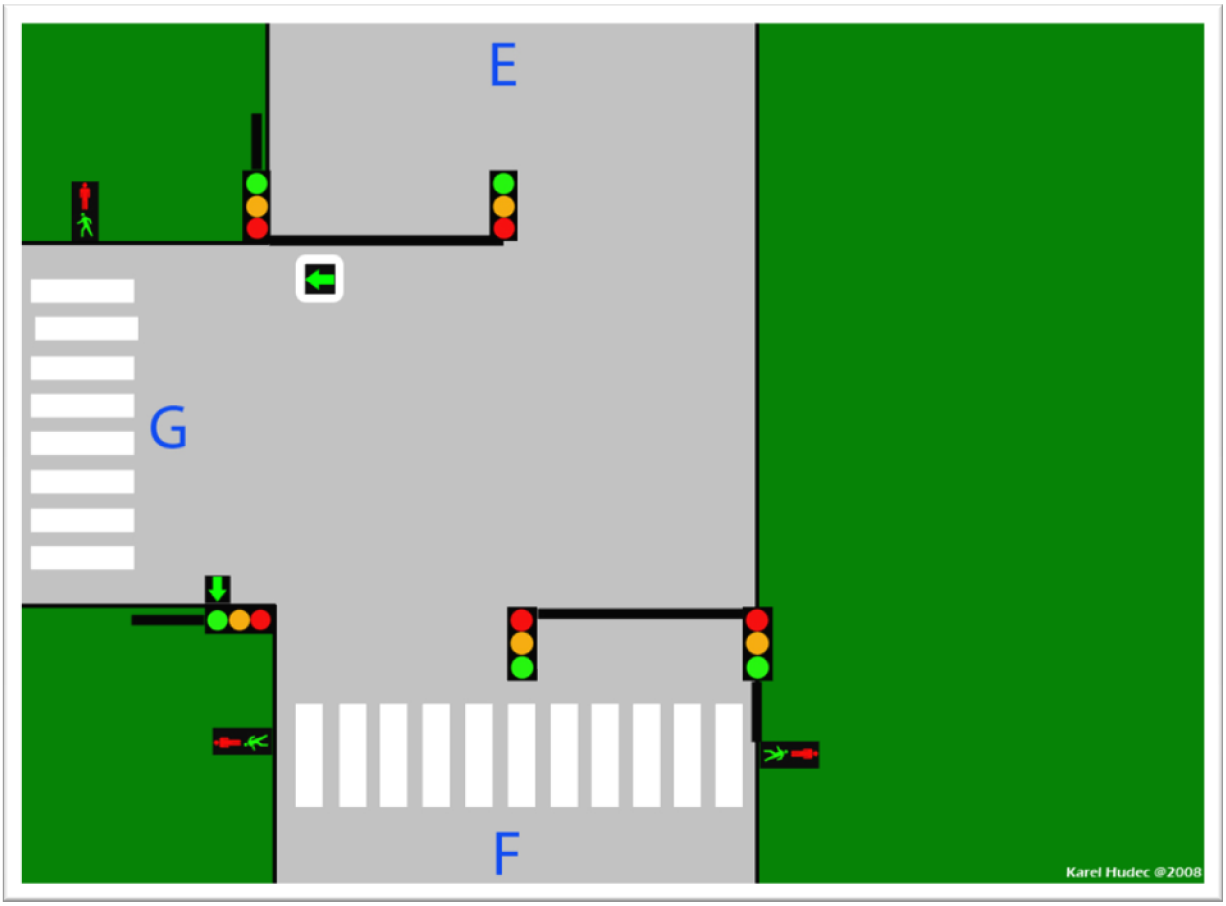
Ve směrech **C** a **D** se nejprve rozsvítí zelená signalizace ve směru **C** a to jak pro směr rovně a doprava, ke které se rozsvítí oranžový výstražný panáček, který signalizuje přednost chodců na přechodu, kam se odbočuje, tak se zelená rozsvítí i pro jízdu vlevo. Na přechodu ve směru **B** se přitom rozsvítí zelená na 0:30 min, ale ve směru **A** bude na přechodu setrvávat červená.

Ve směru **C** ve standardním režimu budou intervaly následující, zelená se na všech zařízeních rozsvítí současně, přičemž na zařízení se světly pro směr doleva bude trvat 0:25 min, pak tam naskočí červená. Na zařízení pro směr rovně a doprava bude trvat zelená 0:50 min.

Ve směru **D** naskočí zelená, jakmile se ve směru **C** rozsvítí červená pro odbočování vlevo (tehdy se také rozsvítí zelená na přechodu pro chodce ve směru **A**) a bude trvat 0:35 min a 0:10 min před koncem tohoto zeleného intervalu se rozsvítí na přechodu ve směru **A** červená a ve směru **D** šipka pro opuštění křižovatky, která pak zhasne se skončením zeleného intervalu v tomto směru.
(po dokončení řízení těchto dvou směrů bude v obou těchto směrech červená na všech semaforech a řízení se předá předchozím dvěma směrům).

Ve specifikaci jednotlivých směrů nejsou zmiňovány signály oranžové barvy a jejich intervaly, ty jsou všude standardní a jejich doba trvání mezi změnou zelené na červenou nebo naopak je 0:03 min.

5.1.2 Specifikace křižovatky II.



Obrázek 5.3 – Grafický model křižovatky II.

Řízená křižovatka II má tvar T. Pro přehlednost jsem si označil směry jako E, F, G. Směry E a F jsou brány jako hlavní průjezd křižovatkou a směr G je napojující komunikace.

Příjezd E

Příjezd E je hlavní příjezd od křižovatky II našeho uzlu. Tento směr bude pro řízení provozu využívat pouze dvou klasických semaforů. Semafory budou zobrazovat to samé. Jsou zdvojené pouze pro větší přehlednost a viditelnost. Na příjezdu z tohoto směru nebude přechod pro chodce.

Příjezd F

V tomto směru bude křižovatka řízena též dvojicí klasických semaforů, které jsou zdvojené pouze pro lepší viditelnost a přehlednost. Navíc tento směr bude obsahovat i přechod pro chodce, který bude řízen klasickým semaforem pro chodce s dvěma panáčky. Bude zde i zelená šipka pro opuštění křižovatky, která bude ulehčovat odbočování vlevo. Je zde také proto, že potom ve směru G

je semaforem řízeno zvlášť odbočování vpravo, což s touto únikovou šipkou také souvisí. (více v řízení).

Příjezd G

Pro řízení provozu z tohoto směru bude použit pouze jeden klasický semafor. Je zde totiž provoz ne tak hustý a pouze jedním pruhem, takže není třeba semafor zdvojovat kvůli lepší viditelnosti. Navíc zde bude ale zvlášť zelená šipka, která bude umožňovat odbočení vpravo i v případě, že na klasickém semaforu bude v tomto směru červená. Tento směr bude také obsahovat klasický přechod pro chodce, který bude řízen klasickými semafory pro chodce.

Řízení

Ze směru **G** se rozsvítí zelená společně se zelenou ve směru **A** a **B** křižovatky **I**. V tuto chvíli bude na semaforech ve směrech **E** a **F** červená. Zároveň se rozsvítí i zelený panáček na přechodu pro chodce ve směru **F**, který po 0:20 min přepne na červeného. Za další 0:10 min se ve směru **G** nahodí semafor do oranžové a standardně pak po 0:03 min do červené. Přesně v tento okamžik, když na semaforu ve směru **G** bude již červená, tak se začne zelený interval ve směrech **E** a **F**. Nejprve začne tím, že se k červené v těchto směrech rozsvítí ještě oranžová a potom po 0:03 min už bude svítit jen zelená. Zelený interval ve směru **E** bude trvat 0:49 min. Po této době se na semaforu ve směru **E** rozsvítí oranžová a klasicky po 0:03 min pak červená. V tuto chvílí, kdy už je na semaforu ve směru E červená se rozsvítí ve směru **G** zelená šipka povolující jízdu vpravo. Zároveň se také rozsvítí signál pro opuštění křižovatky ve směru **F**, který v tomto směru umožní nerušené odbočování vlevo.

Interval zelené ve směru **F** bude po tomto ještě pokračovat 0:38 min. Tehdy se ve směru **F** rozsvítí na semaforu oranžová a spolu s ní zhasne signál pro opuštění křižovatky ve směru **F** a také zhasne zelená šipka pro odbočování vpravo ve směru **G**. Za následující 0:03 min po tomto sena semaforu ve směru **F** rozsvítí červená a zelený interval pro směr **F** zde končí. Zároveň se na semaforu ve směru **G** rozsvítí oranžová k červené a po 0:03 min budeme mít opět na semaforu ve směru **G** zelenou, čímž začíná nový cyklus pro křižovatku **II.**

5.1.3 Specifikace navržených inteligentních řešení

Monitorování hustoty dopravy ze směru do města a reakce na ni.

Ve směru **A** bude hustota provozu monitorována čidlem a v případě hustého provozu bude systém přepnut ze standardního režimu do režimu zácpy ze směru **A**. Bude to dáno tím, že čidlo

zaznamená delší frontu ve směru **A** než 100m. Tuto hodnotu lze samozřejmě v případě potřeby pozměnit. Pokud bude tedy tato dopravní komplikace detekována, tak na ni systém zareaguje tak, že se intervaly zelené ve směrech **A a B** prodlouží o 0:30 min.

Monitorování míry ucpání trasy mezi dvěma světelnými křižovatkami dopravního uzlu.

Mezi dvěma křižovatkami uzlu bude provoz monitorován čidlem, a pokud kolona mezi křižovatkami I a II překročí hranici tři čtvrtě celé délky pozemní komunikace a bude tak hrozit ucpání mezi těmito křižovatkami, tak systém zareaguje tím, že prodlouží interval zelené ve směrech **E** a **F** 0:30 min. Pro zlepšení efektu tohoto opatření se zároveň prodlouží o tu samou dobu interval zelené ve směrech **A** a **B**.

Inteligentní průjezd vozidel s právem přednostní jízdy.

Řidič vozidla s právem přednostní jízdy bude mít možnost při přiblížení se ke křižovatce vyslat signál, který bude inteligentní křižovatkou přijat a ihned bude na světelných signalizačních světlech nastavena na dobu 0:30 min všude červená (včetně přechodů pro chodce). Tam, kde svítila zelená, bude červené předcházet klasicky oranžový signál po standardní dobu 0:03 min.

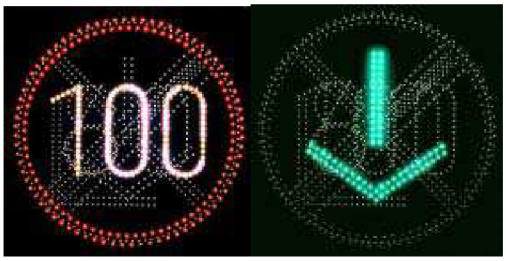
Automatické řízení denního a nočního režimu křižovatky.

Systém řízení uzlu bude automaticky přepínat dopravní uzel mezi režimy denním a nočním. Noční režim bude nastaven na dobu mezi 23:00 – 05:00 hod. Mimo tento časový interval bude uzel v denním režimu. Pokud bude nastaven noční režim, znamená to, že na všech klasických semaforech bude pouze blikat oranžová barva v intervalech po 0:01 min a tím pádem se bude v tuto dobu doprava uzlem řídit pouze svislými dopravními značkami. Přechody pro chodce budou rovněž vyřazeny z činnosti.

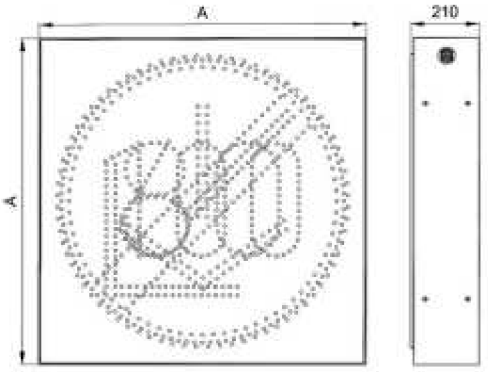
Včasné upozornění řidičů na snížení rychlosti v závislosti na případné tvořící se koloně před křižovatkou za pomoci proměnné dopravní značky.

Systém bude obsahovat proměnnou dopravní značku. Která bude umístěna 300 metrů před křižovatkou I. Na příjezdu z ulice Sportovní(**A**) a bude standardně zobrazovat nejvyšší povolenou rychlost 60km/h. Ovšem jestliže bude systémem řízení dopravního uzlu nastaven příznak, že se tvoří ve směru **A** kolona, tak se tato značka přepne na nejvyšší povolenou rychlost pouze 40km/h. Tím zajistí menší rychlost vozidel přijíždějících ke stojící koloně a tím pádem se zvýší bezpečnost provozu.

Pro realizaci jsem vybral proměnnou dopravní značku typu PDZ/LED od firmy eltodo. [10]



Obrázek 5.4 – Proměnná dopravní značka PDZ/LED. Zdroj [10]



Obrázek 5.5 – Rozměrový náčrt PDZ/LED. Zdroj[10]

Technické parametry navržené proměnné dopravní značky.

Přesný název je „Proměnná dopravní značka/ PDZ LED“. Jako světelný zdroj jsou použity vysoce svítivé LED diody RS485. Příkon je 230V/50Hz(20-125W podle provedení). Krytí IP65. Provozní teplota je -30°C - +55°C. Hmotnost 11-100kg (podle provedení). Rozměry 1400x2000mm, 1400x1000mm, 900x900mm, 900x400mm(dle požadavku). V souladu s **EN 12966-1** „Svislé dopravní značky – Dopravní značky s proměnnými symboly“. **STN EN 12899-1** „Stálé svislé dopravní značení – Část 1: Stálé dopravní značky“.

Automatické přepnutí do nouzového režimu při zjištění chyby v systému a přivolání technika.

Ještě není jisté, zda bude tato funkce do systému implementována, nicméně by šlo pouze o to, že když by se v řídicím systému vyskytla jakákoliv chyba nebo problém, tak by systém nahodil příznak, který by celý dopravní uzel přepnul do nouzového režimu a zároveň by vyslal signál na dispečink, že je potřeba zásah zvenčí.

Nouzový režim

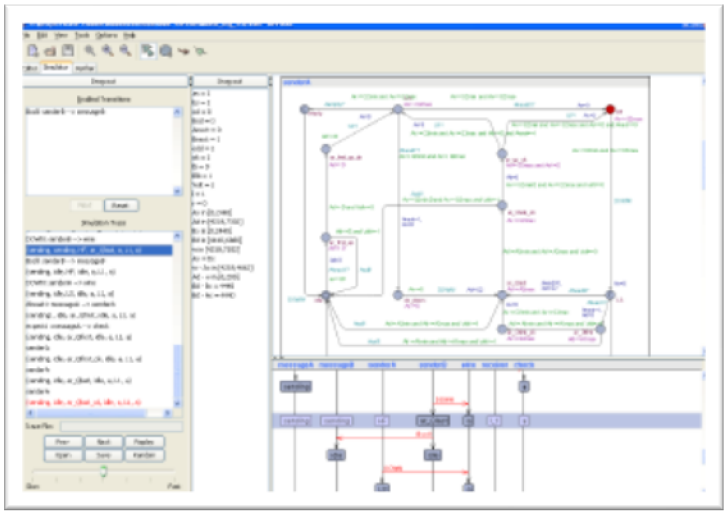
Křižovatku je možno přepnout do tzv. nouzového režimu, což zajistí, že na všech světelných signalizačních prvcích (krom přechodů pro chodce, ty budou vzpnuty) bude blikat oranžové světlo. Interval blikání bude 0:01 min. Tehdy se provoz bude řídit svislými dopravními značkami, které již nejsou součástí našeho systému.

5.2 Model a verifikace navrženého systému

5.2.1 Modelovací a verifikační nástroje

- **UPPAAL**

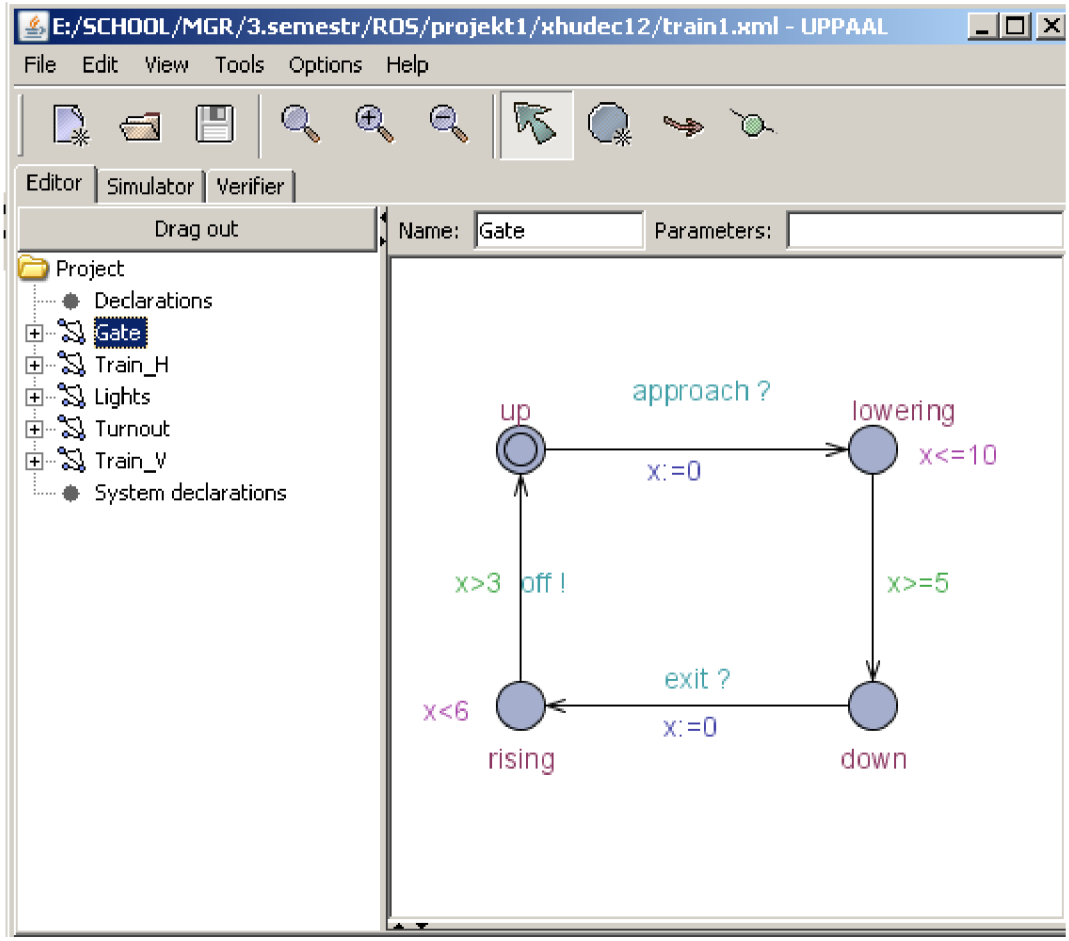
[12] *UPPAAL* je integrovaný nástroj pro modelování, simulaci a verifikaci *RT* systémů. Tento nástroj vznikl na Aalborg university v Dánsku a na Uppsala university ve Švédsku. Tento nástroj je vhodný pro systémy, které mohou být modelovány jako kolekce nedeterministických procesů s konečnou řídící strukturou a skutečnou hodnotou hodin, komunikující přes kanály nebo sdílené proměnné. Typické oblasti aplikace obsahují *RT* řízení a komunikační protokoly, kde časové aspekty jsou kritické.



Obrázek 5.6 – Prostředí nástroje UPPAAL

UPPAAL se skládá ze tří hlavních částí: Jazyk pro popis modelu, simulátor a kontrola modelu. Jazyk pro popis modelu je nedeterministický příkazový jazyk s datovými typy. Slouží jako modelovací nebo návrhový jazyk pro popis chování systému ve formě časovaného automatu s proměnnými. Simulátor je validačním nástrojem, který dovoluje zkoumání možných dynamických kroků modelovaného systému a umožňuje nenákladné odhalení různých chyb již v rané fázi návrhu za pomocí verifikace. Systém kontroly modelu může kontrolovat invarianty a možnosti dostupnosti stavů prozkoumáváním modelu systému.

Ke zjednodušení modelování a následného ladění *UPPAAL* nabízí možnost automatického generování diagnostické trasy, která zobrazuje proč je nebo není určitá podmínka systému splněna. Diagnostické trasy generované systémem kontroly modelu mohou načteny automaticky do simulátoru, kde pak mohou být použity pro vizualizaci a kontrolu systému.



Obrázek 5.7 – Ukázka modelu v UPPAAL

Na obrázku je ukázka modelu závory železničního přejezdu, kterou jsem vytvořil. Jsou zde využity kanály pro synchronizaci a časové invarianty.

UPPAAL je také multiplatformní (Linux, SunOS a Windows 95/98/NT/XP) nástroj. Sám jsem měl možnost ještě před diplomovou prací vyzkoušet si jej v předmětu ROS. Zdál se mi jako vhodný, ale ne však nejvhodnější pro moji aplikaci inteligentního dopravního uzlu. Nicméně jsem v něm udělal základní modely semaforů a vyzkoušel základní verifikační dotazy. Kompletní model a verifikaci systému jsem potom vytvořil za pomocí nástroje *TimesTool*.

- **TIMESTOOL**

[13] *TimesTool* je nástroj pro modelování a implementaci vestavěných systémů. Tento nástroj umožňuje modelování, plánovací analýzu, syntézu plánovače a dokonce za určitých podmínek umí vytvořit i zdrojový kód a spustitelný program.

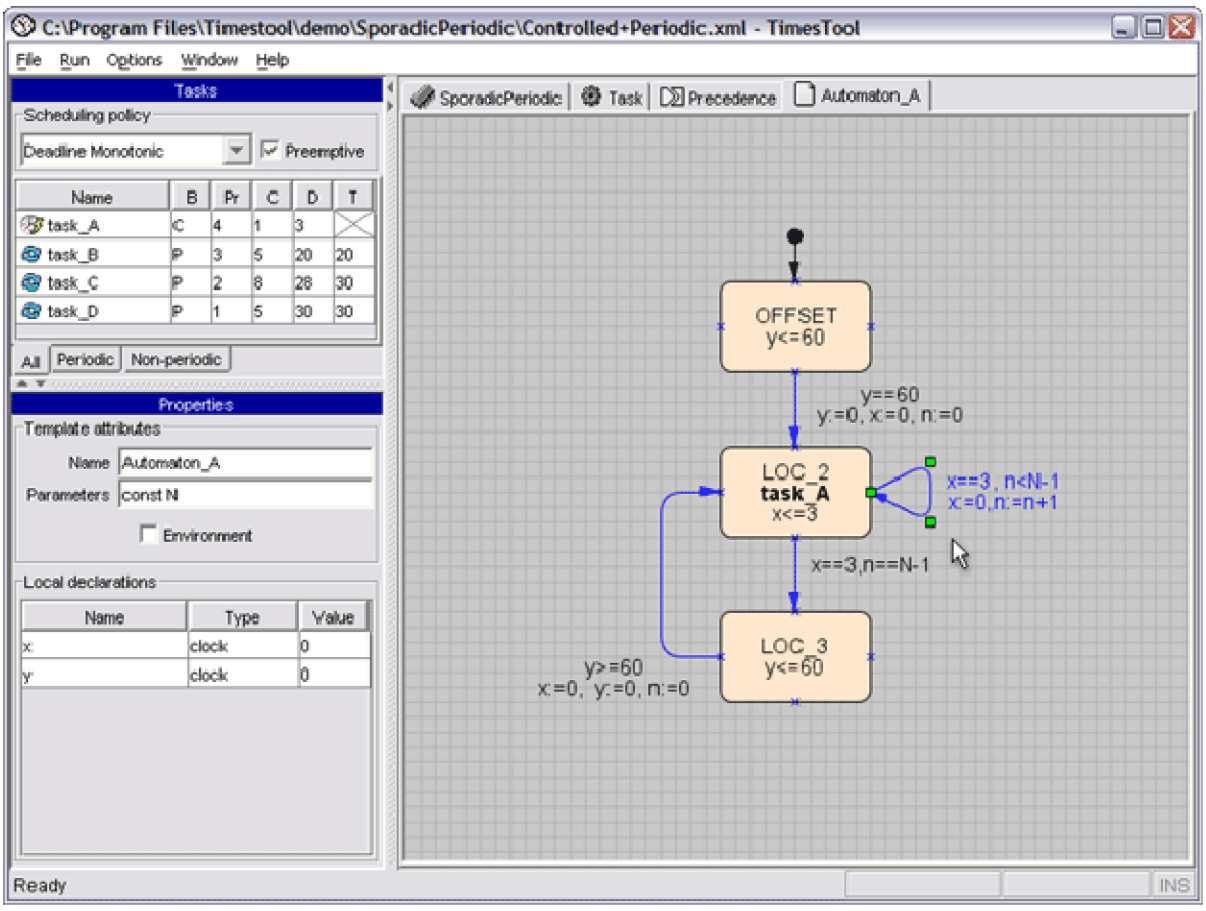
V současnosti *TimesTool* podporuje generaci zdrojového kódu pouze pro platformu *LegoOS* [3]. Specifikace systému v *TimesTool* se skládá ze třech částí: řídící automat modelovaný jako síť

časovaných automatů rozšířená o úlohy, tabulka úloh s informacemi o procesech spouštěných když řídící automat změní pozici a plánovací politika.

Tento nástroj umožňuje modelování a plánovací analýzu pro vestavěné RT systémy. Byl vyvinut na Uppsala university [13]. Je vhodný pro systémy, které mohou být popsány jako množina preemptivních nebo nepreemptivních úloh, které jsou spouštěny periodicky nebo sporadicky v závislosti na čase externích událostí. Tento modelovací systém poskytuje grafické rozhraní pro editaci a simulaci, také umožňuje analýzu plánovatelnosti.

Hlavní rysy *TimesTool*:

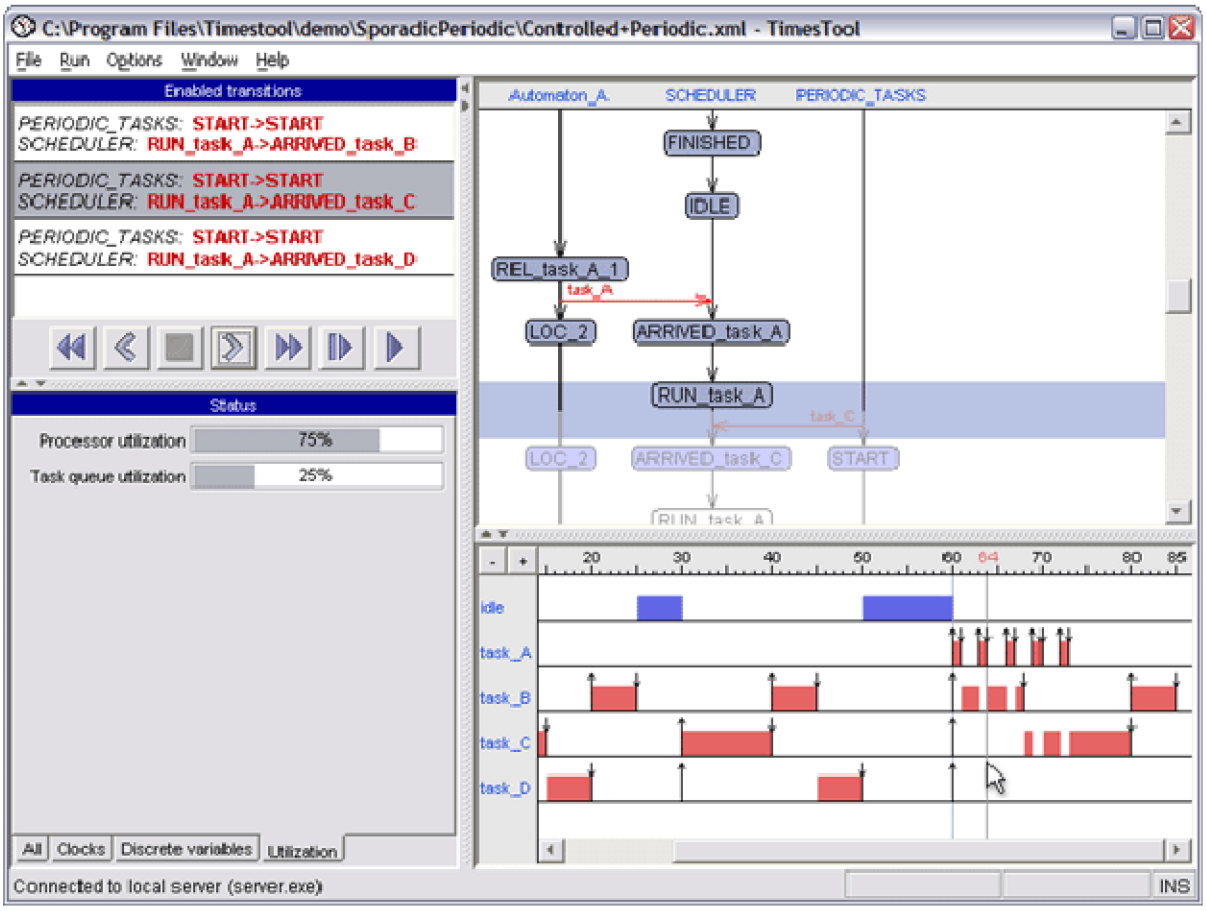
Grafický editor pro časované automaty rozšířené o úlohy, který umožňuje uživateli modelovat systém a chování prostředí systému. Uživatel může taky specifikovat množinu preemptivních nebo nepreemptivních úloh s parametry jako jsou časová mez, čas běhu, priorita a podobně.



Obrázek 5.8 – Prostředí nástroje TimesTool Editor. Zdroj [13]

Na obrázku je vidět výřez obrazovky grafického editoru. Nalevo nahoře definujeme úlohy a jejich parametry uvedené výše a na pravé straně je grafické znázornění systému.

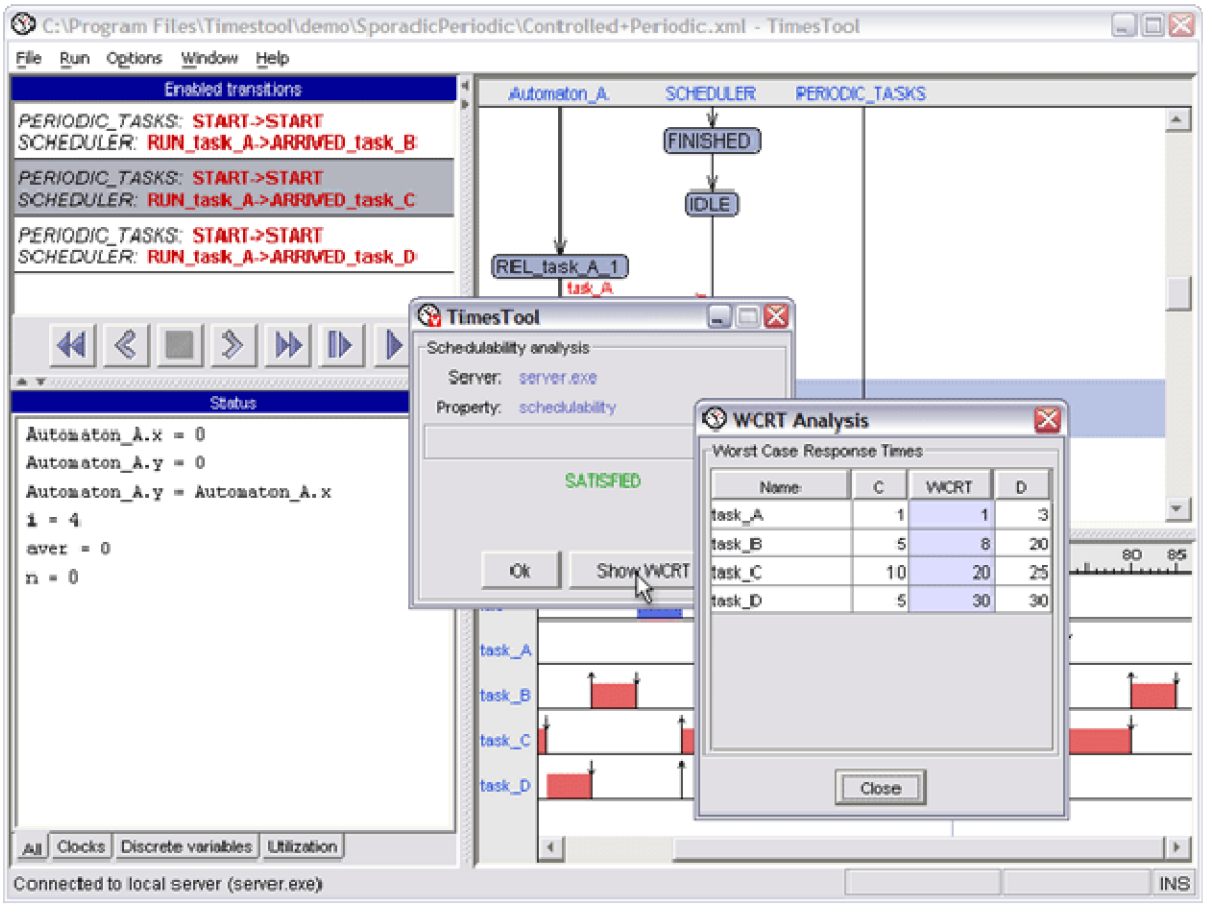
Simulátor, ve kterém může uživatel validovat dynamické chování systému a může vidět, jak se úkoly vykonávají v závislosti na jejich parametrech a na aktuální plánovací politice. Simulátor ukazuje grafickou reprezentaci generované trasy. Ukazuje časové body, kdy jsou spouštěny, suspendovány a dokončovány jednotlivé úlohy.



Obrázek 5.9 – Prostředí nástroje TimesTool Simulátor. Zdroj [13]

V levé horní části obrázku je vidět jak jsou plánovačem spouštěné úlohy. Napravo potom nahoře grafická reprezentace běhu a dole časový plán.

Verifikátor pro plánovací analýzu, který se používá ke zkontrolování, jestli jsou dostupné všechny stavy nebo zda je celý systém plánovatelný. Symbolické algoritmy byly vyvinuty založené na DBM [13] technikách a implementovány v základech jako u verifikátoru z nástroje *UPPAAL*.



Obrázek 5.10 – Prostředí nástroje TimesTool Simulátor. Zdroj [13]

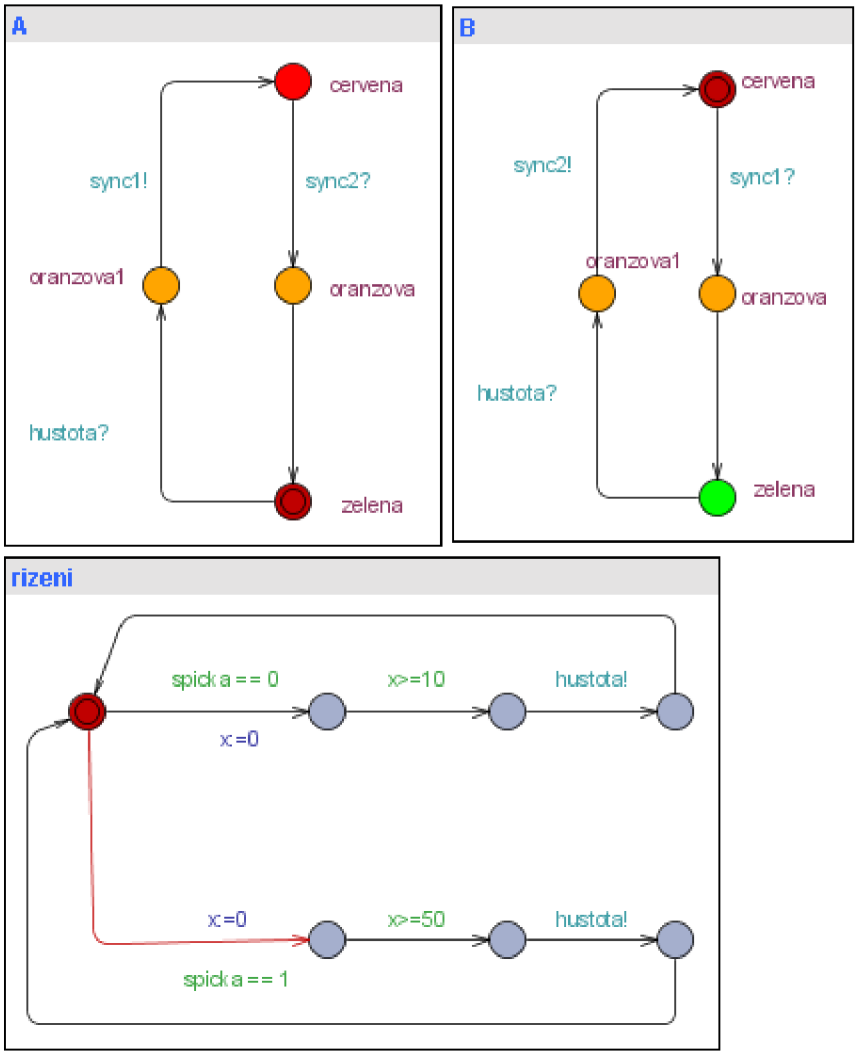
Generátor kódu modelu pro automatickou syntézu kódu v jazyce C na platformu *brickOS* [3]. Pokud je model automatu plánovatelný v závislosti na plánovacím analyzátoru, tak vykonávání generovaného kódu splní všechny podmínky a časové limity specifikované v modelu a v jednotlivých úkolech.

Modelovací a verifikační nástroj *TimesTool* jsem si vybral jako hlavní pro moji aplikaci inteligentního dopravního uzlu. Měl jsem se s ním možnost seznámit již dříve v předmětu ROS.

5.2.2 Model systému

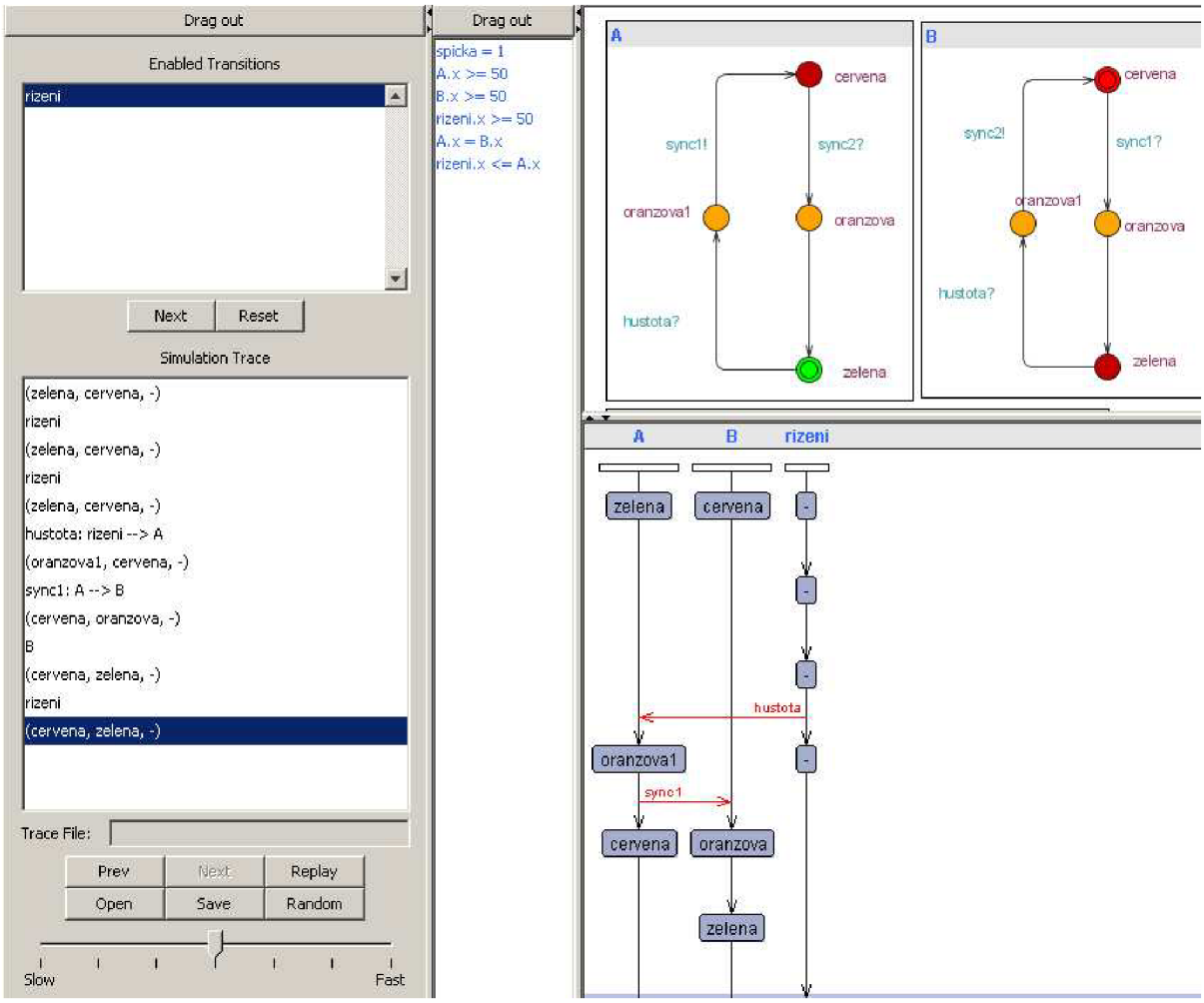
- **Základní model v UPPAAL**

V modelovacím a verifikačním nástroji *UPPAAL* jsem si vyzkoušel udělat základní jednoduché modely spíše na vyzkoušení. Následující obrázky demonstrují, jak jsem tvořil modely.



Obrázek 5.11 – Základní model semaforu a řízení

Na obrázcích jsou nejzákladnější modely v *UPPAAL*. Reprezentují klasické semafory ve dvou směrech a zjednodušený systém jejich řízení na základě jedné podmínky. Tyto modely jsou pouze triviální. Kompletní model jsem vytvořil v nástroji TimesTool.



Obrázek 5.12 – Simulace v UPPAAL

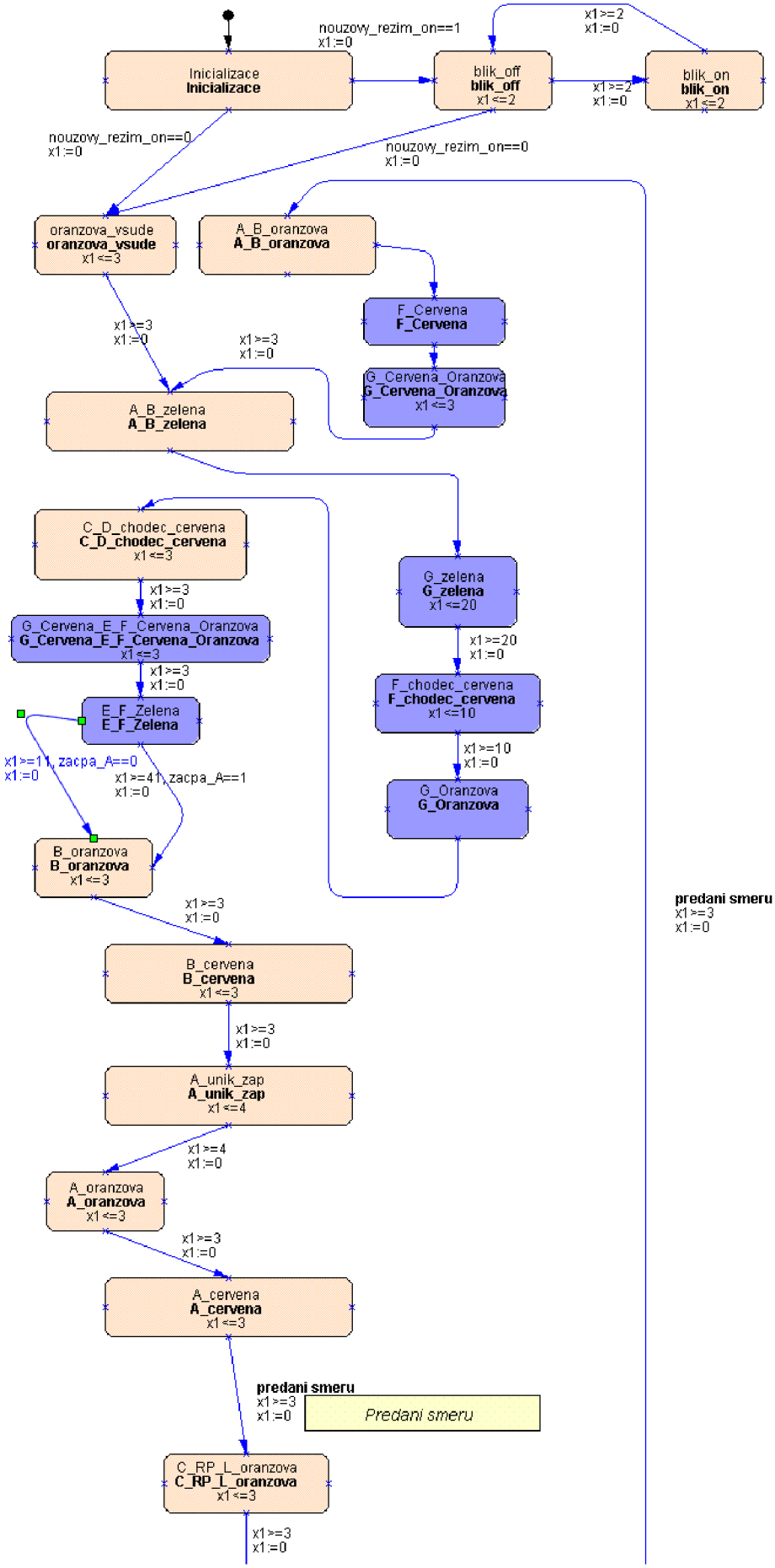
Na obrázku je ukázka simulace systému v *UPPAAL*. Můžeme zde vidět jak grafické vyznačení aktuálních stavů, tak simulační trasu, nebo stavy jednotlivých proměnných systému.

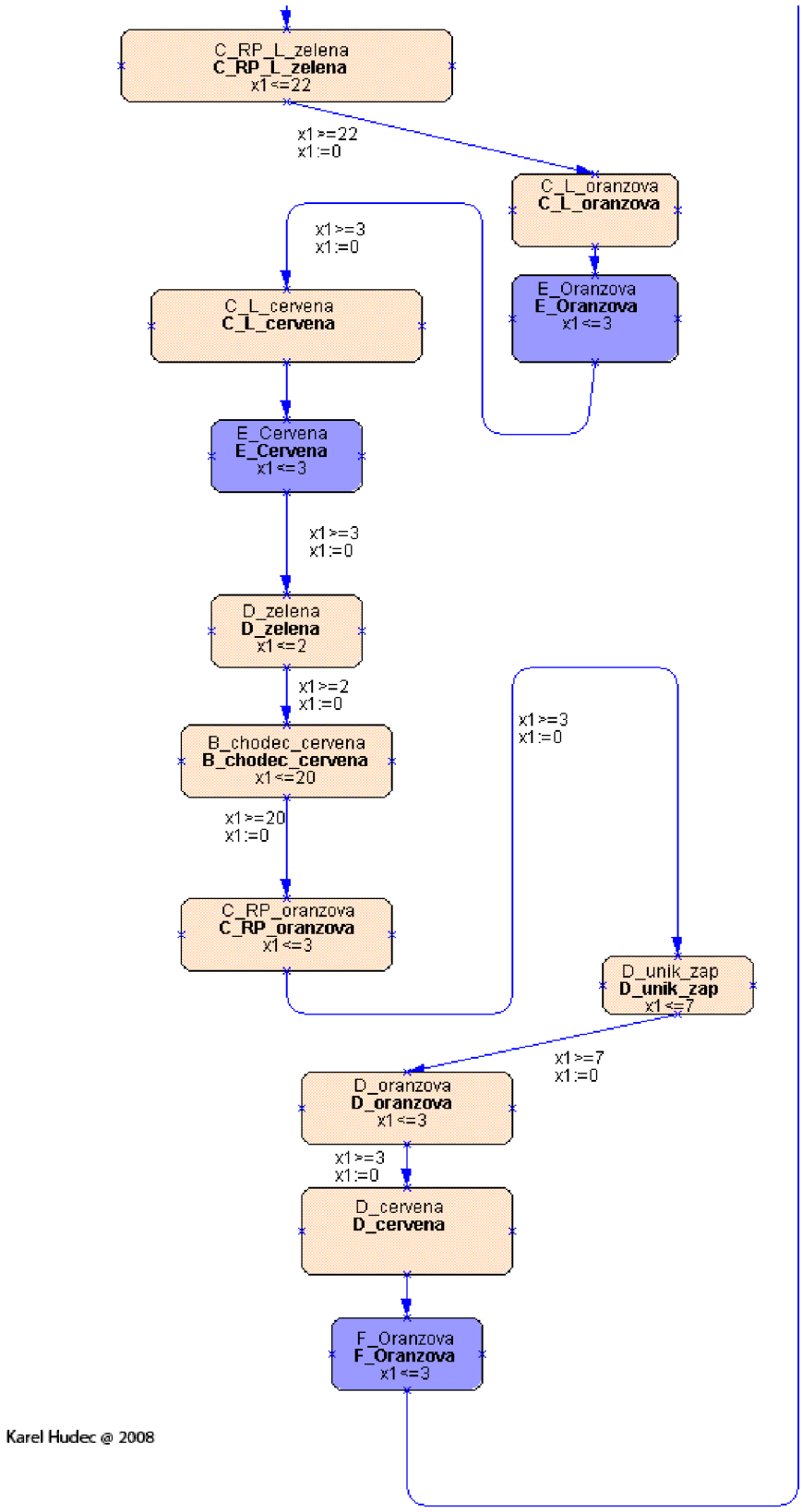
Soubory modelu v *UPPAAL* jsou obsaženy na CD přiloženém k diplomové práci

- **Kompletní model systému v TIMESTOOL**

V tomto nástroji jsem vytvořil hlavní model celého řídicího systému, protože toho dokáže mnohem více než *UPPAAL* a přišel mi více přehledný a pro moji potřebu vhodnější.

Model v tomto nástroji jsem tvořil postupně a zároveň vytvářel proměnné, které byly třeba a stejně tak jednotlivé úlohy včetně jejich parametrů.

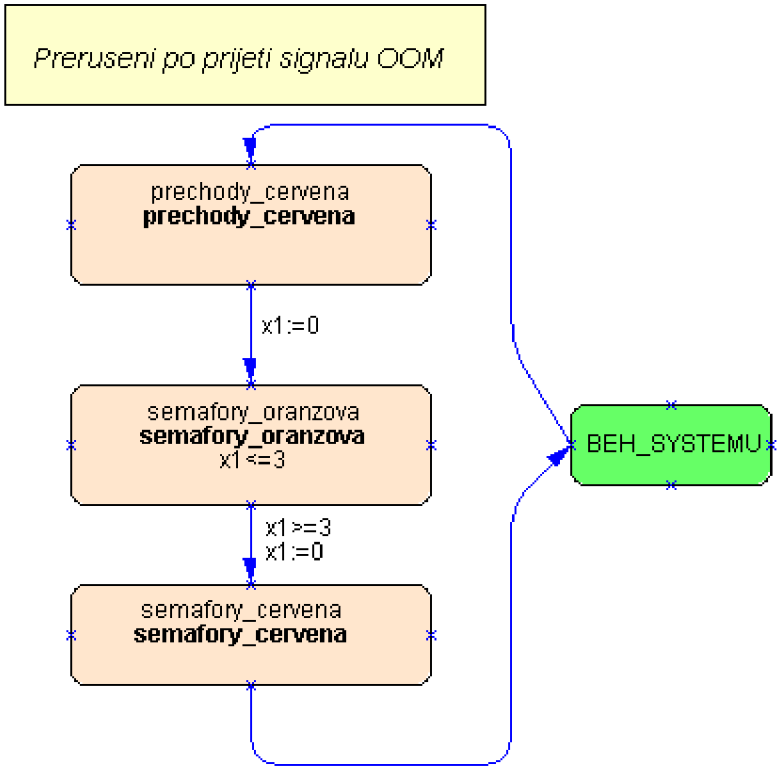

Inicializace
Inicializace
nouzovy_rezim_on==1
x1:=0
blik_off
blik_off
x1<=2
x1>=2
x1:=0
blik_on
blik_on
x1<=2
nouzovy_rezim_on==0
x1:=0
nouzovy_rezim_on==0
x1:=0
oranzova_vsude
oranzova_vsude
x1<=3
A_B_oranzova
A_B_oranzova
F_Cervena
F_Cervena
x1>=3
x1:=0
x1>=3
x1:=0
G_Cervena_Oranzova
G_Cervena_Oranzova
x1<=3
A_B_zelena
A_B_zelena
C_D_chodec_cervena
C_D_chodec_cervena
x1<=3
G_zelena
G_zelena
x1<=20
x1>=3
x1:=0
G_Cervena_E_F_Cervena_Oranzova
G_Cervena_E_F_Cervena_Oranzova
x1<=3
x1>=20
x1:=0
x1>=3
x1:=0
F_chodec_cervena
F_chodec_cervena
x1<=10
E_F_Zelena
E_F_Zelena
x1>=10
x1:=0
x1>=11, zacpa_A==0
x1:=0
x1>=41, zacpa_A==1
x1:=0
G_Oranzova
G_Oranzova
B_oranzova
B_oranzova
x1<=3
x1>=3
x1:=0
predani smeru
x1>=3
x1:=0
B_cervena
B_cervena
x1<=3
x1>=3
x1:=0
A_unik_zap
A_unik_zap
x1<=4
x1>=4
x1:=0
A_oranzova
A_oranzova
x1<=3
x1>=3
x1:=0
A_cervena
A_cervena
x1<=3
predani smeru
x1>=3
x1:=0
Predani smeru
C_RP_L_oranzova
C_RP_L_oranzova
x1<=3
x1>=3
x1:=0



Obrázek 5.12 – Model systému v TimesTool

Kompletní model je vytisknut v přehledné velikosti jako příloha diplomové práce a je také obsažen na položeném CD.

V modelu jsou odlišnou barvou od sebe zvýrazněny uzly týkající se křižovatky I. A II. Dále k modelu patří i část, kdy se obsluhuje takzvaný signál *OOM* . Ten se používá při průjezdu vozidel s právem přednostní jízdy. Úkoly obsažené v jeho obsluze by měli mít vyšší prioritu.



Obrázek 5.13 – Model obsluhy OOM v TimesTool

Dle obrázku modelu je tedy vidět, že při přijetí signálu *OOM* přerušíme běh systému, provedeme obsluhu v podobě tří za sebou jdoucích úloh a následně se vrátíme zpět do bodu, kde byl systém přerušen. Pro lepší bezpečnost silničního provozu se při přechodu zpět, tam kde bude jiná barva než stávající červená, rozsvítí předtím na standardní dobu 0:03 min oranžová z důvodu nenarušení plynulosti dopravy.

Při vytváření celého modelu jsem postupně kompletoval i tabulku úloh. Ta je znázorněna na následujícím obrázku. Jako plánovací politika bylo zvoleno „User defined Priorities“ což znamená uživatelem definované priority. Jelikož spouštění jednotlivých úloh bylo řízeno výhradně časovaným automatem, bylo vhodné použít tuto plánovací politiku. Jednotlivé priority jsem zvolil tak, že standardní úlohy mají shodnou prioritu 0 a úkoly obsluhy *OOM* mají z důvodu bezpečnosti prioritu nastavenu na 10(vyšší priorita než 0), kdyby náhodou došlo rozhodování podle priority.

User-defined Priorities ☑ Preemptive

Name	B	Pr▲	C	D	T
Inicializace	C	0	1	inf	
blik_on	C	0	1	inf	
blik_off	C	0	1	inf	
A_B_zelena	C	0	1	inf	
A_unikova_sipka	C	0	1	inf	
C_D_chodec_cervena	C	0	1	inf	
A_unik_zap	C	0	1	inf	
B_cervena	C	0	1	inf	
A_cervena	C	0	1	inf	
B_oranzova	C	0	1	inf	
A_oranzova	C	0	1	inf	
C_RP_L_zelena	C	0	1	inf	
C_RP_L_oranzova	C	0	1	inf	
C_L_oranzova	C	0	1	inf	
C_L_cervena	C	0	1	inf	
B_chodec_cervena	C	0	1	inf	
D_zelena	C	0	1	inf	
D_unik_zap	C	0	1	inf	
D_oranzova	C	0	1	inf	
D_cervena	C	0	1	inf	
oranzova_vsude	C	0	1	inf	
A_B_oranzova	C	0	1	inf	
C_RP_oranzova	C	0	1	inf	
E_oranzova	C	0	1	inf	
G_zelena	C	0	1	inf	
F_chodec_cervena	C	0	1	inf	
G_cervena_oranzova	C	0	1	inf	
G_oranzova	C	0	1	inf	
G_cervena_E_F_cervena_oranzova	C	0	1	inf	
E_F_zelena	C	0	1	inf	
E_cervena	C	0	1	inf	
F_oranzova	C	0	1	inf	
F_cervena	C	0	1	inf	
prechody_cervena	C	10	1	inf	
semafory_oranzova	C	10	1	inf	
semafory_cervena	C	10	1	inf	

Obrázek 5.14 – Seznam namodelovaných úloh systému v TimesTool

Na obrázku 5.14 je vidět seznam úloh. Nastavované parametry v jednotlivých sloupcích jsou:

B (behavior) – Chování úlohy, zde jsou možnosti C,P,S - automatem řízená, periodická nebo sporadická. První je řízená časovaným automatem, periodická se opakuje stále v nastavených intervalech a sporadická se spouští potom dle nastaveného plánovacího mechanismu.
P (priority) – Priorita úlohy, zde čím vyšší číslo, tím vyšší priorita.
C (execution time) – Doba vykonávání úlohy v časových jednotkách v nejhorším případě.
D (deadline) – Nejzasší možná doba vykonání úlohy.
T (period) – Perioda úlohy. Pro kontrolované úlohy se nepoužívá.

Aby mezi sebou mohly jednotlivé úlohy komunikovat a nastavovat různé příznaky systému, tak bylo třeba zavést sadu proměnných. Za pomocí těchto proměnných probíhala také simulace a následná verifikace systému. Tyto proměnné jsou uvedeny v následující tabulce.

Global declarations

Name	Type	Value	Env
A_semafor_oranzova_on	int[0,1]	0	☐
B_semafor_oranzova_on	int[0,1]	0	☐
nouzovy_rezim_on	int[0,1]	0	☐
C_semafor_RP_oranzova_on	int[0,1]	0	☐
D_semafor_oranzova_on	int[0,1]	0	☐
A_semafor_zelena_on	int[0,1]	0	☐
B_semafor_zelena_on	int[0,1]	0	☐
A_semafor_cervena_on	int[0,1]	0	☐
B_semafor_cervena_on	int[0,1]	0	☐
x2	clock	0	☐
C_chodec_zelena_on	int[0,1]	0	☐
D_chodec_zelena_on	int[0,1]	0	☐
A_chodec_cervena_on	int[0,1]	0	☐
B_chodec_cervena_on	int[0,1]	0	☐
C_chodec_cervena_on	int[0,1]	0	☐
D_chodec_cervena_on	int[0,1]	0	☐
A_unik_on	int[0,1]	0	☐
C_semafor_RP_cervena_on	int[0,1]	0	☐
A_chodec_zelena_on	int[0,1]	0	☐
B_chodec_zelena_on	int[0,1]	0	☐
C_chodec_oranzova_on	int[0,1]	0	☐
C_semafor_RP_zelena_on	int[0,1]	0	☐
C_semafor_L_zelena_on	int[0,1]	0	☐
C_semafor_L_oranzova_on	int[0,1]	0	☐
C_semafor_L_cervena_on	int[0,1]	0	☐
D_semafor_cervena_on	int[0,1]	0	☐
D_semafor_zelena_on	int[0,1]	0	☐
D_unik_on	clock	0	☐
zacpa_A	int[0,1]	0	☐
E_semafor_cervena_on	int[0,1]	0	☐
E_semafor_zelena_on	int[0,1]	0	☐
E_semafor_oranzova_on	int[0,1]	0	☐
F_semafor_cervena_on	int[0,1]	0	☐
F_semafor_zelena_on	int[0,1]	0	☐
F_semafor_oranzova_on	int[0,1]	0	☐
F_chodec_cervena_on	int[0,1]	0	☐
F_chodec_zelena_on	int[0,1]	0	☐
F_unik_on	int[0,1]	0	☐
G_semafor_zelena_on	int[0,1]	0	☐
G_semafor_cervena_on	int[0,1]	0	☐
G_semafor_oranzova_on	int[0,1]	0	☐
G_sipka_on	int[0,1]	0	☐

Obrázek 5.15 – Seznam proměnných v modelu systému v TimesTool

Na obrázku 5.15 je uveden seznam proměnných. V jednotlivých sloupcích se ke každé proměnné nastavuje typ, počáteční hodnota a nastavuje se, zda je proměnná součástí prostředí.

Příklad:

Název proměnné: `A_semafor_oranzova_on`

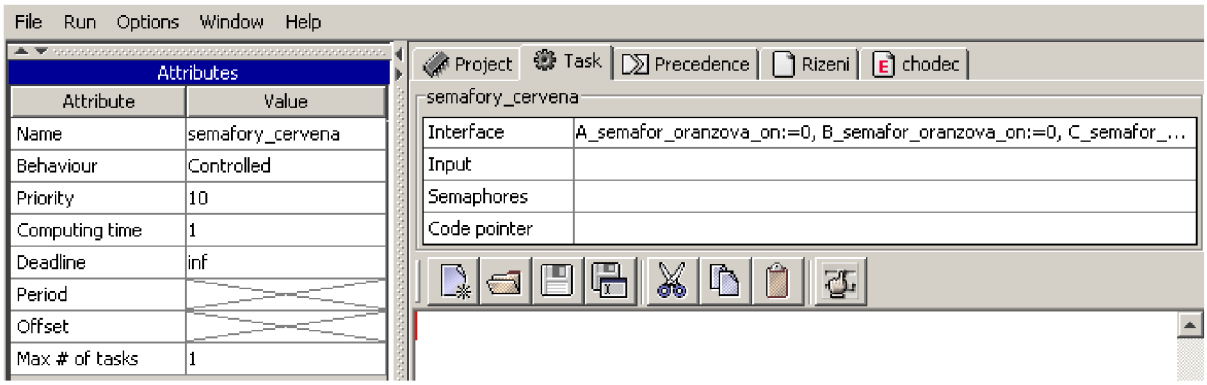
Typ proměnné: `int[0,1]`

Počáteční hodnota: `0`

Proměnná prostředí: `ne`

Použitý typ `int[0,1]` integer v rozmezí 0 – 1. Dalším použitým typem je `clock` , což je proměnná použitá pro hodiny.

Dále bylo třeba říct každé úloze v modelu, co bude vykonávat, to se provádí v editoru úloh.



Obrázek 5.16 – Editace úlohy modelu v TimesTool

Zde máme editovanou jednu úlohu. V řádku interface potom klasickými příkazy napíšeme co má úloha při svém spuštění provádět. V tomto případě šlo o nastavování různých proměnných a příznaků, které se potom využívali při simulaci a verifikaci. Konkrétně interface u této úlohy obsahuje tyto příkazy:

```
A_semafor_oranzova_on:=0, B_semafor_oranzova_on:=0,
C_semafor_L_oranzova_on:=0, C_semafor_RP_oranzova_on:=0,
D_semafor_oranzova_on:=0, A_semafor_cervena_on:=1,
B_semafor_cervena_on:=1, C_semafor_L_cervena_on:=1,
C_semafor_RP_cervena_on:=1, D_semafor_cervena_on:=1
```

Tento kód rozsvítí na všech klasických semaforech červenou. Je zde vidět jakým způsobem jsou nastavovány příslušné proměnné, tím pádem zapínány a vypínány jednotlivá světla na semaforech.

Model tedy máme kompletní, nyní následuje simulace chodu systému a verifikace. Použil jsem opět obou nástrojů. Nástroj *UPPAAL* jsem použil opět spíše v základech na odzkoušení.

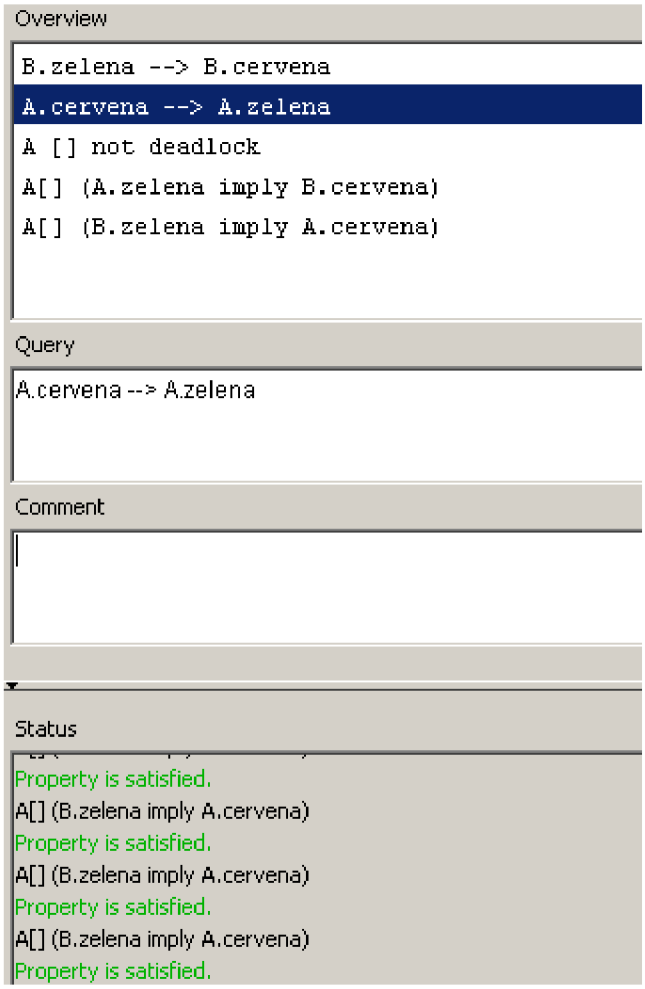
5.2.3 Verifikace

- **Verifikace základního modelu v UPPAAL**

Pro verifikaci základního modelu, který byl později rozšířen pomocí nástroje *TimesTool*, jsem použil verifikátoru aplikace *UPPAAL*.

Ukázky jednotlivých dotazů ve verifikátoru:

B.zelena --> B.cervena	*Bude po zelené ve směru B někdy následovat červená?*
A.cervena --> A.zelena	*Bude po červené ve směru A následovat někdy zelená?*
A [] not deadlock	*Může systém uváznout?*
A[] (A.zelena imply B.cervena)	*Bude ve směru B červená, když v A bude zelená?*
A[] (B.zelena imply A.cervena)	*Bude ve směru A červená, když v B bude zelená?*



Obrázek 5.17 – Ukázka verifikátoru v UPPAAL

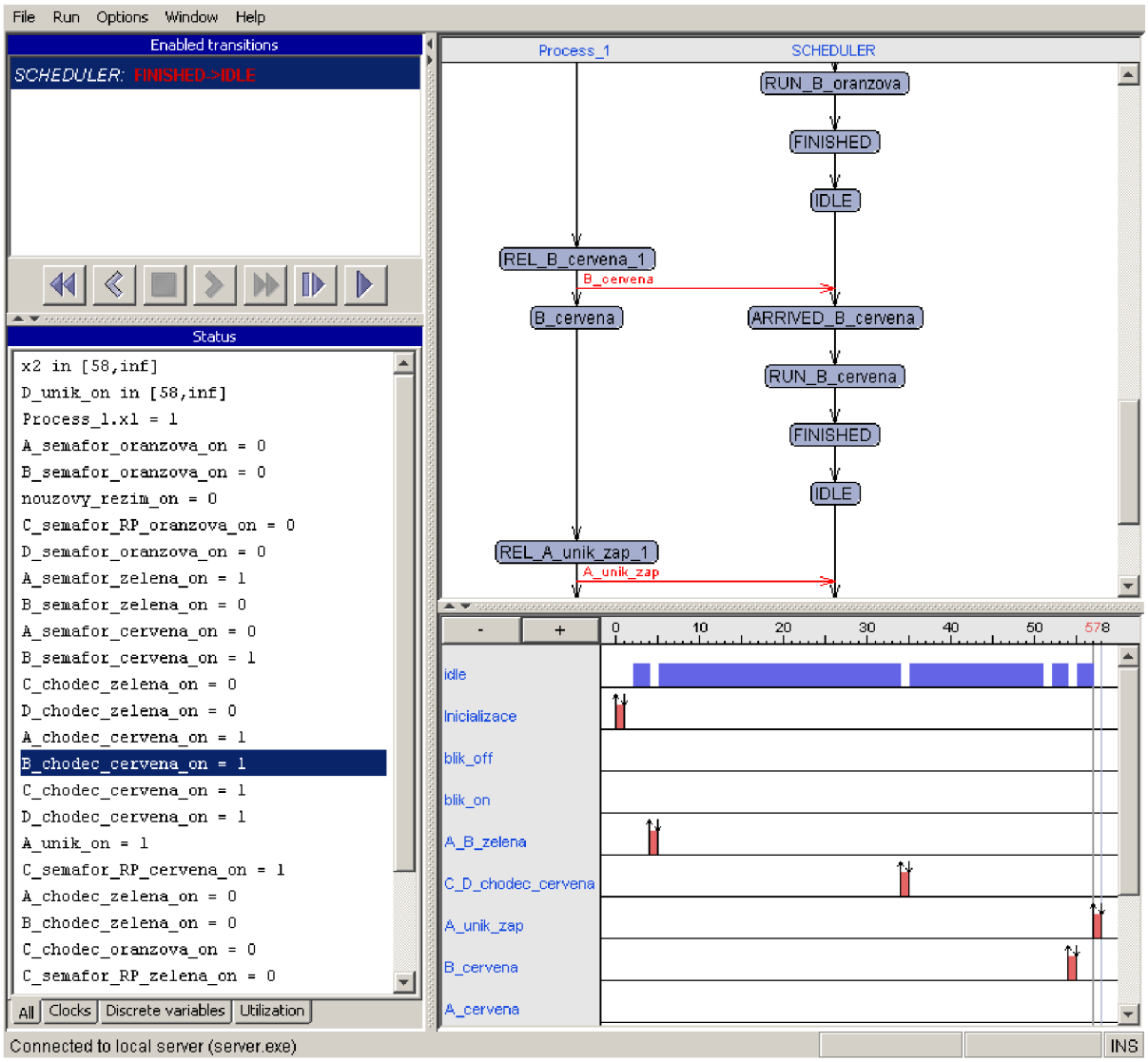
Na obrázku je uvedeno několik málo verifikačních formulí, které jsem pomocí tohoto nástroje ověřoval. Výsledek je vidět ve spodní části. Zde například byly všechny testované vlastnosti s pozitivním výsledkem.

- **Verifikace modelu v TIMESTOOL**

Pomocí nástroje *TimesTool* jsem na kompletním modelu nejdříve provedl kontrolu syntaxe, což pomůže odhalit některé chyby, které vznikly při tvorbě modelu nebo při psaní kódu v jednotlivých úlohách. Následovala analýza plánovatelnosti, což nástroj *TimesTool* také umožňuje.

Následovala simulace celého systému, kde bylo možno zkontrolovat, jestli se jednotlivé úlohy spouští správně a ve správný čas a jestli provádí správné akce. To ve výsledku bude znamenat, že na jednotlivých semaforech bude opravdu ta barva, která má být. Simulátor nástroje *TimesTool* je velice přehledný a vcelku dobře se s ním pracuje. Co bych mu vytkl, je to, že během simulace už neumožňuje ručně zasahovat do proměnných a měnit jejich obsah. Toto je tedy možné pouze na začátku před spuštěním simulace.

Ještě jsem se také setkal s problémem, že někdy prostě simulátor nenaběhl z důvodu chyby serveru. S tímto problémem jsem se dlouhou dobu snažil vypořádat, jenže se nepodařilo, a jelikož *TimesTool* je zatím pouze jen jako beta verze, tak se opravdu spíše jedná o nějakou chybu přímo v tomto nástroji.



Obrázek 5.18 – Simulace systému v TimesTool

Zde na obrázku máme ukázku simulace našeho systému. Na levé straně můžeme vidět ovládací prvky simulace a pod nimi se nachází tabulka proměnných, kde můžeme kontrolovat, zda se nám v průběhu simulace dobře nastavují příznaky a ostatní proměnné. Na pravé straně je pak nahoře diagram, který je generován v reálném čase běhu simulace a ukazuje návaznosti jednotlivých spouštěných úloh procesu a jejich zpracovávání plánovačem. Napravo dole je potom opět v reálném čase běhu simulace vykreslován graficky časový průběh jednotlivých úloh, jejich začátky a konce jsou znázorněny malými šipkami. Lze zde odečíst, kdy je systém nevytížen a v jakých časech se vykonávají jednotlivé úlohy.

Simulací lze krokovat, různě ji pozastavovat a odečítat potřebné kontrolní hodnoty. Zde se konkrétně jedná o spuštění prvních několika úloh od startu systému. Inicializace, A_B_zelená,… atd. Časy spuštění těchto úloh vidíme na časovém diagramu vpravo dole. Správnou kombinaci nastavených příznaků a proměnných potom vlevo v tabulce.

Dále jsem postupoval zkoušením různých verifikačních formulí, které umí nástroj *TimesTool* vyhodnotit a tím jsem vyzkoušel různé správné i nesprávné možnosti chování systému podobně jako výše za pomocí nástroje *UPPAAL*. Je zde použito formulí *CTL* logiky viz [14]. *Satisfied* v následujících formulích znamená, že formule je splněna.

Ve formulích jsou testovány hodnoty proměnných, které za pomocí hodnot 0 a 1 udávají, zda to, či ono světlo na semaforu svítí nebo ne. Jsou testovány různé kombinace, vyhovující bezpečnému řízení uzlu. (například, že nedojde k situaci, že se v určitých směrech, které se protínají, nerozsvítí zelené) Jedná se jen o výsek ověřovaných vlastností pro ilustraci.

Ukázky verifikačních formulí ověřovaných verifikátorem nástroje TimesTool:

Vždy je buď na semaforu ve směru A zelená a ve směru D zelená nesvítí nebo je zelená ve směru D a ve směru A zelená nesvítí nebo nesvítí ani jedna.

A[]((A_semafor_zelena_on==1 and D_semafor_zelena_on==0) or (A_semafor_zelena_on==0 and D_semafor_zelena_on==1) or (A_semafor_zelena_on==0 and D_semafor_zelena_on==0)) SATISFIED

Vždy je buď na semaforu ve směru A zelená a ve směru C na semaforu rovně a doprava zelená nesvítí nebo je zelená ve směru C na semaforu rovně a doprava a ve směru A zelená nesvítí nebo nesvítí ani jedna.

A[]((A_semafor_zelena_on==1 and C_semafor_RP_zelena_on==0) or (A_semafor_zelena_on==0 and C_semafor_RP_zelena_on==1) or (A_semafor_zelena_on==0 and C_semafor_RP_zelena_on==0)) SATISFIED

Vždy je buď na semaforu ve směru D zelená a ve směru C na semaforu doleva zelená nesvítí nebo je zelená ve směru C na semaforu doleva a ve směru D zelená nesvítí nebo nesvítí ani jedna.

A[]((D_semafor_zelena_on==1 and C_semafor_L_zelena_on==0) or (D_semafor_zelena_on==0 and C_semafor_L_zelena_on==1) or (D_semafor_zelena_on==0 and C_semafor_L_zelena_on==0)) SATISFIED

Vždy buď na semaforu ve směru A svítí zelená a v tom samém směru na přechodu zelená nesvítí nebo na semaforu nesvítí zelená a na přechodu je potom zelená nebo nesvítí ani jedna.

A[]((A_semafor_zelena_on==1 and A_chodec_zelena_on==0) or (A_semafor_zelena_on==0 and A_chodec_zelena_on==1) or (A_semafor_zelena_on==0 and A_chodec_zelena_on==0)) SATISFIED

Vždy buď na semaforu ve směru B svítí zelená a v tom samém směru na přechodu zelená nesvítí nebo na semaforu nesvítí zelená a na přechodu je potom zelená nebo nesvítí ani jedna.

A[]((B_semafor_zelena_on==1 and B_chodec_zelena_on==0) or (B_semafor_zelena_on==0 and B_chodec_zelena_on==1) or (B_semafor_zelena_on==0 and B_chodec_zelena_on==0)) SATISFIED

Vždy je buď na semaforu ve směru B zelená a ve směru D zelená nesvítí nebo je zelená ve směru D a ve směru B zelená nesvítí nebo nesvítí ani jedna.

A[]((B_semafor_zelena_on==1 and D_semafor_zelena_on==0) or (B_semafor_zelena_on==0 and D_semafor_zelena_on==1) or (B_semafor_zelena_on==0 and D_semafor_zelena_on==0)) SATISFIED

Vždy je buď na semaforu ve směru G zelená a ve směru E zelená nesvítí nebo je zelená ve směru E a ve směru G zelená nesvítí nebo nesvítí ani jedna.

A[]((G_semafor_zelena_on==1 and E_semafor_zelena_on==0) or (G_semafor_zelena_on==0 and E_semafor_zelena_on==1) or (G_semafor_zelena_on==0 and E_semafor_zelena_on==0)) SATISFIED

Vždy je buď na semaforu ve směru G zelená a ve směru F zelená nesvítí nebo je zelená ve směru F a ve směru G zelená nesvítí nebo nesvítí ani jedna.

A[]((G_semafor_zelena_on==1 and F_semafor_zelena_on==0) or (G_semafor_zelena_on==0 and F_semafor_zelena_on==1) or (G_semafor_zelena_on==0 and F_semafor_zelena_on==0)) SATISFIED

Vždy když svítí ve směru G zelená šipka doprava tak na semaforu ve směru E zelená nesvítí nebo svítí zelená na semaforu ve směru E a nesvítí zelená na semaforu ve směru G nebo nesvítí ani jedna.

A[]((G_sipka_zelena_on==1 and E_semafor_zelena_on==0) or (G_sipka_zelena_on==0 and E_semafor_zelena_on==1) or (G_sipka_zelena_on==0 and E_semafor_zelena_on==0)) SATISFIED

Vždy buď na semaforu ve směru G svítí zelená a v tom samém směru na přechodu zelená nesvítí nebo na semaforu nesvítí zelená a na přechodu je potom zelená nebo nesvítí ani jedna.

A[]((G_semafor_zelena_on==1 and G_chodec_zelena_on==0) or (G_semafor_zelena_on==0 and G_chodec_zelena_on==1) or (G_semafor_zelena_on==0 and G_chodec_zelena_on==0)) SATISFIED

Vždy buď na semaforu ve směru F svítí zelená a v tom samém směru na přechodu zelená nesvítí nebo na semaforu nesvítí zelená a na přechodu je potom zelená nebo nesvítí ani jedna.

A[]((F_semafor_cervena_on==1 and F_chodec_zelena_on==0) or (F_semafor_zelena_on==0 and F_chodec_zelena_on==1) or (F_semafor_cervena_on==0 and F_chodec_zelena_on==0)) SATISFIED

Vždy buď svítí ve směru F úniková šipka a na semaforu ve směru E nesvítí zelená nebo nesvítí úniková šipka ve směru F a na semaforu ve směru E svítí zelená nebo nesvítí ani jedna.

A[]((F_unik_on==1 and E_semafor_zelena_on==0) or (F_unik_on==0 and E_semafor_zelena_on==1) or (F_unik_on==0 and E_semafor_zelena_on==0)) SATISFIED

6 Implementace systému

Systém jsem se rozhodl implementovat pod RT operačním systémem *uC/OS-II* [7], který byl v této diplomové práci již prostudován a prezentován a vybrán z několika možných systémů. *uC/OS-II* jsem nainstaloval na školní server Merlin. Můj adaptivní systém řízení byl implementován v jazyce C v jednom zdrojovém souboru o asi 1600 řádcích. Bylo by ještě možné zdrojové soubory rozdělit do více modulů pro větší přehlednost, ale můj zdrojový kód je přehledný a dobře komentovaný, takže by v tom neměl být problém.

Při implementaci jsem vycházel z malého ukázkového příkladu, ze kterého jsem zachoval základní strukturu psaní zdrojového kódu pro systém *uC/OS-II*. Navíc jsem ale potřeboval využívat pokročilejší funkce zobrazování, takže jsem použil navíc modul „term_display.h“ se kterým jsem se naučil pracovat za pomocí ukázkových příkladů a manuálu systému *uC/OS-II*.

6.1 Obecné deklarace a řídící struktury

Bylo třeba definovat si některé typy a struktury, které potom systém využívá pro řízení i pro vlastní běh. Můj systém řízení jednotlivých signalizačních světel na semaforech spočívá v tom, že mám definovaný výčtový typ, který může nabývat hodnot *ON* nebo *OFF* a pro každé světlo na semaforech, které buď svítí, nebo ne máme jednu proměnnou právě tohoto výčtového typu, která nám udává, jestli světlo svítí nebo ne. Pokud má proměnná hodnotu ON pak toto světlo svítí a pokud má proměnná hodnotu OFF tak světlo nesvítí. Při praktické realizaci by se potom jednoduše tyto signály napojily přímo na fyzické zařízení například za pomocí nějakých portů a tak by přímo spínaly jednotlivé světelné signalizace.

Výčtový typ definovaný pro zapínání a vypínání jednotlivých světelných zařízení:

```
typedef enum {
    ON, OFF
} on_off;
```

Dále jsem definoval strukturu, která představovala jednotlivé celé klasické semafory. To znamená semafor, který má tři barvy (červena, oranžová zelená). Potom každý klasický semafor měl přiřazenu proměnnou, která byla typu přesně této struktury. Toto jsem zavedl pro lepší přehlednost.

Struktura definovaná pro klasický semafor. Využívá výčtového typu definovaného výše:

```
typedef struct {
```

```
        on_off cervena;
        on_off oranzova;
        on_off zelena;
} semafor;
```

Stejně jako pro klasický semafor jsem si zavedl strukturu, která reprezentuje klasický přechod pro chodce.

```
typedef struct {
        on_off cervena;
        on_off zelena;
} prechod;
```

Ostatní světelné signalizační prvky sestávající pouze z jednoho světla jsou potom deklarovány jako proměnná typu on_off definovaného výše.

Při implementaci jsem potřeboval také proměnné pro režii systému a pro jeho vlastní funkci. Tyto řídící proměnné jsem všechny integroval do struktury nazvané řízení.

```
typedef struct {
        on_off nouzovy_rezim;    //signalizace nouzového režimu
        on_off zacpa_A;          //signalizace zácpy ve směru A
        on_off from_init;  // pomocná proměnná programu pro označení
                           //prvního cyklu programu
        on_off vypis;    //proměnná aktivující výpis stavu uzlu po
                         //každé změně
        on_off OOM_signal;     //povolení režimu OOM(průjezd
                               //vozidel s právem přednosti)

} t_rizeni;
```

Posledním úsekem deklarací jsou proměnné, které jsou využívány pro simulaci provozu inteligentního řídicího systému.

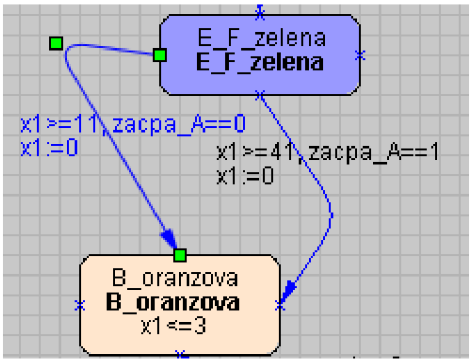
```
    //promenne pro simulaci
    int i;
    int fronta_A = 0;     //Počítadlo fronty aut ve směru A.
    int fronta_mezi = 0;  //Počítadlo fronty tvořící se mezi křižovatkami.
    int prep = 0;//Přepínací proměnná (pouze k přepínání barev v
terminálu)
```

```
int odjezd_A = 0;       //Signalizace odjezdu ve směru A.
int odjezd_E = 0;       //Signalizace odjezdu ve směru E.
int odjezd_D = 0;       //Signalizace odjezdu ve směru D.
int pocitadlo = 0;      //Počítadlo generací simulačního běhu.
#define NORMAL_A 1      //Definice konstant provozu.
#define NORMAL_E 12     //Definice konstant provozu.
//#define SPICKA 9
FILE *soubor;           //Definice souboru pro logování stavu uzlu.
time_t cas;             //Definice hodin.
struct tm *ltmcas;      //Definice hodin.
int hod_aktual;         //Definice hodin.
```

6.2 Deklarace jednotlivých úloh a jejich volání

Tak jak jsem vytvořil kompletní model systému v nástroji *TimesTool*, tak jsem pak přesně podle něj vytvořil jednotlivé úlohy systému pro *uC/OS-II*.

Ukázka převodu jednoduchého modelu z *TimesTool* do úloh pro s *uc/OS-II.*



Obrázek 6.0 – Jednoduchý model TimesTool

Deklarace

```
void E_F_zelena (void *data)
{
  for (;;)
  {
   //zde je komplet telo funkce
.
.
   OSTaskSuspend(OS_PRIO_SELF);
  }
}
```

```
//*************************************
void B_oranzova (void *data)
 {

   for (;;)
   {
     OSTaskSuspend(OS_PRIO_SELF);
   }
 }
```

Vytvoření úloh

```
OSTaskCreate(E_F_zelena, (void *)&TaskData[31], (void *)&TaskStk[31]
 [TASK_STK_SIZE - 1], 31);
OSTaskSuspend(31);

OSTaskCreate(B_oranzova,(void *)&TaskData[7],(void  *)&TaskStk[7]
 [TASK_STK_SIZE - 1], 7);
OSTaskSuspend(7);
```

Spuštění

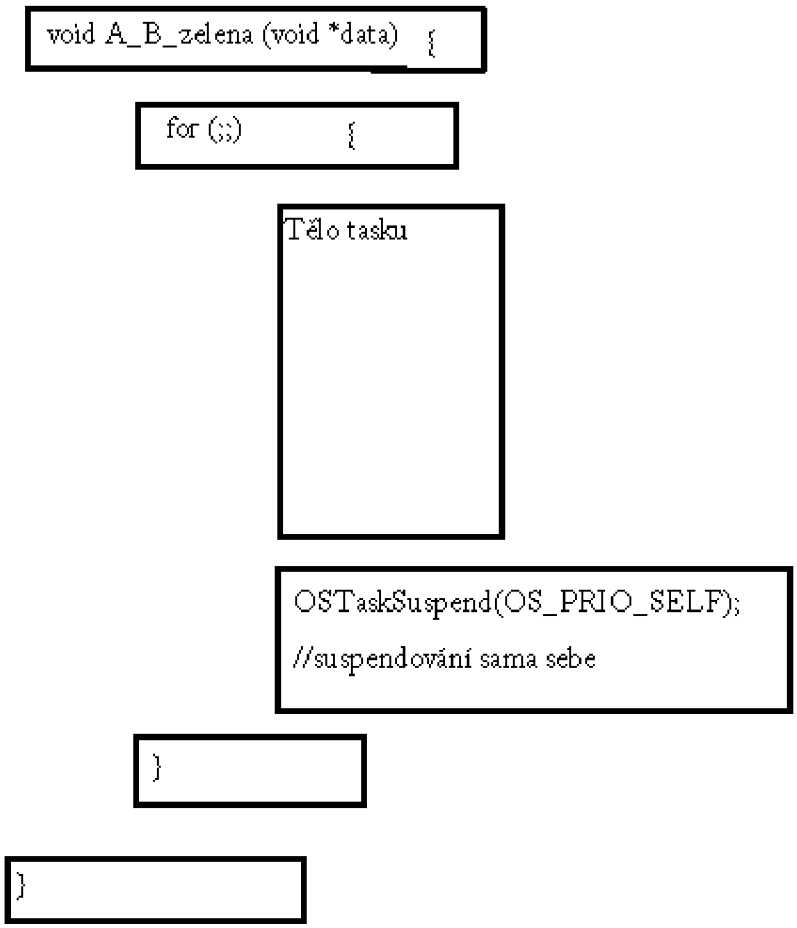
```
OSTaskResume(31);  //E_F_zelena
    if( rizeni.zacpa_A == ON)
    {
       OSTimeDlyHMSM(0,0,41,0);
    } else OSTimeDlyHMSM(0,0,11,0);
OSTaskResume(7); //B_oranzova
```

Deklarace jednotlivých úloh následují v kódu aplikace ihned za deklaracemi proměnných. Úlohy se nadeklarují a poté se v takzvaném StartUpTasku aktivují. StartUpTask je speciální úloha, která se automaticky jako první spustí při startu systému. Úlohy v tomto systému využívám dvěma způsoby. Buď úlohu nadeklaruji, nastavím případnou prodlevu a úlohu aktivuji a ta potom automaticky běží nebo úlohu nadeklaruji, ve startUpTaku aktivuji a ihned zastavím její vykonávání. Potom si ji spustím jen tehdy, kdy potřebuji a ta se pak opět automaticky pozastaví a je připravená na další obnovení.

Každá úloha je v *uC/OS-II* tvořena vlastně funkcí, která obsahuje nekonečnou smyčku, ve které jsou jednotlivé vykonávané činnosti.

6.2.1 Ukázka deklarace úlohy spouštěné voláním v daný čas.

Tato úloha je po deklaraci aktivována ve StartUpTasku a poté hned suspendována a obnovena až na zavolání. Proběhne potom jednou a zase se sama suspenduje a bude čekat na další zavolání. V této úloze se spouští zelený interval ve směrech A a B křižovatky I.



Obrázek 6.1 – Schéma úlohy volané v daný čas

Po deklaraci je úloha aktivována ve StartUpTasku.

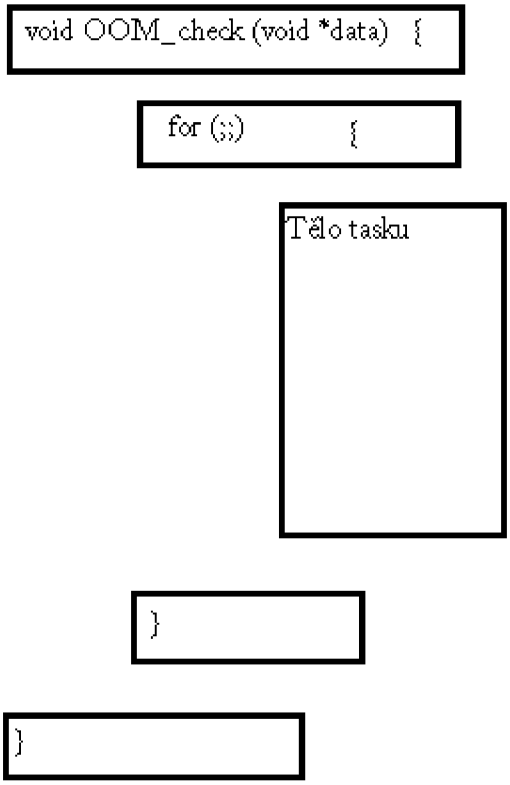
*OSTaskCreate(A_B_zelena, (void *)&TaskData[5], (void *)&TaskStk[5][TASK_STK_SIZE - 1], 5);*
OSTaskSuspend(5);

Parametry funkce jsou ukazatel na zdrojový kód úlohy, ukazatel na volitelný datový prostor úlohy, ukazatel na vrch zásobníku úlohy a posledním je priorita úlohy.

Zde je nastavena i priorita úlohy, za pomocí které je pak úloha později identifikována. Pomocí příkazu *OSTaskSuspend* je pak úloha ihned po vytvoření a aktivaci suspendována a čeká na příkaz *OSTaskResume* , kterým se pak obnoví její běh. TaskSuspend a TaskResume identifikují danou úlohu za pomocí priority úlohy, která je jedinečným klíčem.

6.2.2 Ukázka deklarace úlohy periodické běžící automaticky.

Tato úloha je deklarována, má nastaveno zpoždění a je potom spouštěna automaticky s danou periodou. Na rozdíl od předchozí nesuspenduje sama sebe po doběhnutí a ani po vytvoření a aktivaci. Za pomoci této úlohy je každých pět sekund kontrolován příznak OOM, který pokud je nastaven, tak se uzel přepne do režimu průjezdu vozidla s právem přednostní jízdy.



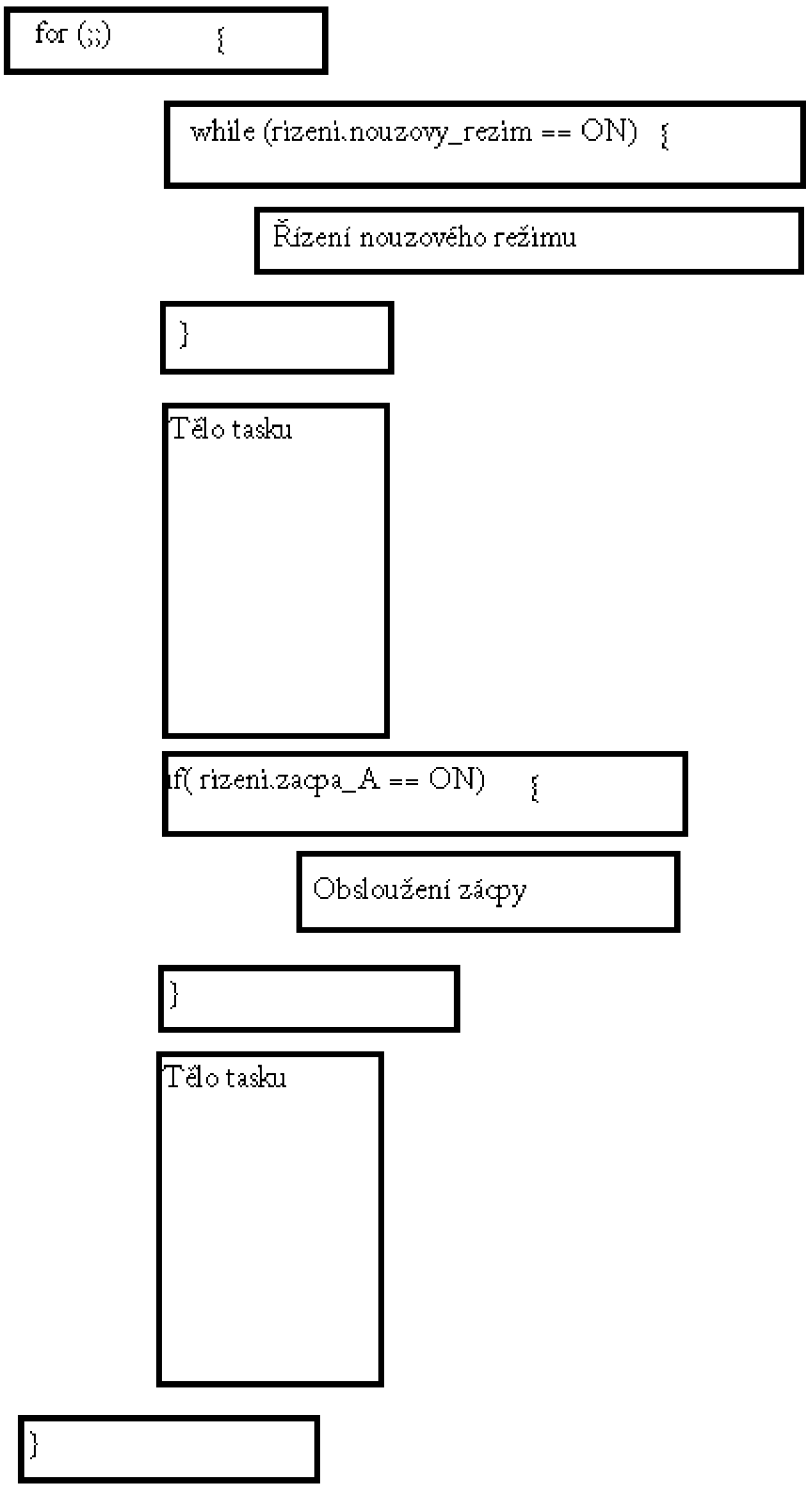
Obrázek 6.2 – Schéma úlohy běžící automaticky

Po deklaraci je úloha aktivována ve StartUpTasku.

*OSTaskCreate(OOM_check, (void *)&TaskData[1], (void *)&TaskStk[1][TASK_STK_SIZE - 1], 1);*
Zde je nastavena priorita úlohy a úloha je spuštěna a automaticky běží.

6.2.3 Hlavní smyčka „programu“ StartUpTasku

Hlavní StartUpTask kromě jiného obsahuje nekonečnou smyčku, která pak automaticky sama donekonečna běží a v ní je obsaženo spouštění jednotlivých úloh, které jsou spouštěny v závislosti na volání. Jsou to všechny úlohy kromě těch, které jsou deklarovány tak, že běží periodicky stále.



Obrázek 6.3 – Schéma hlavní smyčky StartUpTasku

6.3 Implementace inteligentních prvků

V této podkapitole bude vysvětlen postup implementace jednotlivých inteligentních prvků systému.

6.3.1 Monitorování hustoty dopravy ze směru do města nebo mezi křižovatkami dopravného uzlu a reakce na ně.

Ze směru A křižovatky I. Je kontrolována délka fronty čidlem. Pokud fronta překročí určitou mez, tak je nastaven příznak zácpa_A. (obdobně lze rozšířit systém i pro jiné směry) Ten je potom vyhodnocen v hlavní smyčce zde:

```
if( rizeni.zacpa_A == ON)
   {
     OSTimeDlyHMSM(0,0,41,0);
   } else OSTimeDlyHMSM(0,0,11,0);
```

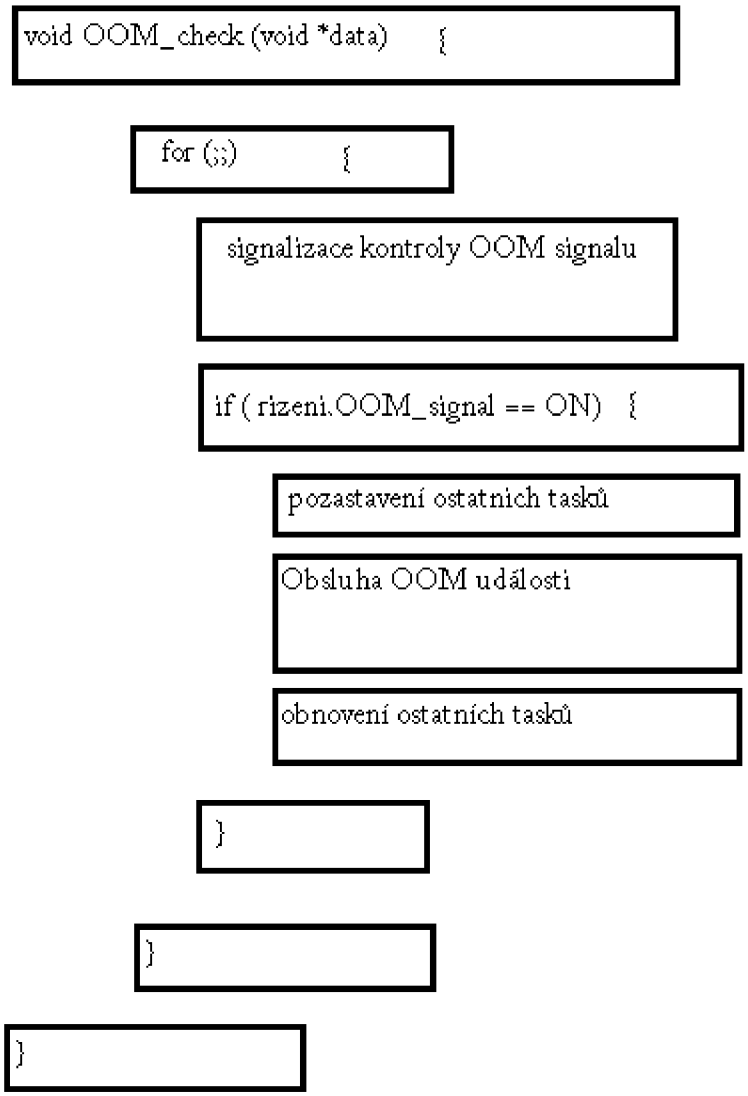
Pokud není zácpa tak je standardně doba před spuštěním další úlohy 11 sekund. Pokud je ovšem nastaven příznak že došlo k překročení fronty, tak se doba před spuštěním další úlohy změní na 41 sekund. Je také možnost povolit nebo zakázat reakci na tvořící se kolonu příznakem ve struktuře řízení. *rizeni.zacpa_A* může mít hodnoty ON nebo OFF.

Délka fronty je také monitorována mezi oběma křižovatkami dopravního uzlu. V případě, že je překročena nastavená mez, tak je nastaven příznak zácpy mezi křižovatkami, který nastaví automaticky příznak stejný jako pro zácpu ze směru A. Tudíž obsluha reakce probíhá stejně. Vypozoroval jsem totiž na modelu, že to bude výhodné. Dle specifikace křižovatek totiž opatření, které se používá při zácpě ve směru A bude mít stejně dobrý vliv na redukci zácpy mezi křižovatkami. Prodloužením intervalu zelené ve směrech AB se totiž zredukuje počet přímo přijíždějících aut do prostoru mezi křižovatkami a urychlí se odjezd z tohoto prostoru.

6.3.2 Inteligentní průjezd vozidel s právem přednostní jízdy.

```
void OOM_check (void *data)
```

Inteligentní průjezd vozidel s právem přednostní jízdy jako jsou policie, záchranná služba nebo hasiči je řešen tak, že tyto složky mají před příjezdem k dopravnímu uzlu možnost vyslat specifický signál, který nazývám OOM (ochrana osob a majetku). Pokud je tento signál vyslán, tak systém nastaví příznak, že byl signál OOM obdržen. Systém kontroluje tento příznak za pomocí jedné periodické úlohy. Tato úloha se spouští pravidelně každých 5 sekund, což bylo vypozorováno jako dostatečný interval.



Obrázek 6.4 – Schéma úlohy obsluhujícího signál OOM

Ukázka obsluhy *OOM*:

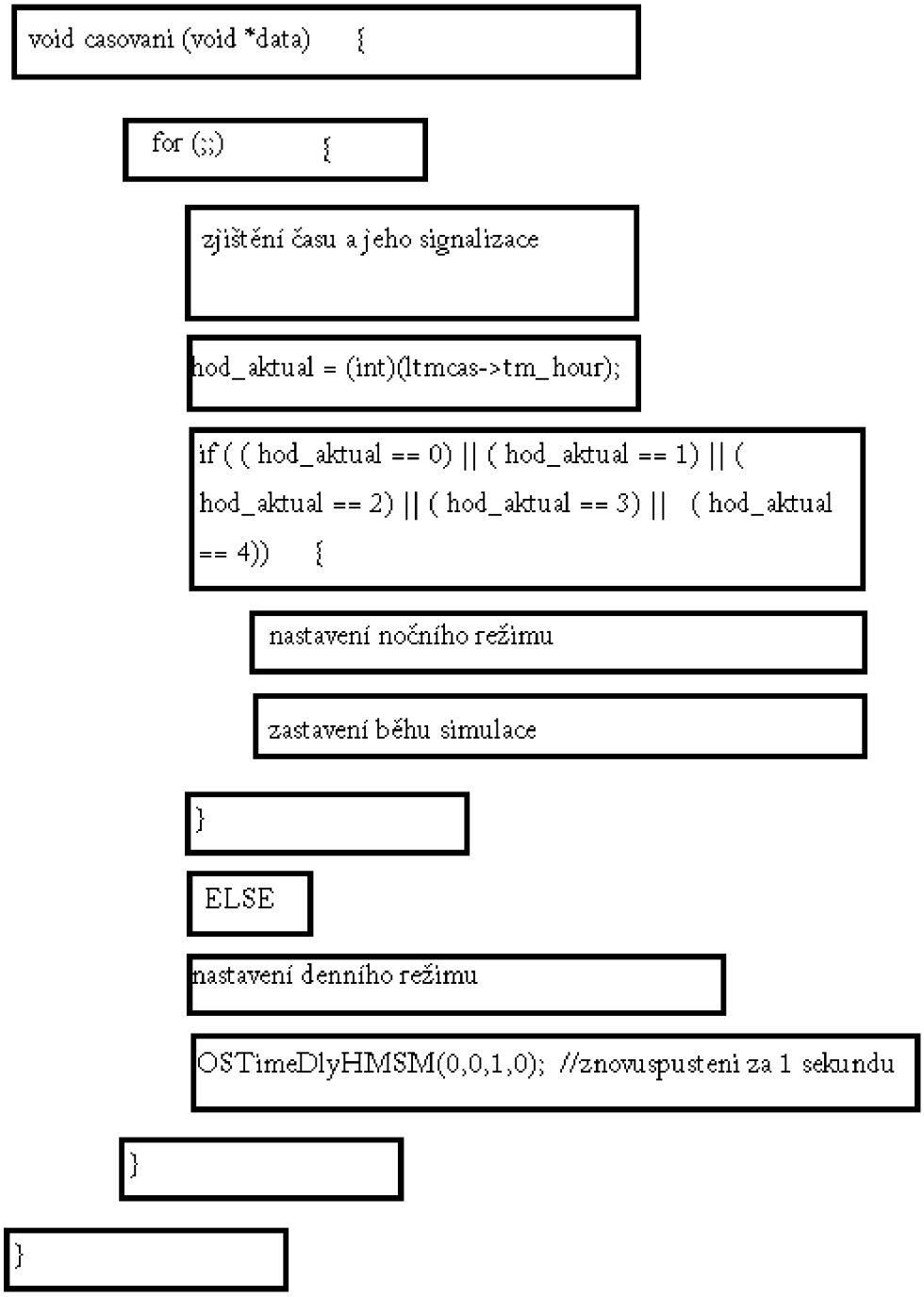
```
if ( rizeni.OOM_signal == ON)
 {
   //OSSchedLock(); //zakaz ostatnich tasku
   OSTaskSuspend(0);
   OSTaskResume(24);
          OSTimeDlyHMSM(0,0,3,0);
          OSTaskResume(25);
          OSTimeDlyHMSM(0,0,20,0);
          OSTaskResume(4); //oranzova vsude
          OSTimeDlyHMSM(0,0,3,0);
          rizeni.OOM_signal = OFF;
          printf("---KONEC OSLUHY OOM---\n");
          OSTaskResume(0);
  } else OSTimeDlyHMSM(0,0,5,0);
```

Pokud OOM kontrolní úloha zjistí nastavení příznaku OOM, tak pozastaví vykonávání všech ostatních úloh a zařídí spuštění třech obslužných úloh. Po jejich skončení opět spustí provádění ostatních řídících úloh v tom okamžiku, kde byly přerušeny. Zároveň taky vynuluje OOM příznak. V případě že tato kontrolní úloha nezjistí nastavení příznaku OOM, tak se nic neprovede a naplánuje se opětovné spuštění OOM kontrolní úlohy znovu za 5 sekund.

6.3.3 Automatické řízení denního a nočního režimu křižovatky.

```
void casovani (void *data)
```

Do systému jsem implementoval i časování a sledování aktuálního času. Aktuální hodiny jsou i zobrazovány na terminálu programu. Je to vyřešeno úlohou, která je automaticky spouštěna každou sekundu běhu programu.



Obrázek 6.5 – Schéma úlohy obsluhujícího časování a hodiny

Ukázka rozlišení denního a nočního režimu:

```
if ( ( hod_aktual == 0) || ( hod_aktual == 1) || ( hod_aktual == 2) || ( hod_aktual == 3) || ( hod_aktual == 4))
              {
                 PC_DispStr(1,24,"NOCNI REZIM",COLOR_BLUE,COLOR_BLACK);
                 rizeni.nouzovy_rezim = ON;
```

```
            }  else
            {
               PC_DispStr(1,24,"DENNI REZIM",COLOR_WHITE,COLOR_BLACK);
               rizeni.nouzovy_rezim = OFF;
        }
```

Tato úloha zjišťuje aktuální čas a zobrazuje jej i v okně programu. Kontroluje, zda je aktuální čas v mezích pro denní nebo noční provoz a nastavuje příznaky, podle kterých potom systém reaguje. Vypisuje také, zda je systém v denním nebo nočním režimu.

6.3.4 Včasné upozornění řidičů na snížení rychlosti v závislosti na případné tvořící se koloně před křižovatkou za pomoci proměnné dopravní značky.

Systém monitoruje velikost fronty před křižovatkou I. Ve směru A a pokud je nastaven příznak zácpy, tak je nastaven signál, který přepíná proměnnou dopravní značku před křižovatkou. Standardně tato značka zobrazuje nejvyšší povolenou rychlost 60km/h a v případě, že je signalizována zácpa ve směru A je tato značka přepnuta na nejvyšší povolenou rychlost 40km/h. Pro tento účel byla vybrána proměnná dopravní značka typu PDZ od firmy Eltodo [10], která je specifikována výše v tomto dokumentu. Systém zároveň signalizuje zobrazovanou rychlost i na terminálu programu.

6.3.5 Nouzový režim

Systém lze přepnout kdykoliv do nouzového režimu nastavením příznaku nouzovy_rezim v řídící struktuře programu. Tento příznak je kontrolován v hlavní smyčce a podle něj je prováděno další řízení.

Ukázka obsluhy nouzového režimu:

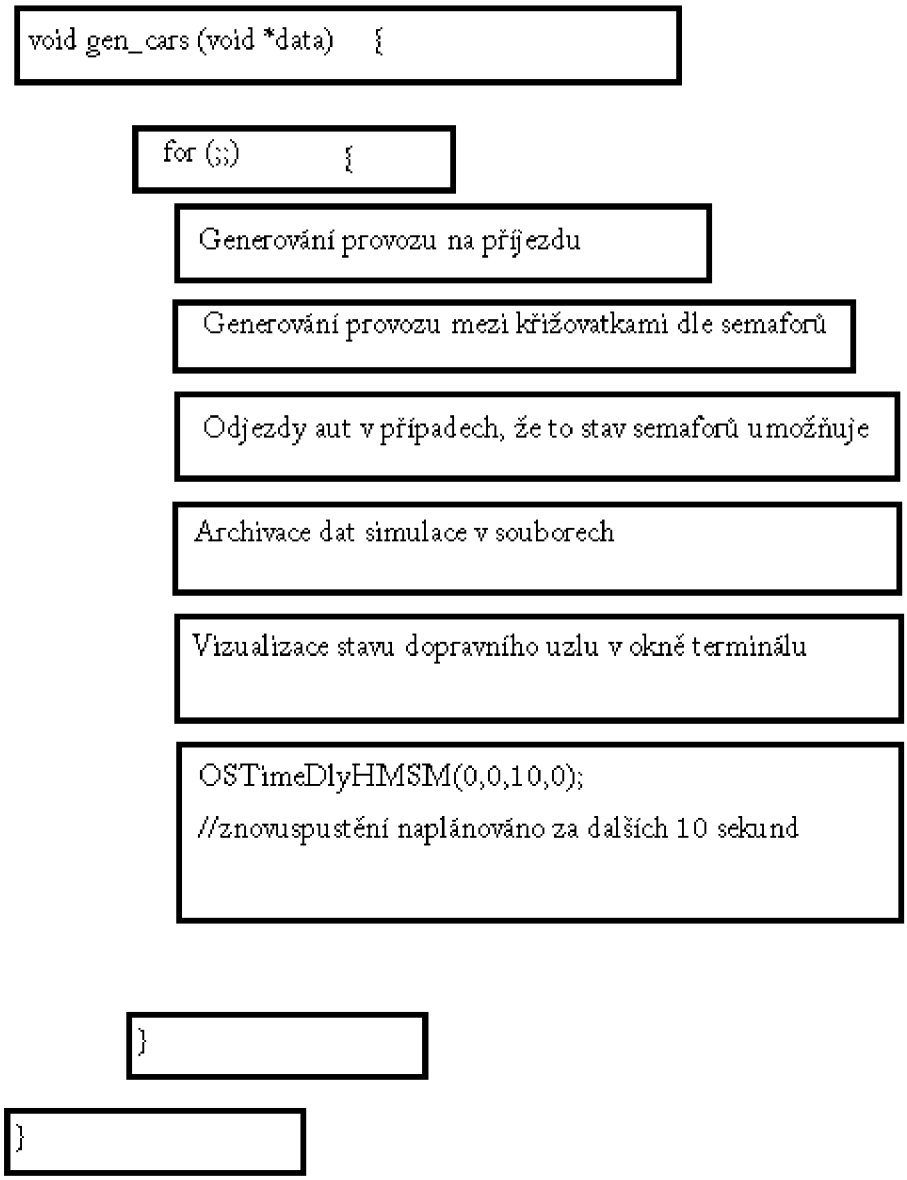
```
while (rizeni.nouzovy_rezim == ON)
     {
     OSTaskResume(2);  //Blik_off
     OSTimeDlyHMSM(0,0,2,0);
     OSTaskResume(3); //Blik_on
     OSTimeDlyHMSM(0,0,2,0);
     from_init = OFF;

     }
```

Nouzový režim je shodný s režimem nočním. Systém by bylo možno ještě rozšířit o možnost automatické diagnostiky a v případě jakéhokoliv problému by byl systém automaticky přepnut do nouzového režimu a byl přivolán technik, ale toto zatím implementováno není.

6.4 Implementace simulace běhu systému.

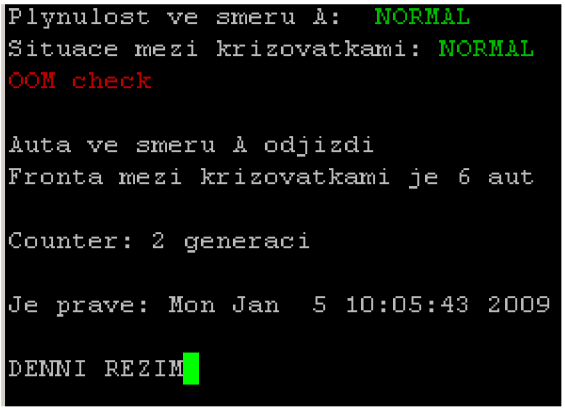
```
void gen_cars (void *data)
```

V systému je implementována i simulace provozu systému aby bylo možné ověřit chování a funkci. Je to vyřešeno za pomocí simulační úlohy.



Obrázek 6.5 – Schéma tasku zajišťujícího simulaci provozu a archivaci simulačních dat

Tato úloha je spouštěna každých 10 sekund. Generuje náhodně provoz jak ve směru A tak mezi křižovatkami. Intervaly generovaných vozidel lze nastavit parametry, které určují to, zda je zrovna provoz ve špičce nebo ne. Například takto. #define NORMAL_A 1; #define NORMAL_E 12; #define SPICKA 9. Generování provozu mezi křižovatkami je také ovlivněno tím, z jakých směrů mohou přijíždět či odjíždět auta (takže podle stavu ostatních semaforů). Systém simuluje i odjezdy aut v monitorovaných směrech podle stavu semaforů. Abychom měli nějaké výstupy, které bude možno později analyzovat, tak systém vytváří LOG soubory a to soubory log1.txt a log2.txt. Do těchto souborů se každých 10sekund zapisuje stav jednotlivých front. Log1 – směr A a Log2 – mezi křižovatkami. Tyto log soubory jsou vytvářeny při každém spuštění systému. Tato simulační úloha zároveň vypisuje do okna terminálu programu počty aut v kolonách a stavy monitorovaných prvků systému.



Obrázek 6.6 – Simulační výstup z terminálu

Na simulačním výstupu z terminálu lze vidět aktuální situaci na příjezdu A, nebo mezi křižovatkami, signalizaci kontroly *OOM,* počty aut ve frontách, počet generací simulátoru, hodiny a aktuální režim.

6.5 Přeložení na serveru Merlin

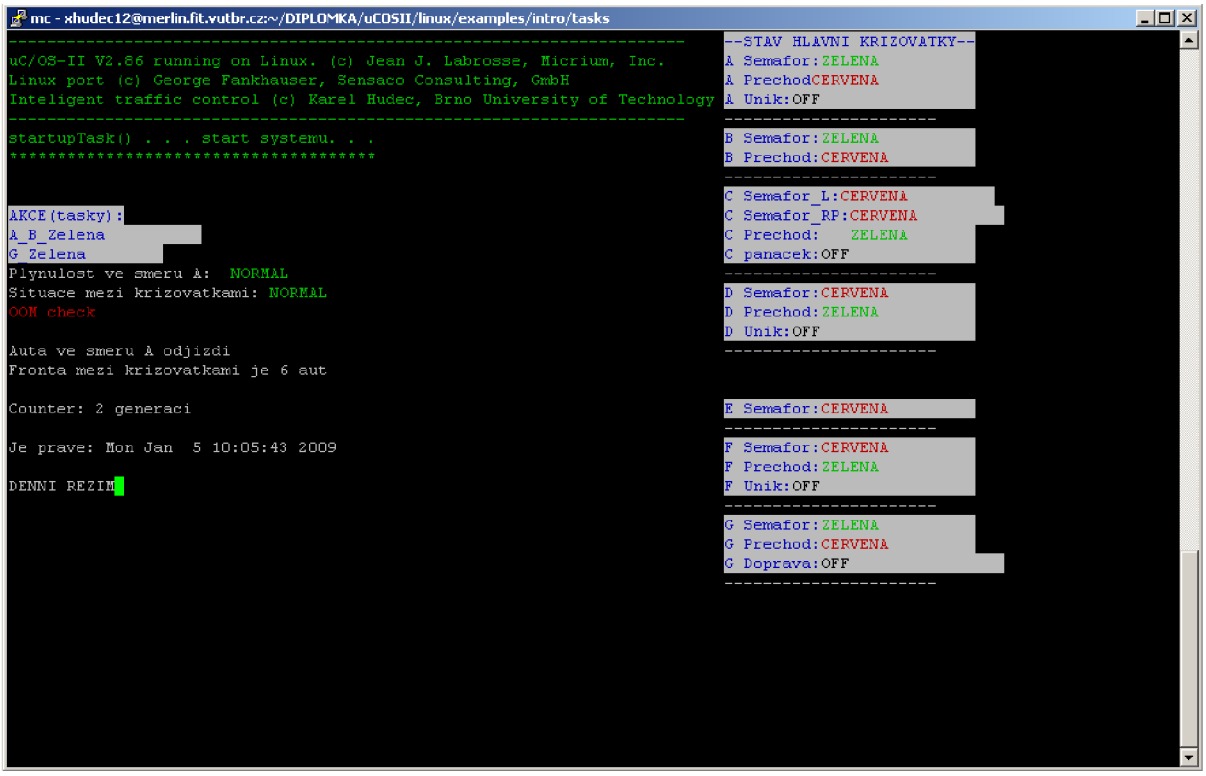
Zdrojový kód je možno přeložit na školním serveru Merlin za použití příkazu *make*. Bylo tedy třeba vytvořit také *makefile* pro překlad pod uC OS/II. Při tvorbě *makefilu* jsem vycházel z dostupných příkladů, které jsem měl k systému.

Makefile i všechny zdrojové kódy jsou obsaženy na CD přiloženém k diplomové práci. Kódy zde mají stejnou strukturu, jako byly zkoušeny přímo na školním serveru merlin.

7 Výsledná aplikace a analýza funkčnosti

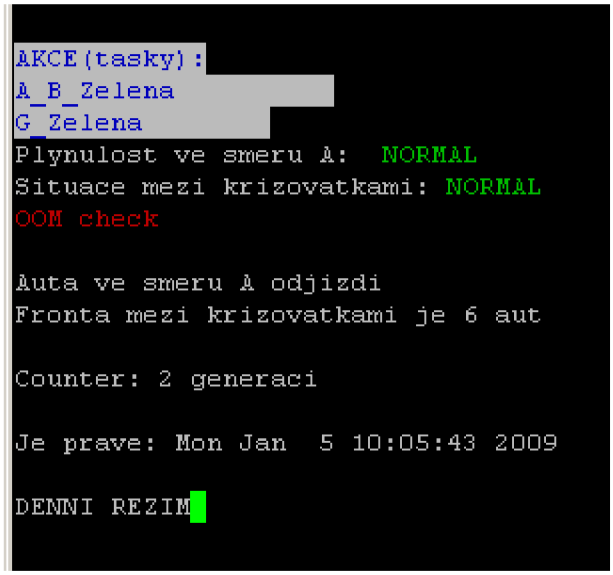
7.1 Vlastní aplikace

Po překladu zdrojového kódu na serveru Merlin je aplikace připravena ke spuštění. To provedeme jednoduše *./itc*. Výsledná aplikace vypadá následovně.



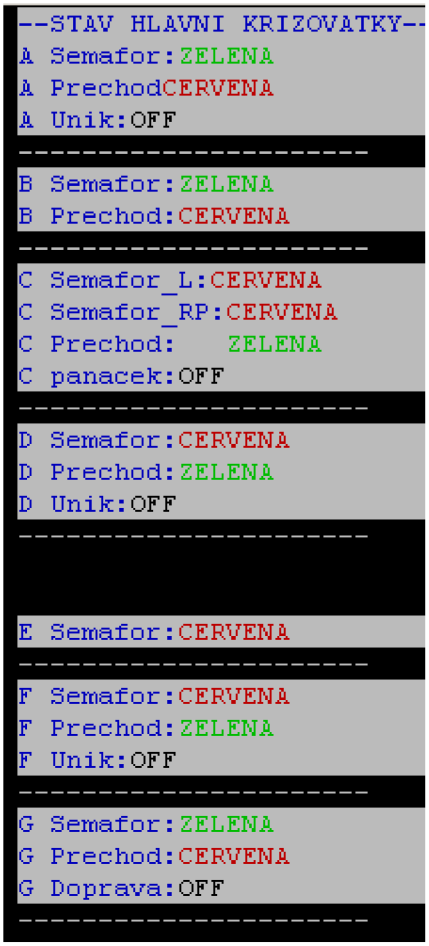
Obrázek 7.1 – Okno terminálu spuštěné aplikace

V pravém okně se nám zobrazuje aktuální stav obou křižovatek, co se týká světelných signálů. V levém sloupci aplikace ukazuje, jaká úloha se zrovna vykonává (naposled vykonala), vypisuje dále plynulost dopravy ze směru A a mezi křižovatkami (NORMAL, ZACPA), signalizuje barevně průběh kontrolní úlohy OOM_check, počty automobilů ve frontách, počítadlo cyklů simulace a ukazuje aktuální čas a režim, ve kterém se uzel zrovna nachází. Jelikož je aplikace určena do RT vestavěného systému tak pracuje v nekonečné smyčce. Takže ji je možno ukončit Control + c.



Obrázek 7.1 – Okno s údaji o úlohách a simulaci

V této části terminálu se zobrazují informace o tom, které úlohy jsou zrovna spouštěny a zobrazují se zde informace o probíhající simulaci běhu systému.



Obrázek 7.2 – Okno s údaji o úlohách a simulaci

Takto se zobrazují informace o současném stavu světelných signalizačních zařízení na křižovatkách uzlu.

7.2 Analýza funkčnosti systému a experimenty

7.2.1 Funkčnost systému obecně

Po implementaci a rozběhnutí systému bylo třeba zjistit, zda navržená a implementovaná „inteligentní řešení“ mají opravdu význam a smysl a zda opravdu napomáhají plynulosti provozu přes řízený dopravní uzel.

Systém průjezdu vozidel s právem přednostní jízdy bylo odzkoušeno a odsimulováno náhodným nastavením OOM příznaku v průběhu programu. Systém zareagoval včas a správně obsloužil průjezd křižovatkou, kde zajistil na 30 sekund všude červené. Poté se vrátilo řízení do stavu, ve kterém se nacházelo před přijetím signálu OOM.

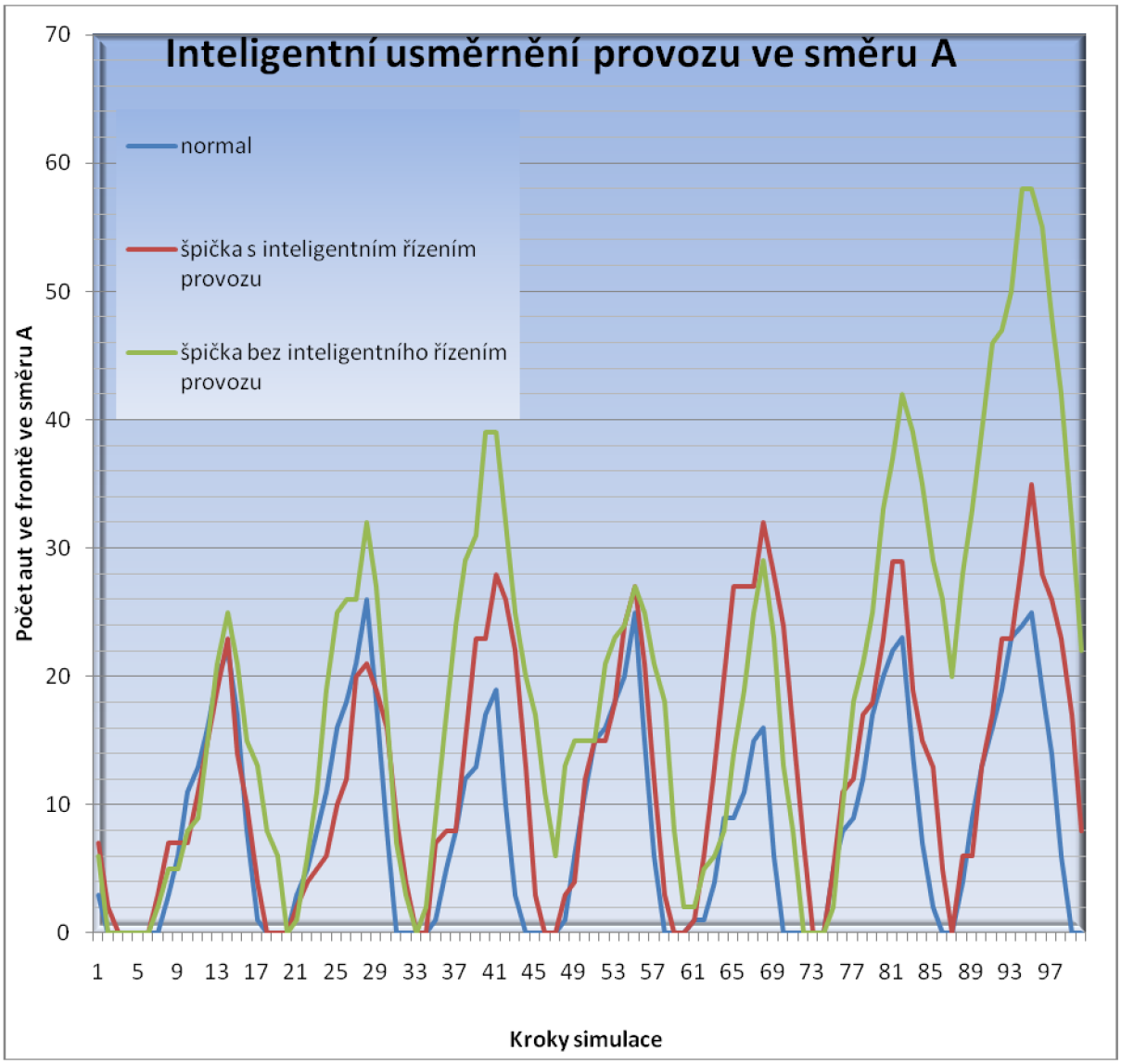
I ostatní „inteligentní“ vlastnosti systému byly jednoduše odzkoušeny, jako je například přepínání proměnné dopravní značky při koloně nebo aktuálním časem řízené přepínání denního a nočního režimu, stejně tak režim nouzový. Myslím ale, že jedno z nejdůležitějších a také pravděpodobně nejzajímavějších byla zkouška, zda funguje správně a jaké má výsledky „inteligentní“ systém zvyšování plynulosti dopravy přes tento uzel. Z tohoto důvodu jsem v simulaci zavedl LOG soubory, do kterých se zapisují stavy provozu přes křižovatky. Je do nich možné *LOGovat* i ostatní události. Například při debutování jsem do nich zapisoval i příchody některých signálu a změny některých příznaků. Z těchto LOG souborů jsem tedy zkoumal, jakým způsobem systém dokáže regulovat nadměrný provoz jednak ze směru A křižovatky I, tak v případě zácpy mezi křižovatkami uzlu. Pro samotné vykreslení do grafů jsem používal kolem 100 vzorků s tím, že nastavením parametrů simulačního běhu bylo možné měnit čas snímaní stavu a generování vozidel. Například 100 vzorků po 10 sekundách s tím, že systém každých 10sekund vygeneruje provoz úměrný stavu dopravy ve špičce nebo mimo ni, což nastavujeme právě parametry simulačního běhu.

7.2.2 Analýza výsledků řízení systému ve směru A

Směr A je směr do města tudíž ve špičkách se tu nárazově tvoří kolony. Simuloval jsem tedy v tomto směru provoz jak ve špičce, tak mimo špičku. Bylo důležité dobře nastavit rozložení příjezdu aut (například kolik aut přijede průměrně ve špičce nebo mimo ni za 10 sekund). Tyto konstanty jsou

již zmíněny výše a byly vypozorovány z reálné situace a případně lehce optimalizovány pro můj systém.

Zde je uveden jeden z prvních experimentů s hustotou provozu ve směru A. Ukazuje dohromady v jednom grafu jak provoz normální, tak provoz ve špičce s inteligentním usměrněním a provoz ve špičce bez inteligentního usměrnění.



Obrázek 7.1 – Graf Inteligentního usměrnění provozu ve směru A

Na grafu je názorně vidět, že co se týká špičkového provozu, tak je jasný rozdíl mezi tím, když byl „inteligentní“ systém řízení použit a když byl odstaven. Opravdu tedy dokázal dopravu usměrnit a zvýšit její plynulost. Vstupem byl LOG soubor hlavní křižovatky, ze kterého bylo vybráno 100 vzorků. Krokem simulace je v tomto případě časový horizont 10 sekund (osa x). Na ose y je pak znázorněn počet vozidel stojících ve frontě.

Modrá – normální provoz mimo špičku.

(0-6/10 sekund, průměrně 3 auta/10 sekund)

Červená – provoz ve špičce se zapnutým int. řízením.

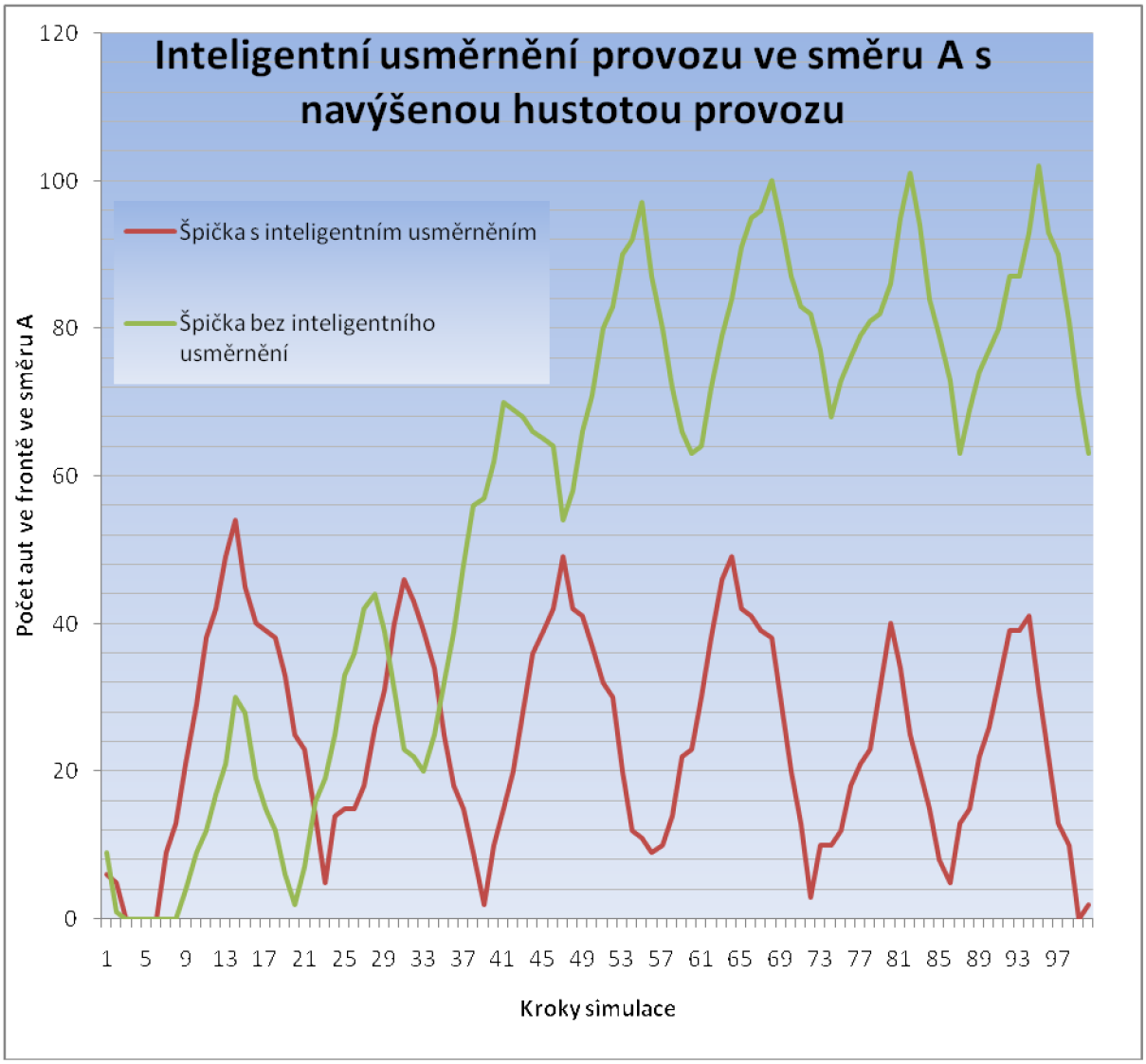
(0-10/10 sekund průměrně 5 aut/10 sekund)

Zelená – provoz ve špičce bez zapnutého int. řízení.

(0-10/10 sekund průměrně 5 aut/10 sekund)

Cílem tohoto experimentu bylo dokázat, že s použitím inteligentního řízení provozu se při husté dopravě zlepší průjezdnost uzlem, což se tedy potvrdilo.

Zde je ještě uveden jeden experiment pouze s provozem ve špičce, kdy byla ještě navýšena hustota přijíždějících aut.



Obrázek 7.2 – Graf Inteligentního usměrnění provozu ve směru A pouze ve špičce

Při tomto experimentu bylo testováno razantnější zvýšení hustoty provozu, kdy bylo testováno pro jak až extrémně hustý provoz bude systém stačit zachovat standardní průjezdnost uzlem. Zde byl konkrétně zvýšen počet aut ještě o třetinu oproti hustému provozu ve špičce. Na grafu je názorně vidět, že i tak systém obstál. Osy jsou popsány stejné jako u předchozího grafu.

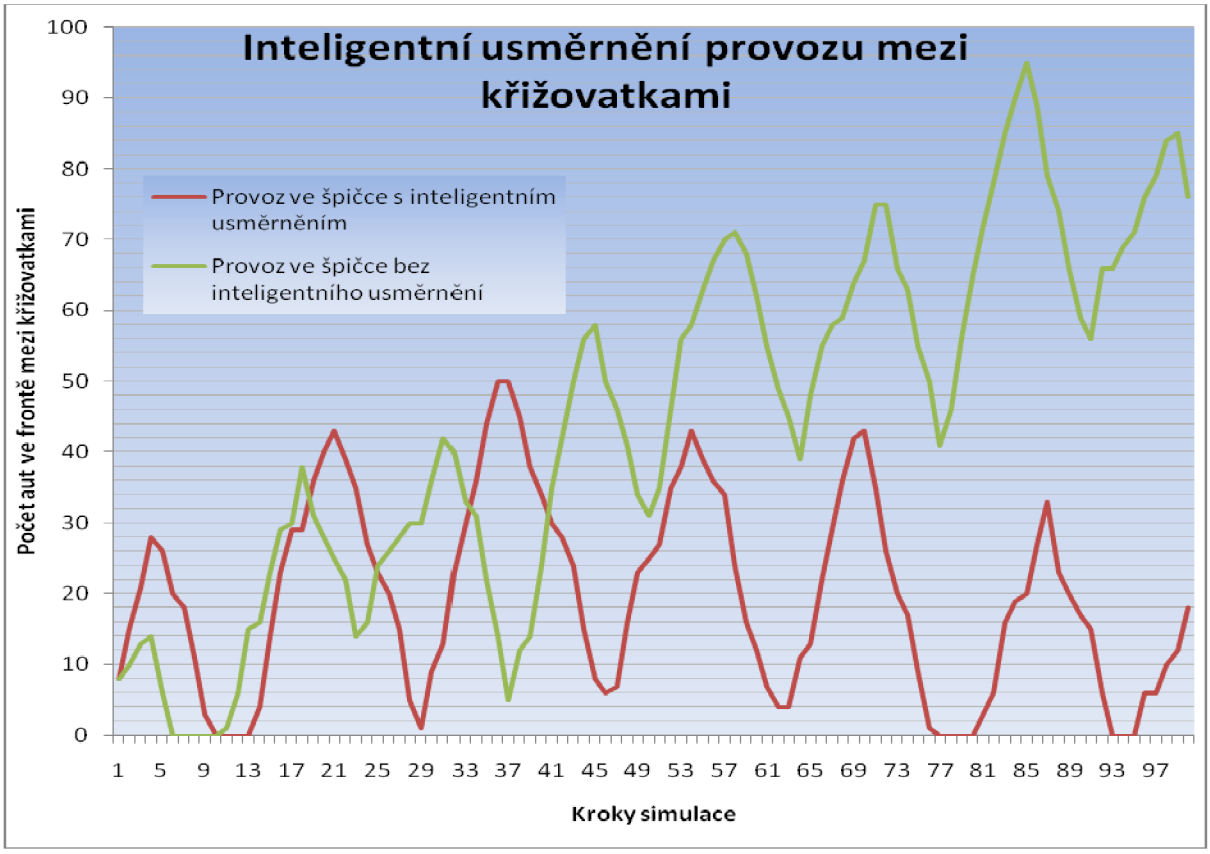
Červená – provoz ve špičce se zapnutým int. řízením.
(0-12/10 sekund průměrně 6 aut/10 sekund)
Zelená – provoz ve špičce bez zapnutého int. řízení.
(0-12/10 sekund průměrně 6 aut/10 sekund)

Výsledkem tohoto experimentu je to, že systém si poradí i s extrémně hustým provozem a stále udrží slušnou průjezdnost uzlem. Při ještě hustších provozech, které jsem taky testoval se pak kolony neúměrně zvyšovaly a tvořily se dlouhé zácpy i při zapnutí systému. Ne ovšem tak dlouhé jako při vypnutí.

7.2.3 Analýza výsledků řízení systému mezi křižovatkami

Pozemní komunikace mezi oběma křižovatkami našeho dopravního uzlu není příliš dlouhá, takže může lehce dojít k jejímu ucpání a tím zhoršení průjezdnosti celého dopravního uzlu. V simulaci systému jsem tedy obsáhl i tuto část uzlu. Simulační úloha generuje provoz mezi křižovatkami v závislosti na stavu ostatních semaforů a možnostech kudy pak auta mohou do tohoto prostoru přijíždět. V normálním provozu je provoz mezi křižovatkami plynulý. Při tomto experimentu se teda nastavily parametry pro provoz ve špičce a ostatní vhodné konstanty.

Na následujícím grafu je názorně vidět jaký je rozdíl mezi tím, když je zapnuto nebo vypnuto inteligentní usměrnění při stejném špičkovém provozu.



Obrázek 7.3 – Graf Inteligentního usměrnění provozu mezi křižovatkami uzlu

Při tomto experimentu byl testován vliv hustoty provozu na kolonu mezi jednotlivými křižovatkami uzlu. Vstupem byl LOG soubor simulačního systému. Bylo použito 100 kroků simulace po 10-ti sekundových intervalech. Ty jsou na ose x. Osa y znázorňuje počet aut v koloně mezi křižovatkami.

Červená – provoz ve špičce se zapnutým int. řízením.
(0-10/10 sekund průměrně 5 aut/10 sekund)
Zelená – provoz ve špičce bez zapnutého int. řízení.
(0-10/10 sekund průměrně 5 aut/10 sekund)

Výsledkem experimentu byl důkaz, že inteligentní systém usměrnění má opravdu vliv na délku kolony mezi křižovatkami při velmi hustém provozu, a to velmi pozitivní vliv. Při jeho odstavení se totiž mezi křižovatkami tvořily dlouhé kolony, případně docházelo i k zablokování komunikace.

7.2.4 Poznámka k simulaci

Všechny zjištěné hodnoty, podle kterých byly vytvořeny grafy, jsou doloženy k diplomové práci jako příloha v souboru log.xls na přiloženém CD v adresáři LOG

8 Závěr

Praktické realizaci této diplomové práce předcházel můj zájem o inteligentní dopravní systémy. Do realizace jsem se tedy pustil s velkým elánem a práce mě bavila. Nejprve bylo nutné prostudovat techniky inteligentního řízení provozu na pozemních komunikacích. Velice mnoho informací jsem získal od firmy *Eltodo*, která se již dlouhou dobu zabývá mimo jiné elektronickým řízením dopravy.

Dopravní uzel, který jsem se rozhodl zpracovat, jsem vybral v Brně na ulicích Sportovní a Pionýrská, kudy jsem denně autem jezdil a začal jsem si všímat jakým způsobem je provoz řízen a jak by se to či ono dalo zlepšit. Finálně vybraný dopravní uzel pak tvořily dvě navazující křižovatky, na kterých jsem strávil i dost času pozorováním a studováním jejich provozu. Sledoval jsem jak systém řízení, tak hustoty provozu ve špičce i mimo ni.

Při návrhu systému jsem se snažil o co největší usnadnění provozu na vybraných křižovatkách a o co největší plynulost provozu i při vysoké hustotě provozu. Hlavní snahou bylo vypořádat se ze zvýšenou hustotou provozu, která nastává hlavně na příjezdu směrem do města. Dalším problémem může být ne příliš velká vzdálenost obou křižovatek, takže se zde může vytvořit zácpa a tím pádem by hrozilo zhoršení průjezdnosti. Tyto problémy jsem vyřešil adaptivním řízením intervalů semaforů právě v závislosti na hustotě dopravy. Dále jsem se snažil do systému zakomponovat i další pomocné prostředky i takové, které si myslím, že se v současnosti ani nepoužívají. Systém zjednodušuje také průjezd vozidel s právem přednostní jízdy a využívá proměnné dopravní značky.

Při návrhu systému jsem si vyzkoušel hned dva nástroje *UPPAAL* a *TimesTool* s takovým výsledkem, že kompletní model jsem vytvořil v *TimesTool*, který je zcela jistě propracovanější a mě se s ním pracovalo opravdu příjemně až na pár chybek co se v něm objevilo, ale to je pravděpodobně dáno tím, že to je zatím beta verze. Systém jsem tedy navrhnul, potom přesně specifikoval a namodeloval v *TimesTool*. Následovala formální verifikace, kterou jsem si vyzkoušel opět jak v UPPAAL tak v *TimesTool* opět s tím výsledkem, že v *TimesTool* mi to bylo příjemnější a tak jsem v něm verifikoval celý model a v UPPAAL jsem vyzkoušel spíše jen základy.

V souvislosti s implementací jsem provedl i malou studii několika *RT* operačních systémů (brickOs, *SHARK, QNX, uC/OS-II*). Každý z těchto systémů jsem vyzkoušel a nakonec se rozhodl pro *uC/OS-II*, který je z mnou prostudovaných pravděpodobně nejvhodnější pro implementaci našeho systému. Systém se mi podařilo kompletně implementovat a rozchodit.

Dále jsem do systému zakomponoval vlastní simulační úlohy, pomocí kterých se mi podařilo ověřit správnou funkci všech navržených a implementovaných podpůrných řešení provozu. Při simulaci generuji provoz jak ve špičce tak mimo ni a nechávám systém normálně běžet a archivuji stav dopravy, ze kterého pak byly sestaveny grafy, ze kterých je jasně vidět že systém inteligentního usměrnění provozu opravdu funguje.

Systém by mohl být do budoucna ještě rozšířen o napojení přímo na dispečink řízení dopravy. Mohly by se také na křižovatky napojit kamery, které by snímaly neustále prostor křižovatek a k tomu by byly schopné být napojeny na řídící inteligentní systém a ve spolupráci s tím by pořizovaly záběry řidičů, kteří by snad projížděli křižovatkou na červenou. Další možnost rozšíření bych viděl v připojení ještě jedné velké křižovatky a to konkrétně světelné křižovatky na rohu ulic Štefánikova a Pionýrská. To by ovšem kvůli způsobu jakým je systém řešen vyžadovalo předělání kompletního modelu systému. Pak by se dle tohoto nového modelu změnila struktura zdrojového kódu aplikace.

Práce na tomto systému mě bavila a dozvěděl jsem se i dost nových věcí ohledně řízení provozu. Věřím, že můj systém by bylo možné bez problémů zprovoznit na nějakém vestavěném systému (uC/OS-II je totiž velice dobře portovatelný) a použít v praxi.

Seznam zkratek

OS – Operační systém.

RT – Real time.

RTOS – Real time operační systém.

RIS – Řídící a informační systém.

DPMB – Dopravní podnik města Brna.

RCX – mikrokontroler od Lego mindstorms

SSZ – Světelné signalizační zařízení.

OOM – Ochrana osob a majetku.

GUI – Grafické uživatelské rozhraní.

Literatura a použité zdroje

[1] Dopravní podnik města Brna. Řídící a informační systém brněnské MHD [cit. 2008-11-25] < http://www.dpmb.cz/ris.asp >

[2] Firma ELTODO, řízení provozu ve městech [cit. 2008-08-10] <http://www.eltodo.cz/produkty-a-sluzby/vyrobni-program/Doprava_IS_RizeniDopravy.pdf>

[3] BrickOS. Web systému BrickOS [cit. 2008-7-5] < http://brickos.sourceforge.net/ >

[4] SHARK. Web syst0mu SHARK [cit. 2008-7-10] < http://shark.sssup.it/ >

[5] Jaroslav Pajer, Diplomová práce, Operační systém QNX, FIT VUT v Brně 2003

[6] QNX, Web systému QNX. [cit. 2008-10-11] < http://www.qnx.com/ >

[7] uC OS/II, Web systému uC OS/II [cit. 2008-09-05] < http://www.micrium.com/products/rtos/kernel/rtos.html >

[8] uC OS/II, User manual < uCOS-II_The_Real-Time_Kernel.pdf > na CD

[9] mapy.cz, Webové stránky mapy.cz < http://www.mapy.cz >

[10] fa.Eltodo, Web produktů firmy Eltodo [cit. 2009-01-05] < http://www.eltodo.cz/produkty-a-sluzby/vyrobni-program/Doprava_promen_znacky_LED.pdf >

[11] Ing. Josef Strnadel Ph.D, Studijní opora předmětu ROS 2008

[12] Uppsala University, Web nástroje UPPAAL [cit. 2008-08-03] < http://www.uppaal.com >

[13] Uppsala University, Web nástroje TIMESTOOL [cit. 2008-08-05] < http://www.timestool.com >

[14] Ing. Aleš Smrčka, Úvod do formální verifikace [cit. 2009-01-10]
< http://www.fit.vutbr.cz/~smrcka/fav/guide/.cs >

Seznam příloh

- Vytisknutý model systému v nástroji TimesTool
- CD se zdrojovými kódy aplikace, LOGovými soubory mého systému, grafy analýzy, se systémem *uC/OS-II* a všemi potřebnými zdroji.

Struktura přiloženého CD:

/publikace – DP v elektronické podobě

/nastroje – nástroje UPPAAL a TimesTool a soubory modelů

/source – zdrojový kód aplikace

/model – hlavní model systému

/LOG – logovací soubory a xls soubor s grafy a jejich daty

/uCOSII – RT systém uCOSII